Harpreet Kaur

Database Design and Implementation

College Enrollment System Case-Study

Table of Content

Introduction

The **College Enrollment System** is designed to simplify the academic management process by efficiently handling student enrollment, faculty assignments, course scheduling, and student performance in exams. This case study presents the database design and implementation, covering table structures, relationships, complex queries, and views.

Mission and Objectives

Mission

To simplify enrollment and academic management while ensuring accurate data handling.

Objectives

- Maintain accurate records of student details, course schedules, and faculty assignments.
- Enable payments and fee management.
- Monitor performance and generate detailed reports.

Database Design

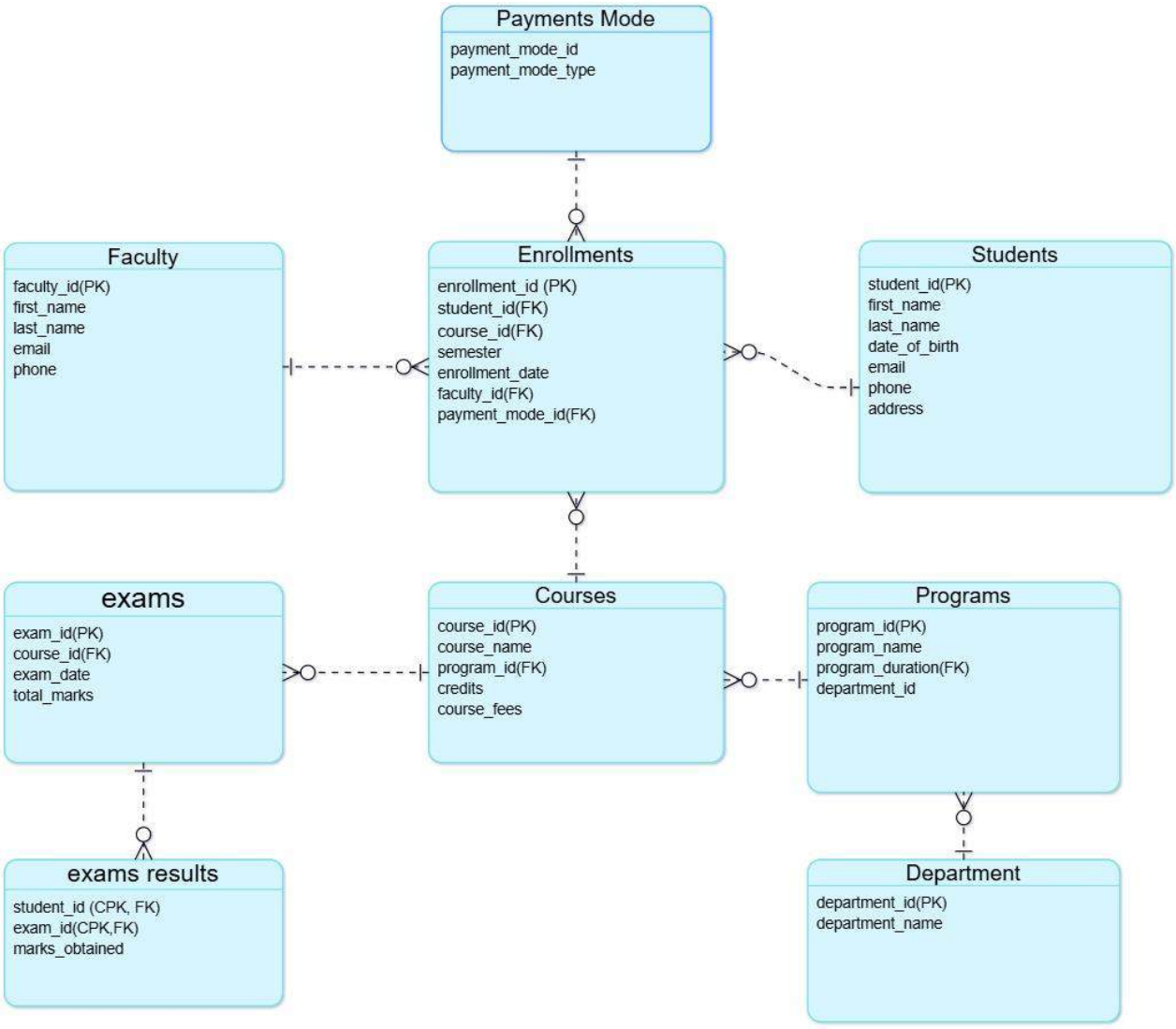
Entities

Entities used in this database are as follows:

