CASE STUDY FOR DATA CONCEPT ON

COLLEGE ENROLLMENT SYSTEM

WRITTEN BY :- PATEL AKSHAR NIMESHKUMAR (http://github.com/Axar3010)

SUBMITTED TO :- JUNAID QAZI (https://www.linkedin.com/in/jqazi)

TABLE OF CONTENT

INTRODUCTION

College Enrolments Systems is a platform it is designed to manage the processes related student admissions and registrations. These systems allow colleges and universities to handle large volumes of student data efficiently and provide a seamless experience for both administrators as well as students.

By implementing a College Enrolment System, higher education institutions can attract and retain more students, manage student data efficiently, and optimize resources. These systems provide a centralized platform for managing all aspects of the enrolment process, from initial inquiries to final registration, ensuring a seamless and efficient experience for everyone involved.

MISSION

The goal of College Enrolment System enables students, institutions, and administrators by offering a smooth, effective, and user-friendly platform.

OBJECTIVES

Simplify enrolling Procedures: Reduce paperwork and delays by streamlining the admission, enrolling, and application processes for staff and potential students.

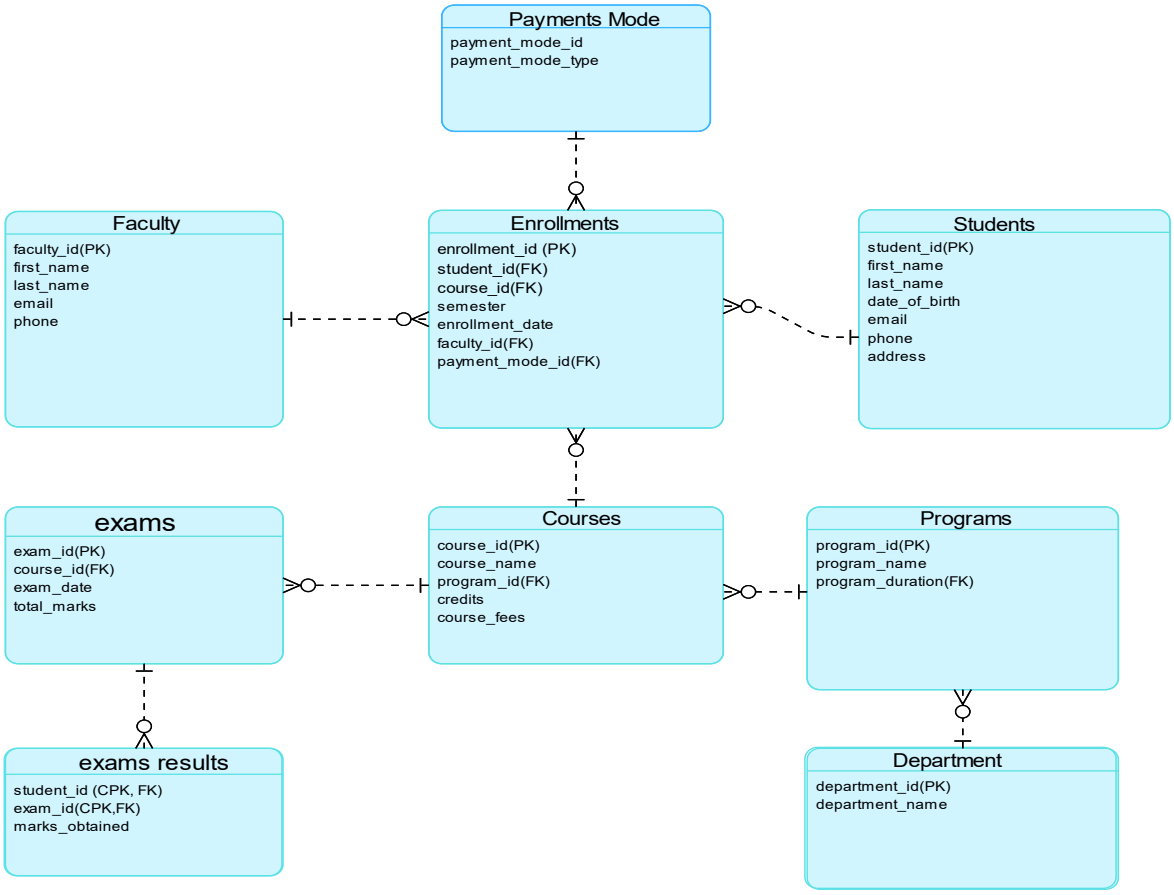
Improve Accessibility: By using an inclusive digital platform, make sure that all students, regardless of background, have fair access to chances for higher education.

