ABSTRACT:

With the increasing popularity of e-learning, online courses and educational resources have become widely available. E-learning databases play a crucial role in managing these resources and providing students with a seamless learning experience. In this report, we will explore the schema of an e-learning database management system, which includes tables for department, courses, enrollment, faculty, students, notes, assignments, and more. We will discuss the purpose of each table and the relationships between them. Additionally, we will cover the use of PL/SQL triggers and procedures to enforce business rules and ensure data integrity. Finally, we will examine the potential benefits of using an e-learning DBMS, such as increased efficiency, enhanced communication, and improved learning outcomes. Overall, this report aims to provide insight into the design and implementation of e-learning DBMS, which are essential tools for modern education.

INTRODUCTION:

E-Learning has become increasingly popular in the education industry, providing students with flexible and cost-effective learning opportunities. E-Learning databases serve as a backbone to this dynamic system, allowing users to interact with the platform and store information such as student details, course material, and course assignments.

This report presents a detailed analysis of an E-Learning database schema that consists of tables related to Departments, Courses, Course Enrollment, Faculty, Student, Note, Assignment, Note_Course_Faculty, and Assignment_Course_Faculty. The schema provides a comprehensive set of data structures that allow users to manage course content, student enrollment, and student progress.

The report discusses the design of the schema, the relationships between its tables, and the business rules enforced by the database triggers. Additionally, the report describes the stored procedures that are used to manage the database and retrieve data.

Overall, this report provides a comprehensive overview of the E-Learning database schema, highlighting its key features, benefits, and limitations. The report will be useful to anyone interested in understanding the architecture of an E-Learning database, its functionality, and how it can be used to support modern-day learning needs.

Analysis of Problem Statement:

The problem statement requires designing the RDBMS architecture for an e-learning platform. This involves creating an efficient and scalable database schema to manage the various aspects of an e-learning platform such as courses, assignments, faculty, students, and departments. The schema must support features such as enrollment, note sharing, assignment submissions, faculty and student management, and data analytics.

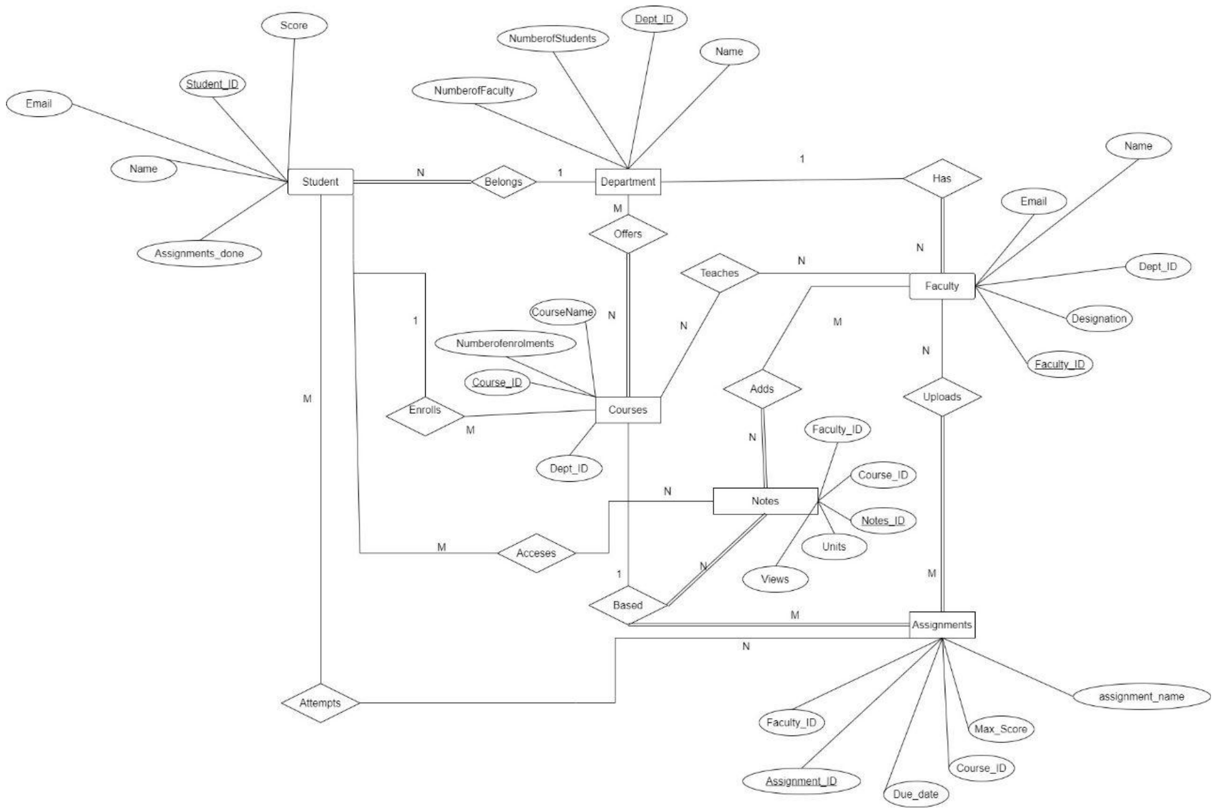
Designing a robust and scalable RDBMS architecture for an e-learning platform can be a challenging task, as it involves creating a schema that can manage a large amount of data and users. The schema must be designed with the future in mind, such that it can handle an increase in the number of users and courses without compromising performance.