- **Students** – Stores student details such as name, contact information, and address.
- **Payment mode** – Records distinct payment modes available for payment.
- **Department -** Stores the details about the various academic departments in the college.
- **Programs** – Defines the academic programs offered by the college (e.g., BSc, BTech).
- **Faculty** – Stores the details of the faculty members who teach courses in the college.
- **Courses** – Stores information about courses within different programs.
- **Exams**– Stores information about the exams associated with each course.
- **Exam Results -** Stores the results of students' exams.
- **Enrollments** – Stores information about the courses each student is enrolled in.

Entity-Relationship Diagram (ERD)

The **ER Diagram** represents the key entities of the system and their relationships.



Relationships

1. Students and Enrollments (1:M)

- **Relationship:** One student can have multiple enrollments, but each enrollment belongs to only one student.
- **Reason:** A student may enroll in multiple courses across different semesters.

2. Courses and Enrollments (1:M)

- **Relationship:** One course can have multiple enrollments, but each enrollment is for only one course.
- **Reason:** Many students can enroll in the same course.

3. Faculty and Enrollments (1:M)

- **Relationship:** One faculty member can oversee multiple enrollments, but each enrollment is handled by one faculty member.
- **Reason:** Faculty members may be assigned to multiple students in various courses.

4. Payments Mode and Enrollments (1:M)

- **Relationship:** One payment mode can be used in multiple enrollments, but each enrollment has only one payment mode.
- **Reason:** Students may pay through different payment methods such as credit card, cash, or online transactions.

5. Programs and Courses (1:M)

- **Relationship:** One program can include multiple courses, but each course belongs to only one program.
- **Reason:** Programs (like Computer Science or Business) consist of multiple courses.

6. Students and Exam Results (1:M)

- **Relationship:** One student can have multiple exam results, but each exam result belongs to only one student.
- **Reason:** A student appears for multiple exams in different courses.

7. Exams and Exam Results (1:M)

- **Relationship:** One exam can generate multiple results, but each result is linked to a single exam.
- **Reason:** Many students attempt the same exam, and each has a separate result.

8. Courses and Exams (1:M)

- **Relationship:** One course can have multiple exams, but each exam belongs to only one course.
- **Reason:** Each course can have multiple exams throughout the semester.

9. Department and Programs (1:M)

- **Relationship:** One department can have multiple programs, but each program belongs to only one department.
- **Reason:** A department (such as Engineering or Science) can offer multiple academic programs.

Data Dictionary

The **relational model** of the College Enrollment System consists of multiple tables, each designed to store structured information and maintain referential integrity. Below is a description of the **key tables** and their relationships:

1. Students Table

Stores student details, including personal information.

This table holds all essential student information, ensuring that every student has a unique `Student_ID`.

- **Primary Key:** `Student_ID`
- **Not null Constraint :** `first_name, last_name, dob, email, phone, address`
- **Unique Constraint :** `email`

Column Name	Data Type	Description
student_id	INT	Unique identifier for each student. It is the primary key.
first_name	VARCHAR	Student's first name.
last_name	VARCHAR	Student's last name.
dob	DATE	Date of birth of the student.
email	VARCHAR	Student's email address.
phone	VARCHAR	Student's phone number.
address	TEXT	Student's residential address.

2. Payment mode Table

Stores information about students who are enrolled in the college.

- **Primary Key:** `Payment_ID`
- **Not null Constraint :** `payment_mode_type`
- **Unique Constraint :** `payment_mode_type`

Column Name	Data Type	Description
payment_mode_id	INT	Unique identifier for each payment mode. It is the primary key.
payment_mode_type	VARCHAR	Type of payment (e.g., Credit Card, PayPal, Bank Transfer).

