SRM UNIVERSITY AP
Andhra Pradesh

Course Code: CSE 209
Course Name: Database Management Systems

PROJECT REPORT

Submitted to
Dr. Edukondalu Chappidi

Rasamsetti Gnana Prudhvi || AP23110010542
Vuppu Kishore || AP23110010563
Navanit Reddy Turpinti || AP23110010578

Batch: 2023-2027
Course-Branch-Section: B.Tech-CSE-H
Year-Semester: 2-IV

INDEX

Identification of Project

The project titled LUMEN (Learning Uplift for Modern Educational Networks) is a comprehensive "Student Information System" designed to streamline the academic and administrative processes of an educational institution. LUMEN is a centralized, relational database-driven application that addresses the growing need for structured and efficient management of student-related data in universities and colleges.

LUMEN aims to replace outdated manual and file-based systems with a secure, scalable, and user-friendly platform that ensures data accuracy, integrity, and quick accessibility. It is built on the foundational concepts of Database Management Systems (DBMS) and demonstrates real-world application of key topics such as the relational model, entity-relationship diagrams, normalization, SQL programming, and server-side logic.

The application is capable of handling:

- Student Data: Personal information, unique identifiers, and contact details.
- Academic Records: Course registrations, grades, semester-wise performance.
- Course Management: Information on available courses and credit structures.
- Faculty Details: Instructor assignments to specific courses.

LUMEN not only reflects a solid understanding of DBMS principles but also serves as a prototype that could be extended to support features like real-time student dashboards, attendance tracking, notifications, and performance analytics. The project embodies the practical integration of database theory into a functional system tailored for academic institutions.

Project Background

In the traditional educational environment, managing student data has long relied on manual record-keeping or file-based systems. While these methods may suffice in small-scale scenarios, they become inefficient, error-prone, and difficult to maintain as institutions grow and academic data becomes increasingly complex. Institutions often struggle with issues such as data redundancy, inconsistency, lack of security, and time-consuming access to student records.

With the increasing digitization of educational systems, there is an urgent need for a more efficient, secure, and scalable solution that can handle large volumes of academic and administrative data. This is where the concept of a Database Management System (DBMS) becomes critical. A DBMS provides a systematic and organized way to store, manipulate, and retrieve data, ensuring data integrity, consistency, and security.

The development of LUMEN (Learning Uplift for Modern Educational Networks) emerges from this need. LUMEN is designed as a centralized Student Information System (SIS) that automates the process of managing student records, course enrollments, instructor details, and academic performance tracking. The system leverages the relational model to represent data in the form of well-structured tables, making it easy to query and maintain.

LUMEN not only simplifies the storage and retrieval of data but also enables academic administrators, faculty, and students to interact with the system in a real-time, user-friendly, and efficient manner. By implementing principles from the DBMS course - such as data modeling, normalization, ER diagrams, and SQL programming - this project bridges the gap between academic theory and real-world application.

This system is a foundational step toward digital transformation in educational institutions, where routine operations such as enrolling students, assigning instructors, calculating grades, and generating reports can be done seamlessly

with just a few queries. LUMEN provides a modular and extensible framework that can be expanded in the future to include features like attendance tracking, notifications, student dashboards, and fee management.

In essence, LUMEN is more than just a course project - it is a vision of how data can empower educational ecosystems when managed with precision and purpose.

Project Description

The project is designed to support the administration of student records, including their personal details, academic performance, course enrollment, and grading. It ensures accurate, accessible, and secure data handling using a relational database system at the core.

Key components of the project include:

- **Student Module:** Maintains student details such as ID, name, age, gender, department, and contact information.
- **Faculty Module:** Stores faculty information including ID, name, department, designation, and contact.
- **Course Module:** Manages course-related data such as course code, name, credits, and faculty assigned.
- **Enrollment Module:** Keeps track of which students are enrolled in which courses during which semester.
- **Performance Module:** Records students' grades and calculates their overall academic performance.

The frontend of LUMEN is developed using HTML, CSS, and JavaScript, while MySQL is used to implement the backend database. The system also demonstrates the use of server-side logic and SQL views for efficient data access and manipulation.

The project emphasizes learning and implementing core DBMS concepts, including:

- Relational modeling
- ER diagram design
- Conversion of ER models into normalized tables (up to 3NF)
- SQL, DDL, and DML queries
- Use of views and triggers
- Server-side programming for backend data operations

Project Representation

Description using ER Diagrams

Entities, Entity Types & Entity Sets

- Student
- Faculty
- Course
- Enrollment
- Performance
- Dependents (Weak Entity)

Entity Sets

- student
- faculty
- course
- enrollment
- performance
- dependents

Attributes

Key Attributes

- Student_ID - Student
- Faculty_ID - Faculty
- Course_Code - Course
- Enrollment_ID - Enrollment
- Performance_ID - Performance
- Dependent_ID - Dependents (Partial key, identified by Student_ID)

Composite Attributes

- Student:
  Name → (First_Name, Last_Name)
  Address → (Street, City, State, Pincode)

- Faculty:
  Name → (First_Name, Last_Name)

