Normal Forms (1NF to BCNF)

Table of contents

Overview . . . 1
What you need before testing . . . 3
First Normal Form (1NF) . . . 3
Full and partial dependencies, and 2NF . . . 3
Transitive dependencies, and 3NF . . . 4
Boyce-Codd Normal Form (BCNF) . . . 5
Determining the highest normal form . . . 6
Denormalisation . . . 7
See also . . . 8

Normalisation is the process of putting a relation schema through a series of tests — defined in terms of its FDs[1] and its keys — to certify which **normal form** it satisfies. A schema that fails a test is decomposed into smaller relations with better properties. Put another way, it is the process of *identifying redundancy in data*: an FD `A → B` is precisely the statement that every tuple sharing an `A` value is forced to repeat the same `B` value, and each normal form rules out a different way that forcing can go wrong. Introduced in 2026-05-04-normalisation-and-relational-database-schema-design[2].

Each normal form names a *class of redundancy* and rules it out. The guarantees stack: `BCNF` ⇒ `3NF` ⇒ `2NF` ⇒ `1NF`, so a BCNF relation is automatically in all the others, and the "highest normal form" of a relation is the strictest test it passes.

Overview

[1] courses/infs1200/functional-dependencies.pdf
[2] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf

NF	Identifies	Test — for every non-trivial `X → A` in `F`
1NF	Non-atomic values	Relation has no multivalued attributes or nested relations
2NF	Partial dependencies (removes *some* anomalies)	`X` is **not** a proper subset of a candidate key, **or** `A` is a prime attribute
3NF	Partial + transitive dependencies (removes *most* anomalies)	`X` is a superkey, **or** `A` is a prime attribute
BCNF	*All* anomalies — at the cost of not preserving all FDs	`X` is a superkey

Note the tests quantify over `F`, not `F` — **implied** FDs count too (see functional-dependencies[3]). Trivial FDs are always exempt.

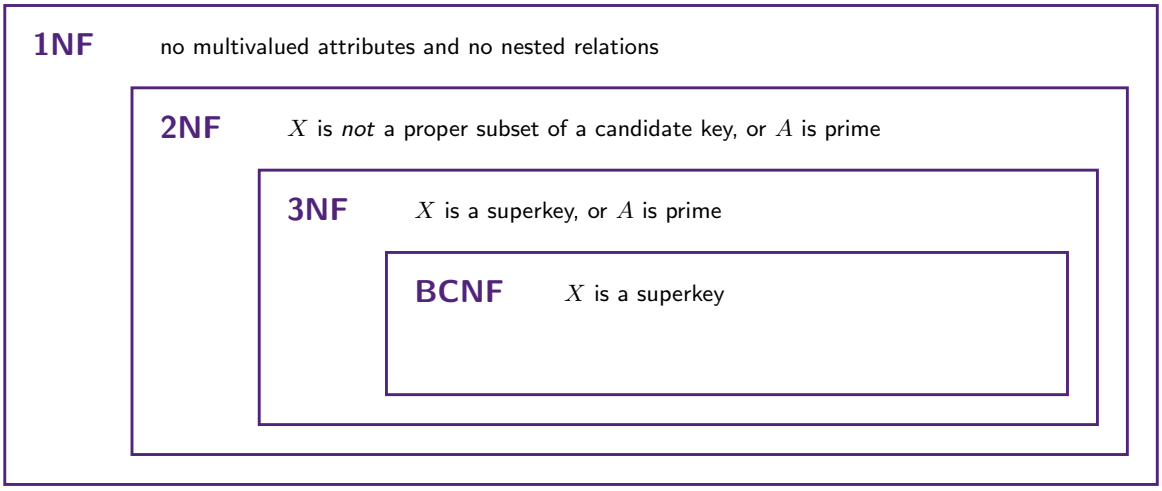


Figure 1: The four normal forms nest: each test is the previous one with an escape hatch removed, so a relation in an inner band is automatically in every band outside it. Each band carries its one-line test, and every test reads *for every non-trivial* `X → A` *in* `F`.