Encourage openness: Clearly communicate and update admissions statuses of students show course availability for admission and institutional requirements in real time.

Encourage the Making of Decisions: Provide schools with data-driven insights to enhance resource providing, optimize admissions procedures.

Assure Security and Reliability: Use strong security measures to protect private student information and keep system uptime high to allow for continuous availability.

ENTITY RELATIONSHIP DIAGRAM



RELATIONSHIPS

Students - Enrollments: One-to-Many (A student can enroll in multiple courses).

Courses - Enrollments: One-to-Many (A course can have multiple enrollments).

Courses - Schedules: One-to-One (Each course has a specific schedule).

Faculty - Schedules: One-to-Many (An Faculty can teach multiple courses).

Classrooms -Schedules: One-to-Many (A classroom can host multiple schedules).

Departments - Courses/Instructors: One-to-Many (A department offers multiple courses and has multiple faculties)

https://github.com/Axar3010

DATABASE DESIGN

TABLES :

1) STUDENTS TABLE

	Field	Type	Null	Key	Default	Extra
1	StudentID	b'int'	NO	PRI	*NULL*	auto_inc…
2	FirstName	b'varchar(50)'	YES		*NULL*	
3	LastName	b'varchar(50)'	YES		*NULL*	
4	Email	b'varchar(50)'	YES		*NULL*	
5	Phone	b'varchar(15)'	YES		*NULL*	
6	DateOfBirthday	b'date'	YES		*NULL*	
7	ProgramID	b'int'	YES		*NULL*	

QUERY TO CREATE STUDENT TABLE –

```
CREATE TABLE STUDENT (
StudentID INT AUTO_INCREMENT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Email VARCHAR(100) UNIQUE,
Phone VARCHAR(15),
DateOfBirth DATE,
ProgramID INT,
FOREIGN KEY (ProgramID) REFERENCES PROGRAM(ProgramID)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	StudentID	Id number of student	Integer
2	FirstName	First Name of student/instructor	Variables
3	LastName	Family Name of student/instructor	Variable
4	Email	Email ID of Student/Instructor	Varchar
5	Phone	Contact Number of student/instructor	Varchar
6	DateOfBirth	Birth Date	Integer
7	ProgramID	ID number of program	Integer

2) COURSE TABLE :

	Field	Type	Null	Key	Default	Extra
1	CourseID	b'int'	NO	PRI	*NULL*	auto_inc…
2	CourseName	b'varchar(50)'	YES		*NULL*	
3	Credits	b'int'	YES		*NULL*	
4	InstructorID	b'int'	YES	MUL	*NULL*	
5	ProgramID	b'int'	YES	MUL	*NULL*	

QUERY TO CREATE COURSE TABLE-

```
CREATE TABLE COURSE (
 CourseID INT AUTO_INCREMENT PRIMARY KEY,
 CourseName VARCHAR(100),
 Credits INT,
 InstructorID INT,
 ProgramID INT,
 FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID),
 FOREIGN KEY (ProgramID) REFERENCES PROGRAM(ProgramID)
 );
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	CourseID	ID number of Course	Integer
2	CourseName	Name of Course	Varchar
3	Credits	Number of credits	Integer
4	InstructorID	ID of instructor	Integer
5	ProgramID	ID of Program	Integer

3) PROGRAM TABLE :

	Field	Type	Null	Key	Default	Extra
1	ProgramID	b'int'	NO	PRI	*NULL*	auto_inc…
2	ProgramName	b'varchar(50)'	YES		*NULL*	
3	Duration	b'int'	YES		*NULL*	
4	Department	b'varchar(15)'	YES		*NULL*	