Multi-valued Attributes

- Student
  Phone_Number → Can have multiple contact numbers
- Faculty
  Subjects_Handled → May teach multiple courses

Derived Attributes

- Student
  Age → Derived from Date_of_Birth
- Performance
  Grade_Status → Derived (e.g., “Pass” if Grade ≥ D)
  GPA → Computed from average grade across courses
- Enrollment
  Course_Duration → Derived from Semester and Year

Relationship Types & Sets

Relationship Types	Participating Entities	Description
Student - Enrollment	Student & Enrollment	A student enrolls in courses
Course - Enrollment	Course & Enrollment	A course has enrollments
Course - Faculty	Course & Faculty	A course is taught by faculty
Student - Performance	Student & Performance	A student has grades for courses
Course - Performance	Course & Performance	A course gives grades in
Student - Dependents	Student & Dependents	A student has dependents

Cardinality

Relationship Types	Cardinality	Description
Student - Enrollment	1 : M	A student can enroll in many courses
Course - Enrollment	1 : M	A course has many enrollments
Course - Faculty	M : 1	Many courses are taught by one faculty
Student - Performance	1 : M	A student has grades for many courses
Course - Performance	1 : M	A course contains multiple performances
Student - Dependents	1 : M	A student can have many dependents

Participation Constraints

Relationship Types	Participating Type	Description
Student - Enrollment	Total on Enrollment	Every enrollment must belong to a student
Course - Enrollment	Total on Enrollment	Every enrollment must refer to a course
Course - Faculty	Partial	A course may or may not be assigned faculty yet
Student - Performance	Partial	Not all students have grades yet
Course - Performance	Partial	Not all courses have performance entries
Student - Dependents	Total on Dependents	A dependent must be associated with a student

