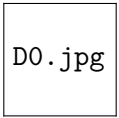
Lecture 33: 4NF, 5NF and Lossless-Join Decomposition

Z2004: Database Management Systems

Dr. Innocent Nyalala



IIT Madras Zanzibar

25 May 2026

Outline

- Beyond BCNF: redundancy survives!
- Multivalued dependencies (MVDs)
- 4NF: eliminate non-trivial MVDs
- 5NF (Project-Join Normal Form)
- Lossless-join decomposition
- Testing for lossless join
- Dependency preservation
- Summary of all normal forms
- Think-Pair-Share

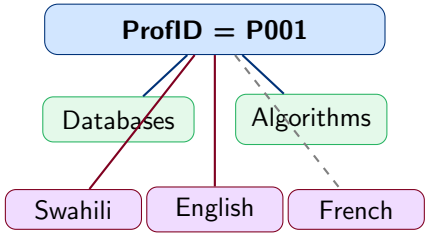
Recall from L32

BCNF eliminates all FD-based redundancy. But what if a BCNF table **still has redundancy**? That is what we address today.

Beyond BCNF: Redundancy Survives!

Professor table — every FD respected, yet...

ProfID	Course	Language
P001	Databases	Swahili
P001	Databases	English
P001	Algorithms	**Swahili**
P001	Algorithms	**English**
P002	OS	French
P002	OS	English



Courses & Languages are independent!

The problem

P001 speaks {Swahili, English}. That appears **once per course**. Add a new course ⇒ must insert 2 new

Multivalued Dependencies (MVDs)

Definition: $X \twoheadrightarrow Y$

X **multi-determines** Y if: for each X-value, the set of Y-values is **independent** of the other attributes (Z).

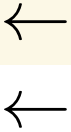
Formal: if tuples t_1, t_2 agree on X, then swapping their Y-values produces two more valid tuples.

In our Professor table:

- ProfID $\twoheadrightarrow$ Course
- ProfID $\twoheadrightarrow$ Language
- The two sets are **independent** of each other

Note

MVD swappi



4NF — Fourth Normal Form

4NF Rule

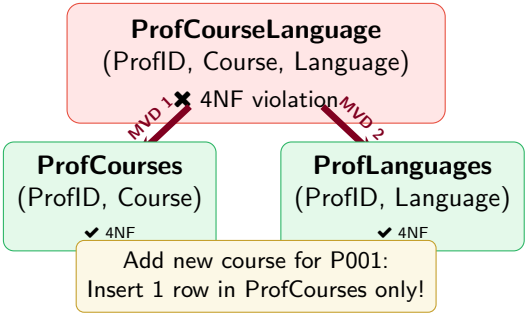
BCNF + no non-trivial MVDs where the left side is not a superkey.

A non-trivial MVD $X \twoheadrightarrow Y$: $Y \not\subseteq X$ and $X \cup Y \neq$ all attrs.

Fix: decompose into two separate tables — one for each independent MVD.

✖ Violates 4NF

`ProfCourseLanguage(ProfID, Course, Language)`
ProfID $\twoheadrightarrow$ Course and ProfID $\twoheadrightarrow$ Language are independent.



5NF — Project-Join Normal Form (Brief)

5NF Rule

4NF + no join dependencies that are not implied by the candidate keys.

A join dependency $\bowtie \{R_1, R_2, \ldots, R_k\}$ holds if $R = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k$.

5NF violations arise from **three-way** (or more) relationships where all three projections are needed.

In Practice

5NF violations are **very rare**. Most real-world schemas stop at BCNF or 4NF. Know the concept; recognise it if you see it.

Classic 5NF example:

`SPJ(Supplier, Part, Project)`

Supplier	Part	Project
S1	P1	J1
S1	P2	J1
S2	P1	J1

Constraint: If supplier s supplies part p, supplier s supplies to project j, and project j uses part p, **then** (s, p, j) must be in the table.

$\Rightarrow$ Join dependency not from any key.
$\Rightarrow$ Decompose into SP, SJ, PJ.

Lossless-Join Decomposition — The Critical Property

Lossless-Join

When we split R into R_1 and R_2, we must be able to **reconstruct the original** by joining:
$R = R_1 \bowtie R_2$

Lossless: join gives back exactly R
Lossy: join gives extra (spurious) rows

✖ LOSSY Example

$R(\underline{A}, B, C)$. Decompose into $R_1(A, B)$ and $R_2(B, C)$.
Join on B may produce **spurious rows**!
Rows that never existed in the original.