3. Department Table

Stores the details about the various academic departments in the college.

- **Primary Key:** `department_id`
- **Not null Constraint :** `department_name`
- **Unique Constraint :** `department_name`

Column Name	Data Type	Description
department_id	INT	Unique identifier for each department. It is the primary key.
department_name	VARCHAR	Name of the department (e.g., Computer Science, Mathematics).

4. Programs Table

Stores information about the academic programs (e.g., BSc, BTech) offered by the college.

- **Primary Key:** `program_id`
- **Foreign Key:** `department_id` references *Department Table* `department_id`
- **Not null Constraint :** `program_name`
- **Unique Constraint :** `program_name`

Column Name	Data Type	Description
program_id	INT	Unique identifier for each program. It is the primary key.
program_name	VARCHAR	Name of the program (e.g., BSc Computer Science, BTech Electrical Engineering).
program_duration_year	INT	Duration of the program in years (e.g., 3 years, 4 years).
department_id	INT	Foreign key linking to the `department` table.

5. Faculty Table

Stores information about faculty members who teach courses.

- **Primary Key:** `Faculty_ID`
- **Not null Constraint :** `first_name, last_name, email, phone`
- **Unique Constraint :** `email, phone`

Column Name	Data Type	Description
faculty_id	INT	Unique identifier for each faculty member. It is the primary key.
first_name	VARCHAR	Faculty member's first name.
last_name	VARCHAR	Faculty member's last name.
email	VARCHAR	Faculty member's email address.
phone	VARCHAR	Faculty member's phone number.

6. Courses Table

Stores details about the courses offered by each program in the college.

Each course belongs to a program (e.g., BSc Computer Science), ensuring a structured academic hierarchy.

- **Primary Key:** `Course_ID`
- **Foreign Key:** `Program_ID` references *Programs table* `Programs_id`
- **Not null Constraint :** `course_name`
- **Unique Constraint :** `course_name`
- **Default :** *True* for `credit` column

Column Name	Data Type	Description
course_id	INT	Unique identifier for each course. It is the primary key.
course_name	VARCHAR	Name of the course (e.g., Data Structures, Calculus).
program_id	INT	Foreign key linking to the `programs` table.
credit	BOOLEAN	Indicates whether the course carries credit (TRUE/FALSE).
course_fee	DECIMAL	Fee for the course.

7. Exam Table

Stores information about the exams associated with each course.

- **Primary Key:** `exam_id`
- **Foreign Key:** `course_id` references *Course Table* `course_id`
- **Not null Constraint:** `course_id`

Column Name	Data Type	Description
exam_id	INT	Unique identifier for each exam. It is the primary key.
course_id	INT	Foreign key linking to the `courses` table.
exam_date	DATE	Date of the exam.
total_marks	INT	Total marks for the exam.

8. Exam Result Table

Stores the results of students' exams.

- **Composite Primary Key:** `student_id` and `exam_id`
- **Foreign Key:** `student_id` references *Student Table* `student_id`.
  `exam_id` references *Student Table* `exam_id`

Column Name	Data Type	Description
student id	INT	Foreign key linking to the students table.
exam id	INT	Foreign key linking to the exams table.
marks obtained	INT	Marks obtained by the student in the exam.

9. Enrollment Table (transaction table)

Stores information about the courses each student is enrolled in.

- **Composite Primary Key:** `student_id` and `exam_id`
- **Foreign Key:** `student_id` references *Student Table* `student_id`.
  `course_id` references *Course Table* `course_id`
  `faculty_id` references *Faculty Table* `faculty_id`
  `payment_mode_id` references *Payment_mode Table* `payment_mode_id`
- **Not null Constraint :** `student_id, course_id`
- **Default :** *curdate() function* for `enrollment_date` column

Column Name	Data Type	Description
enrollment_id	INT	Unique identifier for each enrollment record.
student_id	INT	References the student_id from the Students table, indicating which student is enrolled.
course_id	INT	References the course_id from the Courses table, indicating the enrolled course.
semester	VARCHAR(10)	Represents the semester in which the student is enrolled (e.g., "Fall 2024").
enrollment_date	DATE	The date when the student enrolled in the course.
faculty_id	INT	References the faculty_id from the Faculty table, indicating which faculty member is responsible.
payment_mode_id	INT	References the payment_mode_id from the Payments Mode table, indicating the payment method used for enrollment.