Weak Entity Types

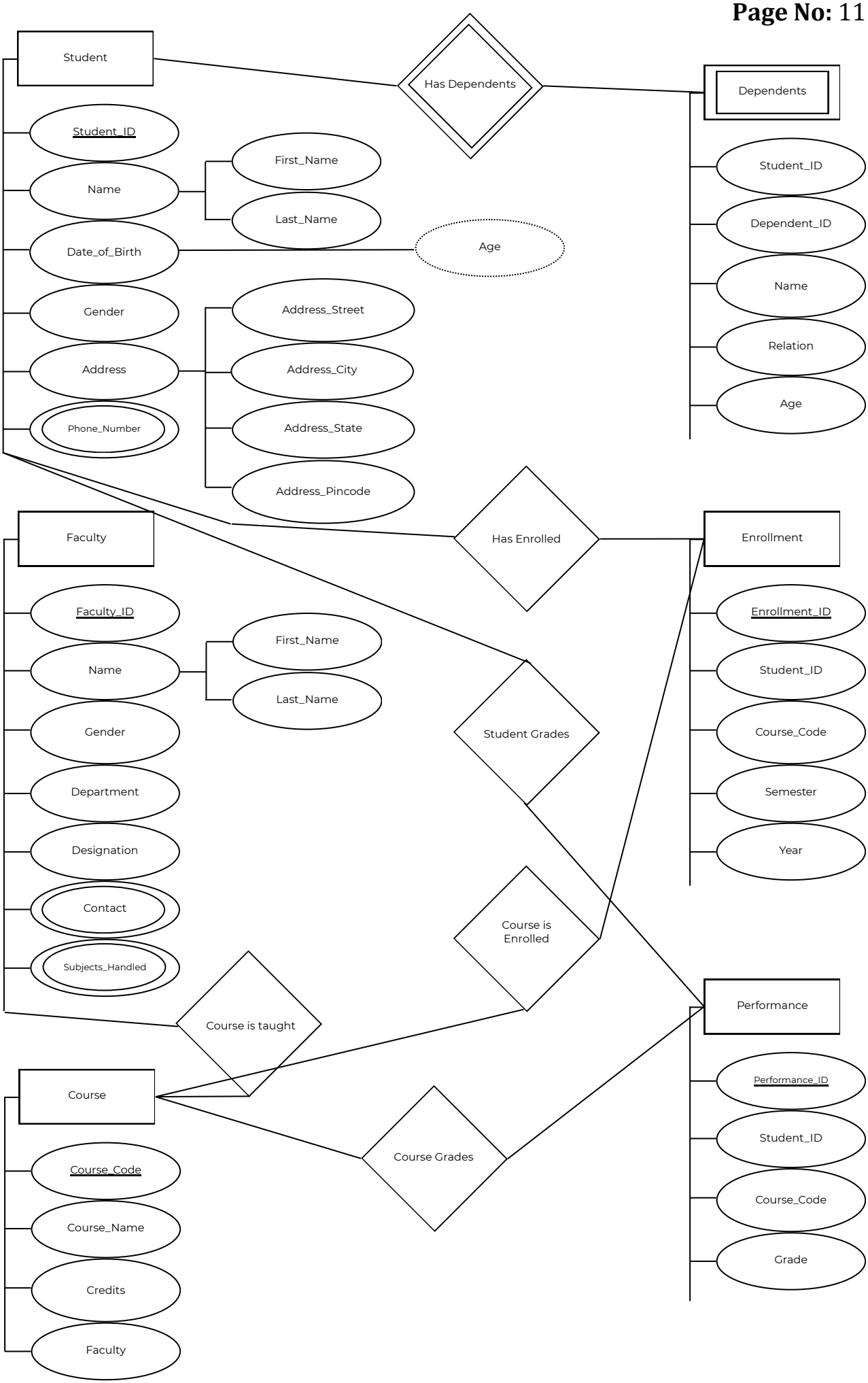
<u>Dependents</u>

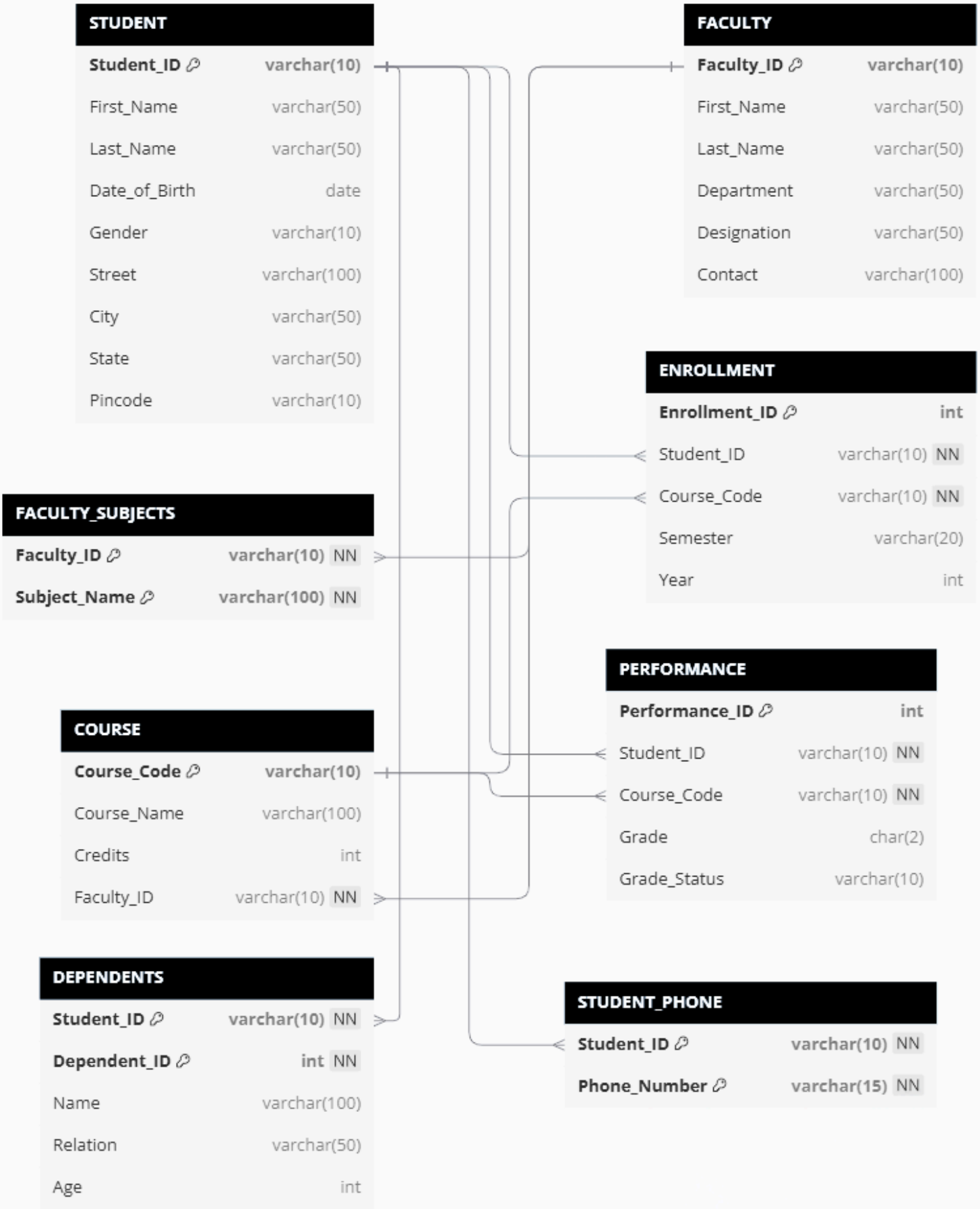
- Has a partial key: Dependent_ID which is dependent on Student_ID for identification.

Identifying Relationships

Has_Dependents

- A student has dependent(s) → identifying relationship
- Dependent entity cannot exist without a Student entity


Student
Has Dependents
Dependents
Student_ID
Name
First_Name
Last_Name
Date_of_Birth
Age
Gender
Address
Address_Street
Address_City
Address_State
Address_Pincode
Phone_Number
Student_ID
Dependent_ID
Name
Relation
Age
Faculty
Has Enrolled
Enrollment
Faculty_ID
Name
First_Name
Last_Name
Gender
Department
Designation
Contact
Subjects_Handled
Student Grades
Enrollment_ID
Student_ID
Course_Code
Semester
Year
Course is Enrolled
Course is taught
Performance
Course
Course Grades
Performance_ID
Student_ID
Course_Code
Grade
Course_Code
Course_Name
Credits
Faculty



Note: This ERD is in updated 3NF.

