Assignment-4: DBMS

Topics: Relational Algebra, ER Models, FDs, and Normalization

1. Basic Operations of Relational Algebra

Relational Algebra provides a set of operations to manipulate relations (tables) in a database.

Operation	Description	Example
SELECT (σ)	Filters rows based on a condition.	σDept = 'CSE'(STUDENT) — selects all CSE students.
PROJECT (π)	Selects specific columns.	πName,Dept(STUDENT) — shows only Name and Dept columns.
UNION (∪)	Combines tuples from two relations.	STUDENT1 ∪ STUDENT2
SET DIFFERENCE (−)	Shows tuples in one relation but not in the other.	STUDENT − ALUMNI
CARTESIAN PRODUCT (×)	Combines every tuple of one relation with another.	STUDENT × DEPARTMENT
RENAME (ρ)	Renames relation or attributes.	ρ(S)(STUDENT) — renames STUDENT as S
JOIN (⋈)	Combines related tuples using common attributes.	STUDENT ⋈STUDENT.DeptID = DEPARTMENT.DeptIDDEPARTMENT

2. Relational Algebra Expressions

Relations:

```
STUDENT(RollNo, Name, DeptID)
DEPARTMENT(DeptID, DeptName)
```

(a) Students in 'CSE' department:

πName(σDeptName='CSE'(STUDENT ⋈STUDENT.DeptID = DEPARTMENT.DeptID DEPARTMENT))

(b) Departments with no students:

πDeptName(DEPARTMENT) − πDeptName(STUDENT ⋈ DEPARTMENT)

3. ER Diagram for Online Shopping System

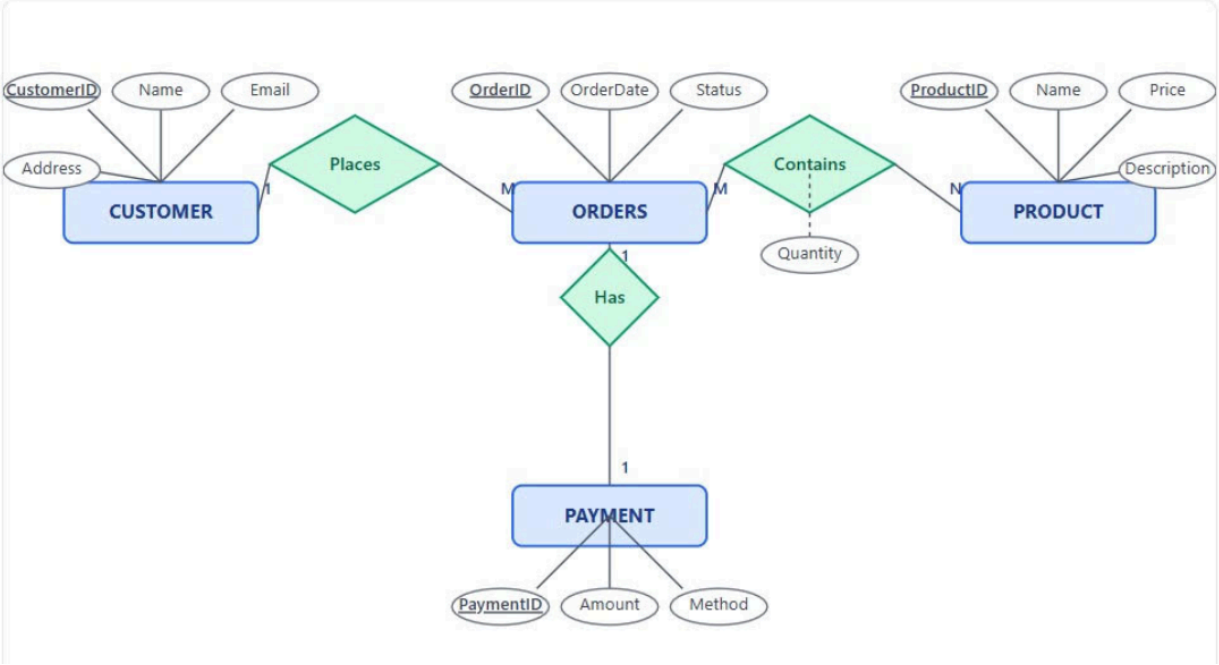
Entities:

- **Customer**(CustomerID, Name, Email, Phone)
- **Product**(ProductID, ProductName, Price, StockQty)
- **Order**(OrderID, OrderDate, CustomerID)
- **Payment**(PaymentID, OrderID, Amount, Date, Mode)

Relationships:

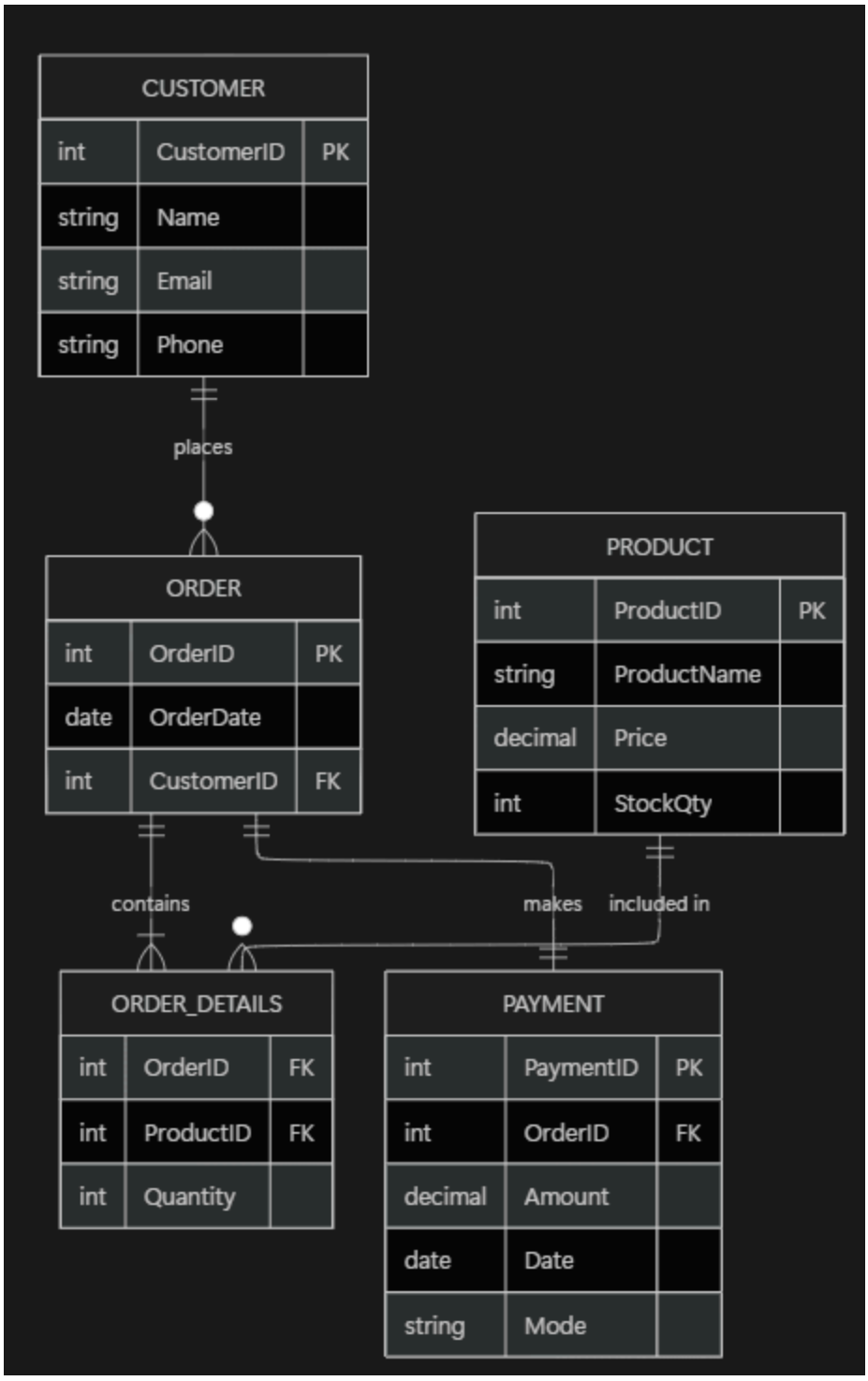
- Customer places Order → *One-to-Many*
- Order contains Product → *Many-to-Many* (Resolved using OrderDetails)
- Order makes Payment → *One-to-One*

ER Model:



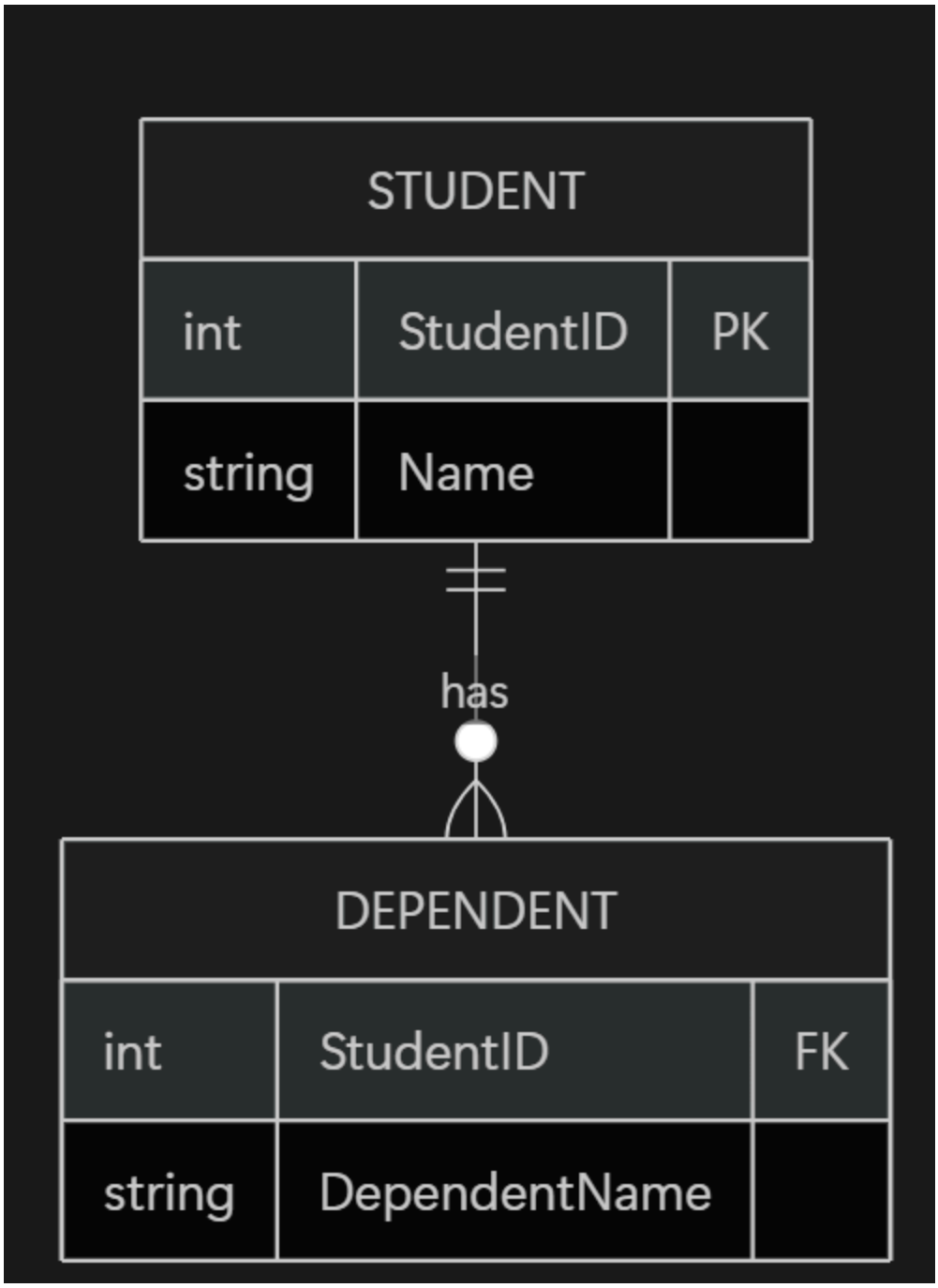
Relational Schema:

CUSTOMER(CustomerID PK, Name, Email, Phone)
PRODUCT(ProductID PK, ProductName, Price, StockQty)
ORDER(OrderID PK, OrderDate, CustomerID FK)
ORDER_DETAILS(OrderID FK, ProductID FK, Quantity)
PAYMENT(PaymentID PK, OrderID FK, Amount, Date, Mode)


CUSTOMER
int
CustomerID
PK
string
Name
string
Email
string
Phone
places
ORDER
int
OrderID
PK
date
OrderDate
int
CustomerID
FK
PRODUCT
int
ProductID
PK
string
ProductName
decimal
Price
int
StockQty
contains
makes
included in
ORDER_DETAILS
int
OrderID
FK
int
ProductID
FK
int
Quantity
PAYMENT
int
PaymentID
PK
int
OrderID
FK
decimal
Amount
date
Date
string
Mode

4. Strong vs Weak Entity Sets

Feature	Strong Entity	Weak Entity
Existence	Independent	Depends on strong entity
Key	Has primary key	Has partial key
Example	STUDENT(StudentID, Name)	DEPENDENT(StudentID FK, DependentName)
Relationship	Identifying not needed	Needs identifying relationship



5. Functional Dependency (FD)

A **Functional Dependency** (FD) is a relationship between attributes such that one attribute uniquely determines another.

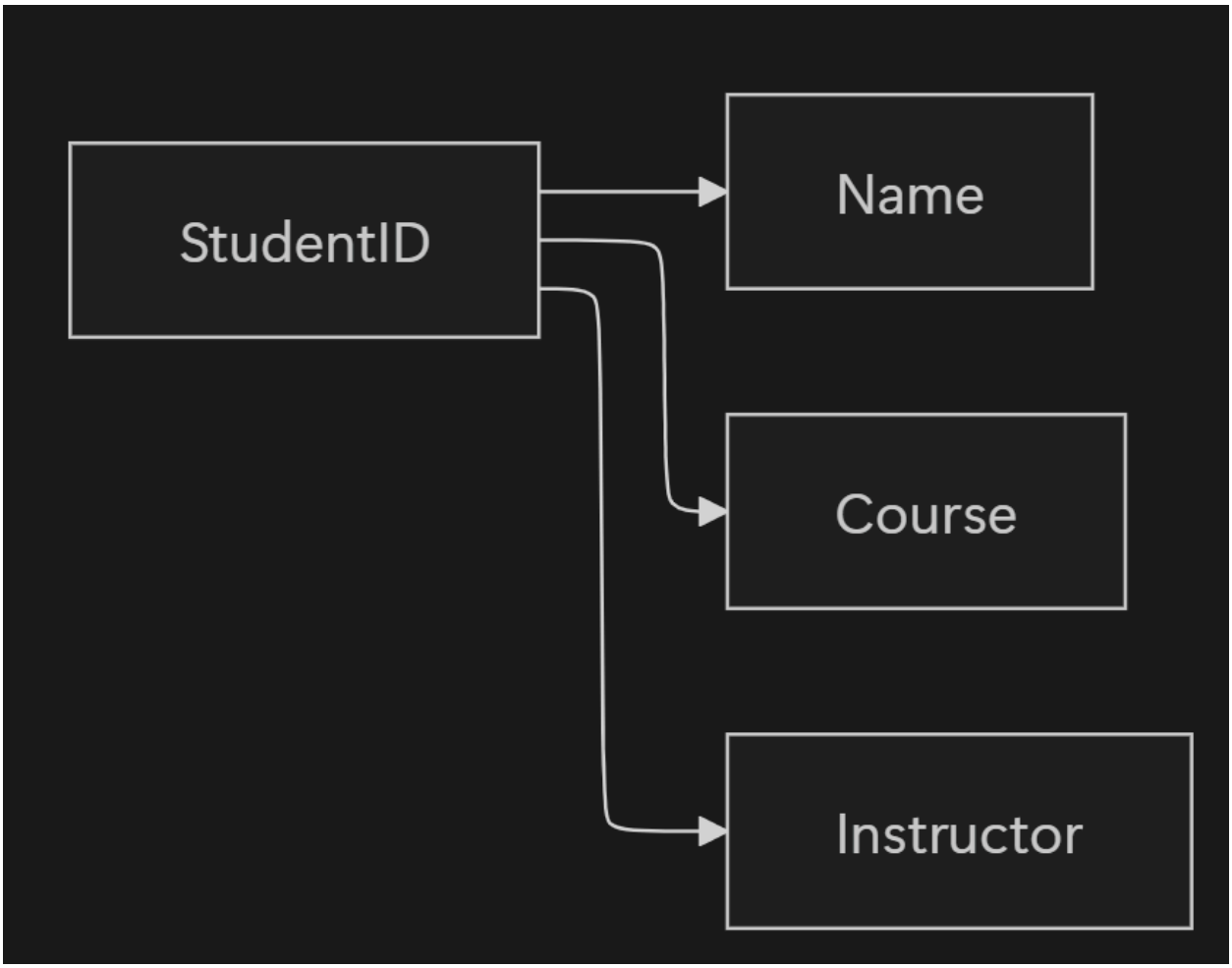
Example:

If A → B, then for every value of A, there is exactly one value of B.