Queries to create Tables

Queries to create tables are as follows :

-- students table :

```
create table students (
student_id int(10) primary key,
first_name varchar(50) not null,
```

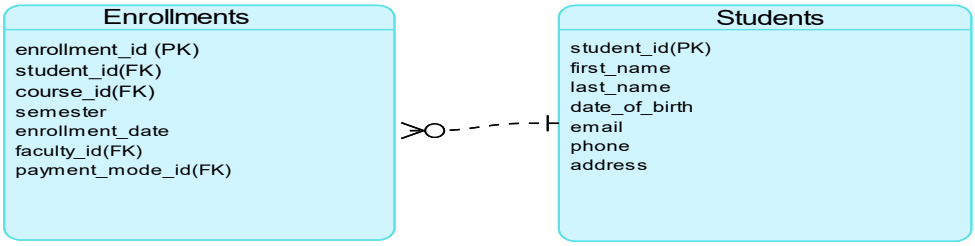
```
last_name varchar(50) not null,
dob date not null,
email varchar(120) not null unique,
phone varchar(17) not null,
address varchar(250) not null
);

-- payment mode table :

create table payment_mode (
payment_mode_id int(2) primary key auto_increment,
payment_mode_type varchar(30) unique not null
);

-- department table :

create table department (
department_id int(2) primary key auto_increment,
department_name varchar(50) not null unique);

-- programs table :

create table programs (
program_id int(4) primary key,
program_name varchar(150) not null unique,
program_duration_year int(1) ,
department_id int(2) ,
foreign key (department_id) REFERENCES department(department_id)
);

-- faculty table :

create table faculty(
faculty_id int(10) primary key auto_increment,
first_name varchar(50) not null ,
last_name varchar(50) not null,
email varchar(100) not null unique,
phone varchar(17) not null unique );

-- courses table :

create table courses (
course_id int(10) primary key ,
course_name varchar(150) not null unique,
program_id int(3) ,
credit boolean default(true),
course_fee int(5) ,
```

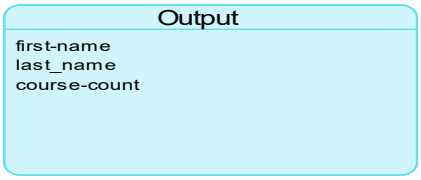
```
foreign key (program_id) references programs(program_id)
);

-- exams table :

create table exams(
exam_id int(15) primary key auto_increment ,
course_id int(10) not null,
exam_date date ,
total_marks int(3),
foreign key (course_id) references courses (course_id)
);

-- exam result table :

create table exam_result(
student_id int(10) ,
exam_id int(15),
marks_obtained int(3),
primary key(student_id , exam_id), -- composite primary key
foreign key (student_id) references students(student_id),
foreign key (exam_id) references exams(exam_id)
);

-- Enrollment table(transaction table) :

create table enrollments(
enrollment_id int(15) primary key auto_increment,
student_id int(10) not null,
course_id int(10) not null,
semester int(2) ,
enrollment_date date default(CURDATE()),
faculty_id int(10) ,
payment_mode_id int(2),
UNIQUE (student_id, course_id),
foreign key (student_id) references students(student_id),
foreign key (course_id) references courses (course_id),
foreign key (faculty_id) references faculty(faculty_id),
foreign key (payment_mode_id) references payment_mode(payment_mode_id)
);
```

Advanced SQL Queries for College Enrollment System

1. Query to list students who have enrolled in multiple courses:

Table used: Enrollments, students





The following query retrieves students who are enrolled in **more than one course**, displaying their first name, last name, and total course count. It joins the **Students** and **Enrollments** tables, groups by `student_id`, and filters results using the `HAVING` clause to show only students with multiple enrollments.

