Assignment-4: DBMS

1. Basic Operations of Relational Algebra

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

Basic Operations

Relational Algebra is a procedural query language that takes instances of relations as input and yields instances of relations as output. The fundamental operations are:

1. SELECT (σ)

This operation selects tuples (rows) from a relation that satisfy a given predicate (condition).

- **Notation:** σpredicate(R)
- **Example:** Given a relation STUDENT(RollNo, Name, Age). To find all students older than 20:

σAge>20(STUDENT)

2. PROJECT (π)

This operation selects specified attributes (columns) from a relation and discards the other columns. It also removes duplicate rows from the result.

- **Notation:** πA1,A2,...(R)
- **Example:** From the STUDENT relation, to get only the names and ages:

πName, Age(STUDENT)

3. UNION (∪)

This operation returns a relation containing all tuples that appear in either or both of two union-compatible relations. (Union-compatible means they must have the same number of attributes and corresponding attributes must have the same domain).

- **Notation:** R ∪ S
- **Example:** Given LOCAL_STUDENTS(StudentID, Name) and FOREIGN_STUDENTS(StudentID, Name):

LOCAL_STUDENTS ∪ FOREIGN_STUDENTS

4. SET DIFFERENCE (−)

This operation returns a relation containing all tuples that are in the first relation (R) but not in the second relation (S). R and S must be union-compatible.

- **Notation:** R − S

- **Example:** To find local students who are not foreign students:

LOCAL_STUDENTS − FOREIGN_STUDENTS

5. CARTESIAN PRODUCT (×)

This operation combines every tuple from the first relation (R) with every tuple from the second relation (S).

- **Notation:** R × S
- **Example:** Given STUDENT(RollNo, Name) and COURSE(CourseID, Title):

STUDENT × COURSE

6. RENAME (ρ)

This operation is used to rename either a relation, its attributes, or both.

- **Notation:** ρNewName(R) or ρA1/B1,A2/B2,...(R)
- **Example:** To rename the STUDENT relation to PUPIL:

ρPUPIL(STUDENT)

2. Relational Algebra Expressions

Relations:

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

(a) Find names of students in the 'CSE' department.

1. First, select the 'CSE' department from the DEPARTMENT relation.
2. Then, join this result with the STUDENT relation on their common attribute, DeptID.
3. Finally, project the Name attribute from the joined result.

Expression:

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

(b) List departments having no students.

1. Find all DeptIDs that are present in the STUDENT relation.
2. Find all DeptIDs in the DEPARTMENT relation.
3. Subtract the student DeptIDs (from step 1) from the department DeptIDs (from step 2).
4. Join this result with the DEPARTMENT relation to get the department names.

Expression:

πDeptName((πDeptID(DEPARTMENT) − πDeptID(STUDENT)) ⋈ DEPARTMENT)

3. ER Diagram for Online Shopping System

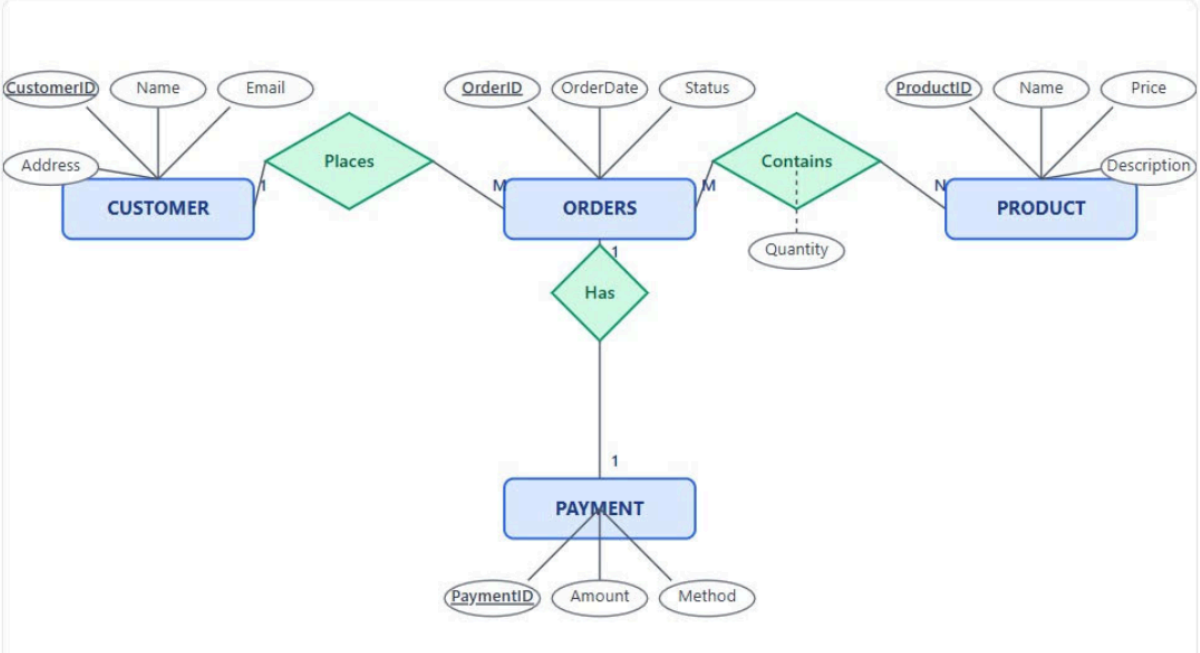
Entities:

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

Relationships:

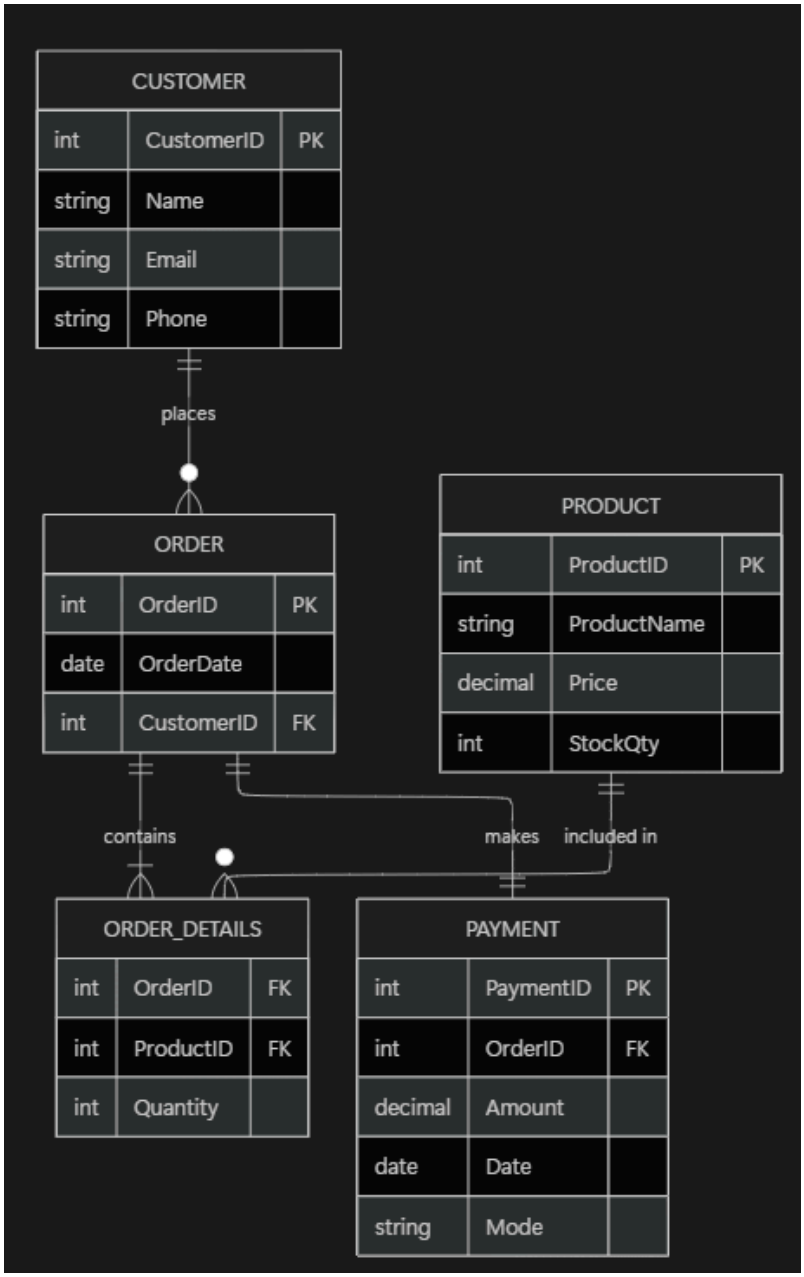
- Customer places Order → *One-to-Many*
- Order contains Product → *Many-to-Many* (Resolved using OrderDetails)
- Order makes Payment → *One-to-One*

ER Model:



Relational Schema:

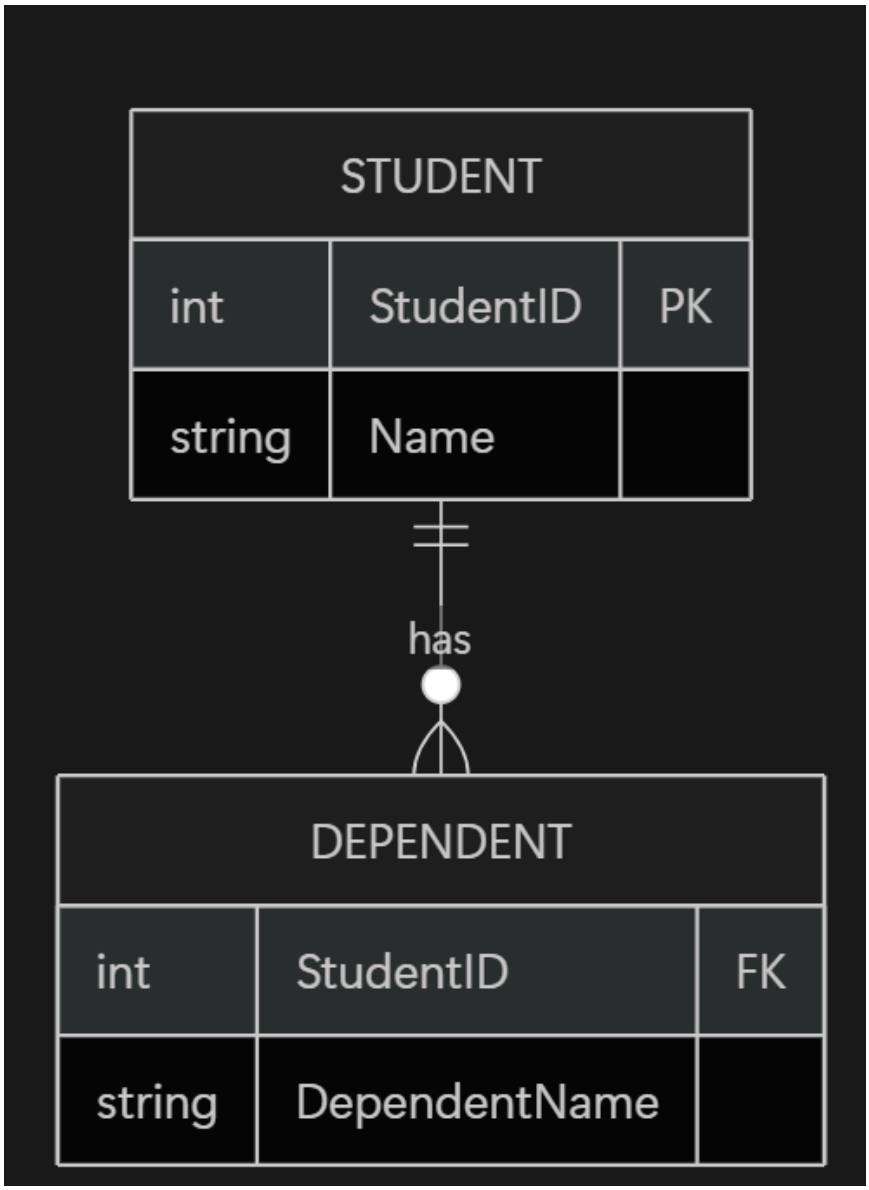
CUSTOMER(CustomerID PK, Name, Email, Phone)
PRODUCT(ProductID PK, ProductName, Price, StockQty)
ORDER(OrderID PK, OrderDate, CustomerID FK)
ORDER_DETAILS(OrderID FK, ProductID FK, Quantity)
PAYMENT(PaymentID PK, OrderID FK, Amount, Date, Mode)



4. Strong vs Weak Entity Sets

Feature	Strong Entity Set	Weak Entity Set
Definition	An entity set that has its own primary key.	An entity set that does not have its own primary key and depends on a strong entity for its existence.

