Unit-4

Schema Refinement & Normalization

Schema Refinement & Normalization

Functional Dependencies · Decomposition · Normal Forms

Course: Database Management Systems

Date: 2026

Contents Outline

1 **Introduction**
Schema Refinement Basics

2 **FDs**
Functional Dependencies

3 **Decomposition**
Properties of Decomposition

4 **Normal Forms**
1NF through 5NF

This unit covers schema refinement techniques in database design. We explore functional dependencies as the central tool for guiding decomposition, examine the properties that ensure decomposition quality, and study the normal forms from 1NF to 5NF that progressively eliminate data anomalies.

PART 01

Understanding the Need for Schema Refinement

Introduction to Schema Refinement

Why we refine schemas, the problems redundancy causes, and the goals of good database design

** This chapter introduces the motivation and core concepts behind schema refinement*

What is Schema Refinement?

- **Definition**: Refining a database schema to improve quality
- **Goal 1**: Eliminate data redundancy across relations
- **Goal 2**: Avoid update, insert, and delete anomalies
- **Goal 3**: Ensure data integrity and consistency
- **Approach**: Decompose relations into smaller, well-structured relations
- **Central Tool**: Functional Dependencies (FDs)

Key Idea: Schema refinement uses FDs to guide the decomposition of relations into well-structured normal forms

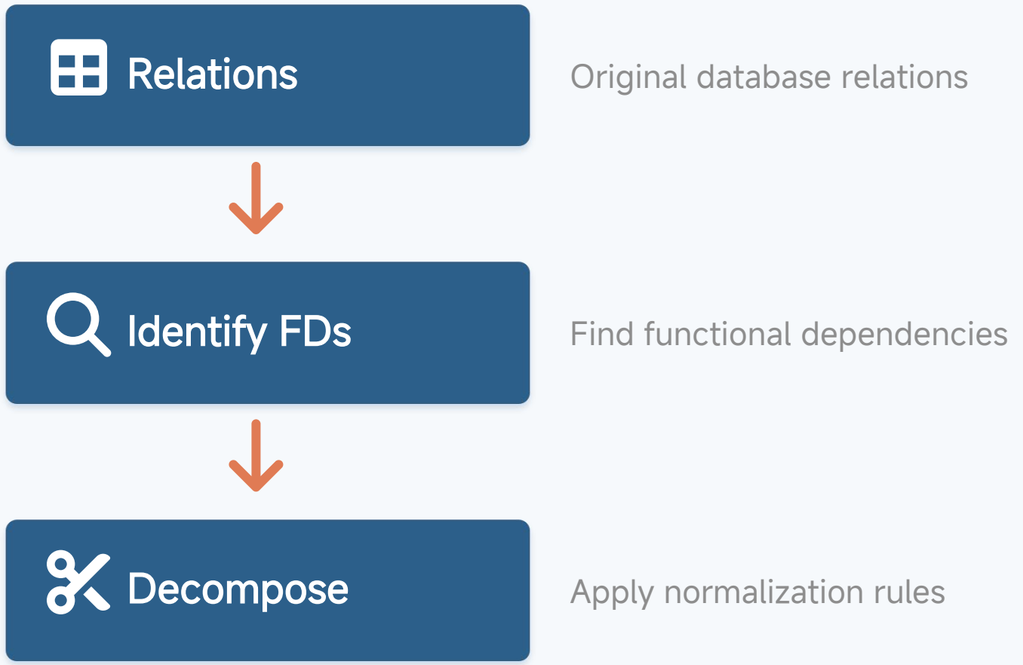

Fig 1: The Schema Refinement Process

Refinement Goals

Schema refinement aims to produce a database design that minimizes redundancy, prevents anomalies, and preserves the original relation's information and dependencies.

Problems Caused by Redundancy

Update Anomaly

Changing one copy of redundant data requires changing all copies. If missed, the database becomes inconsistent. Same information exists in multiple tuples.

Insertion Anomaly

Cannot insert certain data without including other unrelated data. Forces storing null values or artificial combinations just to add valid information.

Deletion Anomaly

Deleting a tuple may unintentionally lose other information stored in that tuple. Facts that should be independent become tied together.

Example

In **(EmpID, EmpName, DeptID, DeptName)**, employee info repeats for each department reference. Updating an employee's name requires changes in multiple tuples.