Candidate Key Identification Example:

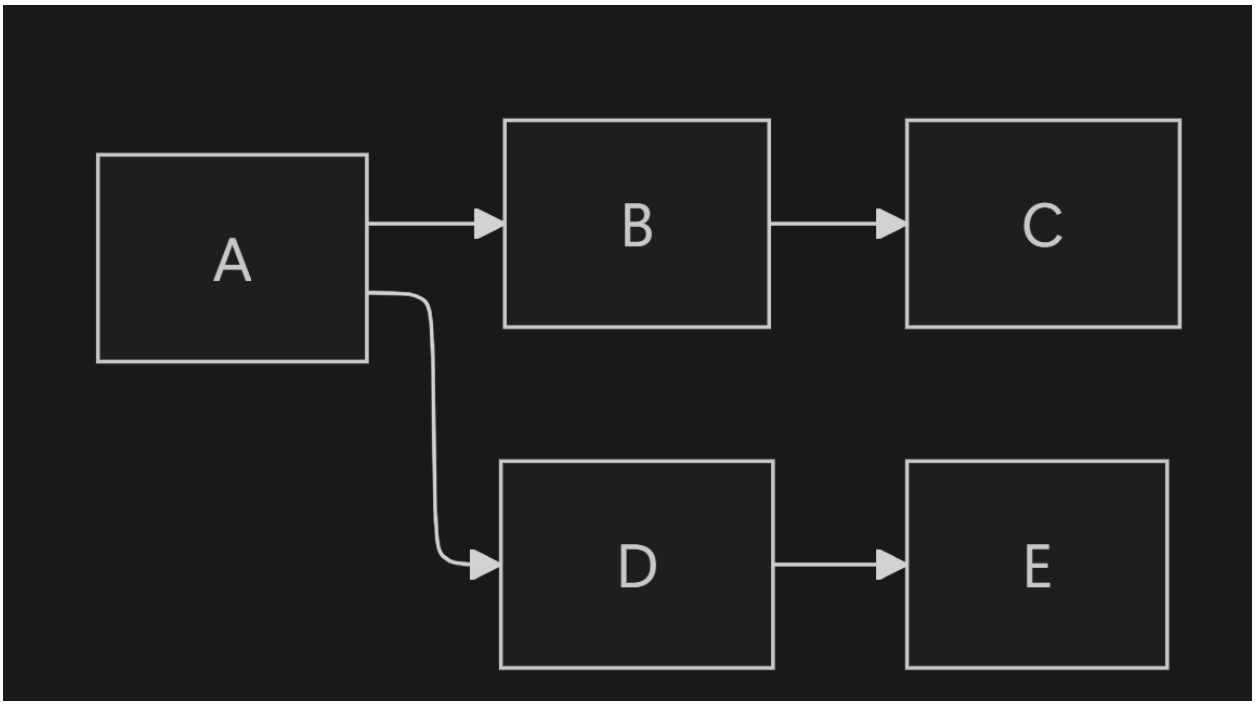
```
STUDENT(StudentID, Name, Course, Instructor)
FDs: StudentID → Name, Course, Instructor
```

Here, StudentID uniquely identifies all attributes → Candidate Key = {StudentID}.



6. Relation R(A, B, C, D, E)

FDs: {A → B, B → C, A → D, D → E}



Step 1: Find Closure of A

A^+ = {A, B, C, D, E} → A is a **Candidate Key**.

Step 2: Normalize

- **1NF:** Already atomic.
- **2NF:** No partial dependency (A is single attribute key).
- **3NF:** Each non-prime attribute depends only on key.

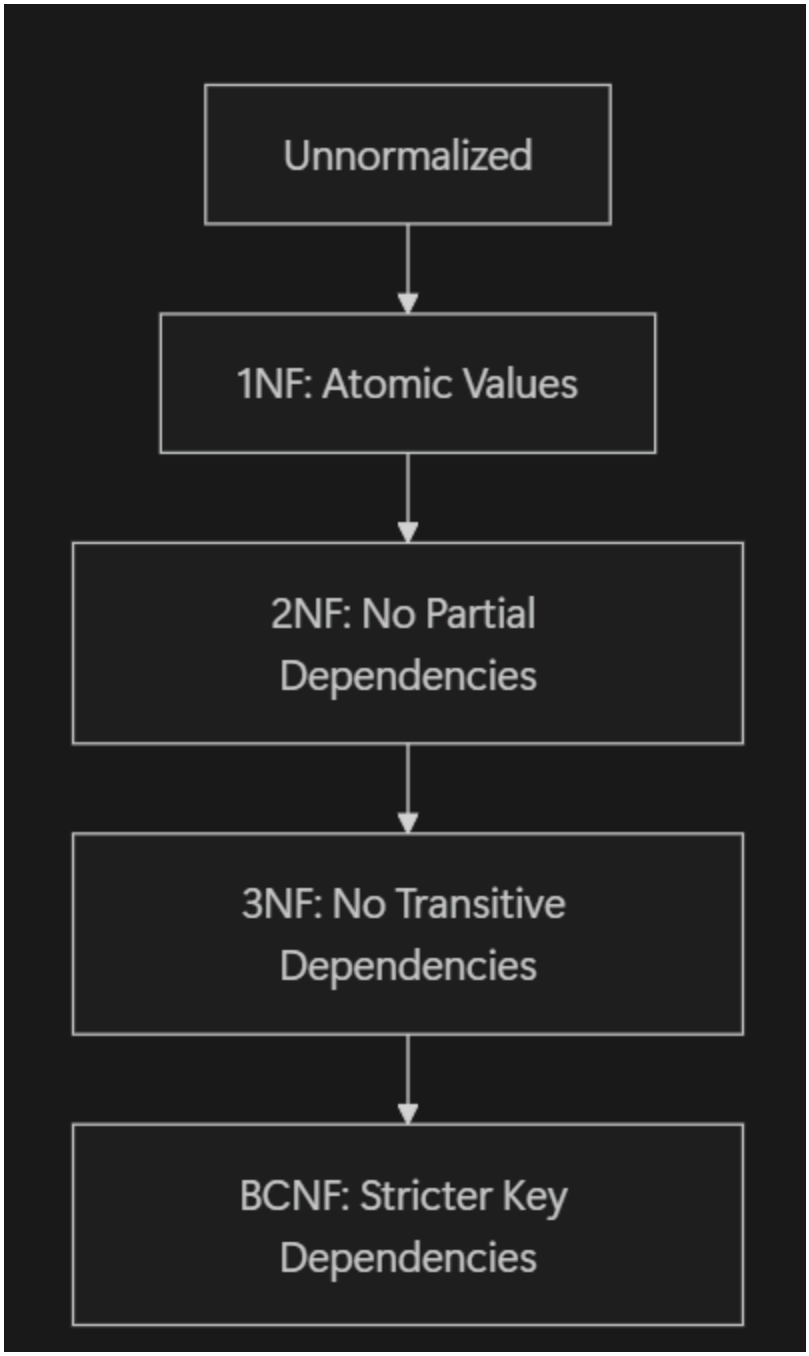
Decomposition:

```
R1(A, B, C, D, E) →
R1(A, B, D)
R2(B, C)
R3(D, E)
```

