3NF Synthesis

Table of contents

The algorithm 1
The key relation step, precisely 2
Worked example — the straightforward shape 3
Worked example — same-LHS combination and subset elimination 3
Worked example — when the key relation swallows everything 4
Synthesis vs decomposition 4
See also 5

3NF synthesis is the bottom-up route from a universal relation to a good schema: instead of repeatedly splitting one big relation apart (see decomposition[1]), it *builds* relations up from a minimal-cover[2] — one relation per dependency. The result is guaranteed lossless-join, **dependency preserving**, and in 3NF[3]. Introduced in 2026-05-04-normalisation-and-relational-database-schema-design[4].

Dependency preservation is the reason to choose synthesis over BCNF decomposition: because every FD in the cover gets its own relation, every FD ends up checkable against a single relation. The price is that 3NF tolerates the residual anomalies BCNF would have removed.

The algorithm

```
S := ;
Compute a minimal cover G of F;
Combine all FDs in G with the same LHS into one;
For each X → Y in G:
    if no relation in S contains X  Y:
        add a relation with schema X  Y to S;
```

[1] courses/infs1200/decomposition.pdf
[2] courses/infs1200/minimal-cover.pdf
[3] courses/infs1200/normal-forms.pdf
[4] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf

```
if any candidate key is missing from the relations:
    add a relation with all prime attributes (i.e. all candidate keys);
Eliminate redundant relations from the resulting set of relations;
```

A relation is **redundant** if it is a *projection of* another relation already in the schema — that is, if its attributes are a subset of another's.

Reading it step by step:

Step	What it does	Why
Minimal cover	Strips redundant FDs and redundant LHS attributes	Redundant FDs would create redundant relations
Combine same LHS	`X → A` and `X → B` become `X → {A, B}`	Otherwise you'd get two relations that both key on `X`
One relation per FD	Schema `X  Y`; `X` is its key	Puts each dependency somewhere it can be enforced — this is what preserves dependencies
Add the key relation	If no relation contains a full candidate key, add one holding all prime attributes	Without it the decomposition need not be lossless — nothing joins the pieces back to the original key
Eliminate redundant relations	Drop any relation that is a *projection of* another — i.e. whose attributes are a subset of another's	A subset relation stores nothing new

The key relation step, precisely

This is the step that trips people up, so it's worth stating exactly.

When it fires. Check whether any relation already in `S` contains a full candidate key of `R` among its attributes. If one does, skip the step entirely. If none does, fire it. Note that the FD-derived relations often *look* key-ish — each has its own LHS as a local key — without any of them containing a candidate key of the original `R`.

What gets added. A relation containing **all the prime attributes** — the union of *every* candidate key of `R`, not one chosen candidate key. When `R` has exactly one candidate key the two readings coincide, which is why the distinction is easy to miss; when it has several, they differ and only the union is correct.

Why. The FD-derived relations are chained together by their shared LHS attributes, but nothing guarantees those chains reach the original key. The prime-attribute relation is what

makes the join reconstruct R — it is a losslessness patch, not an FD-carrying relation, and it usually carries no non-trivial FD at all.

Then re-run elimination. The prime-attribute relation is typically wide, so relations added earlier can become projections of it. Eliminate redundant relations **after** adding it, never before.

Worked example — the straightforward shape

R [A, B, C, D, E] with F = {{A,B} → C, C → D} (already a minimal cover). A, B and E never appear on a RHS, so every key contains them; {A,B,E} = {A,B,C,D,E}, so {A, B, E} is the only candidate key.

1. One relation per FD: R1 [A, B, C] (from {A,B} → C) and R2 [C, D] (from C → D).
2. Neither contains the candidate key {A, B, E} — E appears nowhere — so add R3 [A, B, E].
3. No relation is a subset of another.

Final: R1 [A, B, C], R2 [C, D], R3 [A, B, E].

R3 carries no non-trivial FD; its job is purely to hold the key together so the join is lossless. A relation like that is a normal, and expected, output of synthesis.