Primary Key	Contains a primary key composed of its own attributes.	Its primary key is formed by the primary key of the identifying (strong) entity plus its own partial key (discriminator).
Existence	Can exist independently.	Its existence is dependent on the strong entity. If the "owner" entity is deleted, the weak entity is typically deleted.
ERD Notation	Represented by a single-line rectangle.	Represented by a double-line rectangle.
Relationship	The relationship to other entities is shown by a single-line diamond.	The identifying relationship with its strong entity is shown by a double-line diamond.
Example	EMPLOYEE(EmpID, Name) - EmpID is the primary key. The EMPLOYEE entity can exist on its own.	DEPENDENT(DepName, Age) - This entity is weak and depends on EMPLOYEE. DepName is only a partial key.
Example PK	PK for EMPLOYEE is (EmpID).	The full PK for DEPENDENT would be (EmpID, DepName), where EmpID is the foreign key from EMPLOYEE.



5. Functional Dependency (FD)

A **Functional Dependency** (FD) is a constraint between two sets of attributes in a relation. For a relation R, a functional dependency X → Y (read as "X functionally determines Y") holds if, for any two tuples t1 and t2 in R that have the same value for X (i.e., t1[X] = t2[X]), they must also have the same value for Y (i.e., t1[Y] = t2[Y]).

In simple terms: The value of X uniquely determines the value of Y.

Identifying Candidate Keys using FDs

A **Candidate Key (CK)** is a minimal set of attributes that functionally determines all other attributes in the relation. "Minimal" means that no proper subset of the CK can determine all attributes.

We use FDs to find CKs by calculating the **attribute closure (X^+)** of a set of attributes X. The closure X^+ is the set of all attributes that are functionally determined by X.

Algorithm:

1. Identify candidate attribute sets (X). A good starting point is to find attributes that do not appear on the right-hand side (RHS) of any FD.
2. For a candidate set X, calculate its closure X^+.
3. If X^+ contains all attributes in the relation, then X is a superkey.
4. If X is a superkey and no proper subset of X is also a superkey, then X is a candidate key.

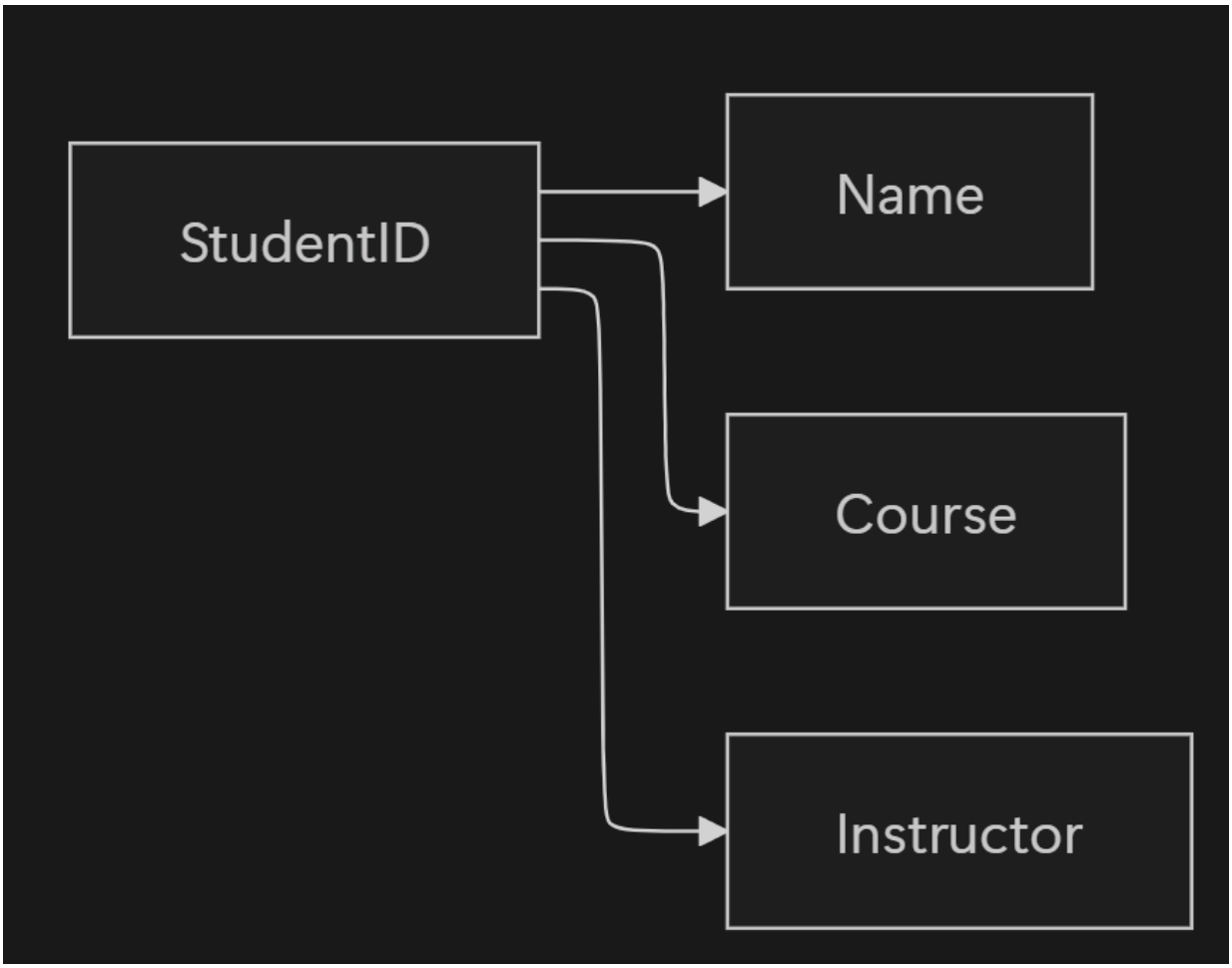
Example:

Given a relation R(A, B, C, D) with FDs:

- A → B
- B → C
- C → D

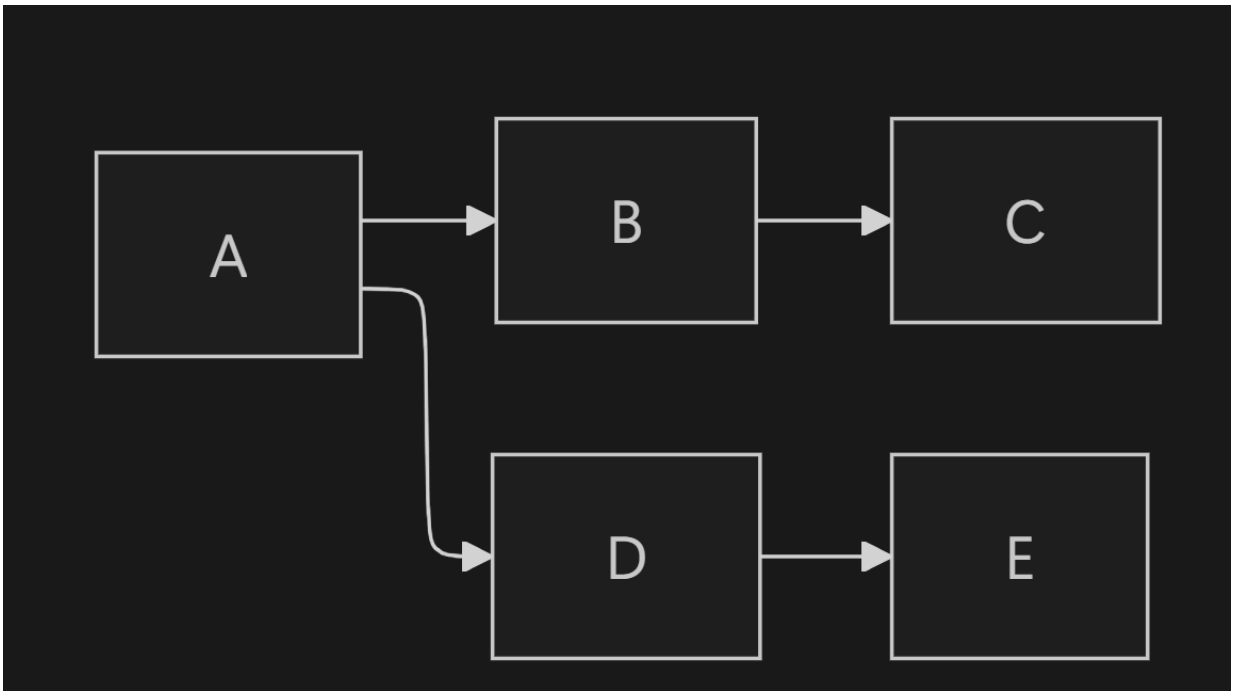
Let's find the candidate key:

1. The attribute A never appears on the RHS of any FD. Therefore, A must be part of any candidate key.
2. Let's find the closure of A, denoted A^+:
    - A^+ = {A} (Start with A)
    - A^+ = {A, B} (Because A → B)
    - A^+ = {A, B, C} (Because B → C)
    - A^+ = {A, B, C, D} (Because C → D)
3. The closure A^+ contains all attributes (A, B, C, D) in the relation. Therefore, A is a superkey.
4. Since A is a single attribute, no proper subset of it can exist. Therefore, A is minimal.
5. **Conclusion:** {A} is the candidate key for the relation R.



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

FDs: {A → B, B → C, A → D, D → E}



1. Find Candidate Keys

1. Identify attributes that are not on the RHS of any FD: A.
2. The attribute A must be part of any candidate key. Let's find the closure of A.
3. $A^+ = \{A\}$
   - $A^+ = \{A, B, D\}$ (using A → B and A → D)
   - $A^+ = \{A, B, C, D\}$ (using B → C)
   - $A^+ = \{A, B, C, D, E\}$ (using D → E)
4. The closure A^+ contains all attributes of R.
5. Since A is a superkey and is minimal (it's a single attribute), {A} is the only candidate key.

2. Normalize to 3NF

- **Relation:** R(A, B, C, D, E)
- **Candidate Key:** {A}
- **Prime Attribute:** {A}
- **Non-Prime Attributes:** {B, C, D, E}
- **FDs:**
  1. A → B
  2. B → C

3. A → D
4. D → E

Check 1NF:

The relation is assumed to be in 1NF (atomic attributes).

Check 2NF (No Partial Dependencies):

A partial dependency occurs when a non-prime attribute depends on part of a candidate key. Since our candidate key {A} is a single attribute, no partial dependencies can exist. Therefore, the relation is in 2NF.

Check 3NF (No Transitive Dependencies):

A transitive dependency exists if X → Y where X is not a superkey and Y is a non-prime attribute.

- A → B: OK (Determinant A is a superkey).
- A → D: OK (Determinant A is a superkey).
- B → C: This is a transitive dependency.
  - (A → B → C) B is not a superkey and C is a non-prime attribute.
- D → E: This is a transitive dependency.
  - (A → D → E) D is not a superkey and E is a non-prime attribute.

Therefore, the relation is NOT in 3NF.

Decomposition to 3NF:

We decompose the relation based on the FDs that cause problems.

1. Create a relation for B → C:
   - R1(B, C)
2. Create a relation for D → E:
   - R2(D, E)
3. Create a relation for the original candidate key {A} and the attributes it determines:
   - R3(A, B, D)

Final 3NF Relations:

- R1(B, C) (with FD B → C)
- R2(D, E) (with FD D → E)
- R3(A, B, D) (with FDs A → B, A → D)

This decomposition is in 3NF, lossless, and dependency-preserving.

7. 1NF, 2NF, 3NF Differences

1NF (First Normal Form)

Rule: A relation is in 1NF if all its attributes contain only atomic (indivisible) values. It must not contain multi-valued attributes or repeating groups.