To design an efficient schema, one must identify the various entities involved in the e-learning platform and the relationships between them. This requires a thorough understanding of the business requirements and the user needs. Once the entities and relationships are identified, the schema can be designed, and appropriate constraints and triggers can be added to maintain data integrity.

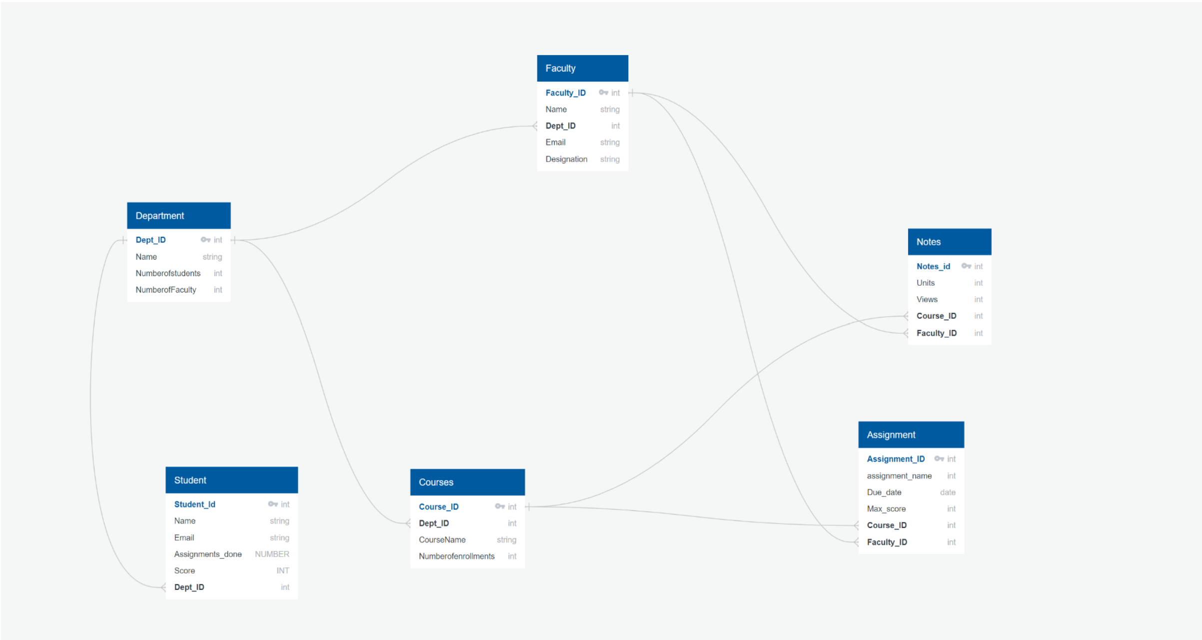
Furthermore, the design of an e-learning platform must consider the security and privacy of the users. This includes the use of encryption, access control mechanisms, and data backup and recovery systems.

Overall, designing the RDBMS architecture for an e-learning platform requires careful consideration of the business requirements, user needs, scalability, and security. The resulting schema must be robust, scalable, and secure to provide an efficient and effective e-learning platform.

