Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)
School of Computing
B.Tech. – Computer Science and Engineering

VTR UGE2021- (CBCS)

Academic Year: 2025–2026

SDG 4: Quality Education
Course Code : 10211CS207

Course Name : Database Management Systems

Slot No : S4-L5

DBMS PROJECT REPORT

Title: **University Academic Information System**

Submitted by:

VTUNO	REGISTER NUMBER	STUDENT NAME
VTU30200	24UECS1453	GANESWARAREDDY UPPALAPALLI

Under the guidance of:
Dr V Senthil kumar

INDEX **PAGE**

1. Introduction

The **University Academic Information System (UAIS)** is a comprehensive database management solution that automates and manages the core academic activities of a university. Universities handle a large volume of academic data every semester — such as student enrollments, course offerings, instructor assignments, attendance records, and examination results. Managing all of these manually or with outdated tools can lead to inefficiencies, data redundancy, and loss of information integrity.

The proposed system is designed to **digitize and centralize academic operations** using **relational database management principles**. It aims to provide a structured and consistent way to store, update, and retrieve academic data. Each major entity—such as **Students**, **Courses**, **Instructors**, and **Results**—is represented in the database with clearly defined relationships that enforce referential integrity through **foreign keys** and **constraints**.

The system also supports **indexing** for faster query performance, particularly on frequently accessed fields such as *student_id*, *course_id*, and *semester*. Additionally, **transaction management** ensures that operations like grade updates, enrollments, or deletions are logged securely for auditing and recovery purposes. To enhance flexibility, the system integrates **NoSQL storage (MongoDB)** to maintain unstructured data such as transaction logs or activity tracking, complementing the relational Oracle database.

Overall, this system forms the backbone of an efficient academic management process, ensuring data accuracy, easy access for administrators, and transparency in university operations.

2. Problem Statement

Universities face significant challenges in handling academic data manually or through loosely connected systems. Traditional record-keeping using paper files or spreadsheets makes it difficult to ensure **accuracy**, **security**, and **timely access** to information.

Some of the major issues include:

1) **Data Redundancy and Inconsistency:**
When student or course information is stored in multiple places, discrepancies may arise. Updating a student's details in one record but not another can lead to conflicting information.

2) **Manual Processing and Errors:**
Recording marks, attendance, or grades manually increases the chances of human error. It also consumes valuable administrative time.

3) **Lack of Real-Time Information:**
Students and instructors often cannot access updated academic information instantly. This delay affects decision-making for academic planning and performance evaluation.

4) **Poor Data Security and Integrity:**
Without proper database constraints or access control, there is a risk of unauthorized access, data tampering, or accidental deletions.

5) **Scalability Issues:**
As the number of students, courses, and semesters increases, traditional methods become inefficient in managing large datasets.

6) **Lack of Transaction Management:**
Important operations like grade updates or course enrollments need to be traceable. Manual systems fail to maintain proper transaction logs, which can lead to accountability issues.

3. Objectives

· To develop a centralized database for managing university academic data.

· To maintain student, course, instructor, and result records efficiently.

· To automate operations such as course enrollment, grading, and attendance.

· To ensure data integrity using relational modeling and constraints.

· To improve performance using indexing on key attributes.

· To integrate SQL and NoSQL databases for hybrid data management.

4. System Requirements

Hardware Requirements:

- Processor: Intel i5 or higher
- RAM: 8 GB or more
- Hard Disk: 250 GB

Software Requirements:

- OS: Windows 10 or Linux
- Database: Oracle 12c or above
- Front-End: Java / Web Interface
- NoSQL: MongoDB 6.0
- Tools: Oracle SQL Developer, MongoDB Compass

5. System Analysis and Design

The system identifies the main entities and their relationships. Major entities include:

Students enroll in multiple **Courses**.

· **Instructors** teach one or more **Courses**.