Example:

NOT in 1NF:

STUDENT(StudentID, Name, Courses)

StudentID	Name	Courses
S101	Ann	{CS101, MATH202}
S102	Tom	{PHYS101}

In 1NF:

STUDENT_COURSES(StudentID, Name, Course)

StudentID	Name	Course
S101	Ann	CS101
S101	Ann	MATH202
S102	Tom	PHYS101

2NF (Second Normal Form)

Rule: A relation must be in 1NF, AND all its non-prime attributes must be fully functionally dependent on the entire candidate key. It must not have any partial dependencies.

Example:

NOT in 2NF:

ENROLLMENT(StudentID, CourseID, StudentName, Grade)

- Candidate Key: {StudentID, CourseID}
- FD: StudentID → StudentName
- This is a partial dependency because StudentName (a non-prime attribute) depends on StudentID (only part of the candidate key).

StudentID	CourseID	StudentName	Grade
S101	CS101	Ann	A
S101	MATH202	Ann	B
S102	CS101	Tom	B

Decomposition to 2NF:

- STUDENT(StudentID, StudentName)
- ENROLLMENT_INFO(StudentID, CourseID, Grade)

Relation 1: STUDENT

StudentID	StudentName
S101	Ann
S102	Tom

Relation 2: ENROLLMENT_INFO

StudentID	CourseID	Grade
S101	CS101	A
S101	MATH202	B
S102	CS101	B

3NF (Third Normal Form)

Rule: A relation must be in 2NF, AND it must not have any transitive dependencies. A transitive dependency occurs when a non-prime attribute depends on another non-prime attribute, rather than directly on the key.

Example:

NOT in 3NF:

EMPLOYEE(EmpID, Name, DeptID, DeptName)

- Candidate Key: {EmpID}
- FDs: EmpID → DeptID and DeptID → DeptName
- This forms a transitive dependency: EmpID → DeptID → DeptName.

EmpID	Name	DeptID	DeptName
E101	Alice	D01	Sales
E102	Bob	D02	Marketing
E103	Carol	D01	Sales

Decomposition to 3NF:

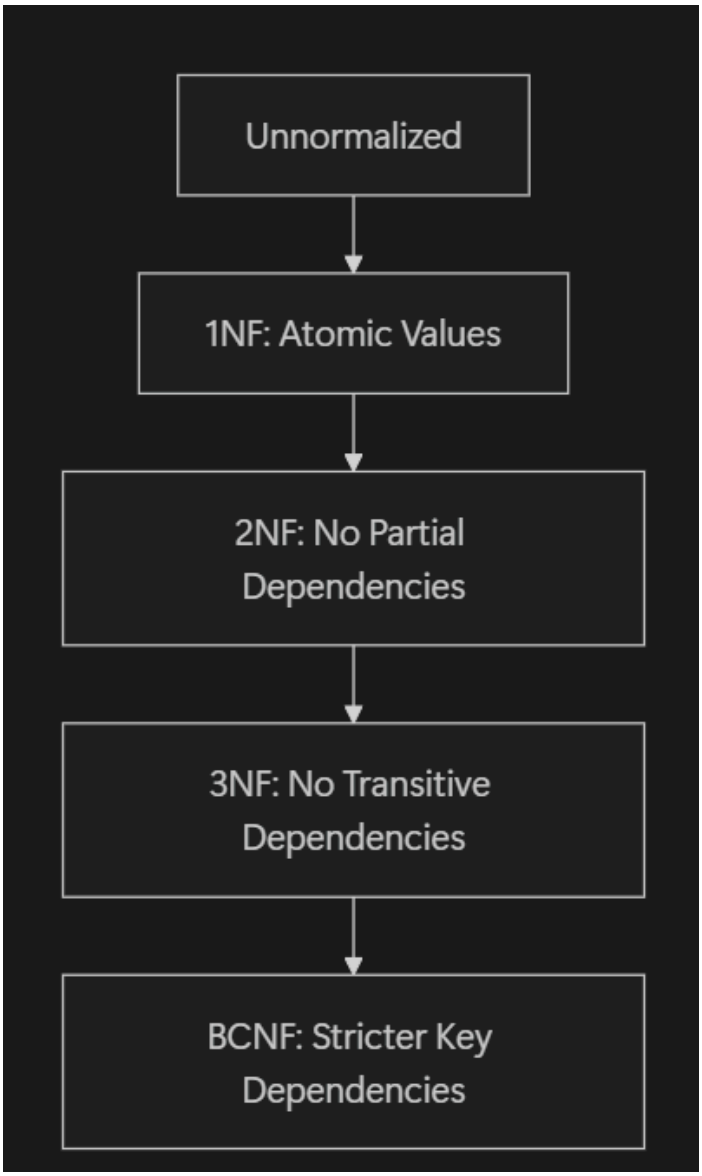
- EMP_DEPT(EmpID, Name, DeptID)
- DEPARTMENT(DeptID, DeptName)

Relation 1: EMP_DEPT

EmpID	Name	DeptID
E101	Alice	D01
E102	Bob	D02
E103	Carol	D01

Relation 2: DEPARTMENT

DeptID	DeptName
D01	Sales
D02	Marketing



8. Relational Algebra

(a) Retrieve employees whose salary is greater than 50,000.

Assuming relation: EMPLOYEE(EmpID, Name, Salary, ...)

σSalary>50000(EMPLOYEE)

(b) Find departments having more than five employees.

Assuming relations: EMPLOYEE(EmpID, DeptID, ...) and DEPARTMENT(DeptID, DeptName)

This requires aggregation, which is an extended relational algebra operation.

Notation:

πDeptName((σNumEmployees>5 (DeptIDGcount(EmpID)→NumEmployees(EMPLOYEE))) ⋈ DEPARTMENT)