E-R DIAGRAM



E-R To Tables



Normalization:

- Functional Dependency

For the Student table:

Student_Id -> Name, Email, Assignments_done, Score, Dept_ID

Email -> Student_Id

Dept_ID -> Department.Dept_ID

For the Department table:

Dept_ID -> Name, Numberofstudents, NumberofFaculty

For the Courses table:

Course_ID -> Dept_ID, CourseName, Numberofenrollments

Dept_ID -> Department.Dept_ID

For the Faculty table:

Faculty_ID -> Name, Email, Designation, Dept_ID

Dept_ID -> Department.Dept_ID

For the Notes table:

Notes_id -> Units, Views, Course_ID, Faculty_ID

Course_ID -> Courses.Course_ID

Faculty_ID -> Faculty.Faculty_ID

For the Assignment table:

Assignment_ID -> assignment_name, Due_date, Max_score, Course_ID, Faculty_ID

Course_ID -> Courses.Course_ID

Faculty_ID -> Faculty.Faculty_ID

Note: PK stands for Primary Key and FK stands for Foreign Key.

First Normal Form:

If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in first normal form if it does not contain any composite or multi-valued attribute.

All the tables in the given RDBMS schema satisfy the requirements of first normal form (1NF).

Second Normal form :

To check if each table is in second normal form (2NF), we need to ensure that the table is in 1NF and that each non-primary key attribute is fully functionally dependent on the primary key.

The original Courses table is not in 2NF because the Numberofenrollments attribute is partially dependent on the primary key.

To bring the Courses table into 2NF without introducing an additional table, we would need to remove the partially dependent attribute (Numberofenrollments) from the Courses table and place it in a separate table along with the foreign key to the Courses table

Notes table is not in 2NF in the original schema. The table has five attributes: Notes_id, Units, Views, Course_ID, and Faculty_ID. The primary key of the table is Notes_id, and all the other attributes are functionally dependent on it.

There is a partial dependency between Course_ID and Notes_id as well as between Faculty_ID and Notes_id.

To bring the Notes table into 2NF, we need to remove the partial dependencies by splitting the table into two tables. One table will contain the Notes_id, Units, and Views attributes, and the other table will contain the Course_ID and Faculty_ID attributes, as well as foreign keys to the Courses and Faculty tables.

The Assignment table is not in 2NF,there is a partial dependency between Course_ID and Assignment_id as well as between Faculty_ID and Assignment_id.To bring the Assignment table into 2NF, we need to remove the partial dependencies by splitting the table into two tables. One table will contain the Assignment_ID, assignment_name, Due_date, and Max_score attributes, and the other table will contain the Course_ID and Faculty_ID attributes, as well as foreign keys to the Courses and Faculty tables.

UPDATED SCHEMA IN 2NF

Student

—-------

Student_Id PK int

Name string

Email string

Dept_ID int FK >- Department.Dept_ID

Department

Dept_ID PK int

Name string

Numberofstudents int

NumberofFaculty int

Courses

Course_ID PK int

Dept_ID int FK >- Department.Dept_ID

CourseName string

Course_Enrollment

Course_ID FK int >- Courses.Course_ID

Numberofenrollments int

Faculty

Faculty_ID PK int

Name string

Dept_ID int FK >- Department.Dept_ID

Email string

Designation string

Note

Note_id PK int

Units int

Views int

Note_Course_Faculty

Note_id PK int FK >- Note_Metadata.Note_id

Course_ID int FK >- Courses.Course_ID

Faculty_ID int FK >- Faculty.Faculty_ID

Assignment

Assignment_ID PK int

assignment_name int

Due_date date

Max_score int

Assignment_Course_Faculty

Assignment_ID PK int FK >- Assignment_Metadata.Assignment_ID

Course_ID int FK >- Courses.Course_ID

Faculty_ID int FK >- Faculty.Faculty_ID

Third Normal Form:

The Student table has a single composite primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Department table has a single primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Courses table has a single primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Course_Enrollment table has a single composite primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Faculty table has a single primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Note table has a single primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Note_Course_Faculty table has a composite primary key that includes foreign keys, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Assignment table has a single primary key, and all non-key attributes are dependent on the entire key. It is in 3NF.

The Assignment_Course_Faculty table has a composite primary key that includes foreign keys, and all non-key attributes are dependent on the entire key. It is in 3NF.

Therefore, the updated schema is in 3NF.