Origi

π_{AB}
$R_1(A, B)$

Testing for Lossless Join

Binary Decomposition Test

For R decomposed into R_1 and R_2, the join is **lossless** iff:

$(R_1 \cap R_2) \rightarrow R_1$

or

$(R_1 \cap R_2) \rightarrow R_2$

i.e. the shared attributes functionally determine one of the two schemas.

The Chase Algorithm

For multi-way decompositions (> 2 parts), use the **chase**: fill a tableau with symbols, apply FDs, check if a row becomes all-subscript-free.

Example: $R(A, B, C)$, FD: $A \rightarrow BC$
Decompose into $R_1(A, B)$ and $R_2(A, C)$

$R_1 \cap R_2 = \{A\}$
Does $\{A\} \rightarrow R_1$? i.e. $A \rightarrow AB$?
$A \rightarrow B$ (from FD) + $A \rightarrow A$ (trivial)
$\Rightarrow$ **Yes!**

✔ **Lossless-join!**

Countercheck: $R_1 \bowtie R_2$ on A gives back exactly R. ✔

Dependency Preservation

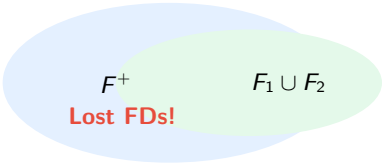
Definition

Decomposition $\{R_1, R_2, \ldots, R_k\}$ is **dependency-preserving** if every FD in F^+ can be checked using only the individual schemas (no join needed).

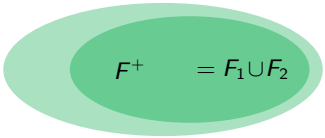
Formally: $(F_1 \cup F_2 \cup \cdots \cup F_k)^+ = F^+$
where F_i = FDs in F^+ restricted to R_i.

Why it matters

If not dependency-preserving, enforcing an FD requires **joining tables** every time a row is inserted or updated. Expensive!



vs. Preserved:



✔ **All FDs preserved**

Summary: All Normal Forms at a Glance

NF	Requirement	Eliminates	FD/MVD concept	Practicality
1NF	Atomic values	Multi-valued cells	—	Always required
2NF	1NF + no partial deps	Partial FD	Part-of-key $\to$ non-key	Common
3NF	2NF + no transitive deps	Transitive FD	Non-key $\to$ non-key	Very common
BCNF	Every determinant is a key	All FD anomalies	$X \to Y$, X not a key	Most systems
4NF	BCNF + no non-trivial MVDs	MVD redundancy	$X \twoheadrightarrow Y$, X not a key	Occasionally needed
5NF	4NF + no non-key join deps	Join dependencies	$\bowtie \{R_1, \ldots, R_k\}$	Rarely encountered

Design Advice

- Most practical schemas: **3NF or BCNF**
- MVD redundancy seen in fact tables, catalogs
- Always verify **lossless-join**
- Check **dependency preservation**

Two Critical Properties

- **Lossless-join**: can we reconstruct?
- **Dependency preservation**: can we enforce FDs without joins?
- BCNF: lossless always; dep. pres. not always
- 3NF: both always guaranteed

Think-Pair-Share (5 minutes)

Is this decomposition lossless? Is it dependency-preserving?

Given:

$R(A, B, C, D)$ with FDs: $A \to B$, $B \to C$, $C \to D$, $D \to A$

Decomposition 1: $R_1(A, B)$, $R_2(B, C)$, $R_3(C, D)$

Decomposition 2: $R_1(A, B, C)$, $R_2(A, C, D)$

- For each: what are the shared attrs?
- Test lossless join for each pair
- Which FDs are in each R_i?
- Union of all F_i equals F^+?

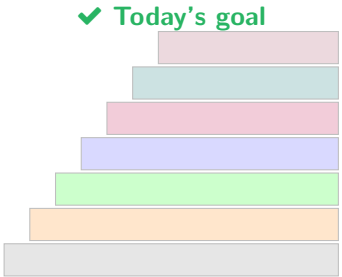
Note

Here all 4 candidate keys exist: {A}, {B}, {C}, {D}!

Summary

Key Concepts Today

- **MVD** $X \twoheadrightarrow Y$: independent multi-sets
- **4NF**: remove non-trivial MVDs
- **5NF**: remove join dependencies
- **Lossless join**: no spurious rows after join
- **Dep. preservation**: enforce FDs locally
- BCNF: lossless always, dep. pres. sometimes
- 3NF: both always guaranteed



Next Lecture — L34: Storage, Disk, Files, Buffer Pool

We leave normalisation and enter **Module 8: Storage and Indexing**. How does a DBMS physically store data on disk?