9. Library Management System (ER + Schema)

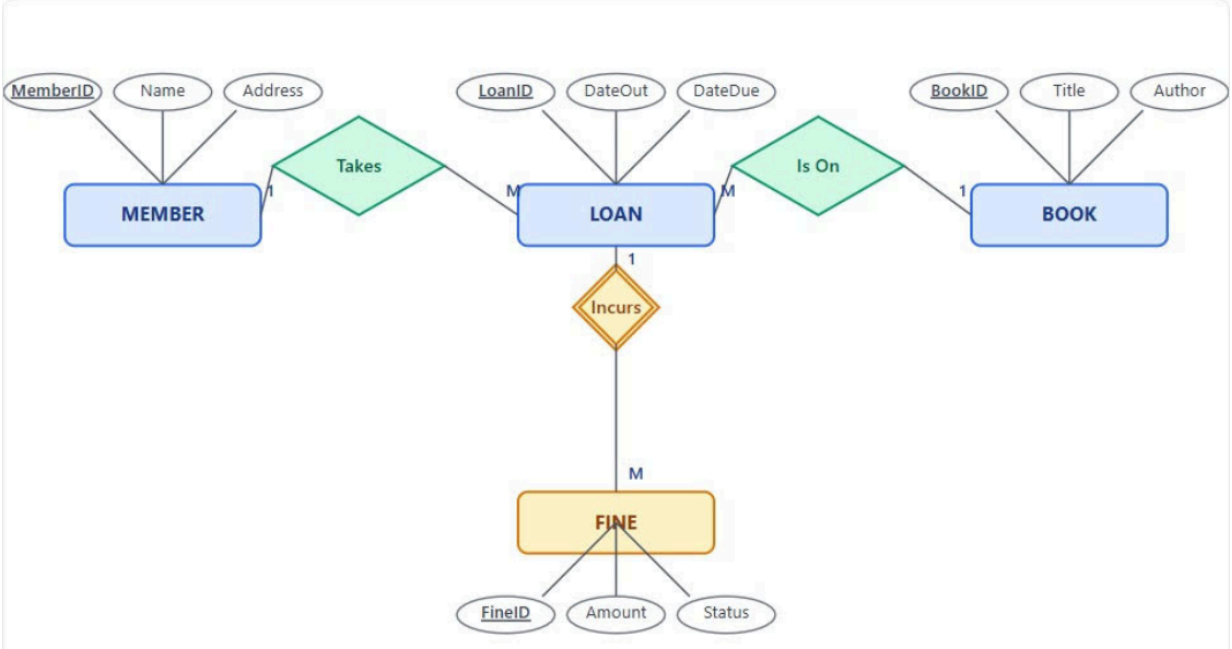
Entities:

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

Relationships:

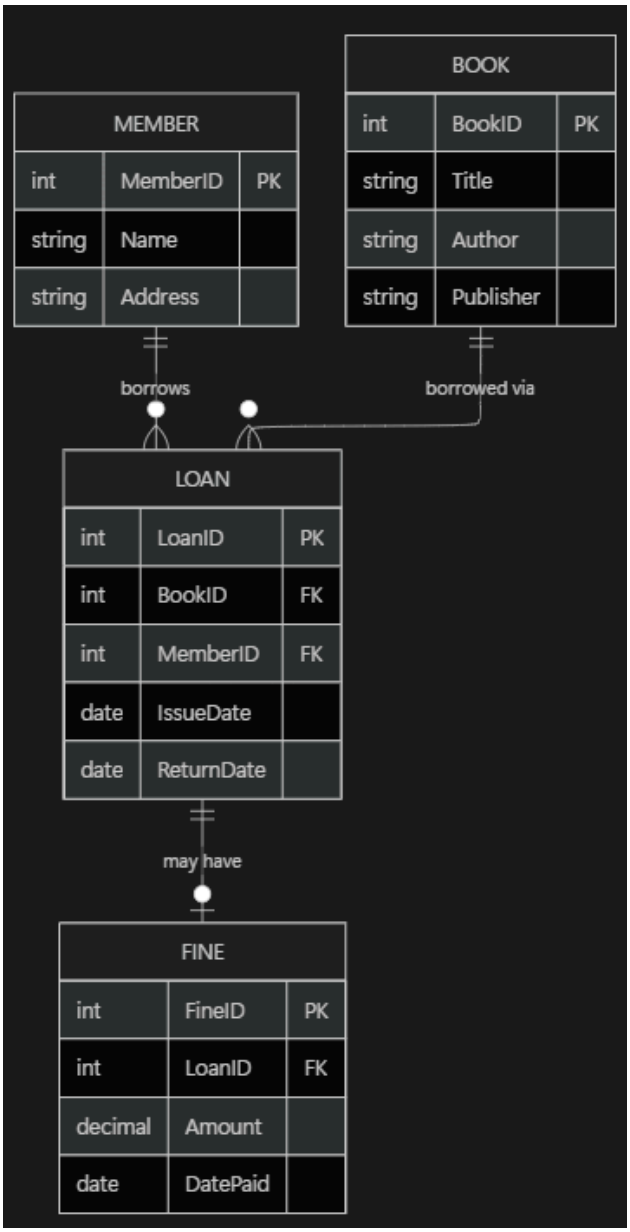
- MEMBER borrows BOOK (via LOAN)
- LOAN may have FINE

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)



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

Example Data in Original Relation R:

StudentID	CourseID	InstructorID	InstructorName	CourseName
S101	CS101	I20	Dr. Codd	Database Systems

S101	MATH202	I30	Dr. Turing	Algorithms
S102	CS101	I20	Dr. Codd	Database Systems
S103	MATH202	I30	Dr. Turing	Algorithms

1. Find Candidate Key

- Attributes not on RHS: StudentID. So, StudentID must be part of the CK.
- Let's check StudentID$^+$: StudentID$^+$ = {StudentID} (Not a key).
- Let's check {StudentID, CourseID}$^+$:
  - {StudentID, CourseID}$^+$ = {StudentID, CourseID}
  - ... = {..., InstructorID} (from FD 1)
  - ... = {..., InstructorName} (from FD 2)
  - ... = {..., CourseName} (from FD 3)
- The closure contains all attributes.
- **Candidate Key: {StudentID, CourseID}**

2. Normalize to 3NF

- **Relation:** R(StudentID, CourseID, InstructorID, InstructorName, CourseName)
- **Prime Attributes:** {StudentID, CourseID}
- **Non-Prime Attributes:** {InstructorID, InstructorName, CourseName}

Step 1: Check 2NF (Remove Partial Dependencies)

- CourseID → CourseName: This is a partial dependency.
  - CourseName (non-prime) depends on CourseID (part of the CK).

Decomposition for 2NF:

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

Step 2: Check 3NF (Remove Transitive Dependencies)

We check the new relations R1 and R2.