Worked example — same-LHS combination and subset elimination

R [A, B, C, D, E] with {A} → {D,E}, {D} → {A}, {B} → {C}, {B,C} → {A,D}, {E,A} → {D} (from week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis[5]).

{B} = {A,B,C,D,E}, and B never appears on a RHS, so {B} is the only candidate key.

Minimal cover: after RHS-splitting, {E,A} → D simplifies to {A} → D (E is redundant — A alone already reaches D) and the resulting duplicate collapses into the existing {A} → {D,E}; {B,C} → A and {B,C} → D both lose their C (redundant, since B → C), becoming {B} → A and {B} → D; and {B} → D is then dropped outright, being derivable via {B} → A then {A} → D. What remains is {A} → {D, E}, {B} → {A, C}, {D} → {A}.

Synthesis: one relation per FD gives [A, D, E], [A, B, C] and [A, D]. The last is a subset of [A, D, E], so it goes. The candidate key {B} is already inside [A, B, C], so no extra relation is needed.

Final: R1 [A, D, E] (carrying {A} → {D,E} and {D} → {A}), R2 [A, B, C] (carrying {B} → {A,C}). Both are in 3NF.

[5] courses/infs1200/week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis.pdf

Worked example — when the key relation swallows everything

Given the minimal cover `{A,C} → E`, `{B,D} → A`, `A → B`, `E → {C, F}` for `R [A, B, C, D, E, F]`. The candidate keys are `{A,C,D}`, `{A,D,E}`, `{B,C,D}` and `{B,D,E}` — note `F` appears in none of them, so the prime attributes are `{A, B, C, D, E}`.

1. One relation per FD: `[A, C, E]`, `[B, D, A]`, `[A, B]`, `[E, C, F]`.
2. `[A, B]` is a subset of `[B, D, A]` — drop it.
3. No remaining relation contains a *full* candidate key: every key contains `D`, but `[A,C,E]` and `[E,C,F]` lack `D`, and `[A,B,D]` lacks both `C` and `E`. So add `R5` with all prime attributes: `[A, B, C, D, E]`.
4. Eliminate subsets again: `[A, C, E]` and `[A, B, D]` are now both subsets of `R5` — drop both. `[E, C, F]` is not (it has `F`) — keep.

Final: `[E, C, F]` and `R5 [A, B, C, D, E]`.

This is the case that catches people out. Skipping the second subset-elimination pass leaves three redundant relations behind; skipping step 3 leaves a decomposition that isn't lossless.

Synthesis vs decomposition

	BCNF decomposition (top-down)	3NF synthesis (bottom-up)
Input	`R` + `F`	A **minimal cover** of `F`
Method	Split on each violating FD	One relation per FD, plus a key relation
Normal form reached	BCNF	3NF
Lossless join	✓	✓
Dependency preserving	✗ (not guaranteed)	✓
Anomaly-free	✓	Mostly — 3NF's prime-attribute clause can let some through

Neither is deterministic — synthesis depends on which minimal cover you land on, decomposition on the order violating FDs are processed — and both inherit any error in the FD set: a missing FD produces a sub-optimal or incorrect design either way.

The two often converge. In week11-tutorial-case-study-9-payroll-system[6], `TripExpenseAllocations` decomposes to the same three relations — `Trip`, `Expense`, `TripExpenseAllocations` — whether you take the BCNF or the synthesis route. When they differ, the difference is exactly the FDs BCNF had to sacrifice.

[6] courses/infs1200/week11-tutorial-case-study-9-payroll-system.pdf

See also

- minimal-cover[7] — compute this first; synthesis is meaningless without it
- decomposition[8] · normal-forms[9] · attribute-closure[10] · functional-dependencies[11]
- 2026-05-04-normalisation-and-relational-database-schema-design[12] — Module 4, part 2
- week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis[13] — Section B is pure synthesis drill

[7] courses/infs1200/minimal-cover.pdf
[8] courses/infs1200/decomposition.pdf
[9] courses/infs1200/normal-forms.pdf
[10] courses/infs1200/attribute-closure.pdf
[11] courses/infs1200/functional-dependencies.pdf
[12] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf
[13] courses/infs1200/week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis.pdf