Read the figure from the inside out and the whole module fits on one line: BCNF asks only that every determinant is a superkey; 3NF adds the escape hatch "…or the determined attribute is prime"; 2NF weakens the superkey demand to "not a *proper subset* of a key"; 1NF stops asking about FDs at all and only asks that the values be atomic.

[3] courses/infs1200/functional-dependencies.pdf

What you need before testing

Every test above mentions candidate keys or prime attributes, so the first step is always the same: compute **all** candidate keys with attribute-closure[4], then mark each attribute **prime** (in at least one candidate key) or **non-prime**. Doing this against only the *primary* key is the most common source of wrong answers — 2NF's "proper subset of a candidate key" clause and 3NF's "or `A` is a prime attribute" escape hatch both depend on *every* candidate key, and a relation with several keys can pass against one and fail against another.

First Normal Form (1NF)

A relation is in 1NF if every attribute's domain contains only **atomic (simple, indivisible) values** — no set of values, no tuple of values (nested attributes), and no combination of the two.

An `items` column holding a comma-separated list violates 1NF; so does a `name` attribute that nests into `{firstName, familyName}`. Both standard ways of fixing it are unsatisfying:

- **Repeat** the non-nested columns once per value — one row per `(customerName, orderNum, item)` — which introduces redundancy; or
- **Flatten** into fixed columns (`item1`, `item2`, …) — which caps how many items an order can have and forces nulls on the shorter ones (guideline 3 of update-anomalies[5]).

That neither option is good is exactly the motivation for 2NF/3NF/BCNF, which say *how* to decompose an already-1NF relation.

Full and partial dependencies, and 2NF

- `X → Y` is a **full functional dependency** if removing *any* attribute from `X` breaks it — no proper subset of `X` determines `Y`.
- `X → Y` is a **partial dependency** if some attribute can be dropped from `X` and the FD still holds.

A relation `R` is in **2NF** if every **non-prime** attribute is **fully** functionally dependent on the primary key — informally, *no partial dependency on a key.*

Primary key or all candidate keys? The informal sentence above says "the primary key". The **formal** definition — the one given alongside every exam question, and the one that decides marks — is stated over *candidate keys*:

[4] courses/infs1200/attribute-closure.pdf
[5] courses/infs1200/update-anomalies.pdf

> `R` with FD set `F` is in 2NF iff, for every non-trivial `X → A` in `F` with `X  R` and `A  R`, either (1) `X` is **not** a proper subset of a candidate key for `R`, **or** (2) `A` is a prime attribute.

Use the formal version. It differs from the primary-key reading whenever a relation has more than one candidate key: a partial dependency on *any* candidate key breaks 2NF, not just one on the key you happened to nominate as primary. It also gives you a free shortcut in the other direction — if **every** attribute is prime, clause (2) is satisfied by every FD, so the relation is automatically in 2NF (and in 3NF).

Example — `Address [houseNum, street, postcode, state, value]` with `{houseNum, street, postcode} → {state, value}` and `postcode → state`. The only key is `{houseNum, street, postcode}`.

- `{houseNum, street, postcode} → value` is **full** — no subset of the LHS determines `value`.
- `{houseNum, street, postcode} → state` is **partial** — `postcode` alone already suffices.

`state` is non-prime and only partially dependent on the key, so `Address` is **not** in 2NF. Decompose into `Address [houseNum, street, postcode, value]` and `Postcodes [postcode, state]` with `Address.postcode references Postcodes.postcode`.

2NF doesn't fix much. `Employee [id, name, level, salary]` with `level → salary` satisfies 2NF — `level` isn't a *proper subset* of the key `{id}`, it's a whole non-key attribute — yet it still has all three anomalies from update-anomalies[6]. 2NF only catches redundancy hanging off *part of a composite key.*

Transitive dependencies, and 3NF

`X → Y` in `R` is a **transitive dependency** if some attribute set `Z` exists such that `Z` is neither a candidate key nor a subset of one, and both `X → Z` and `Z → Y` hold — i.e. `Y` depends on `X` only *through* an intermediate that isn't a key.