- R1(CourseID, CourseName): This relation is in 3NF.
- R2(StudentID, CourseID, InstructorID, InstructorName):
  - CK: {StudentID, CourseID}
  - FDs on R2: {StudentID, CourseID} → InstructorID and InstructorID → InstructorName

- This relation has a transitive dependency: {StudentID, CourseID} → InstructorID → InstructorName

Decomposition for 3NF:

- Split R2 based on the transitive dependency:
- R2a(InstructorID, InstructorName)
- R2b(StudentID, CourseID, InstructorID)

3. Final 3NF Relations and Justification

The final set of 3NF relations is:

1. R1(CourseID, CourseName)
2. R2a(InstructorID, InstructorName)
3. R2b(StudentID, CourseID, InstructorID)

Example Data in 3NF Relations:

Relation 1: R1

CourseID	CourseName
CS101	Database Systems
MATH202	Algorithms

Relation 2: R2a

InstructorID	InstructorName
I20	Dr. Codd
I30	Dr. Turing

Relation 3: R2b

StudentID	CourseID	InstructorID
S101	CS101	I20
S101	MATH202	I30
S102	CS101	I20
S103	MATH202	I30

Justification:

The original relation R was not in 2NF due to a partial dependency. After decomposing to 2NF, the relation R2 was not in 3NF due to a transitive dependency. The final decomposition removes both, resulting in relations that are all in 3NF.

11. Redundancy & Anomalies Reduction

Normalization is the process of structuring a database to reduce data redundancy and improve data integrity. Un-normalized relations suffer from:

- **Redundancy:** The same piece of information is stored multiple times unnecessarily.
- **Anomalies:** This redundancy leads to problems when trying to insert, update, or delete data.
  - **Insertion Anomaly:** Inability to add new data because some other related data is not available.
  - **Update Anomaly:** A change to a single piece of data requires updating multiple rows. Missing an update leads to inconsistent data.
  - **Deletion Anomaly:** Deleting a row of data unintentionally causes other, unrelated information to be lost.

Illustration (Example)

Consider this un-normalized relation (not in 3NF):

EMPLOYEE_DEPT(EmpID, EmpName, DeptID, DeptName)

With FDs: EmpID → EmpName, DeptID and DeptID → DeptName

Data Example:

EmpID	EmpName	DeptID	DeptName
E101	Alice	D01	Sales
E102	Bob	D02	Marketing
E103	Carol	D01	Sales

Problems:

1. **Redundancy:** The fact that 'D01' is 'Sales' is stored twice.
2. **Update Anomaly:** If 'Sales' is renamed to 'Global Sales', we must update rows E101 and E103. If we miss one, the data is inconsistent.
3. **Insertion Anomaly:** We cannot add a new department (e.g., 'D03', 'HR') until at least one employee is assigned to it.
4. **Deletion Anomaly:** If we delete employee 'Bob' (E102), we lose the information that 'D02' is named 'Marketing'.

Solution: Normalization to 3NF

We decompose the relation to remove the transitive dependency (EmpID → DeptID → DeptName).

Relation 1: EMPLOYEE(EmpID, EmpName, DeptID (FK))

EmpID	EmpName	DeptID
E101	Alice	D01
E102	Bob	D02
E103	Carol	D01

Relation 2: DEPARTMENT(DeptID, DeptName)

DeptID	DeptName
D01	Sales
D02	Marketing

How Anomalies are Solved:

1. **Redundancy:** The name 'Sales' is now stored only once.
2. **Update Anomaly:** To rename 'Sales', we change only one row in the DEPARTMENT table.
3. **Insertion Anomaly:** We can easily add a new department ('D03', 'HR') to the DEPARTMENT table.
4. **Deletion Anomaly:** If we delete 'Bob' from EMPLOYEE, the DEPARTMENT table is unaffected.

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

Identify the anomalies present in this relation. (Anomalies are described in the decomposition steps below)

Decompose the relation step-by-step up to 3rd Normal Form (3NF).

Step 0: Original Relation (Not in 2NF)

- **Candidate Key:** {StudentID, CourseID}
- **FDs:** StudentID → StudentName, CourseID → CourseName, InstructorID → InstructorName, CourseID → InstructorID

Example Data:

STUDENT_INFO

StudID	CourseID	StudName	CourseName	InstrID	InstrName
S101	CS101	Ann	Databases	I20	Dr. Codd
S101	MATH202	Ann	Algorithms	I30	Dr. Turing
S102	CS101	Tom	Databases	I20	Dr. Codd
S102	PHYS101	Tom	Physics	I40	Dr. Curie

Anomalies:

- Redundancy (e.g., 'Ann', 'Databases', 'Dr. Codd' all repeated).
- Update anomalies (changing 'Ann's name requires 2 updates).
- Insertion anomalies (cannot add student 'S103' without a course).
- Deletion anomalies (dropping 'PHYS101' loses 'Dr. Curie').

Step 1: 1NF → 2NF (Remove Partial Dependencies)

- **Partial Dependencies:**
  1. StudentID → StudentName
  2. CourseID → CourseName, InstructorID, InstructorName (derived)
- **Decomposition for 2NF (Intermediate Tables):**
  1. STUDENT(StudentID, StudentName)
  2. COURSE_DETAILS(CourseID, CourseName, InstructorID, InstructorName)
  3. ENROLLMENT(StudentID, CourseID)

Data in 2NF Relations:

STUDENT

StudentID	StudentName
S101	Ann
S102	Tom

COURSE_DETAILS

CourseID	CourseName	InstructorID	InstructorName
CS101	Databases	I20	Dr. Codd
MATH202	Algorithms	I30	Dr. Turing
PHYS101	Physics	I40	Dr. Curie

ENROLLMENT

StudentID	CourseID
S101	CS101
S101	MATH202
S102	CS101
S102	PHYS101

Step 2: 2NF → 3NF (Remove Transitive Dependencies)

- STUDENT and ENROLLMENT are in 3NF.
- COURSE_DETAILS is not in 3NF.
  - CK: CourseID
  - FDs: CourseID → CourseName, InstructorID and InstructorID → InstructorName
  - This has a transitive dependency: CourseID → InstructorID → InstructorName.
- **Decomposition for 3NF:**
  - Split COURSE_DETAILS.
  - INSTRUCTOR(InstructorID, InstructorName)
  - COURSE(CourseID, CourseName, InstructorID)

Final 3NF Relations (with data)