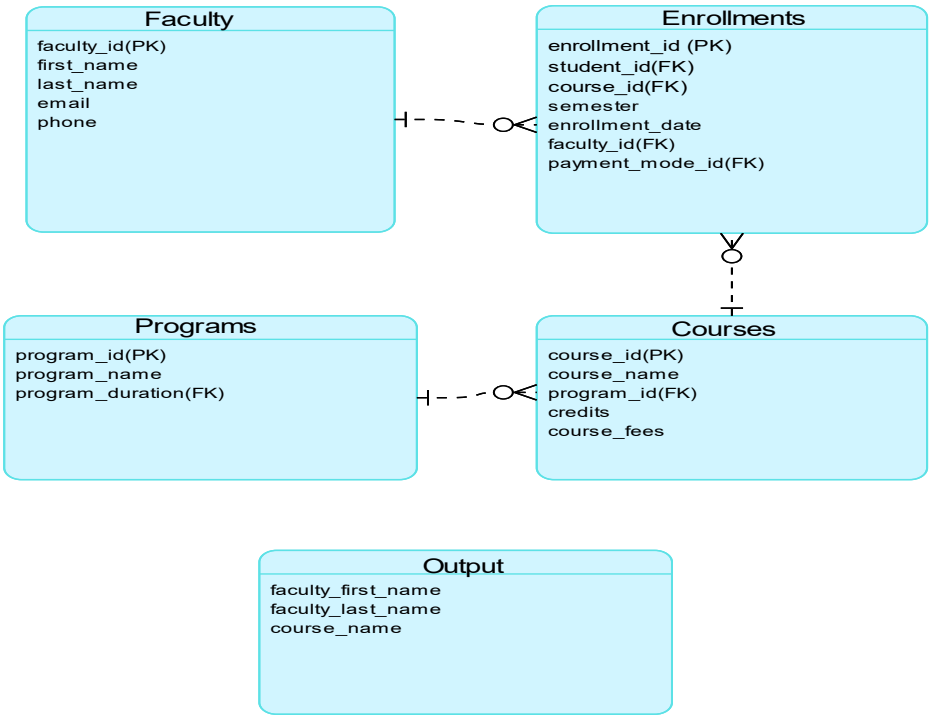
```
212    --Query to List Students Who Have Enrolled in Multiple Courses
213    SELECT s.first_name, s.last_name, COUNT(e.course_id) AS course_count
214    FROM enrollments e
215    JOIN student s ON e.student_id = s.student_id
216    GROUP BY s.student_id
217    HAVING COUNT(e.course_id) > 1;
218
219    --Query to Get Faculty Information for Courses in a Specific Program
```

Results | Messages

	first_name	last_name	course_count
1	Nikhil	Choudhary	4
2	Mohan	Smith	3
3	Sharanpreet	Kaur	3
4	Jaspreet	Singh	4
5	Akshar	Sharma	2
6	Vishal	Wilson	3
7	Harpreet	Kaur	3
8	Sashi	Sharma	2

2. Query to get faculty information for courses in a specific program

- **Table used:** Enrollments, students



This query retrieves the **first and last names of faculty members** along with the **course names** they teach in the **BSc Computer Science** program.

- It joins the **Faculty**, **Enrollments**, **Courses**, and **Programs** tables based on their relationships.
- The `WHERE` clause filters results to include only courses from the **BSc Computer Science** program.