QUERY TO CREATE PROGRAM TABLE-

```
CREATE TABLE PROGRAM (
ProgramID INT AUTO_INCREMENT PRIMARY KEY,
ProgramName VARCHAR(100),
Duration INT,
Department VARCHAR(100)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	ProgramID	ID of Program	Integer
2	ProgramName	Name of program	Varchar
3	Duration	Duration of program	Integer
4	Department	Department name	Varchar

4) ENROLLMENT TABLE :

	Field	Type	Null	Key	Default	Extra
1	EnrollmentID	b'int'	NO	PRI	*NULL*	auto_inc…
2	StudentID	b'int'	YES	MUL	*NULL*	
3	CourseID	b'int'	YES	MUL	*NULL*	
4	EnrollmentDate	b'date'	YES		*NULL*	
5	Grade	b'varchar(5)'	YES		*NULL*	

QUERY TO CREATE ENROLLMENT TABLE-

```
CREATE TABLE ENROLLMENT (
EnrollmentID INT AUTO_INCREMENT PRIMARY KEY,
StudentID INT,
CourseID INT,
EnrollmentDate DATE,
Grade VARCHAR(5),
FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	EnrollmentID	ID of enrollment	Integer
2	StudentID	ID of student	Integer
3	CourseID	ID of Course	Integer
4	EnrollmentDate	Date of Enrollment	Date
5	Grade	Marks	Varchar

5) INSTRUCTOR TABLE :

	Field	Type	Null	Key	Default	Extra
1	InstructorID	b'int'	NO	PRI	*NULL*	auto_inc…
2	FirstName	b'varchar(50)'	YES		*NULL*	
3	LastName	b'varchar(50)'	YES		*NULL*	
4	Email	b'varchar(50)'	YES	UNI	*NULL*	
5	Phone	b'varchar(15)'	YES		*NULL*	
6	Department	b'varchar(50)'	YES		*NULL*	

QUERY TO CREATE INSTRUCTOR TABLE-

```
CREATE TABLE INSTRUCTOR (
InstructorID INT AUTO_INCREMENT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Email VARCHAR(100) UNIQUE,
Phone VARCHAR(15),
Department VARCHAR(100)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	InstructorID	Id number of instructor	Integer
2	FirstName	First Name of student/instructor	Variables
3	LastName	Family Name of student/instructor	Variable
4	Email	Email ID of Student/Instructor	Varchar
5	Phone	Contact Number of student/instructor	Varchar
6	Department	Department name	Varchar

6) PAYMENT TABLE :

	Field	Type	Null	Key	Default	Extra
1	PaymentID	b'int'	NO	PRI	*NULL*	auto_inc…
2	StudentID	b'int'	YES	MUL	*NULL*	
3	Amount	b'decimal(10,2)'	YES		*NULL*	
4	PaymentMethod	b'varchar(50)'	YES		*NULL*	

QUERY TO CREATE PAYMENT TABLE:

```
CREATE TABLE PAYMENT (
PaymentID INT AUTO_INCREMENT PRIMARY KEY,
StudentID INT,
Amount DECIMAL(10, 2),
PaymentMethod VARCHAR(50),
FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	PaymentID	ID number of the payment	Integer
2	StudentID	ID of the student	Integer
3	Amount	Fees amoount	decimal
4	PaymentMethod	Mode of payment	Variable

7) Department Table

	COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	IS_NULLABLE	COLUMN_DEFAULT
1	department_id	int	*NULL*	NO	*NULL*
2	department_na…	varchar	50	NO	*NULL*

Query to create department table:-

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

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	DepartmentID	Duration of program	Integer
2	DepartmentName	Department name	Varchar

8) Exam Table:-

	COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	IS_NULLABLE	COLUMN_DEFAULT
1	course_id	int	*NULL*	NO	*NULL*
2	exam_date	date	*NULL*	YES	*NULL*
3	exam_id	int	*NULL*	NO	*NULL*
4	total_marks	int	*NULL*	YES	*NULL*

Query to create 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)
);
```

DATA DICTIONARY :-

SR NO	FIELD NAME	MEANING	TYPE
1	Course_ID	ID number of Course	Integer
2	Exam_ID	Exam ID OF Course	Varchar
3	total_marks	Total marks	Integer
4	Exam_date	Date of exams	date

9) Exams result table:-

	COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	IS_NULLABLE	COLUMN_DEFAULT
1	exam_id	int	*NULL*	NO	*NULL*
2	marks_obtained	int	*NULL*	YES	*NULL*
3	student_id	int	*NULL*	NO	*NULL*

Query 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 student(student_id),
foreign key (exam_id) references exams(exam_id)
);
```