Conversion of ER Diagrams into Tables

Student Entity

Attribute	Type	Description
Student_ID	VARCHAR (PK)	Primary Key
First_Name	VARCHAR	Composite part of Name
Last_Name	VARCHAR	Composite part of Name
Date_of_Birth	DATE	For deriving Age
Street	VARCHAR	Composite part of Address
City	VARCHAR	Composite part of Address
State	VARCHAR	Composite part of Address
Pincode	VARCHAR	Composite part of Address
Phone_Number	VARCHAR	One student can have multiple phone numbers

Derived Attribute (Not Stored):

- Age

Faculty

Attribute	Type	Description
Faculty_ID	VARCHAR (PK)	Primary Key
First_Name	VARCHAR	Composite part of Name
Last_Name	VARCHAR	Composite part of Name
Department	VARCHAR	Faculty department
Designation	VARCHAR	Designation
Contact	VARCHAR	Phone or Email
Subjects_Handled	VARCHAR	Subjects the faculty handles

Course

Attribute	Type	Description
Course_Code	VARCHAR (PK)	Primary Key
Course_Name	VARCHAR	Name of the course
Credits	INT	Credit value
Faculty_ID	VARCHAR (FK)	Foreign Key to Faculty

Enrollment

Attribute	Type	Description
Enrollment_ID	INT (PK)	Primary Key
Student_ID	VARCHAR (FK)	Foreign Key to Student
Course_Code	VARCHAR (FK)	Foreign Key to Course
Semester	VARCHAR	e.g., "Fall", "Spring"
Year	INT	Academic year

Performance

Attribute	Type	Description
Performance_ID	INT (PK)	Primary Key
Student_ID	VARCHAR (FK)	Foreign Key to Student
Course_Code	VARCHAR (FK)	Foreign Key to Course
Grade	CHAR (2)	e.g., A, B, C, etc.

Derived Attribute (Not Stored):

- Grade_Status ("Pass" or "Fail")
- GPA (computed from multiple grades)

Dependents (Weak Entity)

Attribute	Type	Description
Student_ID	VARCHAR (FK)	Foreign Key & part of Composite Key
Dependent_ID	INT	Partial Key
Name	VARCHAR	Name of the dependent
Relation	VARCHAR	e.g., Parent, Sibling
Age	INT	Age

Description of the Tables

Description of the Entities

Student Entity

Represents an individual enrolled in the institution.

- Key Attribute: Student_ID - uniquely identifies a student.
- Composite Attribute: Name → includes First_Name and Last_Name.
- Composite Attribute: Address → includes Street, City, State, Pincode.
- Multi-valued Attribute: Phone_Number - a student may have multiple contact numbers.
- Derived Attribute: Age - calculated from the Date_of_Birth.

Faculty Entity

Represents teaching staff at the institution.

- Key Attribute: Faculty_ID - uniquely identifies a faculty member.
- Composite Attribute: Name → includes First_Name and Last_Name.
- Multi-valued Attribute: Subjects_Handled - a faculty member may handle multiple subjects.

Course Entity

Represents a subject or paper offered.

- Key Attribute: Course_Code – uniquely identifies each course.
- Attributes include Course_Name, Credits, and a reference to the teaching Faculty_ID.

Enrollment Entity

Represents a student's registration in a course.

- Key Attribute: Enrollment_ID.
- Includes foreign keys: Student_ID, Course_Code.
- Attributes include Semester, Year.
- Derived Attribute: Course_Duration – inferred from semester and year.

Performance Entity

Captures results of students in courses.

- Key Attribute: Performance_ID.
- Includes foreign keys: Student_ID, Course_Code.
- Attribute: Grade.
- Derived Attributes:
- Grade_Status (e.g., Pass/Fail based on grade).
- GPA (aggregated from multiple performances).

Dependencies (Weak Entity)

Represents people (like parents or siblings) who are associated with a student.

- Has no unique identity of its own.
- Identified by a composite key using Dependent_ID and Student_ID.
- Attributes: Name, Relation, and Contact.

Description of the Relationships and Constraints

Student - Enrollment

- Type: One-to-Many (1:M)
- Meaning: A student can enroll in many courses.
- Participation: Total on Enrollment side (every enrollment must belong to a student).

Course - Enrollment

- Type: One-to-Many (1:M)
- Meaning: A course can have many enrollments.
- Participation: Total on Enrollment side.

Course - Faculty

- Type: Many-to-One (M:1)
- Meaning: Many courses may be taught by one faculty.
- Participation: Partial - a course may not yet be assigned a faculty.

Student - Performance

- Type: One-to-Many (1:M)
- Meaning: A student can have performance records for many courses.
- Participation: Partial - some students may not yet have grades.