PART 02

The Central Tool for Guiding Schema Design

Functional Dependencies

Defining constraints, types of FDs, Armstrong's axioms, and attribute closure

** This chapter covers the theoretical foundation for schema refinement and normalization*

Functional Dependencies — Definition

A Functional Dependency $X \rightarrow Y$ holds on relation R if for any two tuples t1, t2 in R: whenever **t1[X] = t2[X]**, then **t1[Y] = t2[Y]**. X is the **determinant**; Y is the **dependent**.

- **Trivial FD**: $Y \subseteq X$ (e.g., $AB \rightarrow A$)
- **Non-trivial FD**: $Y \not\subseteq X$ (e.g., $A \rightarrow B$)
- **Completely non-trivial**: $X \cap Y = \varnothing$

Example

In **Employee(EmpID, Name, DeptID)**:

EmpID → Name, EmpID → DeptID

EmpID determines all other attributes

EmpID	Name	DeptID
E101	Alice	D01
E102	Bob	D02
E103	Carol	D01
E104	David	D03

Fig 2: Employee relation — EmpID → {Name, DeptID}

Key Insight: FDs capture semantic constraints about the real world. They tell us which attributes determine others, guiding how we should decompose relations.

Types of Functional Dependencies

Dependency Types

Classifying Functional Dependencies

Full FD: $X \rightarrow Y$ where no proper subset of X determines Y

Partial FD: $X \rightarrow Y$ where some proper subset of X determines Y

Transitive FD: $X \rightarrow Y$ and $Y \rightarrow Z$ implies $X \rightarrow Z$

Trivial FD: $Y \subseteq X$, always holds

Non-trivial FD: $Y \not\subseteq X$, carries information

Keys & Attributes

Key Concepts for Normalization

Superkey: A set of attributes that determines all attributes

Candidate Key: A minimal superkey (no subset is a superkey)

Primary Key: The chosen candidate key for the relation

Prime Attribute: Part of any candidate key

Non-prime Attribute: Not part of any candidate key

Reasoning About FDs — Armstrong's Axioms

Reflexivity

If $Y \subseteq X$, then $X \rightarrow Y$

Trivial FDs always hold

Augmentation

If $X \rightarrow Y$, then $XZ \rightarrow YZ$

Adding attributes preserves the FD

Transitivity

If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

FDs can be chained together

Derived Rules

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Pseudo-transitivity**: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Soundness & Completeness

Armstrong's axioms are **sound** (only derive true FDs) and **complete** (can derive all true FDs). Together, they form a complete inference system for functional dependencies.

How to use: Start with a given set of FDs F. Apply reflexivity, augmentation, and transitivity (and derived rules) to compute F^+ — the closure of F containing all FDs logically implied by F.

Closure of Attributes & FDs

Attribute Closure X^+

The set of all attributes functionally determined by X, given a set of FDs F.

FD Closure F^+

The set of all FDs logically implied by F, computed using Armstrong's axioms.

Computing Attribute Closure

1. Initialize: $X^+ = X$
2. For each FD $U \rightarrow V$ in F: if $U \subseteq X^+$, then $X^+ = X^+ \cup V$
3. Repeat step 2 until X^+ no longer changes
4. Return X^+ as the final closure

Uses of Attribute Closure

- Test if X is a superkey (X^+ contains all attrs)
- Find all candidate keys of a relation
- Check if an FD $X \rightarrow Y$ is implied by F (is $Y \subseteq X^+$?)

Canonical Cover (F_c)

A **minimal set** of FDs equivalent to F, with no redundant FDs or extraneous attributes.

Requirements:

1. F_c logically implies all FDs in F
2. F logically implies all FDs in F_c
3. No FD in F_c contains extraneous attributes
4. Each left side is unique in F_c

PART 03

Ensuring Decomposition Preserves Information and Dependencies

Properties of Decomposition

Lossless join decomposition and dependency preserving decomposition

** This chapter covers the two critical properties every decomposition must satisfy*

Lossless Join Decomposition

A decomposition of R into R1 and R2 is **lossless** if $R = R1 \bowtie R2$ (natural join recovers the original relation exactly).

Lossless Join Test

Decomposition is lossless if:

$R1 \cap R2 \rightarrow R1$

or equivalently:

$R1 \cap R2 \rightarrow R2$

Key: The common attributes must form a superkey of at least one decomposed relation.

Example

R(A,B,C) with FD $A \rightarrow B$. Decompose into R1(A,B) and R2(A,C). Lossless because **A** is key for R1.