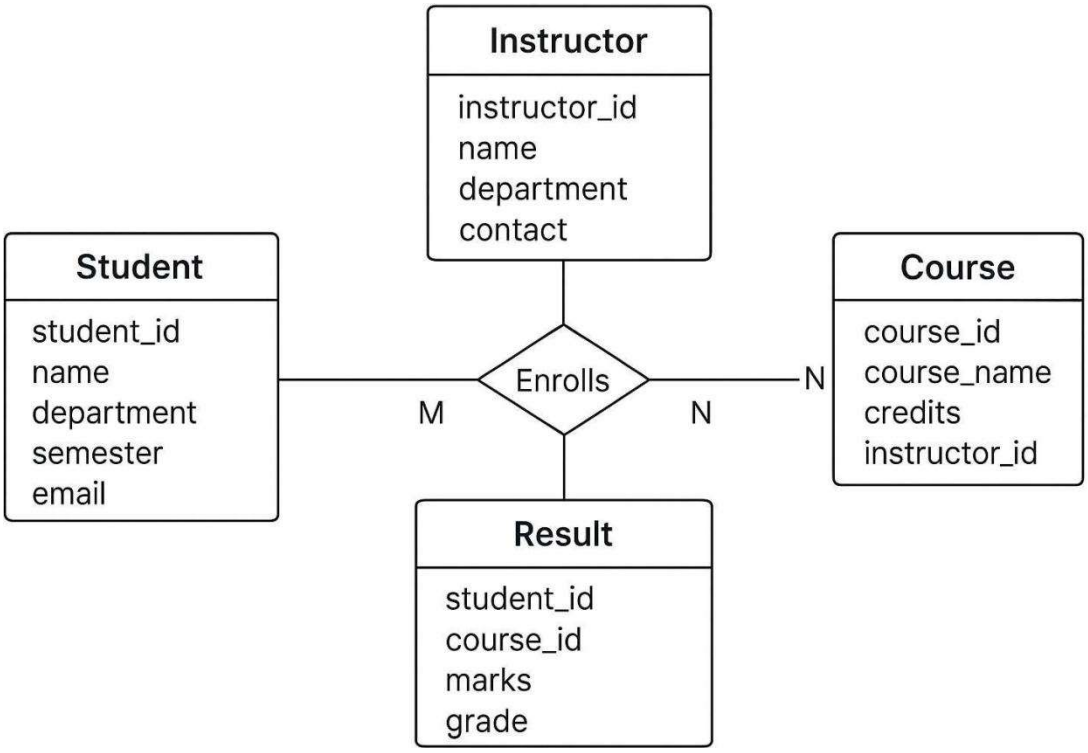
· **Results** store marks and grades for each student per course.

· **Attendance** is tracked per course and per session.

6. ER Diagram (Conceptual Design)

Entities:

- Student (student_id, name, department, semester, email)
- Course (course_id, course_name, credits, instructor_id)
- Instructor (instructor_id, name, department, contact)
- Result (student_id, course_id, marks, grade)
- Attendance (student_id, course_id, date, status)