7. 1NF, 2NF, 3NF Differences

Normal Form	Condition	Example

1NF	No multivalued attributes.	StudentID, Subjects {Math, CS}
2NF	1NF + No partial dependency on part of composite key.	(RollNo, Subject) → Marks
3NF	2NF + No transitive dependency.	RollNo → Dept → HOD



8. Relational Algebra

(a) Employees with salary > 50,000:

σSalary > 50000(EMPLOYEE)

(b) Departments with >5 employees:

πDeptID(σCOUNT(EmpID) > 5(EMPLOYEE))

(Using aggregation equivalent).

9. Library Management System (ER + Schema)

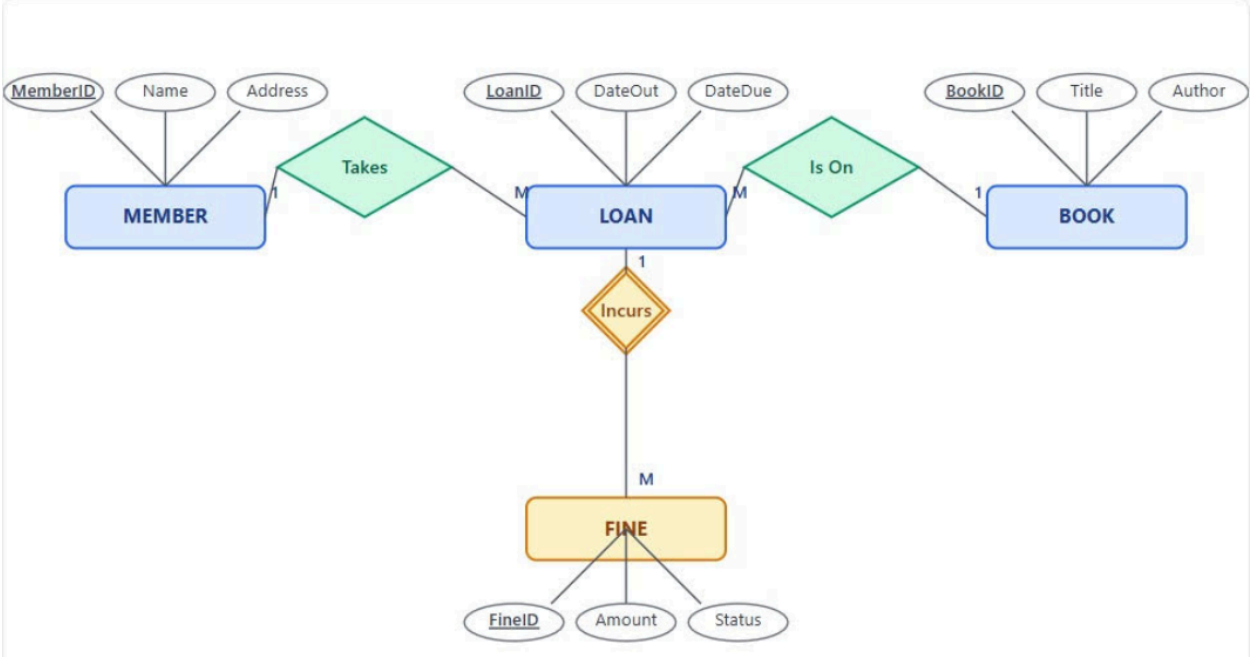
Entities:

- BOOK(BookID, Title, Author, Publisher)
- MEMBER(MemberID, Name, Address)
- LOAN(LoanID, BookID, MemberID, IssueDate, ReturnDate)
- FINE(FineID, LoanID, Amount, DatePaid)

Relationships:

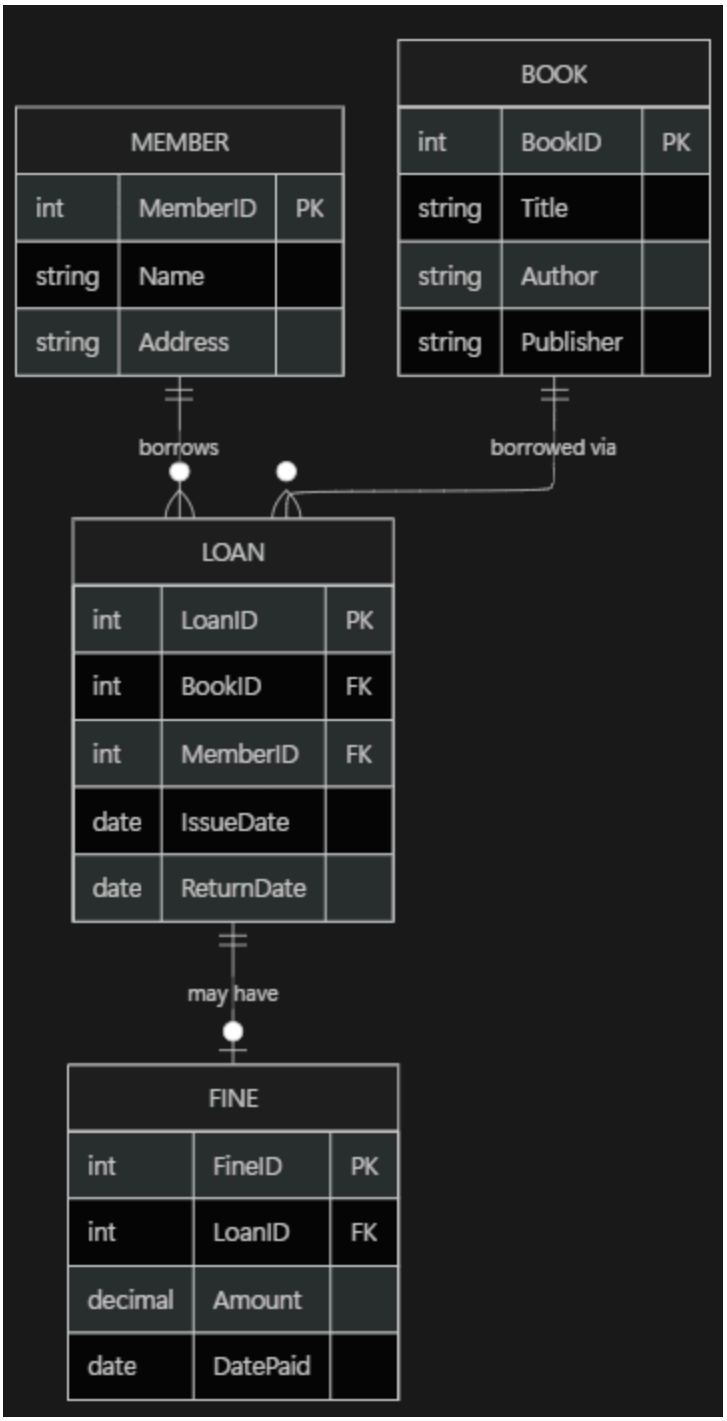
- MEMBER borrows BOOK (via LOAN)
- LOAN may have FINE

ER Model:



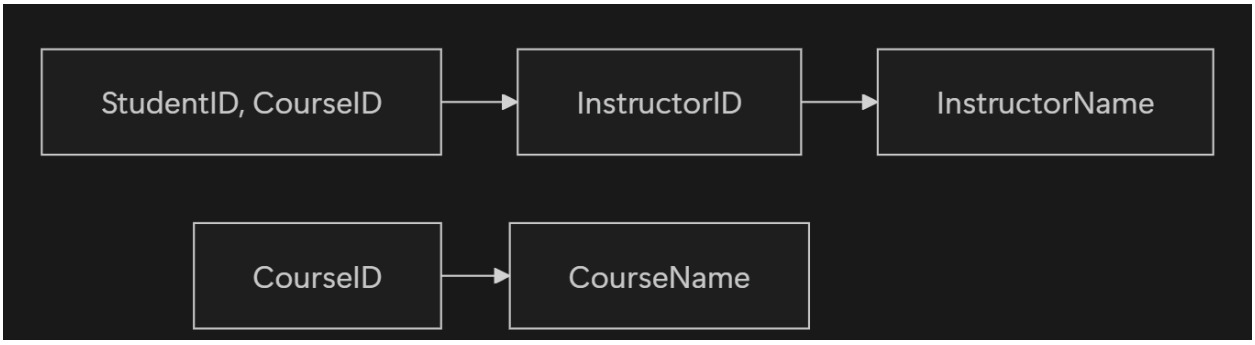
Schema:

BOOK(BookID PK, Title, Author, Publisher)
MEMBER(MemberID PK, Name, Address)
LOAN(LoanID PK, BookID FK, MemberID FK, IssueDate, ReturnDate)
FINE(FineID PK, LoanID FK, Amount, DatePaid)


BOOK
int BookID PK
string Title
string Author
string Publisher
MEMBER
int MemberID PK
string Name
string Address
borrows
borrowed via
LOAN
int LoanID PK
int BookID FK
int MemberID FK
date IssueDate
date ReturnDate
may have
FINE
int FineID PK
int LoanID FK
decimal Amount
date DatePaid

10. Relation R(StudentID, CourseID, InstructorID, InstructorName, CourseName)

FDs: {StudentID, CourseID → InstructorID; InstructorID → InstructorName; CourseID → CourseName}



3NF Decomposition:

R1(StudentID, CourseID, InstructorID)
R2(InstructorID, InstructorName)
R3(CourseID, CourseName)

→ Each table now satisfies 3NF (no transitive dependencies).

11. Redundancy & Anomalies Reduction

Redundancy: Repetition of data (e.g., same InstructorName stored multiple times).

Anomalies:

- **Insertion:** Cannot insert course without student.
- **Updation:** Change in instructor name requires multiple updates.
- **Deletion:** Deleting last student may delete course info.

Normalization splits data into logical tables → removes these anomalies.