Course - Performance

- Type: One-to-Many (1:M)
- Meaning: A course can have performance records for many students.
- Participation: Partial - some courses may not yet be graded.

Student - Dependents

- Type: One-to-Many (1:M)
- Meaning: A student can have multiple dependents.
- Weak Entity: Dependents depend on Student for identification.
- Identifying Relationship: The connection from Student to Dependent is identifying.

Normalization of the Tables upto 3 NF

Student Entity

1NF

- Composite attribute Name split into First_Name and Last_Name.
- Composite attribute Address split into Street, City, State, and Zipcode.
- Multi-valued attribute Phone_Number moved to a separate table.

2NF

- All attributes fully functionally dependent on the primary key Student_ID.

3NF

- Derived attribute Age excluded (derived from DOB).

Final Tables

Attribute	Data Type	Key Type	Description
Student_ID	VARCHAR	Primary Key	Unique ID for each student
First_Name	VARCHAR	—	Student's first name
Last_Name	VARCHAR	—	Student's last name
DOB	DATE	—	Date of birth
Street	VARCHAR	—	Address - Street
City	VARCHAR	—	Address - City
State	VARCHAR	—	Address - State
Pincode	VARCHAR	—	Address - Zipcode

Attribute	Data Type	Key Type	Description
Student_ID	VARCHAR	Foreign Key	Refers to STUDENT(Student_ID)
Phone_Number	VARCHAR	—	One or more contact numbers

Faculty Entity

1NF

- Composite attribute Name split into First_Name, Last_Name.

- Multi-valued attribute Subjects_Handled moved to a separate table.

2NF

- Attributes depend fully on Faculty_ID.

3NF

- All non-key attributes are non-transitively dependent.

Final Tables

Attribute	Data Type	Key Type	Description
Faculty_ID	INT	Primary Key	Unique ID for each faculty
First_Name	VARCHAR	—	Faculty's first name
Last_Name	VARCHAR	—	Faculty's last name
Department	VARCHAR	—	Faculty's department
Designation	VARCHAR	—	Position/title held
Contact	VARCHAR	—	Contact number or email

Attribute	Data Type	Key Type	Description
Faculty_ID	INT	Foreign Key	Refers to FACULTY(Faculty_ID)
Subjects_Handled	VARCHAR	—	Subjects handled by faculty

Course Entity

1NF - 3NF

- All attributes atomic and fully dependent on the primary key Course_Code.
- Foreign key Faculty_ID links to FACULTY table.

Final Table

Attribute	Data Type	Key Type	Description
Course_Code	VARCHAR	Primary Key	Unique code for each course
Course_Name	VARCHAR	—	Name of the course
Credits	INT	—	Number of credits
Faculty_ID	INT	Foreign Key	Refers to FACULTY(Faculty_ID)

Enrollment Entity

1NF

- Attributes are atomic.

2NF

- No partial dependencies on Enrollment_ID.

3NF

- Derived attribute Course_Duration excluded.

Final Tables

Attribute	Data Type	Key Type	Description
Enrollment_ID	INT	Primary Key	Unique enrollment ID
Student_ID	VARCHAR	Foreign Key	Refers to STUDENT(Student_ID)
Course_Code	VARCHAR	Foreign Key	Refers to COURSE(Course_Code)
Semester	VARCHAR	—	Enrolled semester
Year	INT	—	Academic year

Performance Entity

1NF – 3NF

- All attributes atomic and dependent on the primary key.
- Derived attributes GPA and Grade_Status excluded.

Final Table

Attribute	Data Type	Key Type	Description
Performance_ID	INT	Primary Key	Unique ID for each performance record
Student_ID	VARCHAR	Foreign Key	Refers to STUDENT(Student_ID)
Course_Code	VARCHAR	Foreign Key	Refers to COURSE(Course_Code)
Grade	VARCHAR	—	Grade received

Dependents (Weak Entity)

1NF:

- Attributes are atomic.

2NF:

- Composite key formed with Student_ID and Dependent_Name.

3NF:

- Fully normalized with no transitive dependencies.

Final Table

Attribute	Data Type	Key Type	Description
Dependent_ID	VARCHAR	Part of PK	Unique ID for each faculty
Student_ID	VARCHAR	Part of PK, FK	Refers to STUDENT(Student_ID)
Name	VARCHAR	—	Name of the dependent
Relation	VARCHAR	—	Relationship to the student
Age	INT	—	Age of dependent

Creation of Data in the Tables

Student Table

Student_ID	First_ Name	Last_ Name	Gender	Date_of_Birth	Address_ Street	Address_ City	Address_ State	Address_ Pincode
S101	Aryan	Mehta	Male	2006-02-10	14A/7 Sector 9	Delhi	Delhi	110077
S102	Priya	Sharma	Female	2006-07-11	22 MG Road	Chennai	Tamil Nadu	600001
S103	Rahul	Reddy	Male	2006-09-02	10 Lake View	Hyderabad	Telangana	500032