Why Lossless Join Matters

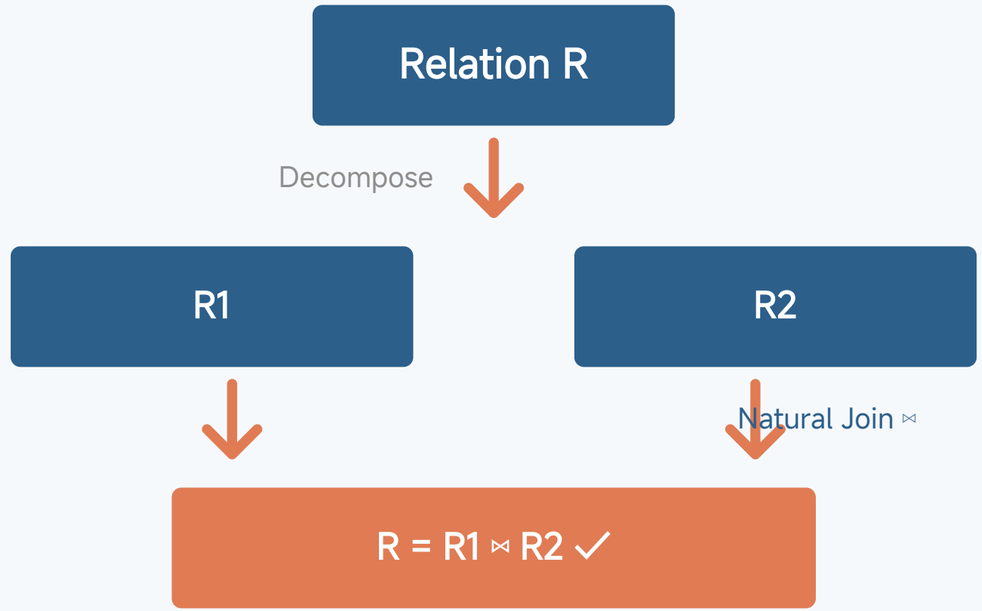

Fig 3: Lossless join guarantees the original relation is exactly recovered by joining the decomposed relations.

Dependency Preserving Decomposition

A decomposition **preserves dependencies** if all FDs in the original relation can be enforced by checking only the decomposed relations.

Testing Preservation

- Compute projection of F onto each Ri
- $F_i = \{ X \rightarrow Y \in F^+ \mid X, Y \subseteq Ri \}$
- Check if union of all F_i implies F

Important Trade-off

Not all lossless decompositions preserve dependencies.

Sometimes we sacrifice dependency preservation to achieve **BCNF**.

Comparison

Property	Required?	Always Possible?
Lossless Join	Yes	Yes
Dependency Preserving	Desirable	No (with BCNF)
3NF	Minimum goal	Yes, both

Fig 4: Decomposition properties at a glance

3NF Guarantee: Any relation can be decomposed into 3NF with both lossless join and dependency preservation.

PART 04

Progressive Elimination of Anomalies

Normalization & Normal Forms

From 1NF to 5NF — understanding each normal form and its requirements

** This chapter covers 1NF, 2NF, 3NF, BCNF, 4NF, and 5NF with definitions and examples*

First Normal Form (1NF)

A relation is in **1NF** if all attribute values are **atomic** (indivisible). No repeating groups, no multi-valued attributes.

Requirements

- All values are atomic (single-valued)
- No repeating groups or arrays
- No composite attributes
- Foundation for all higher normal forms

Violation Example

StudentID	Name	Phone (Multi-valued!)
S101	Alice	555-0101, 555-0102
S102	Bob	555-0201, 555-0202, 555-0203

Solution

StudentID	Name	PhoneID	Phone
S101	Alice	P1	555-0101
S101	Alice	P2	555-0102
S102	Bob	P3	555-0201

Fig 5: Convert multi-valued attribute to separate tuples

1NF is the foundation. Every relation in a relational database must at least be in 1NF. All higher normal forms (2NF, 3NF, BCNF, etc.) require 1NF as a prerequisite.

Second Normal Form (2NF)

A relation is in **2NF** if it is in 1NF and every non-prime attribute is **fully functionally dependent** on every candidate key.

Eliminates Partial Dependencies

A **partial dependency** exists when a non-key attribute depends on only *part* of a composite key, not the entire key.

Note: If all candidate keys are single attributes, 1NF automatically implies 2NF.

Violation Example

EmpID	ProjID	EmpName	Hours
E101	P01	Alice	40
E101	P02	Alice	35

Fix: Decompose

R1 (EmpID, EmpName)

EmpID	EmpName
E101	Alice
E102	Bob

R2 (EmpID, ProjID, Hours)

EmpID	ProjID	Hours
E101	P01	40
E101	P02	35

Why: EmpName depends only on EmpID, not the full key (EmpID, ProjID). Splitting removes the partial dependency.