In `Employee [id, name, level, salary]` with `level → salary`: `level` is neither a candidate key nor part of one, `id → level` holds and `level → salary` holds, so `id → salary` is a transitive dependency.

`R` is in **3NF** iff, for every non-trivial `X → A` in `F` , `X` is a superkey **or** `A` is a prime attribute. Equivalently: whenever `A` is non-prime, `X` must be a superkey.

`level → salary` fails both halves — `level` isn't a superkey and `salary` isn't prime — so `Employee` violates 3NF. Decompose into `StaffAppointment [id, name, level]` and `StaffIncome [level, salary]`.

[6] courses/infs1200/update-anomalies.pdf

3NF still isn't always enough. `Teach [studentID, courseID, lecturer]` with `{studentID, courseID} → lecturer` and `lecturer → courseID` (keys: `{studentID, courseID}` and `{studentID, lecturer}`) **is** in 3NF — `lecturer → courseID` escapes via the prime-attribute clause, since `courseID` is prime. Yet:

- **Deletion anomaly** — deleting the last row for a `(studentID, courseID)` pair can destroy the only record of who teaches that course.
- **Insertion anomaly** — you can't record which lecturer teaches a course that currently has no enrolled students.

Partial vs transitive, in one line each

The distinction is a recurring exam question, and the marked answer is phrased like this:

> A **partial** dependency is where an attribute depends on only a *subpart of the primary key*; normalising to **2NF** solves it. A **transitive** dependency is where a non-prime attribute depends on *other non-prime attributes*; normalising to **3NF** solves it.

The trap in the distractors is the swap — attaching partial dependencies to 3NF and transitive ones to 2NF.

Boyce-Codd Normal Form (BCNF)

`R` is in **BCNF** iff, for every non-trivial `X → A` in `F` , `X` is a superkey. Informally: *whenever a set of attributes determines anything, it must determine everything.* BCNF is 3NF with the prime-attribute escape hatch removed.

In `Teach`, `lecturer → courseID` now fails — `lecturer` is not a superkey — so `Teach` is not in BCNF even though it is in 3NF. That gap between 3NF and BCNF is exactly the set of relations where 3NF's prime-attribute clause is letting an anomaly through.

BCNF is strictly stronger, but not strictly better: decomposing to reach it can lose FDs that 3NF would have kept.

3NF or BCNF? What each one buys

Real designs aim for one of these two, and the choice is a straight swap:

	Removes all anomalies	Preserves all FDs
BCNF		
3NF		

Neither dominates. BCNF eliminates the redundancy but a BCNF decomposition may scatter an FD's attributes across relations so that the constraint can no longer be checked without a join; 3NF keeps every FD checkable inside a single relation but leaves the `Teach`-shaped anomalies above in place. See decomposition[7] for dependency preservation and the BCNF algorithm, and 3nf-synthesis[8] for the algorithm that guarantees the 3NF side of the table.

Determining the highest normal form

1. Compute **all** candidate keys (attribute-closure[9]); mark prime and non-prime attributes.
2. Split every FD so its RHS is a single attribute — the tests are stated per-attribute, and a compound RHS can pass on one attribute and fail on another.
3. Test each non-trivial FD `X → A` from the strictest end down: is `X` a superkey (BCNF)? If not, is `A` prime (3NF)? If not, is `X` a proper subset of a candidate key (breaks 2NF) or not (breaks 3NF only)?
4. The relation's normal form is the strictest one **no** FD violates.

