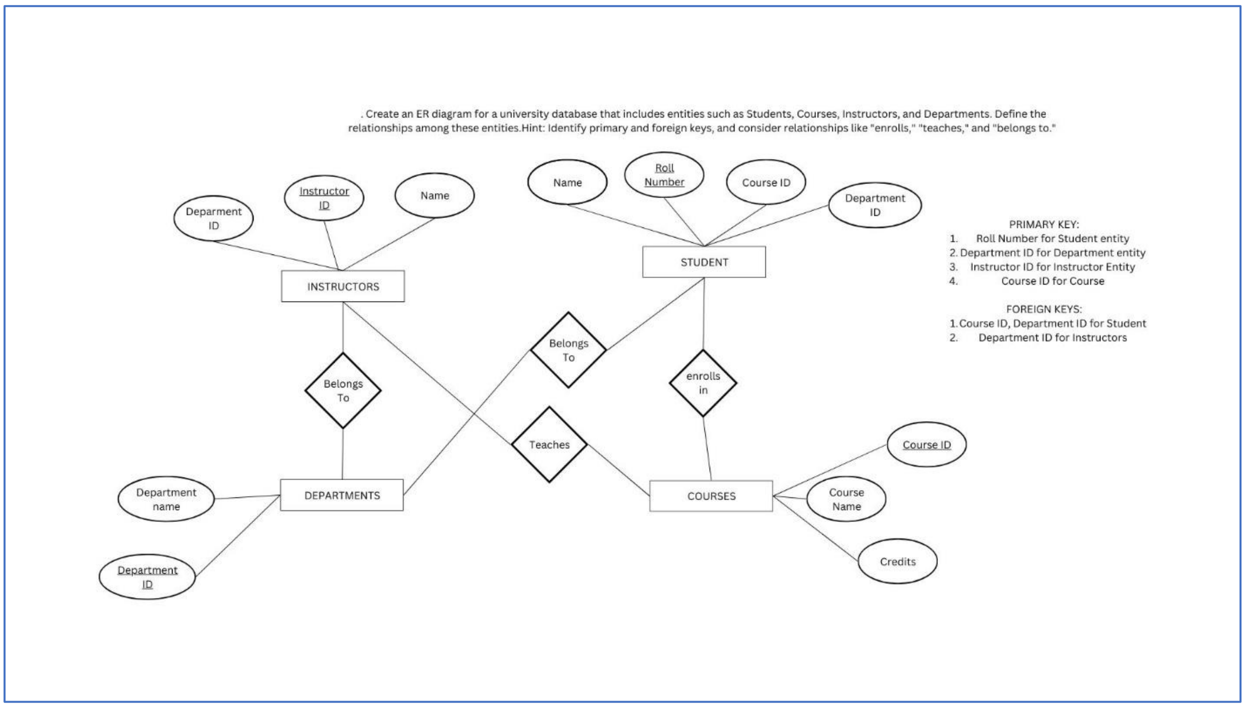
Create an ER diagram for a university database that includes entities such as Students, Courses, Instructors, and Departments. Define the relationships among these entities. Hint: Identify primary and foreign keys, and consider relationships like "enrolls," "teaches," and "belongs to."



. Create an ER diagram for a university database that includes entities such as Students, Courses, Instructors, and Departments. Define the relationships among these entities.Hint: Identify primary and foreign keys, and consider relationships like "enrolls," "teaches," and "belongs to."

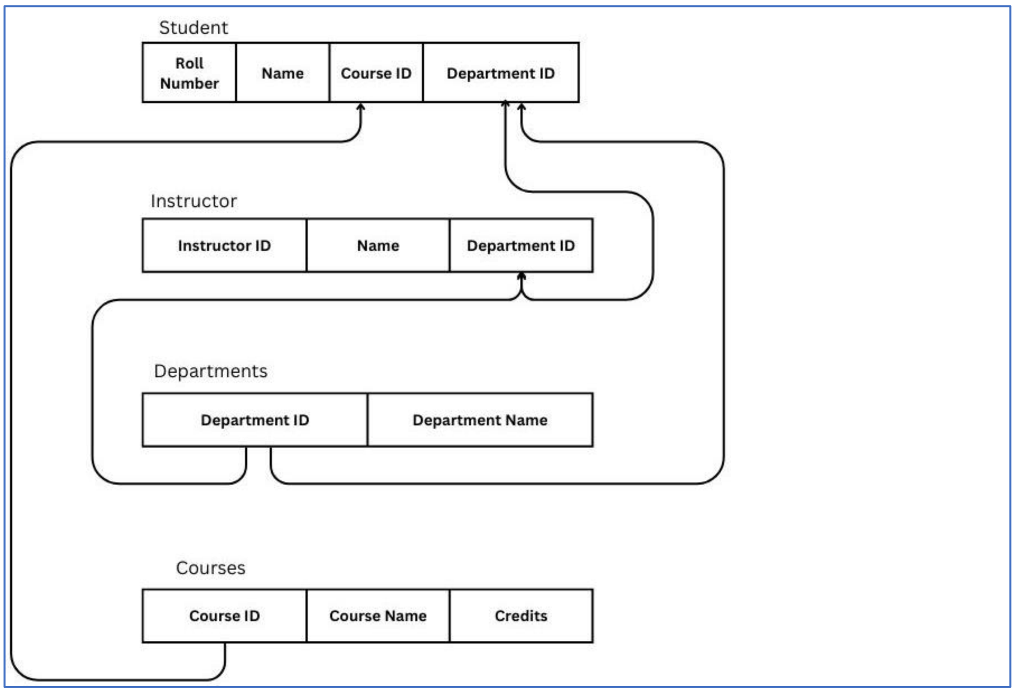
PRIMARY KEY:

1. Roll Number for Student entity
2. Department ID for Department entity
3. Instructor ID for Instructor Entity
4. Course ID for Course

FOREIGN KEYS:

1. Course ID, Department ID for Student
2. Department ID for Instructors

Convert the ER diagram you created into a relational schema. Clearly define tables, primary keys, and foreign keys.Hint: Break down the entities into tables and show how they relate through keys.



Given a table with the following structure, normalize it to 3NF: Employee (EmployeeID, EmployeeName, DepartmentID, DepartmentName, ProjectID, ProjectName, HoursWorked) Hint: Start by removing partial and transitive dependencies.

---

**Candidate Keys** = {EmployeeID, DepartmentID, ProjectID}

**Functional Dependencies:**

EmployeeID → EmployeeName, DepartmentID

DepartmentID → DepartmentName

ProjectID → ProjectName

EmployeeId, ProjectID →HoursWorked

---

*Assuming that the table is in 1NF, Converting to 2NF –*

Forming 4 tables:

1. Employee (EmployeeID, EmployeeName, DepartmentID)
2. EmployeeProject (EmployeeID, ProjectID, HoursWorked)
3. Project (ProjectID, ProjectName)
4. Department (DepartmentID, DepartmentName)

---

*Converting to 3NF –*

1. EmployeeID → EmployeeName, DepartmentID
2. EmployeeID, ProjectID → HoursWorked
3. ProjectID → ProjectName
4. DepartmentID → DepartmentName

*Since none of the tables have any transitive dependencies, they are all in 3NF*

Design a simple transaction in SQL to transfer money between two accounts. Ensure that your transaction adheres to the ACID properties.Hint: Use BEGIN TRANSACTION, COMMIT, and ROLLBACK.

```
begin transaction;
declare @sender_acc = 1;
declare @receiver_acc = 2;
declare @amount = 100;

if (select balance from account where account = @sender_acc) >=
@amount
begin
    update account
    set balance = balance - @amount
    where account = @sender_acc;

    update account
    set balance = balance + @amount
    where account = @receiver_acc;

    commit;
end

else
begin
    rollback;
end
```

## Discuss different types of indexes and their use cases

1. UNIQUE INDEX: it ensures that the column that has been indexed upon, only consists of unique values.
   USES: they are generally used on primary keys to keep the record unique. It also maintain the integrity of the database.

   ```
   Syntax:
   CREATE UNIQUE INDEX index_name
   on table (col);
   ```

2. SINGLE – COLUMN INDEX: this indexing is done on only one column.
   USES: It speeds up the queries that is based on the indexed column, speeds up join operations, improves sorting.

   ```
   Syntax:
   CREATE INDEX index_name
   ON table(col);
   ```

3. COMPOSITE INDEX: this indexing is done on multiple columns.
   USES: Similar to single column indexing, composite indexing improves sorting, speeds up join operations, and other queries that are based upon the indexed columns.