12. Relational Algebra Queries

Given:

```
EMPLOYEE(EmpID, Name, DeptID, Salary)
DEPARTMENT(DeptID, DeptName, Location)
```

(a) Average salary per department:

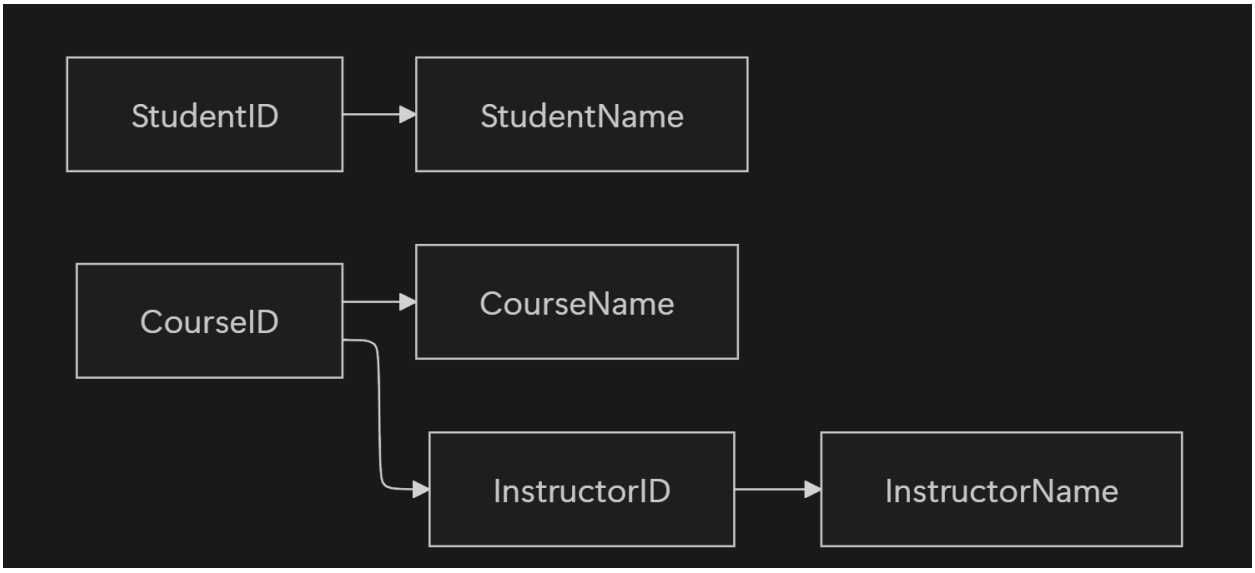
γDeptID, AVG(Salary)(EMPLOYEE)

(b) Names of employees in 'IT' dept:

πName(σDeptName='IT'(EMPLOYEE ⋈EMPLOYEE.DeptID=DEPARTMENT.DeptID DEPARTMENT))

13. STUDENT_INFO Decomposition

FDs: {StudentID → StudentName, CourseID → CourseName, InstructorID → InstructorName, CourseID → InstructorID}



Anomalies:

- Redundant instructor and course names.
- Update anomalies.

Normalization:

1. **1NF:** Already atomic.
2. **2NF:** Key = (StudentID, CourseID). Split partial dependencies.

```
R1(StudentID, StudentName)
R2(CourseID, CourseName, InstructorID)
```

3. **3NF:** InstructorID → InstructorName → Separate table.

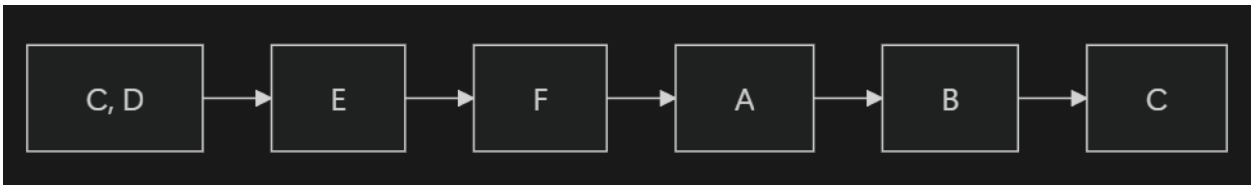
```
R3(InstructorID, InstructorName)
```

Final 3NF:

```
R1(StudentID, StudentName)
R2(CourseID, CourseName, InstructorID)
R3(InstructorID, InstructorName)
```

14. Relation R(A, B, C, D, E, F)

FDs: {A → B, B → C, CD → E, E → F, F → A}



Step 1: Closure

From F → A → B → C, CD → E → F forms a loop.

Candidate Key = **{C, D}**.

Step 2: Highest Normal Form

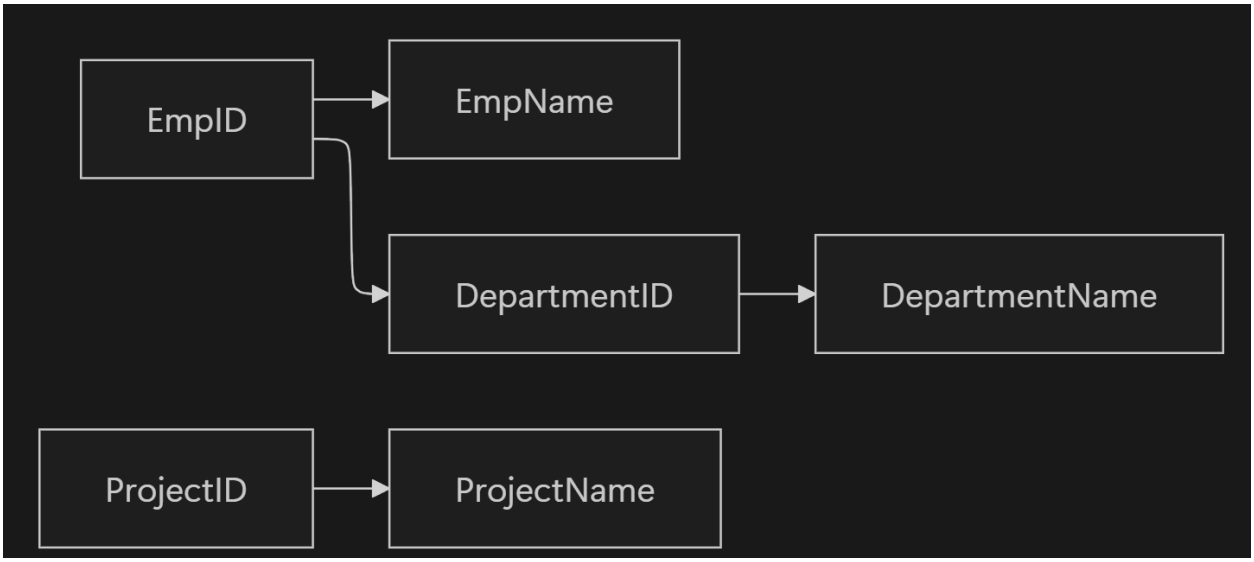
There are transitive dependencies → Not in 3NF → Normalize.

