Minimal Cover

Table of contents

Equivalence of FD sets 1
Definition 2
Finding a minimal cover — three steps 2
Worked example — the mechanical case 3
Worked example — a wide right-hand side 3
Worked example — spotting a false answer 4
Worked example — on a real schema 6
Redundancy can hide behind a chain 6
See also 7

A **minimal cover** (or canonical cover) `G` of an FD set `F` is an equivalent set of FDs with all the slack squeezed out — nothing can be deleted or shortened without changing what the set implies. It's the required input to 3nf-synthesis[1], and it makes reasoning about BCNF[2] easier by hand too. Introduced in 2026-05-04-normalisation-and-relational-database-schema-design[3].

Equivalence of FD sets

Two FD sets are **equivalent** when they imply exactly the same FDs: `F = G`. Equivalence is what makes the reduction safe — a minimal cover constrains the data identically to the original set, it just says so with fewer, smaller rules.

[1] courses/infs1200/3nf-synthesis.pdf
[2] courses/infs1200/normal-forms.pdf
[3] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf

Definition

The goal is to **transform the FDs to be as small as possible** while still implying exactly the same things.

`G` is a minimal cover for `F` iff:

1. **Equivalent** — `F = G`.
2. **Nothing redundant** — deleting any FD from `G`, or any attribute from any FD's LHS, gives some `G'` with `G' ≠ G`. Every FD and every LHS attribute is *necessary*.

Informally: every FD in `G` is needed, and is as small as possible, while still closing to the same set as `F`.

The smallest illustration: `F = {A → B, B → C, A → {B, C}}` reduces to `G = {A → B, B → C}` — the third FD asserts nothing the first two don't already imply.

The textbook definition adds a third condition — **single-attribute RHS**, every FD in `G` having exactly one attribute on its right — but this course explicitly does **not require** it. Splitting the RHS is still the first working step, because condition 2's tests are only correct per-attribute; but the answer may then be written back up with same-LHS FDs recombined (`{E,F} → G` and `{E,F} → H` as `{E,F} → {G, H}`). Both presentations are accepted, and 3nf-synthesis[4] recombines them anyway as its second step.

Finding a minimal cover — three steps

The steps run in this order, and each is decided by an attribute-closure[5] computation:

Step	What to do	Test
1. RHS simplification	Split every FD so its RHS is a single attribute: `X → {Y, Z}` becomes `X → Y` and `X → Z`	— (always applies)
2. LHS simplification	For an FD `X → A` whose LHS has several attributes, drop any attribute `B` that isn't pulling its weight	Replace `X → A` with `(X - {B}) → A` when `(X - {B}) = X`
3. FD set simplification	Delete any FD that the rest of the set already implies	Delete `X → A` if `X`, computed *without* that FD, still contains `A`

Two things make this reliable in practice:

[4] courses/infs1200/3nf-synthesis.pdf
[5] courses/infs1200/attribute-closure.pdf

- **Do step 1 first, completely.** A compound RHS can hide the fact that one of its attributes is redundant while another isn't — the case-study `Department` example below is exactly that.
- **In steps 2 and 3, recompute the closure against the FD set as it currently stands**, including edits already made. Each removal changes what the remaining FDs can derive.

Worked example — the mechanical case

`R [A, B, C, D, E, F, G, H]` with `F = {A → B, {A,B,C,D} → E, {E,F} → G, {E,F} → H, {A,C,D,F} → {E, G}}`:

Step 1: RHS split	Step 2: LHS simplify	Step 3: delete redundant
`A → B`	`A → B`	`A → B`
`{A,B,C,D} → E`	`{A,C,D} → E` (`B` redundant, since `A → B`)	`{A,C,D} → E`
`{E,F} → G`	`{E,F} → G`	`{E,F} → G`
`{E,F} → H`	`{E,F} → H`	`{E,F} → H`
`{A,C,D,F} → E`	`{A,C,D,F} → E`	*dropped* — `{A,C,D,F}` already contains `E` without it
`{A,C,D,F} → G`	`{A,C,D,F} → G`	*dropped* — same reason (`E` then `{E,F} → G`)

Minimal cover: `{A → B, {A,C,D} → E, {E,F} → G, {E,F} → H}`, often written with the last two recombined as `{E,F} → {G, H}`.