Student_ID	Phone_Number
S101	9876543210
S101	9812345678
S102	9898989898
S103	9000001111

```
CREATE TABLE Student (
   Student_ID VARCHAR(10) PRIMARY KEY,
   First_Name VARCHAR(50) NOT NULL,
   Last_Name VARCHAR(50) NOT NULL,
   Date_of_Birth DATE NOT NULL,
   Gender VARCHAR(10),
   Street VARCHAR(100),
   City VARCHAR(50),
   State VARCHAR(50),
   Pincode VARCHAR(10)
);

CREATE TABLE Student_Phone (
   Student_ID VARCHAR(10),
   Phone_Number VARCHAR(15),
   PRIMARY KEY (Student_ID, Phone_Number),
   FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID)
);

INSERT INTO Student VALUES ('S101', 'Aryan', 'Mehta', '2006-02-10', 'Male', '14A/7 Sector 9', 'Delhi', 'Delhi', '110077');
INSERT INTO Student VALUES ('S102', 'Priya', 'Sharma', '2006-07-11', 'Female', '22 MG Road', 'Chennai', 'Tamil Nadu', '600001');
INSERT INTO Student VALUES ('S103', 'Rahul', 'Reddy', '2006-09-02', 'Male', '10 Lake View', 'Hyderabad', 'Telangana', '500032');

INSERT INTO Student_Phone VALUES ('S101', '9876543210');
INSERT INTO Student_Phone VALUES ('S101', '9812345678');
INSERT INTO Student_Phone VALUES ('S102', '9888888888');
INSERT INTO Student_Phone VALUES ('S103', '9000001111');
```

Faculty Table

Faculty_ID	First_Name	Last_Name	Department	Designation	Contact
F01	Neha	Joshi	Computer Sci.	Assistant Prof.	9011122233
F02	Arjun	Verma	Mathematics	Professor	9223344556
F03	Divya	Kapoor	Humanities	Lecturer	9334455667

Faculty_ID	Subjects_Handled
F01	DBMS
F01	Data Structures
F02	Discrete Mathematics
F02	Calculus
F03	Technical Writing

```
CREATE TABLE Faculty (
   Faculty_ID VARCHAR(10) PRIMARY KEY,
   First_Name VARCHAR(50) NOT NULL,
   Last_Name VARCHAR(50) NOT NULL,
   Department VARCHAR(50),
   Designation VARCHAR(50),
   Contact VARCHAR(100)
);

CREATE TABLE Faculty_Subjects (
   Faculty_ID VARCHAR(10),
   Subject_Name VARCHAR(100),
   PRIMARY KEY (Faculty_ID, Subject_Name),
   FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID)
);

INSERT INTO Faculty VALUES ('F01', 'Neha', 'Joshi', 'Computer Science', 'Assistant Prof.', 'neha.joshi@example.com');
INSERT INTO Faculty VALUES ('F02', 'Arjun', 'Verma', 'Mathematics', 'Professor', 'arjun.verma@example.com');
INSERT INTO Faculty VALUES ('F03', 'Divya', 'Kapoor', 'Humanities', 'Lecturer', 'divya.kapoor@example.com');

INSERT INTO Faculty_Subjects VALUES ('F01', 'DBMS');
('F01', 'Data Structures');
('F02', 'Discrete Mathematics');
('F02', 'Calculus');
('F03', 'Technical Writing');
```

Course Table

Course_Code	Course_Name	Credits	Faculty_ID
CSE101	Database Management System	4	F01
MAT102	Discrete Mathematics	3	F02
HUM103	Technical Communication	2	F03

```
CREATE TABLE Course (
   Course_Code VARCHAR(10) PRIMARY KEY,
   Course_Name VARCHAR(100) NOT NULL,
   Credits INT,
   Faculty_ID VARCHAR(10),
   FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID)
);


INSERT INTO Course VALUES ('CSE101', 'Database Management System', 4, 'F01');
INSERT INTO Course VALUES ('MAT102', 'Discrete Mathematics', 3, 'F02');
INSERT INTO Course VALUES ('HUM103', 'Technical Communication', 2, 'F03');
```

Enrollment Table

Enrollment_ID	Student_ID	Course_Code	Semester	Year	Course_Duration
E201	S101	CSE101	Fall	2024	6 months
E202	S101	MAT102	Fall	2024	4 months
E203	S102	CSE101	Fall	2024	6 months
E204	S103	HUM103	Spring	2025	3 months

```
CREATE TABLE Enrollment (
   Enrollment_ID INT PRIMARY KEY,
   Student_ID VARCHAR(10),
   Course_Code VARCHAR(10),
   Semester VARCHAR(20),
   Year INT,
   FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
   FOREIGN KEY (Course_Code) REFERENCES Course(Course_Code),
   UNIQUE (Student_ID, Course_Code)  -- Prevents duplicate enrollments
);

INSERT INTO Enrollment VALUES ('E201', 'S101', 'CSE101', 'Fall', 2024);
INSERT INTO Enrollment VALUES ('E202', 'S101', 'MAT102', 'Fall', 2024);
INSERT INTO Enrollment VALUES ('E203', 'S102', 'CSE101', 'Fall', 2024);
INSERT INTO Enrollment VALUES ('E204', 'S103', 'HUM103', 'Spring', 2025);
```