Relation and FDs	Candidate keys	Highest NF	Why
`R [A,B,C,D,E]`, `A → {B,C,D,E}`, `E → A`	`{A}`, `{E}`	**BCNF**	Every FD's LHS is itself a superkey
`R [A,B,C,D]`, `{A,B} → {C,D}`, `{C,D} → A`, `D → B`	`{A,B}`, `{C,D}`, `{A,D}`	**3NF**	Every attribute is prime, so nothing can break 3NF; but `D → B` has a non-superkey LHS, breaking BCNF
`R [A,B,C,D]`, `{A,C,D} → B`, `{A,C} → D`, `D → C`, `{A,C} → B`	`{A,C}`, `{A,D}`	**3NF**	`D → C`: `D` isn't a superkey (not BCNF), but `C` is prime (3NF survives)
`R [A,B,C,D,E,F]`, `{A,B} → {C,D,E}`, `C → F`, `E → {A,B}`	`{A,B}`, `{E}`	**2NF**	No FD's LHS is a *proper subset* of a key, so 2NF holds; `C → F` has a non-superkey LHS and non-prime RHS, breaking 3NF

[7] courses/infs1200/decomposition.pdf
[8] courses/infs1200/3nf-synthesis.pdf
[9] courses/infs1200/attribute-closure.pdf

Relation and FDs	Candidate keys	Highest NF	Why
`R [A,B,C,D,E]`, `B → {C,D}`, `A → E`	`{A,B}`	**1NF**	Both `A` and `B` are proper subsets of the key with non-prime RHSs — partial dependencies, breaking 2NF
`R [A,B,C,D,E,F]`, `{A,E} → D`, `{B,C} → A`, `B → F`, `F → E`	`{B,C}`	**1NF**	`B → F` is a partial dependency: `B` `{B,C}` and `F` is non-prime

The fourth and fifth rows are the pair worth internalising — the *same* shape of violation (non-superkey LHS, non-prime RHS) lands on 3NF or 2NF purely depending on whether the LHS sits inside a candidate key.

A subtlety from the tutorial: in `R [A,B,C,D,E,F,G]` with `A → {B,C,D}`, `D → A`, `C → {F,G}` (keys `{A,E}` and `{D,E}`), splitting `A → {B,C,D}` gives `A → B`, `A → C` and `A → D`. The first two are partial dependencies (`A` `{A,E}`, and `B`/`C` are non-prime) so the relation is only in 1NF — but `A → D` is *not* a partial dependency, because `D` is prime (it appears in key `{D,E}`). That's why step 2 above insists on splitting the RHS first.

Denormalisation

Since anomalies come from redundancy it's tempting to decompose as aggressively as possible — but a heavily decomposed schema needs more JOIN[10]s to answer the same query. **Denormalisation** is the *deliberate*, controlled relaxation of a normal form to improve query performance: fewer joins (faster queries) and fewer foreign keys (less storage and maintenance overhead), which suits analytical workloads or a small set of very frequent pre-joined queries.

The word "deliberate" is load-bearing. Denormalisation is a trade-off made knowingly, accepting the anomalies it reintroduces — not an excuse for a schema that was never normalised in the first place.

[10] courses/infs1200/sql-joins.pdf

See also

- functional-dependencies[11] · attribute-closure[12] · decomposition[13] · 3nf-synthesis[14] · minimal-cover[15]
- update-anomalies[16] — the anomalies each normal form is removing
- 2026-05-04-normalisation-and-relational-database-schema-design[17] — Module 4, part 2
- week10-tutorial-applied-class-9-functional-dependencies[18] · week10-tutorial-case-study-8-dirt-road-driving[19] — highest-normal-form practice

[11] courses/infs1200/functional-dependencies.pdf
[12] courses/infs1200/attribute-closure.pdf
[13] courses/infs1200/decomposition.pdf
[14] courses/infs1200/3nf-synthesis.pdf
[15] courses/infs1200/minimal-cover.pdf
[16] courses/infs1200/update-anomalies.pdf
[17] courses/infs1200/2026-05-04-normalisation-and-relational-database-schema-design.pdf
[18] courses/infs1200/week10-tutorial-applied-class-9-functional-dependencies.pdf
[19] courses/infs1200/week10-tutorial-case-study-8-dirt-road-driving.pdf