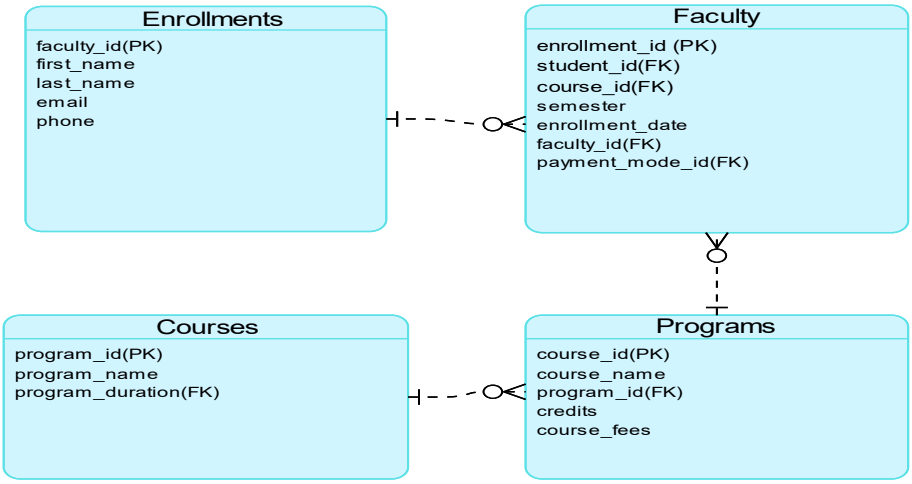
```
219  --Query to Get Faculty Information for Courses in a Specific Program
220  SELECT DISTINCT f.first_name AS faculty_first_name, f.last_name AS faculty_last_name, c.course_name
221  FROM faculty f
222  JOIN enrollments en ON f.faculty_id = en.faculty_id
223  JOIN courses c ON en.course_id = c.course_id
224  JOIN programs p ON c.program_id = p.program_id
225  WHERE p.program_name = 'BSc Computer Science';
226
227  -- Query to Find the Total Fees Collected for Each Program
```

Results | Messages

	faculty_first_name	faculty_last_name	course_name
1	Junaid	Qazi	Data Structures and Algorithms
2	Priyanka	Miller	Introduction to Python

3. Query to find the total fees collected for each program

- **Table used:** Enrollments, students



This query calculates the **total fees collected** for each academic program.

- It joins the **Enrollments**, **Courses**, and **Programs** tables to link students' enrollments with their respective courses and programs.
- The `SUM(c.course_fee)` function computes the total fees collected per program.
- The `GROUP BY p.program_name` ensures the total fees are grouped by each program.

```
226
227    -- Query to Find the Total Fees Collected for Each Program
228    SELECT p.program_name, SUM(c.course_fee) AS total_fees_collected
229    FROM enrollments e
230    JOIN courses c ON e.course_id = c.course_id
231    JOIN programs p ON c.program_id = p.program_id
232    GROUP BY p.program_name;
```

