CSLR51 – Database Management Systems Laboratory

Session: 2

FILE PROCESSING

1. Develop an implementation package using 'C' program to process a FILE containing student details for the given queries.

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

#define MAX_STUDENTS 100
#define MAX_COURSES 4
#define NAME_LENGTH 50
#define DEPT_LENGTH 20

typedef struct {
   char course_name[NAME_LENGTH];
   int credits;
   int grade;
} Course;

typedef struct {
   int roll_no;
   char name[NAME_LENGTH];
   char dept[DEPT_LENGTH];
   Course courses[MAX_COURSES];
   int num_courses;
   float gpa;
} Student;

Student students[MAX_STUDENTS];
int student_count = 0;

void load_students(const char* filename) {
```

```
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Failed to open file");
        return;
    }
    student_count = 0;
    while (fscanf(file, "%d,%49[^,],%19[^,],",
&students[student_count].roll_no,
students[student_count].name,
students[student_count].dept) != EOF) {
        students[student_count].num_courses = 0;
        for (int i = 0; i < MAX_COURSES; ++i) {
            if (fscanf(file, "%49[^,],%d,%d,",
students[student_count].courses[i].course_name,
&students[student_count].courses[i].credits,
&students[student_count].courses[i].grade) == EOF) {
                break;
            }
            students[student_count].num_courses++;
        }
        student_count++;
    }
    fclose(file);
}

void save_students(const char* filename) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Failed to open file");
        return;
    }
    for (int i = 0; i < student_count; ++i) {
        fprintf(file, "%d,%s,%s,", students[i].roll_no,
students[i].name, students[i].dept);
```

```
        for (int j = 0; j < students[i].num_courses; ++j)
{
            fprintf(file, "%s,%d,%d,",
students[i].courses[j].course_name,
students[i].courses[j].credits,
students[i].courses[j].grade);
        }
        fprintf(file, "\n");
    }
    fclose(file);
}

void calculate_gpa(Student* student) {
    int total_credits = 0;
    int total_points = 0;
    for (int i = 0; i < student->num_courses; ++i) {
        total_credits += student->courses[i].credits;
        total_points += student->courses[i].credits *
student->courses[i].grade;
    }
    student->gpa = (float)total_points / total_credits;
}

void add_student() {
    if (student_count >= MAX_STUDENTS) {
        printf("Maximum number of students reached.\n");
        return;
    }
    Student new_student;
    printf("Enter roll number: ");
    scanf("%d", &new_student.roll_no);
    printf("Enter name: ");
    scanf("%s", new_student.name);
    printf("Enter department: ");
    scanf("%s", new_student.dept);
```

```
    printf("Enter number of courses (3-4): ");
    scanf("%d", &new_student.num_courses);
    for (int i = 0; i < new_student.num_courses; ++i) {
        printf("Enter course name: ");
        scanf("%s", new_student.courses[i].course_name);
        printf("Enter course credits: ");
        scanf("%d", &new_student.courses[i].credits);
        printf("Enter course grade: ");
        scanf("%d", &new_student.courses[i].grade);
    }
    calculate_gpa(&new_student);
    students[student_count++] = new_student;
}

void delete_course(int roll_no) {
    for (int i = 0; i < student_count; ++i) {
        if (students[i].roll_no == roll_no) {
            if (students[i].num_courses > 3) {
                students[i].num_courses--;
                printf("Course deleted.\n");
                return;
            } else {
                printf("Cannot delete course. Student must
have at least 3 courses.\n");
                return;
            }
        }
    }
    printf("Student not found.\n");
}

void insert_course(int roll_no) {
    for (int i = 0; i < student_count; ++i) {
        if (students[i].roll_no == roll_no) {
            if (students[i].num_courses < 4) {
```

```
                printf("Enter new course name: ");
                scanf("%s",
students[i].courses[students[i].num_courses].course_name)
;
                printf("Enter new course credits: ");
                scanf("%d",
&students[i].courses[students[i].num_courses].credits);
                printf("Enter new course grade: ");
                scanf("%d",
&students[i].courses[students[i].num_courses].grade);
                students[i].num_courses++;
                printf("Course added.\n");
                return;
            } else {
                printf("Cannot add course. Student already
has 4 courses.\n");
                return;
            }
        }
    }
    printf("Student not found.\n");
}

void update_course_name(int roll_no, const char*
old_name, const char* new_name) {
    for (int i = 0; i < student_count; ++i) {
        if (students[i].roll_no == roll_no) {
            for (int j = 0; j < students[i].num_courses;
++j) {
                if
(strcmp(students[i].courses[j].course_name, old_name) ==
0) {

strcpy(students[i].courses[j].course_name, new_name);
                    printf("Course name updated.\n");
```