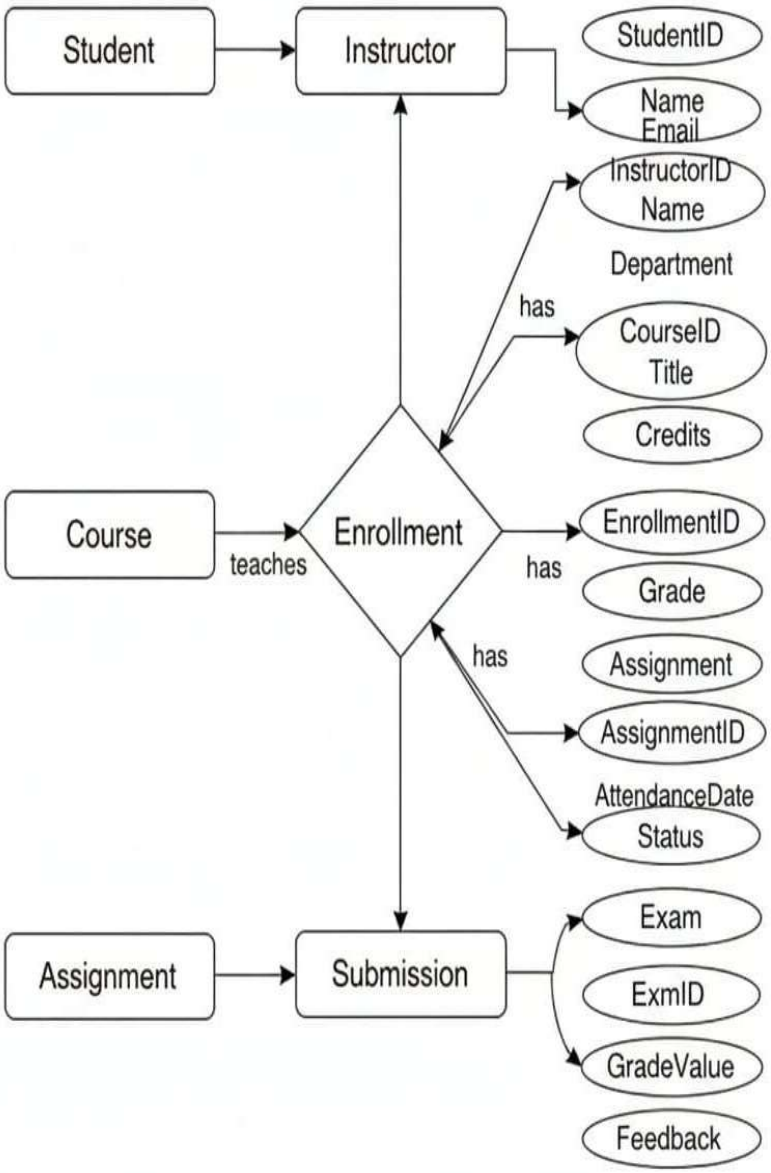


-

Relationships:

- Student ↔ Course (Many-to-Many via Result)
- Instructor ↔ Course (One-to-Many)
- Student ↔ Attendance (One-to-

Many) Figure: ER Diagram


Student
Instructor
StudentID
Name
Email
InstructorID
Name
Department
has
CourseID
Title
Credits
Course
teaches
Enrollment
EnrollmentID
has
Grade
has
Assignment
AssignmentID
AttendanceDate
Status
Assignment
Submission
Exam
ExmID
GradeValue
Feedback

7. Schema Design (Oracle)

```
SQL> CREATE TABLE Student (
    student_id NUMBER PRIMARY KEY,
    name VARCHAR2(50),
    department VARCHAR2(30),
    semester NUMBER,
    email VARCHAR2(50)
);
```

Output:

```
SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 STUDENT_ID                                NOT NULL NUMBER
 NAME                                               VARCHAR2(50)
 DEPARTMENT                                         VARCHAR2(30)
 SEMESTER                                           NUMBER
 EMAIL                                              VARCHAR2(50)
```

```
SQL> CREATE TABLE Instructor (
    instructor_id NUMBER PRIMARY KEY,
    name VARCHAR2(50),
    department VARCHAR2(30),
    contact VARCHAR2(15)
);
```

Output:

```
SQL> DESC instructor;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 INSTRUCTOR_ID                             NOT NULL NUMBER
 NAME                                               VARCHAR2(50)
 DEPARTMENT                                         VARCHAR2(30)
 CONTACT                                            VARCHAR2(15)
```

```
SQL> CREATE TABLE Course (
    course_id NUMBER PRIMARY KEY,
    course_name VARCHAR2(50),
    credits NUMBER,
    instructor_id NUMBER REFERENCES Instructor(instructor_id)
);
```

Output:

```
SQL> desc course;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 COURSE_ID                                 NOT NULL NUMBER
 COURSE_NAME                                        VARCHAR2(50)
 CREDITS                                            NUMBER
 INSTRUCTOR_ID                                      NUMBER
```

```
SQL> CREATE TABLE Result (
    student_id NUMBER REFERENCES Student(student_id),
    course_id NUMBER REFERENCES Course(course_id),
    marks NUMBER,
    grade CHAR(2),
```

PRIMARY KEY(student_id, course_id)

);

Output:

```
SQL> desc result;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 STUDENT_ID                                NOT NULL NUMBER
 COURSE_ID                                 NOT NULL NUMBER
 MARKS                                              NUMBER
 GRADE                                              CHAR(2)
```

SQL> CREATE TABLE Attendance (

student_id NUMBER REFERENCES Student(student_id),

course_id NUMBER REFERENCES Course(course_id),

date DATE,

status CHAR(1),

PRIMARY KEY(student_id, course_id, date)

);

Output:

```
SQL> desc attendance;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------
 STUDENT_ID                                NOT NULL NUMBER
 COURSE_ID                                 NOT NULL NUMBER
 DATE                                      NOT NULL DATE
 STATUS                                             CHAR(1)
```

8. Normalization

All tables are normalized up to **3rd Normal Form (3NF)**:

No repeating groups (1NF).