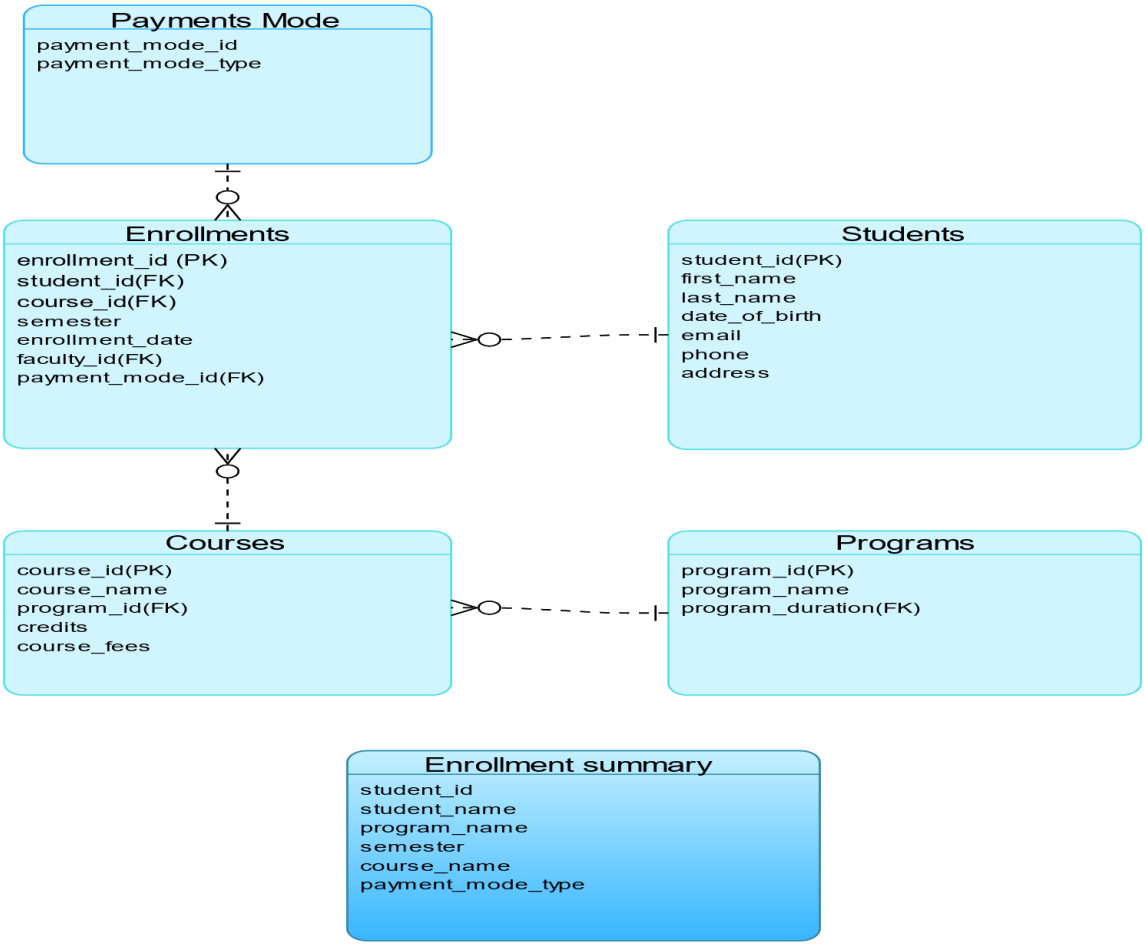
Results | Messages

	program_name	total_fees_collected
1	BSc Biology	15600
2	BSc Chemistry	19200
3	BSc Computer Science	15100
4	BSc Mathematics	9200
5	BSc Physics	22000
6	BTech Artificial Intelligence	15300
7	BTech Astrophysics	15900
8	BTech BioMechanical Engineering	10000

Views

STUDENT ENROLLMENT SUMMARY VIEW

Tables used :



Scenario: Track student enrollments by course, program, semester and payment mode.
Purpose: Monitor students enrollment and specify reporting

Syntax :

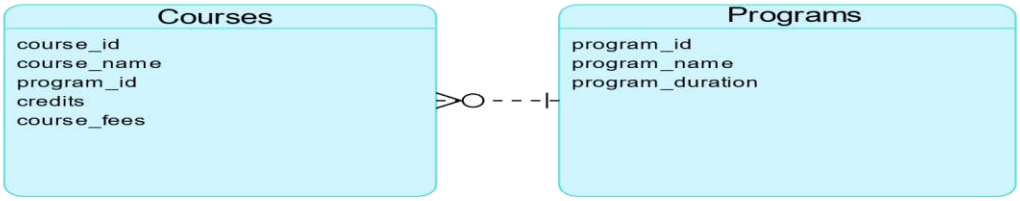
```
CREATE VIEW enrollment_summary_view AS
SELECT
    s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name,
    p.program_name,
    e.semester,
    c.course_name,
    pm.payment_mode_type
FROM enrollments e
```

JOIN students s ON e.student_id = s.student_id
JOIN courses c ON e.course_id = c.course_id
JOIN programs p ON c.program_id = p.program_id
JOIN payments_mode pm ON e.payment_mode_id = pm.payment_mode_id;
Explanation:

- **Joins multiple tables** to gather all necessary information.
- **Concatenates first name and last name** for `student_name`.
- **Uses a view** to simplify future queries without needing complex joins

COURSES IN PROGRAM VIEW

Tables used :



Scenario: Display all courses under a specific program.
Purpose: Help students & faculty to view available course in program.

Syntax:

```
CREATE VIEW courses_in_program AS
SELECT
    p.program_id,
    p.program_name,
    c.course_id,
    c.course_name,
    c.course_fees,
    c.credits
FROM courses c
JOIN programs p ON c.program_id = p.program_id;
```