SQL and PL/SQL

1.Creation of Schema

Statement 1

Edit

```
CREATE TABLE Department (
  Dept_ID INT PRIMARY KEY,
  Name VARCHAR2(50) NOT NULL,
  Numberofstudents INT,
  NumberofFaculty INT
)
```

Table created.

Statement 2

Edit

```
CREATE TABLE Courses (
  Course_ID INT PRIMARY KEY,
  Dept_ID INT NOT NULL,
  CourseName VARCHAR2(50) NOT NULL,
  CONSTRAINT FK_Dept_ID_Courses FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)
)
```

Table created.

```
CREATE TABLE Course_Enrollment (
  Course_ID INT NOT NULL,
  Numberofenrollments INT NOT NULL,
  CONSTRAINT PK_Course_Enrollment PRIMARY KEY (Course_ID),
  CONSTRAINT FK_Course_ID_Course_Enrollment FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID)
)
```

Table created.

```
CREATE TABLE Faculty (
  Faculty_ID INT PRIMARY KEY,
  Name VARCHAR2(50) NOT NULL,
  Dept_ID INT NOT NULL,
  Email VARCHAR2(50) NOT NULL,
  Designation VARCHAR2(50) NOT NULL,
  CONSTRAINT FK_Dept_ID_Faculty FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)
)
```

Table created.

```
CREATE TABLE Student (
  Student_Id INT PRIMARY KEY,
  Name VARCHAR2(50) NOT NULL,
  Email VARCHAR2(50) NOT NULL,
  Dept_ID INT NOT NULL,
  CONSTRAINT FK_Dept_ID_Student FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)
)
```

Table created.

```sql
CREATE TABLE Note (
  Note_id INT PRIMARY KEY,
  Units INT NOT NULL,
  Views INT NOT NULL
)
```

Table created.

```sql
CREATE TABLE Note_Course_Faculty (
  Note_id INT NOT NULL,
  Course_ID INT NOT NULL,
  Faculty_ID INT NOT NULL,
  CONSTRAINT PK_Note_Course_Faculty PRIMARY KEY (Note_id),
  CONSTRAINT FK_Note_id_Note_Course_Faculty FOREIGN KEY (Note_id) REFERENCES Note(Note_id),
  CONSTRAINT FK_Course_ID_Note_Course_Faculty FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID),
  CONSTRAINT FK_Faculty_ID_Note_Course_Faculty FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID)
)
```

Table created.

```sql
CREATE TABLE Assignment (
  Assignment_ID INT PRIMARY KEY,
  assignment_name VARCHAR2(50) NOT NULL,
  Due_date DATE NOT NULL,
  Max_score INT NOT NULL
)
```

Table created.

```sql
CREATE TABLE Assignment_Course_Faculty (
  Assignment_ID INT NOT NULL,
  Course_ID INT NOT NULL,
  Faculty_ID INT NOT NULL,
  CONSTRAINT PK_Assignment_Course_Faculty PRIMARY KEY (Assignment_ID),
  CONSTRAINT FK_Assignment_ID_Assignment_Course_Faculty FOREIGN KEY (Assignment_ID) REFERENCES Assignment(Assignment_ID),
  CONSTRAINT FK_Course_ID_Assignment_Course_Faculty FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID),
  CONSTRAINT FK_Faculty_ID_Assignment_Course_Faculty FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID)
)
```

Table created.

2.Triggers

a)Trigger to Implement Dept_ID as Foreign Key in Student Table

```sql
CREATE OR REPLACE TRIGGER student_dept_fk_trg
BEFORE INSERT OR UPDATE ON Student
FOR EACH ROW
DECLARE
  dept_count NUMBER;
BEGIN
  -- Check if the new department ID exists in the Department table
  SELECT COUNT(*) INTO dept_count
  FROM Department
  WHERE Dept_ID = :NEW.Dept_ID;

  -- If department ID does not exist, raise an application error
  IF dept_count = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Foreign key constraint violation: Dept_ID does not exist in Department table');
  END IF;
END;
```

Trigger created.

```sql
INSERT INTO Student (Student_Id, Name, Email, Dept_ID)
VALUES (6, 'Jane Doe', 'janedoe@example.com', 999)
```