1. STUDENT(StudentID, StudentName)

StudentID	StudentName
S101	Ann
S102	Tom

2. ENROLLMENT(StudentID, CourseID)

StudentID	CourseID
S101	CS101
S101	MATH202
S102	CS101

S102	PHYS101

3. INSTRUCTOR(InstructorID, InstructorName)

InstructorID	InstructorName
I20	Dr. Codd
I30	Dr. Turing
I40	Dr. Curie

4. COURSE(CourseID, CourseName, InstructorID (FK))

CourseID	CourseName	InstructorID
CS101	Databases	I20
MATH202	Algorithms	I30
PHYS101	Physics	I40

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

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

Step 1: Determine all candidate keys

1. Attributes never on the RHS: C and D.
2. Any candidate key must contain {C, D}. Let's find its closure.
3. $\{C, D\}^+$ = {C, D, E, F, A, B}. So {C, D} is a candidate key.
4. Let's check other keys that must include D but substitute C:
   - Check $\{B, D\}^+$: {B, D, C, E, F, A}. So {B, D} is a CK.
   - Check $\{A, D\}^+$: {A, D, B, C, E, F}. So {A, D} is a CK.
   - Check $\{F, D\}^+$: {F, D, A, B, C, E}. So {F, D} is a CK.
   - Check $\{E, D\}^+$: {E, D, F, A, B, C}. So {E, D} is a CK.

All Candidate Keys: {A, D}, {B, D}, {C, D}, {E, D}, {F, D}

Step 2: Identify the highest normal form of R

- **Prime Attributes:** {A, B, C, D, E, F} (Union of all CKs).
- **Non-Prime Attributes:** {} (None).
- Since there are no non-prime attributes, the relation cannot have partial or transitive dependencies.
- Therefore, the relation is guaranteed to be in 3NF.
- Check BCNF: For every FD X → Y, X must be a superkey.

- A → B: Is A a superkey? No. Violates BCNF.
- B → C: Is B a superkey? No. Violates BCNF.
- CD → E: Is CD a superkey? Yes, it is a CK. (OK)
- E → F: Is E a superkey? No. Violates BCNF.
- F → A: Is F a superkey? No. Violates BCNF.

The relation is in 3NF but not in BCNF.

Highest Normal Form: 3NF

Step 3: Decompose the relation into 3NF relations

Since the relation is already in 3NF, no decomposition is *required*. However, to create a set of relations using the 3NF Synthesis Algorithm:

1. Find a canonical cover: F = {A → B, B → C, CD → E, E → F, F → A}
2. Create a relation for each FD:
   - R1(A, B)
   - R2(B, C)
   - R3(C, D, E)
   - R4(E, F)
   - R5(F, A)
3. Check if a candidate key is preserved:
   - The relation R3(C, D, E) contains the candidate key {C, D}.

Final 3NF Relations:

```
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}

Find the candidate key for this relation

1. We can infer a transitive dependency: EmpID → DepartmentID → DepartmentName.

2. The attributes EmpID and ProjectID are not on the RHS of any FD. They must be part of the candidate key.
2. Let's find the closure of {EmpID, ProjectID}⁺:
    - {EmpID, ProjectID}⁺ = {EmpID, ProjectID}
    - ... = {..., EmpName, DepartmentID}
    - ... = {..., ProjectName}
    - ... = {..., DepartmentName}
3. The closure contains all attributes.

Candidate Key: {EmpID, ProjectID}

Normalize the relation step-by-step up to 3NF

Step 0: Original Relation (Not in 2NF)

Example Data:

EMP_PROJECT

EmpID	ProjectID	EmpName	ProjectName	DeptID	DeptName
E101	P1	Alice	'Atlas'	D01	Sales
E101	P2	Alice	'Zeus'	D01	Sales
E102	P1	Bob	'Atlas'	D02	Marketing
E103	P2	Carol	'Zeus'	D01	Sales

Step 1: 1NF → 2NF (Remove Partial Dependencies)

- **Partial Dependencies:**
    1. EmpID → EmpName, DepartmentID, DepartmentName (derived)
    2. ProjectID → ProjectName
- **Decomposition for 2NF (Intermediate Tables):**
    1. EMP_DETAILS(EmpID, EmpName, DepartmentID, DepartmentName)
    2. PROJECT(ProjectID, ProjectName)
    3. EMP_PROJECT_LINK(EmpID, ProjectID)

Data in 2NF Relations:

EMP_DETAILS

EmpID	EmpName	DeptID	DeptName

E101	Alice	D01	Sales
E102	Bob	D02	Marketing
E103	Carol	D01	Sales

PROJECT

ProjectID	ProjectName
P1	'Atlas'
P2	'Zeus'

EMP_PROJECT_LINK

EmpID	ProjectID
E101	P1
E101	P2
E102	P1
E103	P2

Step 2: 2NF → 3NF (Remove Transitive Dependencies)

- PROJECT and EMP_PROJECT_LINK are in 3NF.
- EMP_DETAILS is not in 3NF.
  - CK: EmpID
  - FDs: EmpID → EmpName, DepartmentID and DepartmentID → DepartmentName
  - This has a transitive dependency: EmpID → DepartmentID → DepartmentName.
- **Decomposition for 3NF:**
  - Split EMP_DETAILS.
  - DEPARTMENT(DepartmentID, DepartmentName)
  - EMPLOYEE(EmpID, EmpName, DepartmentID)

Final 3NF Relations (with data)

1. PROJECT(ProjectID, ProjectName)

ProjectID	ProjectName
P1	'Atlas'
P2	'Zeus'

2. DEPARTMENT(DepartmentID, DepartmentName)

DepartmentID	DeptName
D01	Sales
D02	Marketing

3. EMPLOYEE(EmpID, EmpName, DepartmentID (FK))

EmpID	EmpName	DeptID
E101	Alice	D01
E102	Bob	D02
E103	Carol	D01

4. EMP_PROJECT_LINK(EmpID, ProjectID)

EmpID	ProjectID
E101	P1
E101	P2
E102	P1
E103	P2

Explain how redundancy is reduced through normalization

In the original EMP_PROJECT relation, EmpName and DepartmentName were repeated for every project an employee worked on. ProjectName was repeated for every employee on a project.

After normalizing to 3NF, this redundancy is eliminated. For example, ProjectName 'Atlas' is stored only once. EmpName 'Alice' is stored only once. DeptName 'Sales' is stored only once.

This solves all update, insertion, and deletion anomalies.