```
                return;
            }
        }
    }
    printf("Course not found for the student.\n");
}

void upgrade_grade_point(int roll_no) {
    for (int i = 0; i < student_count; ++i) {
        if (students[i].roll_no == roll_no) {
            for (int j = 0; j < students[i].num_courses;
++j) {
                if (students[i].courses[j].grade == 7) {
                    students[i].courses[j].grade = 8;
                    printf("Grade upgraded.\n");
                    calculate_gpa(&students[i]);
                    return;
                }
            }
        }
    }
    printf("Student or grade not found.\n");
}

void generate_grade_report(int roll_no) {
    for (int i = 0; i < student_count; ++i) {
        if (students[i].roll_no == roll_no) {
            printf("Roll No: %d\n", students[i].roll_no);
            printf("Name: %s\n", students[i].name);
            printf("Department: %s\n", students[i].dept);
            for (int j = 0; j < students[i].num_courses;
++j) {
                printf("Course: %s, Credits: %d, Grade:
%d\n", students[i].courses[j].course_name,
```

```
students[i].courses[j].credits,
students[i].courses[j].grade);
            }
            printf("GPA: %.2f\n", students[i].gpa);
            return;
        }
    }
    printf("Student not found.\n");
}

int main() {
    const char* filename = "students.txt";
    load_students(filename);

    int choice, roll_no;
    char old_name[NAME_LENGTH], new_name[NAME_LENGTH];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert student records\n");
        printf("2. Calculate GPA for all students\n");
        printf("3. Delete a course for a student\n");
        printf("4. Insert a new course for a student\n");
        printf("5. Update course name for a student\n");
        printf("6. Upgrade grade point for a student\n");
        printf("7. Generate grade report for a
student\n");
        printf("8. Save and Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_student();
                break;
```

```
            case 2:
                for (int i = 0; i < student_count; ++i) {
                    calculate_gpa(&students[i]);
                }
                printf("GPA calculated for all
students.\n");
                break;
            case 3:
                printf("Enter roll number of the student:
");
                scanf("%d", &roll_no);
                delete_course(roll_no);
                break;
            case 4:
                printf("Enter roll number of the student:
");
                scanf("%d", &roll_no);
                insert_course(roll_no);
                break;
            case 5:
                printf("Enter roll number of the student:
");
                scanf("%d", &roll_no);
                printf("Enter old course name: ");
                scanf("%s", old_name);
                printf("Enter new course name: ");
                scanf("%s", new_name);
                update_course_name(roll_no, old_name,
new_name);
                break;
            case 6:
                printf("Enter roll number of the student:
");
                scanf("%d", &roll_no);
                upgrade_grade_point(roll_no);
```

```
                break;
            case 7:
                printf("Enter roll number of the student:
");
                scanf("%d", &roll_no);
                generate_grade_report(roll_no);
                break;
            case 8:
                save_students(filename);
                printf("Data saved. Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try
again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
Menu:
1. Insert student records
2. Calculate GPA for all students
3. Delete a course for a student
4. Insert a new course for a student
5. Update course name for a student
6. Upgrade grade point for a student
7. Generate grade report for a student
8. Save and Exit
```

STRUCTURED QUERY LANGUAGE

1. Create a Student schema using the student details given in Q.No.1 and execute the following basic queries.

Create the Student schema excluding Course_credit and Course_grade and define the necessary constraints

```
CREATE TABLE Student (
     Std_rollno INT PRIMARY KEY,
     Std_name VARCHAR(50),
     Dept CHAR(10),
     Course1 CHAR(50),
     Course2 CHAR(50),
     Course3 CHAR(50),
     Course4 CHAR(50)
);
```

a. Insert at least 5 student records into the Student table.