The `B` in `{A,B,C,D} → E` is the classic step-2 case: `A → B` already puts `B` in the closure of anything containing `A`, so naming it in the LHS adds nothing.

Worked example — a wide right-hand side

Here step 1 does almost all the work. `R [C, S, J, D, P, Q, V]` with

`F = {C → {S, J, D, P, Q, V}, {J,P} → C, {S,D} → P, J → S}`.

Step 1: RHS split	Step 2: LHS simplify	Step 3: delete redundant
`C → S`	`C → S`	*dropped*
`C → J`	`C → J`	`C → J`
`C → D`	`C → D`	`C → D`
`C → P`	`C → P`	*dropped*

Step 1: RHS split	Step 2: LHS simplify	Step 3: delete redundant
C → Q	C → Q	C → Q
C → V	C → V	C → V
{J,P} → C	{J,P} → C	{J,P} → C
{S,D} → P	{S,D} → P	{S,D} → P
J → S	J → S	J → S

- **Step 2 changes nothing.** Only {J,P} → C and {S,D} → P have a compound LHS, and neither attribute can go: {J} = {J,S} and {P} = {P} both miss C; {S} = {S} and {D} = {D} both miss P.
- **Step 3 deletes two.** C → S goes because C → J then J → S already reaches S. C → P then goes too: with it removed, C still reaches S (via C → J, J → S) and D directly, and {S,D} → P supplies P. Neither removal depends on the other being made first — test them in the opposite order and both still come out redundant.

Minimal cover: {C → J, C → D, C → Q, C → V, {J,P} → C, {S,D} → P, J → S}, or with same-LHS FDs recombined, {C → {J, D, Q, V}, {J,P} → C, {S,D} → P, J → S}.

With C → P removed, the *only* surviving statement about P is {S,D} → P, which is exactly the FD 3nf-synthesis[6] will turn into its own relation. Steps 2 and 3 are not tidying — they decide the shape of the schema you end up with.

Worked example — spotting a false answer

R [A, B, C, D, E, F] with {D,E,F} → C, {A,B} → {D, C}, D → F.

- **Step 1** gives {D,E,F} → C, {A,B} → D, {A,B} → C, D → F.
- **Step 2**: in {D,E,F} → C, is F redundant? {D,E} already contains F via D → F, so {D,E} = {D,E,F} — yes. It becomes {D,E} → C.
- **Step 3**: nothing further can go; all four are necessary.

Minimal cover: {{D,E} → C, {A,B} → D, {A,B} → C, D → F}.

The wrong answers to this question are instructive: keeping the redundant F in the first LHS (step 2 skipped), dropping {A,B} → C (it is *not* implied by the others — {A,B} without it reaches D and F, never C), or dropping D → F (which the simplified {D,E} → C now depends on).