ORA-20001: Foreign key constraint violation: Dept_ID does not exist in Department table ORA-06512: at "SQL_BINUUHEVRXYEKBCFGMCCMIPJN.STUDENT_DEPT_FK_TRG", line 11 ORA-06512: at "SYS.DBMS_SQL", line 1721

b)Trigger to assign NumberofStudents and NumberofFaculty to zero when inserting a new Department.

```sql
CREATE OR REPLACE TRIGGER department_new_dept_trg
BEFORE INSERT ON Department
FOR EACH ROW
BEGIN
   :NEW.Numberofstudents := 0;
   :NEW.NumberofFaculty := 0;
END;
```

```sql
INSERT INTO Department (Dept_ID, Name)
VALUES (16, 'Computer Science')
```

1 row(s) inserted.

```sql
SELECT * FROM Department Where Dept_ID is 16
```

ORA-00908: missing NULL keyword

```sql
SELECT * FROM Department Where Dept_ID = 16
```

DEPT_ID	NAME	NUMBEROFSTUDENTS	NUMBEROFFACULTY
16	Computer Science	0	0

Download CSV

c)Trigger to check that updated due date must be after the current date

```
CREATE OR REPLACE TRIGGER trg_assignment_due_date
BEFORE UPDATE ON Assignment
FOR EACH ROW
DECLARE
    curr_date DATE := SYSDATE;
BEGIN
    IF :NEW.Due_date < curr_date THEN
        RAISE_APPLICATION_ERROR(-20001, 'New due date cannot be in the past.');
    END IF;
END;
```

Trigger created.

```
UPDATE Assignment
SET Due_date = TO_DATE('2022-01-01', 'YYYY-MM-DD')
WHERE Assignment_ID = 1
```

ORA-20001: New due date cannot be in the past. ORA-06512: at "SQL_BINUUHEVRXYEKBCFGMCCMIPJN.TRG_ASSIGNMENT_DUE_DATE", line 5 ORA-06512: at "SYS.DBMS_SQL", line 1721

d)Trigger to check that Max_score for an assignment is not negative

```
CREATE OR REPLACE TRIGGER trg_assignment_max_score
BEFORE INSERT OR UPDATE ON Assignment
FOR EACH ROW
BEGIN
    IF :NEW.Max_score < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Max score for an assignment cannot be negative.');
    END IF;
END;
```

Trigger created.

```
INSERT INTO Assignment (Assignment_ID, assignment_name, Due_date, Max_score)
VALUES (29, 'Negative Score Assignment', SYSDATE, -10)
```

ORA-20001: Max score for an assignment cannot be negative. ORA-06512: at "SQL_BINUUHEVRXYEKBCFGMCCMIPJN.TRG_ASSIGNMENT_MAX_SCORE", line 3 ORA-06512: at "SYS.DBMS_SQL", line 1721

2. Procedures

a)Procedure to insert into department table.

```
CREATE OR REPLACE PROCEDURE insert_department(
    p_dept_id IN Department.Dept_ID%TYPE,
    p_name IN Department.Name%TYPE,
    p_num_students IN Department.Numberofstudents%TYPE,
    p_num_faculty IN Department.NumberofFaculty%TYPE
)
AS
BEGIN
    INSERT INTO Department(Dept_ID, Name, Numberofstudents, NumberofFaculty)
    VALUES (p_dept_id, p_name, p_num_students, p_num_faculty);
    COMMIT;
END;
```

Procedure created.

```
BEGIN
    insert_department(526, 'History', 50, 5);
END;
```

Statement processed.

526	History	0	0

b)Procedure to Update course_name

```sql
CREATE OR REPLACE PROCEDURE update_course_name(
    p_course_id IN Courses.Course_ID%TYPE,
    p_new_name IN Courses.CourseName%TYPE
)
AS
BEGIN
    UPDATE Courses
    SET CourseName = p_new_name
    WHERE Course_ID = p_course_id;
    COMMIT;
END;
```

```sql
SELECT * FROM Courses
```

COURSE_ID	DEPT_ID	COURSENAME
1	1	Database Management
2	1	Computer Networks
3	2	Analog Circuits
4	2	Digital Electronics
5	3	Thermodynamics

Download CSV

rows selected.

```
BEGIN
    update_course_name(2, 'Algorithms');
END;
```

Statement processed.

```
SELECT * FROM Courses
```

COURSE_ID	DEPT_ID	COURSENAME
1	1	Database Management
2	1	Algorithms
3	2	Analog Circuits
4	2	Digital Electronics
5	3	Thermodynamics