```
     INSERT INTO Student (Std_rollno, Std_name, Dept,
Course1, Course2, Course3, Course4) VALUES
(1, 'John Doe', 'CSE', 'Math', 'Physics', 'Chemistry',
'Computer Science'),
(2, 'Jane Smith', 'ECE', 'Math', 'Electronics', 'Digital
Logic', 'Networks'),
(3, 'Alice Johnson', 'ME', 'Thermodynamics', 'Mechanics',
'Physics', 'Chemistry'),
(4, 'Bob Brown', 'CIVIL', 'Civil Engineering', 'Math',
'Physics', 'Structural Analysis'),
(5, 'Charlie Davis', 'EEE', 'Circuits', 'Electronics',
'Control Systems', 'Signals');
```

```
mysql> INSERT INTO Student (Std_rollno, Std_name, Dept, Course1, Course2, Course
3, Course4) VALUES
    -> (1, 'John Doe', 'CSE', 'Math', 'Physics', 'Chemistry', 'Computer Science'
),
    -> (2, 'Jane Smith', 'ECE', 'Math', 'Electronics', 'Digital Logic', 'Network
s'),
    -> (3, 'Alice Johnson', 'ME', 'Thermodynamics', 'Mechanics', 'Physics', 'Che
mistry'),
    -> (4, 'Bob Brown', 'CIVIL', 'Civil Engineering', 'Math', 'Physics', 'Struct
ural Analysis'),
    -> (5, 'Charlie Davis', 'EEE', 'Circuits', 'Electronics', 'Control Systems',
 'Signals');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

b. Delete Course2 and Course3 attributes from the Student table.

```
ALTER TABLE Student DROP COLUMN Course2;
ALTER TABLE Student DROP COLUMN Course3;
```

```
mysql> ALTER TABLE Student DROP COLUMN Course2;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Student DROP COLUMN Course3;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

c. Insert two new columns DoB and email into the Student table.

```
ALTER TABLE Student ADD DoB DATE NOT NULL;
ALTER TABLE Student ADD email VARCHAR(50) CHECK
(email LIKE '%@nitt.edu');
```

```
mysql> ALTER TABLE Student ADD email VARCHAR(50) CHECK (email LIKE '%@nitt.edu')
;
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Student ADD DoB DATE;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

d. Change Course1 datatype to varchar2.

```
ALTER TABLE Student MODIFY Course1 VARCHAR2(50);
```

```
mysql> ALTER TABLE Student MODIFY Course1 VARCHAR(50);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

e. Update the column name ‘Std_rollno’ to ‘Std_rno’.

```
ALTER TABLE Student RENAME COLUMN Std_rollno TO Std_rno;
```

```
mysql> ALTER TABLE Student RENAME COLUMN Std_rollno TO Std_rno;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

f. Update all student records who pursue a course named “DBMS” to “OS”.

```
UPDATE Student SET Course1 = 'OS' WHERE Course1 = 'DBMS';
UPDATE Student SET Course4 = 'OS' WHERE Course4 = 'DBMS';
```

```
mysql> UPDATE Student SET Course4 = 'OS' WHERE Course4 = 'DBMS';
Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

g. Delete a student record with student name starting with letter ‘S’.

```
DELETE FROM Student WHERE Std_name LIKE 'S%';
```

```
mysql> DELETE FROM Student WHERE Std_name LIKE 'S%';
Query OK, 0 rows affected (0.00 sec)
```

h. Display all records in which a student has born after the year 2005.

```
SELECT * FROM Student WHERE YEAR(DoB) > 2005;
```

```
mysql> SELECT * FROM Student WHERE YEAR(DoB) > 2005;
Empty set (0.00 sec)
```

i. Simulate RENAME, TRUNCATE and DROP.

```
ALTER TABLE Student RENAME TO StudentDetails;
```

```
mysql> ALTER TABLE Student RENAME TO StudentDetails;
Query OK, 0 rows affected (0.02 sec)
```

```
TRUNCATE TABLE StudentDetails;
```

```
mysql> TRUNCATE TABLE StudentDetails;
Query OK, 0 rows affected (0.03 sec)
```

```
DROP TABLE StudentDetails;
```

```
mysql> DROP TABLE StudentDetails;
Query OK, 0 rows affected (0.02 sec)
```