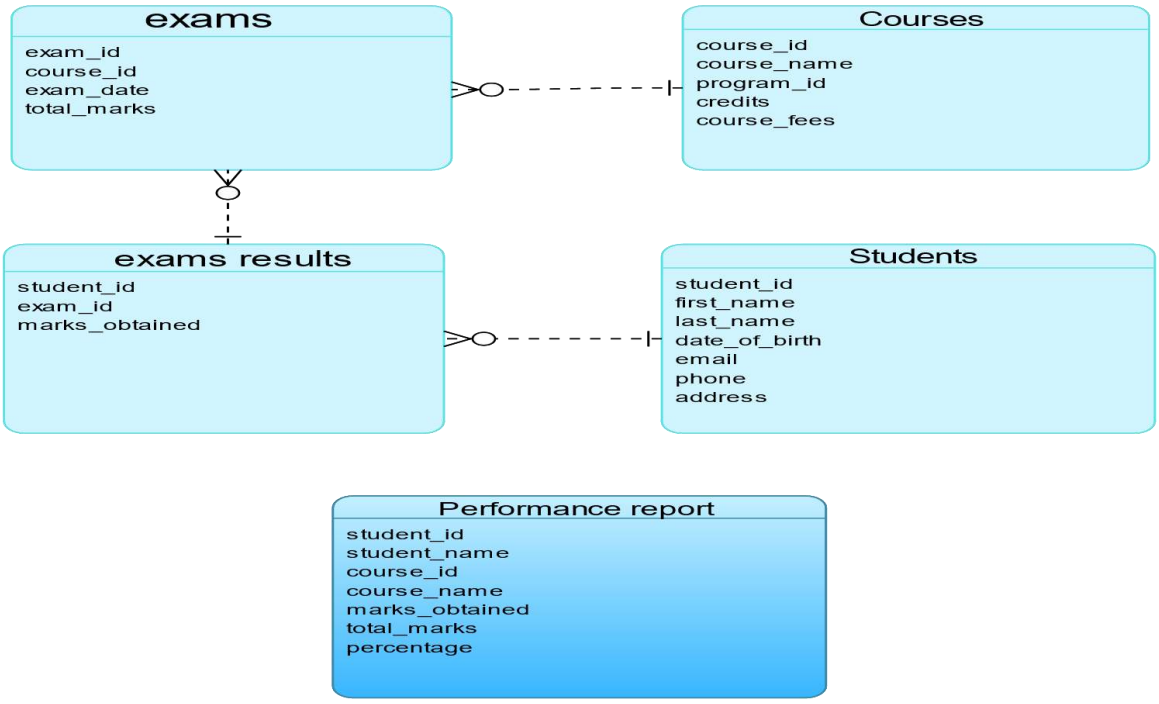
Explanation:

- **Joins `courses` and `programs`** using `program_id` to fetch related data.

- **Includes relevant columns** like program name, course details, fees, and credits.
- **Simplifies queries** for retrieving program-wise course details.

STUDENT PERFORMANCE REPORT

Tables used:



Scenario: Display students exam result along with course details
Purpose: Track student performance in exams Identify top & struggling students Provide insights for academic improvements

Syntax:

```
CREATE VIEW performance_report AS
SELECT
    s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name,
    c.course_id,
    c.course_name,
    er.marks_obtained,
    e.total_marks,
    ROUND((er.marks_obtained / e.total_marks) * 100, 2) AS percentage
FROM students s
JOIN exams_results er ON s.student_id = er.student_id
JOIN exams e ON er.exam_id = e.exam_id
```

JOIN courses c ON e.course_id = c.course_id;

Explanation:

- **Joins the `students`, `exams_results`, `exams`, and `courses` tables** to collect all necessary details.
- **Uses `CONCAT(s.first_name, ' ', s.last_name)`** to display the student's full name.
- **Calculates the percentage** using `ROUND((marks_obtained / total_marks) * 100, 2)`.
- **Includes all key details** such as student ID, name, course details, obtained marks, total marks, and percentage.

Conclusion

The College Enrollment System implements a well-structured database design comprising nine essential tables that effectively manage student enrollments, course offerings, faculty assignments, and examination records. Through carefully defined relationships and constraints, the system maintains data integrity while handling core academic operations. The implementation of advanced SQL queries and specialized views - including enrollment summaries, course listings, and performance reports - demonstrates the system's capability to provide meaningful insights for administrative decision-making. The database successfully achieves its objectives of maintaining accurate records, managing fees, and monitoring student performance, serving as an efficient solution for academic institution management.