3NF Decomposition:

```
R1(A, B, C)
R2(C, D, E)
R3(E, F)
```

15. EMP_PROJECT(EmpID, EmpName, ProjectID, ProjectName, DepartmentID, DepartmentName)

FDs: {EmpID → EmpName, ProjectID → ProjectName, DepartmentID → DepartmentName, EmpID → DepartmentID}



Candidate Key: {EmpID, ProjectID}

Normalization:

1. **1NF:** Atomic.
2. **2NF:** Remove partial dependency on EmpID.

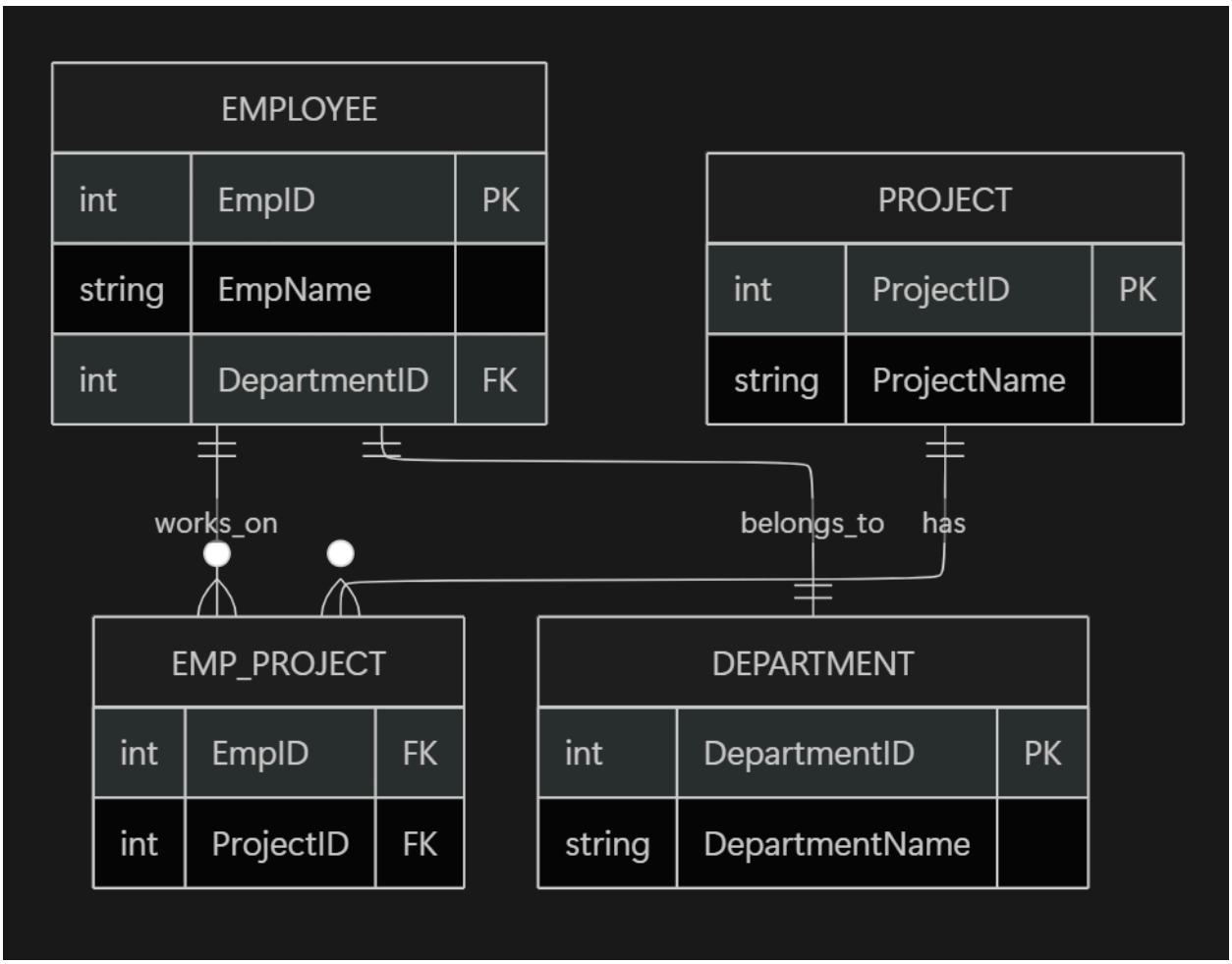
```
R1(EmpID, EmpName, DepartmentID)
R2(ProjectID, ProjectName)
```

3. **3NF:** DepartmentID → DepartmentName ⇒ separate table.

```
R3(DepartmentID, DepartmentName)
R4(EmpID, ProjectID)
```

Final 3NF:

R1(EmpID, EmpName, DepartmentID)
R2(ProjectID, ProjectName)
R3(DepartmentID, DepartmentName)
R4(EmpID, ProjectID)



Normalization eliminates redundant department and project data.