Download CSV

5 rows selected.

c)Procedure to delete student

```
CREATE OR REPLACE PROCEDURE delete_student(
    p_student_id IN Student.Student_Id%TYPE
)
AS
BEGIN
    DELETE FROM Student
    WHERE Student_Id = p_student_id;
    COMMIT;
END;
```

Procedure created.

```
SELECT * FROM Student
```

STUDENT_ID	NAME	EMAIL	DEPT_ID
1	Alice Johnson	alicejohnson@university.edu	1
2	Bob Smith	bobsmith@university.edu	2
3	Emily Davis	emilydavis@university.edu	1
4	Jake Lee	jakelee@university.edu	3
5	Sophia Patel	sophiapatel@university.edu	2

Download CSV

```
BEGIN
    delete_student(3);
END;
```

Statement processed.

```
SELECT * FROM Student
```

STUDENT_ID	NAME	EMAIL	DEPT_ID
1	Alice Johnson	alicejohnson@university.edu	1
2	Bob Smith	bobsmith@university.edu	2
4	Jake Lee	jakelee@university.edu	3
5	Sophia Patel	sophiapatel@university.edu	2

Download CSV

4 rows selected.

3.Cursors

a) Getting department names using cursor

```
DECLARE
  CURSOR dept_cursor IS
    SELECT Name FROM Department;
  dept_name Department.Name%TYPE;
BEGIN
  OPEN dept_cursor;
  LOOP
    FETCH dept_cursor INTO dept_name;
    EXIT WHEN dept_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(dept_name);
  END LOOP;
  CLOSE dept_cursor;
END;
```

Statement processed.
Computer Science
Electronics
Mechanical Engineering
Civil Engineering
Chemical Engineering
Computer Science
History

b) **Fetching Content of Course Table using Cursor**

```
DECLARE
  -- Declare the cursor
  CURSOR course_cursor IS
    SELECT *
    FROM Courses;

  -- Declare variables to store the column values
  course_id Courses.Course_ID%TYPE;
  dept_id Courses.Dept_ID%TYPE;
  course_name Courses.CourseName%TYPE;

BEGIN
  -- Open the cursor
  OPEN course_cursor;

  -- Loop through the cursor and display the data
  LOOP
    -- Fetch the next row from the cursor
    FETCH course_cursor INTO course_id, dept_id, course_name;

    -- Exit the loop if there are no more rows to fetch
    EXIT WHEN course_cursor%NOTFOUND;

    -- Display the row data
    DBMS_OUTPUT.PUT_LINE('Course ID: ' || course_id);
    DBMS_OUTPUT.PUT_LINE('Department ID: ' || dept_id);
    DBMS_OUTPUT.PUT_LINE('Course Name: ' || course_name);
    DBMS_OUTPUT.PUT_LINE('------------------');
  END LOOP;

  -- Close the cursor
  CLOSE course_cursor;
END;
```

```
Statement processed.
Course ID: 1
Department ID: 1
Course Name: Database Management
------------------
Course ID: 2
Department ID: 1
Course Name: Algorithms
------------------
Course ID: 3
Department ID: 2
Course Name: Analog Circuits
------------------
Course ID: 4
Department ID: 2
Course Name: Digital Electronics
------------------
Course ID: 5
Department ID: 3
Course Name: Thermodynamics
------------------
```

Conclusion:

In conclusion, the design of an DBMS architecture for an e-learning platform is a complex task that involves the consideration of various factors such as user requirements, scalability, security, and performance. The schema provided in this report aims to address these factors by creating a robust and flexible database structure that can efficiently manage the platform's data.

The schema includes several tables such as Department, Courses, Course_Enrollment, Faculty, Student, Note, Note_Course_Faculty, Assignment, and Assignment_Course_Faculty. Stored procedures and triggers are also included to ensure data consistency, accuracy, and security.

The assumptions and limitations of this schema include the assumption that the platform will have a finite number of departments, courses, faculty members, and students. Additionally, the schema assumes that there will be no significant changes to the platform's data requirements in the future.

Overall, this report presents a well-designed DBMS architecture for an e-learning platform that considers essential factors such as user requirements, scalability, security, and performance. The schema, along with its corresponding SQL/PLSQL commands, provides a robust and flexible database structure that can effectively manage an e-learning platform's data.