DATA DICTIONARY :

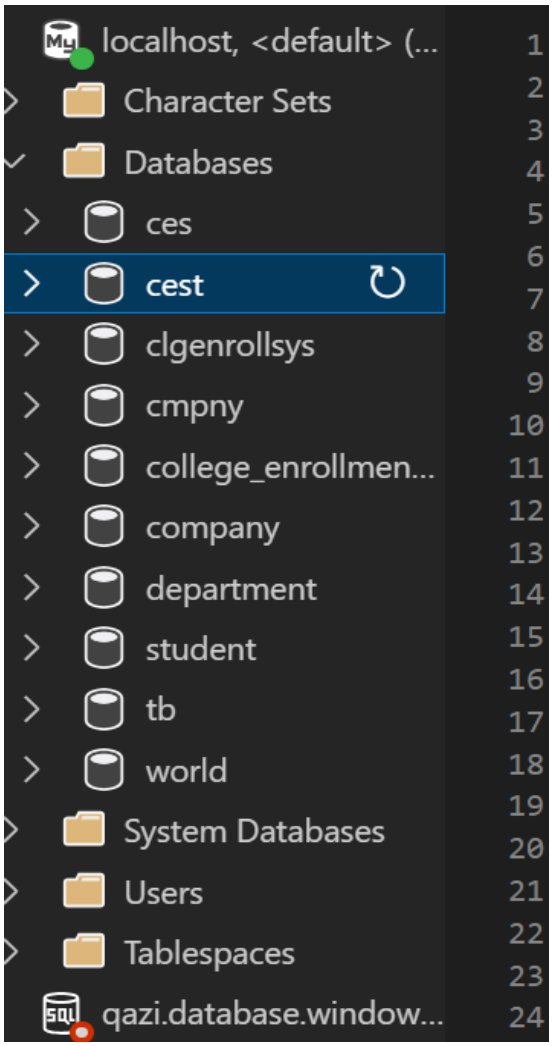
SR NO	FIELD NAME	MEANING	TYPE
1	StudentID	Id number of student	Integer
2	Exam_id	ID of exam taken	Integer
3	Marks_obtained	Total marks obtained	Integer

DATABASE DEVELOPMENT

1) Query to create database:-

```
Create Database CEST;
Use CEST;
```

Output:



INNER JOIN QUERY

1) Query to Get Faculty Information for Courses in a Specific Program

```
SELECT DISTINCT f.first_name AS faculty_first_name, f.last_name AS
faculty_last_name, c.course_name

FROM faculty f

JOIN enrollments en ON f.faculty_id = en.faculty_id

JOIN courses c ON en.course_id = c.course_id

JOIN programs p ON c.program_id = p.program_id

WHERE p.program_name = 'BSc Computer Science';
```

Output:-

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

Explanation of query:

The query essentially fetches the unique first and last names of faculty members and the names of the courses they teach, specifically for the "BSc Computer Science" program. The JOIN operations link the relevant tables (faculty, enrollments, courses, and programs) to combine the necessary information from the mentioned tables.

GROUPBY QUERY

1) Query to Find the Total Fees Collected for Each Program

```
SELECT p.program_name, SUM(c.course_fee) AS total_fees_collected

FROM enrollments e

JOIN courses c ON e.course_id = c.course_id

JOIN programs p ON c.program_id = p.program_id

GROUP BY p.program_name;
```

Output:-

	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

Explanation of the query:

The query essentially fetches the program names and calculates the total fees collected by summing up the course fees for each program. The JOIN operations link the relevant tables (enrollments, courses, and programs) to combine the necessary information.

CONCLUSION

To conclude, the College Enrolment System stands as an important tool for modern higher education premises. By simplifying the processes related to student admissions and registrations, these systems enable colleges and universities to handle large volumes of student data with greater efficiency and accuracy. This platform supports the entire enrolment process, from initial inquiries to final registration, ensuring a smooth and satisfying experience for both administrators and students.