Performance Table

Performance_ID	Student_ID	Course_Code	Grade
P301	S101	CSE101	A
P302	S101	MAT102	B
P303	S102	CSE101	C
P304	S103	HUM103	F

```
CREATE TABLE Performance (
   Performance_ID INT PRIMARY KEY,
   Student_ID VARCHAR(10),
   Course_Code VARCHAR(10),
   Grade CHAR(2),
   FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
   FOREIGN KEY (Course_Code) REFERENCES Course(Course_Code)
);

INSERT INTO Performance VALUES ('P301', 'S101', 'CSE101', 'A');
INSERT INTO Performance VALUES ('P302', 'S101', 'MAT102', 'B');
INSERT INTO Performance VALUES ('P303', 'S102', 'CSE101', 'C');
INSERT INTO Performance VALUES ('P304', 'S103', 'HUM103', 'F');
```

Dependents Table

Student_ID	Dependent_ID	Name	Relationship	Age
S101	D1	Riya Mehta	Mother	44
S102	D2	Ravi Sharma	Father	50
S103	D3	Krishna Reddy	Father	49

```
CREATE TABLE Dependents (
   Student_ID VARCHAR(10),
   Dependent_ID INT,
   Name VARCHAR(100),
   Relation VARCHAR(50),
   Age INT,
   PRIMARY KEY (Student_ID, Dependent_ID),
   FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID) ON DELETE CASCADE
);

INSERT INTO Dependents VALUES ('S101', 'D1', 'Riya Mehta', 'Mother', 44);
INSERT INTO Dependents VALUES ('S102', 'D2', 'Ravi Sharma', 'Father', 50);
INSERT INTO Dependents VALUES ('S103', 'D3', 'Krishna Reddy', 'Father', 49);
```

SQL Queries on the Created Tables

Data Retrieval

```
--Get all student details

SELECT * FROM Student;

--Get students enrolled in a specific course

SELECT S.Student_ID, S.First_Name, S.Last_Name, E.Course_Code
FROM Student S
JOIN Enrollment E ON S.Student_ID = E.Student_ID
WHERE E.Course_Code = 'CSE101';

--List all courses along with faculty names

SELECT C.Course_Code, C.Course_Name, F.First_Name, F.Last_Name
FROM Course C
JOIN Faculty F ON C.Faculty_ID = F.Faculty_ID;
```

Aggregate Functions

```
--Count students enrolled in each course

SELECT Course_Code, COUNT(Student_ID) AS Enrolled_Students
FROM Enrollment
GROUP BY Course_Code;

--Calculate average GPA of each student

SELECT Student_ID, AVG(GPA) AS Average_GPA
FROM Performance
GROUP BY Student_ID;
```

Conditional Queries

```
--Find students who scored below 5 GPA

SELECT Student_ID, Course_Code, GPA
FROM Performance
WHERE GPA < 5;

--Fetch students with age greater than 21

SELECT * FROM Student
WHERE TIMESTAMPDIFF(YEAR, Date_of_Birth, CURDATE()) > 21;
```

Update and Delete

```
--Update a student's contact number

UPDATE Student
SET Contact = '9998877665'
WHERE Student_ID = 'S101';

--Delete a specific enrollment

DELETE FROM Enrollment
WHERE Enrollment_ID = 'E105';
```

Joins and Multi-table Queries

```
--Display student names with their grades and course names

SELECT S.First_Name, S.Last_Name, C.Course_Name, P.Grade
FROM Student S
JOIN Performance P ON S.Student_ID = P.Student_ID
JOIN Course C ON C.Course_Code = P.Course_Code;
```

Server Side Programming

Stored Procedure to Enroll a Student in a Course

```
DELIMITER //
CREATE PROCEDURE Enroll_Student(
  IN p_Student_ID VARCHAR(10),
  IN p_Course_Code VARCHAR(10),
  IN p_Semester VARCHAR(10),
  IN p_Year INT
)
BEGIN
  INSERT INTO Enrollment (Student_ID, Course_Code, Semester, Year)
  VALUES (p_Student_ID, p_Course_Code, p_Semester, p_Year);
END;
//
DELIMITER ;
```

Stored Function to Call Student GPA

```
DELIMITER //
CREATE FUNCTION Calculate_GPA(p_Student_ID VARCHAR(10))
RETURNS FLOAT
DETERMINISTIC
BEGIN
  DECLARE total_gpa FLOAT;

  SELECT AVG(GPA) INTO total_gpa
  FROM Performance
  WHERE Student_ID = p_Student_ID;

  RETURN total_gpa;
END;
//
DELIMITER ;
```

Error Handling via Transaction

```
START TRANSACTION;

--Example: Add a new enrollment and related performance entry

INSERT INTO Enrollment (Enrollment_ID, Student_ID, Course_Code, Semester, Year)
VALUES ('E110', 'S102', 'CSE103', 'Spring', 2025);
```

```
INSERT INTO Performance (Performance_ID, Student_ID, Course_Code, Grade, GPA)
VALUES ('P110', 'S102', 'CSE103', 'B', 7.0);

COMMIT;
```