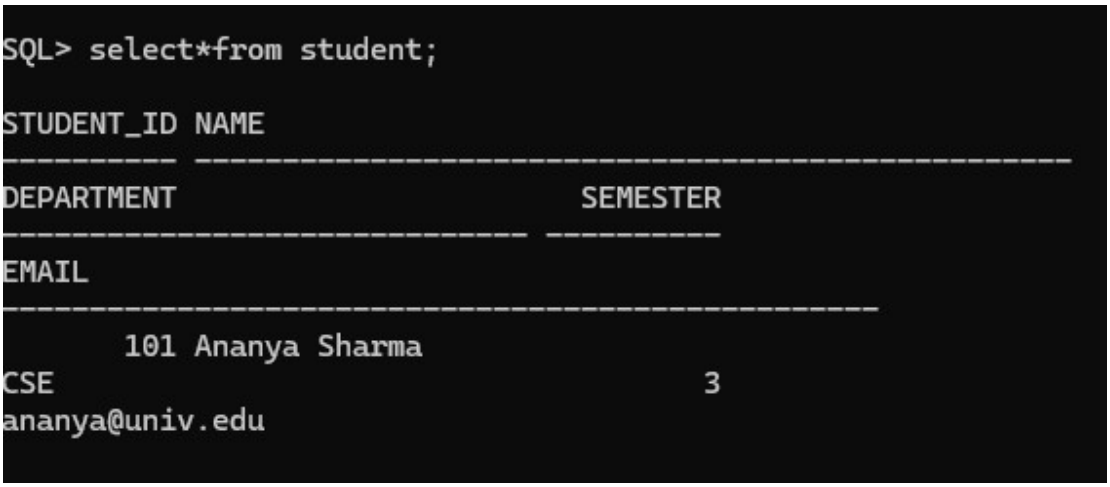
Non-key attributes depend on the primary key (2NF).

No transitive dependencies (3NF).

This ensures minimal redundancy and optimal data integrity.

9. Implementation (SQL Queries)

INSERT INTO Student VALUES (101, 'Ananya Sharma', 'CSE', 3, 'ananya@univ.edu');



INSERT INTO Instructor VALUES (201, 'Dr. Rao', 'CSE', '9876543210');

```
SQL> select *from instructor;

INSTRUCTOR_ID NAME
------------- ----------------------------------
DEPARTMENT                      CONTACT
------------------------------ ---------------
          201 Dr. Rao
CSE                            9876543210
```

INSERT INTO Course VALUES (301, 'Database Systems', 4, 201);

```
SQL> select*from course;

 COURSE_ID COURSE_NAME                                          CREDITS
---------- ---------------------------------------------------- ----------
INSTRUCTOR_ID
-------------
       301 Database Systems                                            4
           201
```

INSERT INTO Result VALUES (101, 301, 85, 'A');

```
SQL> select*from result
  2  ;

STUDENT_ID  COURSE_ID      MARKS GR
---------- ---------- ---------- --
       101        301         85 A
```

INSERT INTO Attendance VALUES (101, 301, SYSDATE, 'P');

```
SQL> select*from attendance;

STUDENT_ID  COURSE_ID DATE      S
---------- ---------- --------- -
       101        301 21-OCT-25 P
```

10. Input and Output

Input: Student details, course registration, marks, attendance records.

OUTPUT:-

Query:

```
SELECT s.name, c.course_name, r.marks, r.grade
FROM Student s
JOIN Result r ON s.student_id = r.student_id
JOIN Course c ON r.course_id = c.course_id;
```

Output:

```
NAME
--------------------------------------------------
COURSE_NAME                                        MARKS GR
-------------------------------------------------- ---------- --
Ananya Sharma
Database Systems                                      85 A
```

11. Integration with MongoDB (NoSQL)

Overview

MongoDB is a **NoSQL, document-oriented database** designed to store large volumes of **unstructured or semi-structured data**. Unlike traditional relational databases such as Oracle, which organize data in tables with rows and columns, MongoDB stores data in **flexible JSON-like documents (BSON format)**.

This flexibility makes MongoDB ideal for handling **dynamic, evolving datasets** such as student activity logs, course materials, online feedback, and audit trails — where data may not fit neatly into a rigid schema.

In the **University Academic Information System**, MongoDB is used to **complement the Oracle database** by managing:

1. **Transaction logs** (e.g., grade updates, enrollments)
2. **Learning materials** (e.g., PDFs, videos)
3. **Feedback and discussions**
4. **System usage analytics**

This hybrid SQL–NoSQL integration ensures that the system maintains both **data integrity (Oracle)** and **scalability with flexibility (MongoDB)**.