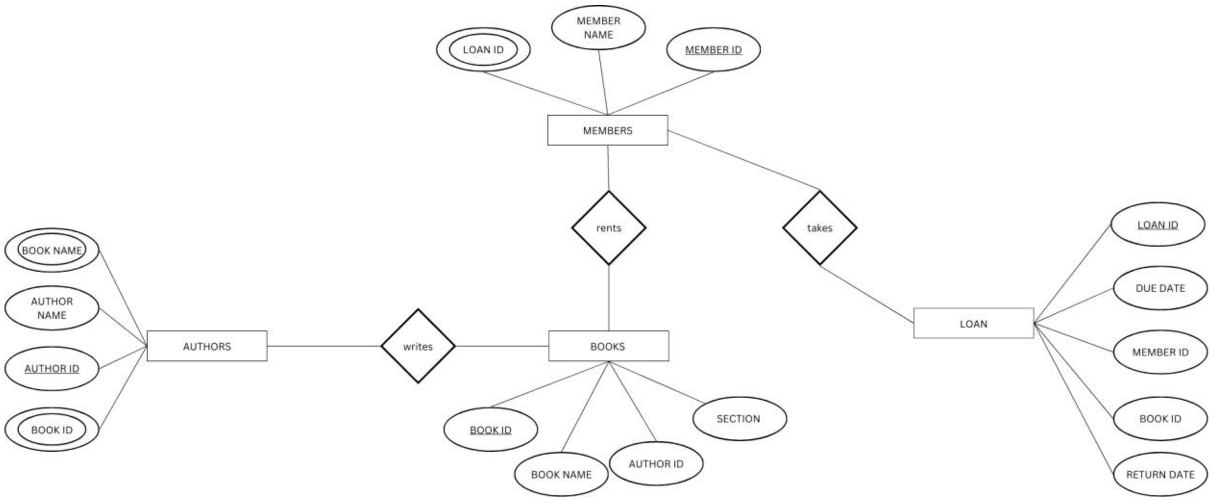
   ```
   Syntax:
   CREATE INDEX index_name
   ON table(col1, col2…);
   ```

4. IMPLICIT INDEX: this is the automatic indexing done by the system when some constraints like unique, primary etc are used on the table. They are not created by the user.

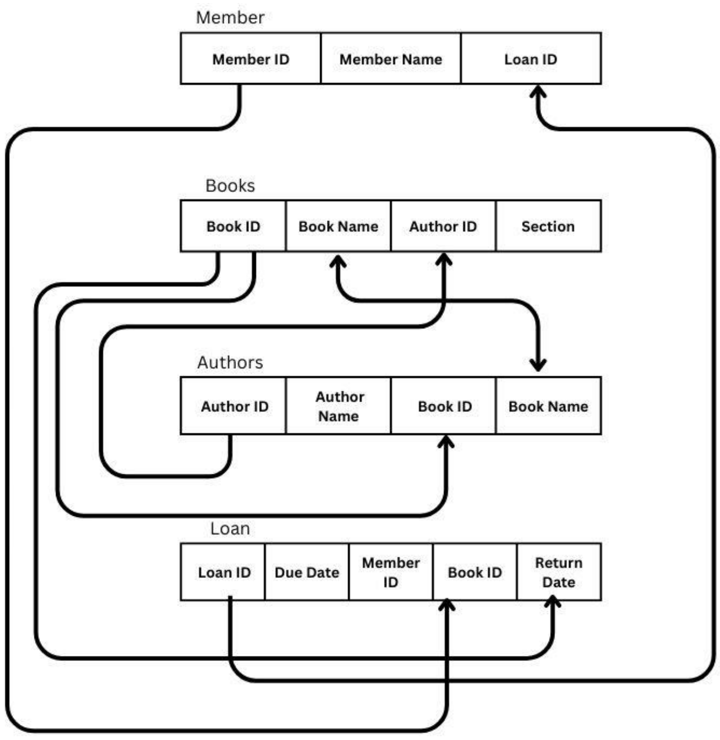
Design a small database for a library system, including entities such as Books, Members, Loans, and Authors. Provide the ER diagram, relational schema, and sample SQL queries to manage the system. Hint: Include queries for checking out books, returning books, and querying overdue loans

---

ER DIAGRAM:



Relational Schema:



SAMPLE QUERIES:

```
--SAMPLE VARIABLE--
declare @current_date = '10-09-24'
declare @due_date = '17-09-24'
declare @returndate = NULL
declare @memberID = 1234
declare @bookID = 987
```

```
-- Check out a book
INSERT INTO Loans (LoanID, Due_date, MemberId, BookID, Return_date)
VALUES (1, @due_date, @memberID, @bookID, @returndate);

UPDATE Members
SET LoanID = (
    SELECT LoanID
    FROM Loans
    WHERE Loans.BookID = @bookId AND Loans.MemberID =
Members.MemberID
)
WHERE MemberID = @memberID;
```

```
--returning a book
UPDATE Loans
SET Return_date = @current_date
WHERE MemberID = @memberID AND BookID = @bookID;

UPDATE Members
SET LoanID = NULL
WHERE MemberID = @memberID AND LoanID = (
    SELECT LoanID FROM Loans WHERE MemberID = @memberID AND BookID =
@bookID);
```

```
--querying overdue loans
select * from Loans
where Loans.MemberID = @memberID
```