Trigger 1: Log Enrollments to an Audit Table

```
CREATE TABLE Enrollment_Log (
  Log_ID INT AUTO_INCREMENT PRIMARY KEY,
  Student_ID VARCHAR(10),
  Course_Code VARCHAR(10),
  Log_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER after_enrollment_insert
AFTER INSERT ON Enrollment
FOR EACH ROW
BEGIN
  INSERT INTO Enrollment_Log (Student_ID, Course_Code)
  VALUES (NEW.Student_ID, NEW.Course_Code);
END;
//
DELIMITER ;
```

Trigger 2: Auto Calculate Grade_Status (Derived Attribute)

```
ALTER TABLE Performance ADD Grade_Status VARCHAR(10);

DELIMITER //
CREATE TRIGGER before_insert_grade_status
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
  IF NEW.Grade IN ('A', 'B', 'C') THEN
    SET NEW.Grade_Status = 'Pass';
  ELSE
    SET NEW.Grade_Status = 'Fail';
  END IF;
END;
//
DELIMITER ;
```

Trigger 3: Prevent Duplicate Enrollment

```
DELIMITER //
CREATE TRIGGER prevent_duplicate_enrollment
BEFORE INSERT ON Enrollment
FOR EACH ROW
BEGIN
  IF EXISTS (
    SELECT 1 FROM Enrollment
    WHERE Student_ID = NEW.Student_ID AND Course_Code = NEW.Course_Code
  ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Student is already enrolled in this course';
  END IF;
END;
//
DELIMITER ;
```

Transaction: Enroll and Record Performance Together

```
START TRANSACTION;

INSERT INTO Enrollment (Enrollment_ID, Student_ID, Course_Code, Semester, Year)
VALUES ('E999', 'S001', 'C101', 'Spring', 2025);

INSERT INTO Performance (Performance_ID, Student_ID, Course_Code, Grade)
VALUES ('P999', 'S001', 'C101', 'B');

COMMIT;
```

For Rolling Back Changes in Case of Error:

```
--On error
ROLLBACK;
```

Creation of Views using the Tables

View: Student Grade Report

```
CREATE VIEW Student_Grade_Details AS
SELECT s.Student_ID,
    CONCAT(s.First_Name, ' ', s.Last_Name) AS Student_Name,
    c.Course_Name,
    p.Grade,
    p.Grade_Status
FROM Student s
JOIN Performance p ON s.Student_ID = p.Student_ID
JOIN Course c ON c.Course_Code = p.Course_Code;
```

View: Course Enrollment Info

```
CREATE VIEW Course_Enrollment_Info AS
SELECT C.Course_Code, C.Course_Name, COUNT(E.Student_ID) AS Total_Enrolled
FROM Course C
JOIN Enrollment E ON C.Course_Code = E.Course_Code
GROUP BY C.Course_Code, C.Course_Name;
```

View: Faculty Course List

```
CREATE VIEW Faculty_Courses AS
SELECT F.Faculty_ID, F.First_Name, F.Last_Name, C.Course_Code, C.Course_Name
FROM Faculty F
JOIN Course C ON F.Faculty_ID = C.Faculty_ID;
```

View: Student Performance Overview

```
CREATE VIEW Student_Performance_Overview AS
SELECT s.Student_ID,
    CONCAT(s.First_Name, ' ', s.Last_Name) AS Student_Name,
    c.Course_Name,
    p.Grade
FROM Student s
JOIN Performance p ON s.Student_ID = p.Student_ID
JOIN Course c ON p.Course_Code = c.Course_Code;
```

View: Faculty Course Assignment

```
CREATE VIEW Faculty_Course_Assignment ASSELECT f.Faculty_ID,
 CONCAT(f.First_Name, ' ', f.Last_Name) AS Faculty_Name,
 c.Course_Name,
 c.Credits
```

```
FROM Faculty f
JOIN Course c ON f.Faculty_ID = c.Faculty_ID;
```

View: Course Enrollment Count

```
CREATE VIEW Course_Enrollment_Count AS
SELECT c.Course_Name,
    COUNT(e.Student_ID) AS Enrollment_Count
FROM Course c
LEFT JOIN Enrollment e ON c.Course_Code = e.Course_Code
GROUP BY c.Course_Name;
```

View: Course Grades Summary

```
CREATE VIEW Course_Grades_Summary AS
SELECT c.Course_Name,
    AVG(CASE WHEN p.Grade = 'A' THEN 1
        WHEN p.Grade = 'B' THEN 2
        WHEN p.Grade = 'C' THEN 3
        WHEN p.Grade = 'D' THEN 4
        ELSE 5 END) AS Average_Grade
FROM Course c
JOIN Performance p ON c.Course_Code = p.Course_Code
GROUP BY c.Course_Name;
```

View: Calculating Age

```
CREATE VIEW Student_Age_View AS
SELECT Student_ID,
    CONCAT(First_Name, ' ', Last_Name) AS Name,
    TIMESTAMPDIFF(YEAR, Date_of_Birth, CURDATE()) AS Age
FROM Student;
```