MongoDB Database Structure for the Project

Database Name: UniversityDB

Collections and Example Documents

1. students Collection

{

```
 "_id": 1,
 "student_id": "STU101",
 "name": "Priya Sharma",
 "email": "priya@university.edu",
 "department": "Computer Science",
 "semester": 5,
 "enrolled_courses": ["C101", "C102"],
 "activity_logs": [
   {"action": "enrolled", "course_id": "C101", "timestamp": "2025-10-10T09:30:00Z"},
   {"action": "submitted_assignment", "course_id": "C101", "timestamp": "2025-10-15T12:45:00Z"}
 ]
}
```


2. courses Collection


```
{
 "_id": 1,
 "course_id": "C101",
 "course_name": "Database Management Systems",
 "instructor": "Dr. Ramesh",
 "credits": 4,
 "materials": [
   {"type": "video", "title": "ER Diagram Lecture", "link": "https://univ.edu/dbms/video1"},
   {"type": "pdf", "title": "Normalization Notes", "link": "https://univ.edu/dbms/notes.pdf"}
```

```
  ]
}
```


3. logs Collection


```
{
  "_id": 1,
  "operation": "Grade Update",
  "student_id": "STU101",
  "course_id": "C101",
  "timestamp": "2025-10-20T14:35:00Z",
  "performed_by": "ExamController",
  "status": "Success"
}
```


4. feedback Collection


```
{
  "_id": 1,
  "feedback_id": "FDB001",
  "student_id": "STU101",
  "course_id": "C101",
  "message": "The course content was clear and interactive.",
  "rating": 5,
  "date": "2025-10-18"
}
```

MongoDB Operations (Examples)

1. Inserting a Student Document

```
db.students.insertOne({
student_id: "STU102",
  name: "Arun Kumar",
  email: "arun@university.edu",
  department: "Information Technology",
  semester: 4,
  enrolled_courses: ["C101", "C103"]
});
```

Output:

Acknowledged: Inserted document with _id: ObjectId("...")

2. Fetching All Courses with Instructor Details

```
db.courses.find({}, {course_name: 1, instructor: 1, _id: 0});
```

Output:

```
[
  {"course_name": "Database Management Systems", "instructor": "Dr. Ramesh"},
  {"course_name": "Web Technologies", "instructor": "Prof. Karthik"}
]
```

3.Updating a Student's Enrolled Courses

```
db.students.updateOne(
  {student_id: "STU101"},
```

```
  {$push: {enrolled_courses: "C103"}}
);
```

Output:

Matched: 1 document, Modified: 1 document

4. Retrieving Students with Marks Above 40

```
db.results.find(
  {marks: {$gte: 40}},
  {student_id: 1, course_id: 1, marks: 1, grade: 1, _id: 0}
);
```

Output:

```
[
  {"student_id": "STU101", "course_id": "C101", "marks": 85, "grade": "A"},
  {"student_id": "STU102", "course_id": "C101", "marks": 42, "grade": "C"}
]
```

5.Recording a Transaction Log

```
db.logs.insertOne({
  operation: "Course Enrollment",
  student_id: "STU103",
  course_id: "C104",
  timestamp: new Date(),
  performed_by: "Admin",
  status: "Successful"
});
```

Output:

Transaction log added successfully.

Advantages of MongoDB Integration

1. **Flexibility:** Supports dynamic data (e.g., feedback, resources, or logs).
2. **Scalability:** Easily handles growing volumes of data.
3. **Performance:** Fast read/write operations for frequently changing data.
4. **Complementary Design:** Works alongside Oracle for hybrid data management (structured + unstructured).
5. **Audit Support:** Maintains transaction logs for accountability and analysis.

12. Results and Discussion

The system successfully integrates relational and non-relational data.

- Query performance improved using indexes on student_id, course_id, and semester.
- Referential constraints maintain data consistency.
- MongoDB integration allows efficient log management.
  Overall, the system ensures reliability, scalability, and user-friendly access.

13. Conclusion

The University Academic Information System effectively automates university academic operations. It ensures data integrity, supports efficient queries, and integrates hybrid data storage. The system provides a strong foundation for future expansion such as AI-based analytics and academic forecasting.

14. References

- · Oracle SQL Developer Documentation
- · MongoDB Manual – MongoDB University
- · Ramez Elmasri & Shamkant Navathe, *Fundamentals of Database Systems*
- · University Data Management Research Papers