Third Normal Form (3NF)

A relation is in **3NF** if for every non-trivial FD X → A, either X is a superkey or A is a **prime attribute**.

Eliminates Transitive Dependencies

A **transitive dependency** exists when a non-key attribute depends on another non-key attribute, creating a chain: Key → Non-key → Non-key.

Key → Non-key A → Non-key B ✗

3NF Guarantee: Any relation can be decomposed into 3NF that is both lossless and dependency-preserving.

Violation Example

In **(EmpID, DeptID, DeptName)**: EmpID → DeptID → DeptName. DeptName is transitively dependent on EmpID.

Fix: Decompose

R1 (EmpID, DeptID)

EmpID	DeptID
E101	D01
E102	D02

R2 (DeptID, DeptName)

DeptID	DeptName
D01	Sales
D02	Engineering

Why: DeptName depends on DeptID, not directly on EmpID. Separating removes the transitive dependency.

Boyce-Codd Normal Form (BCNF)

A relation is in **BCNF** if for every non-trivial FD X → Y, **X is a superkey**. Every determinant must be a superkey.

BCNF vs 3NF

BCNF is stricter than 3NF

BCNF: every determinant is a superkey

3NF: allows prime attributes as dependents

BCNF ⊆ 3NF ⊆ 2NF ⊆ 1NF

Trade-off

BCNF decomposition is **always lossless**

BCNF may **not preserve** all dependencies

3NF always preserves dependencies

Choose based on requirements

Classic BCNF Violation Example

Relation: **(Student, Course, Instructor)** with FDs: **(Student, Course) → Instructor** and **Instructor → Course**

This is in **3NF** (Course is prime), but **not BCNF** because Instructor is not a superkey. Fix: decompose to **(Student, Instructor)** and **(Instructor, Course)**.

Fourth Normal Form (4NF)

A relation is in **4NF** if it is in BCNF and has no non-trivial **multivalued dependencies (MVDs)**.

Multivalued Dependency (MVD)

X →→ Y means for each X value, the set of Y values is independent of R - X - Y values.

MVD Rules:

1. If X → Y, then X →→ Y (every FD is an MVD)
2. If X →→ Y, then X →→ (R - X - Y)

Purpose: 4NF eliminates redundancy from independent multi-valued facts about the same key.

Violation Example

In **(EmpID, Skill, Language)**, skills and languages are independent multi-valued facts.

Fix: Decompose

R1 (EmpID, Skill)

EmpID	Skill
E101	Java
E101	Python

R2 (EmpID, Language)

EmpID	Language
E101	English
E101	Spanish

Why: Skills and languages are independent. Separating them removes the MVD.

Fifth Normal Form (5NF)

A relation is in **5NF** (PJ/NF) if it is in 4NF and has no non-trivial **join dependencies** except those implied by superkeys.

Join Dependency

A **join dependency** *{R1, R2, ..., Rn} states that R is the join of its projections R1, R2, ..., Rn.

Key Points:

5NF is the **ultimate normal form**

A 5NF relation cannot be non-trivially decomposed

Rarely needed in practice

Practical Note: 3NF/BCNF are usually sufficient. 5NF is primarily of theoretical interest.

Example Scenario

Relation: **(Agent, Company, Product)**

Constraint: Agent sells Product for Company, but only certain combinations are valid.

Agent	Company	Product
Smith	Ford	Car
Smith	Ford	Truck
Jones	Toyota	Car

Decompose to 5NF:

(Agent, Company), **(Agent, Product)**, **(Company, Product)** — only valid combinations can be joined back.

Normal Forms — Summary & Comparison

NF	Definition	Eliminates	Requirement
1NF	All values are atomic	Repeating groups	Atomic values only
2NF	1NF + no partial dependencies	Partial dependencies	Full FD on candidate key
3NF	2NF + no transitive dependencies	Transitive dependencies	Non-key → non-key FDs
BCNF	Every determinant is a superkey	All FD anomalies	Determinant = superkey
4NF	BCNF + no non-trivial MVDs	Multi-valued dependencies	No independent MVDs
5NF	4NF + no join dependencies	Join dependencies	Only superkey JDs allowed

Hierarchy

Practical Guideline: Aim for **3NF or BCNF** in most cases. Go to 4NF/5NF only when specific multi-valued or join anomalies persist.

Thank You

Schema Refinement & Normalization

Database Management Systems · UNIT IV

FDs · Decomposition · Normal Forms

Questions? Let's discuss schema refinement and normalization!