[6] courses/infs1200/3nf-synthesis.pdf

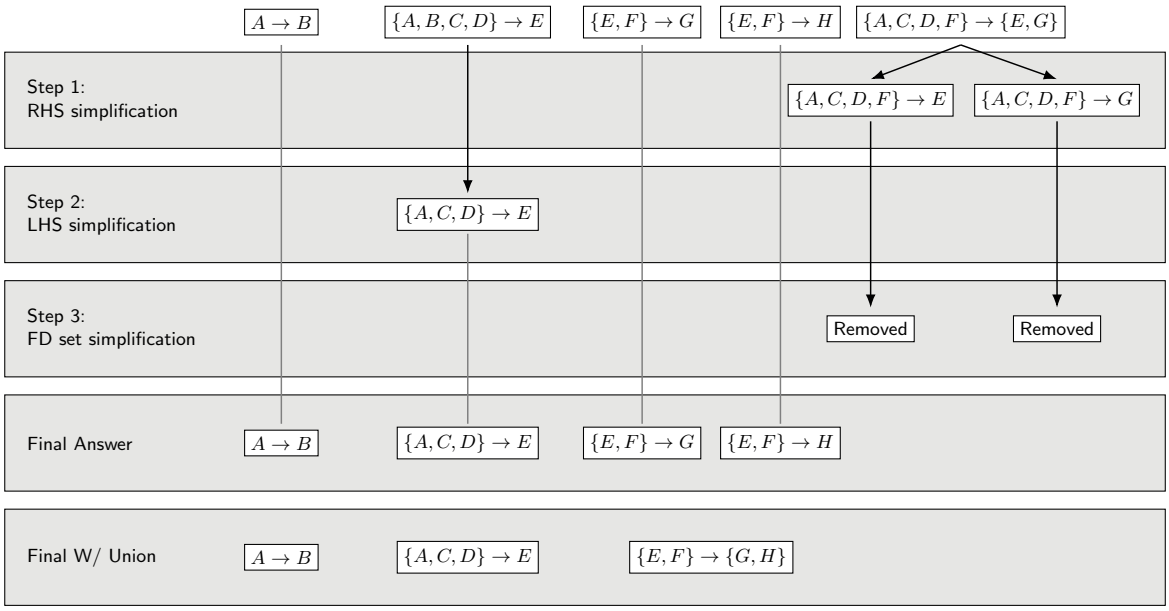


Figure 1: The same reduction seen as a flow, one band per step. Each original FD enters at the top; a black arrow means that step transformed it (split it, or shortened its LHS), a thin grey line means it passed through untouched, and the `Removed` boxes are where step 3's closure test deleted an FD outright.

Worked example — on a real schema

From week11-tutorial-case-study-9-payroll-system[7], the `Department` relation `[code, name, manager, buildingID, buildingName, buildingLocation, floor]` with `code` as the key, plus `buildingID → {buildingLocation, buildingName}` and `buildingName → {buildingLocation, buildingID}`.

After splitting the key's FD into single-attribute FDs, step 3 removes three of them:

- `code → buildingName` — derivable via `code → buildingID` then `buildingID → buildingName`.
- `code → buildingLocation` — derivable the same way.
- `buildingID → buildingLocation` — derivable via `buildingID → buildingName` then `buildingName → buildingLocation`.

Minimal cover: `{code} → {name, manager, buildingID, floor}`, `{buildingID} → {buildingName}`, `{buildingName} → {buildingLocation, buildingID}`.

Feeding that straight into 3nf-synthesis[8] yields `Department [code, name, manager, buildingID, floor]` and `Building [buildingName, buildingLocation, buildingID]` — the transitive `code → building…` redundancy removed exactly because step 3 removed the transitive FDs.

Redundancy can hide behind a chain

Step 3's test is "is `A` still reachable without this FD", which catches FDs whose work is done indirectly by two or more others. From week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis[9], `R [A, B, C, D, E, F]` with `A → {B,C,D,E,F}`, `{B,C} → A`, `{D,E} → B`, `C → D`:

- `A → B` is redundant — `A → D`, `A → E`, then `{D,E} → B`.
- `A → D` is redundant — `A → C`, then `C → D`.

leaving `A → {C, E, F}`, `{B,C} → A`, `{D,E} → B`, `C → D`. Neither removal is visible from a single FD; both need the closure test.

[7] courses/infs1200/week11-tutorial-case-study-9-payroll-system.pdf
[8] courses/infs1200/3nf-synthesis.pdf
[9] courses/infs1200/week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis.pdf

See also

- 3nf-synthesis[10] — the algorithm that consumes a minimal cover
- attribute-closure[11] · functional-dependencies[12] · normal-forms[13] · decomposition[14]
- 2026-05-04-normalisation-and-relational-database-schema-design[15] — Module 4, part 2
- week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis[16] · week11-tutorial-case-study-9-payroll-system[17] — practice

[10] courses/infs1200/3nf-synthesis.pdf
[11] courses/infs1200/attribute-closure.pdf
[12] courses/infs1200/functional-dependencies.pdf
[13] courses/infs1200/normal-forms.pdf
[14] courses/infs1200/decomposition.pdf
[15] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf
[16] courses/infs1200/week11-tutorial-applied-class-10-normalisation-bcnf-3nf-synthesis.pdf
[17] courses/infs1200/week11-tutorial-case-study-9-payroll-system.pdf