Functional Dependencies

Table of contents

Formal definition . . . 1
An FD is a statement about *all* instances . . . 2
Reading candidate FDs off a sample instance . . . 2
Trivial and non-trivial FDs . . . 3
Explicit and implicit FDs; the closure of F . . . 3
Armstrong's axioms and what is actually examinable . . . 4
Why FDs create redundancy . . . 4
See also . . . 6

A **functional dependency (FD)** is the formal way to state a business rule of the form "if you know *this*, you know *that*". It's the unit of information every design-theory tool in this course consumes: closures, keys, normal forms and both schema-design algorithms are all defined in terms of a set of FDs. Introduced in 2026-04-27-database-design-guidelines-and-functional-dependencies[1].

The motivating question is the one left hanging by update-anomalies[2]: how can we be *sure* every employee at the same level earns the same salary, rather than it merely happening to be true of today's rows? By declaring `level → salary` — "level determines salary".

Formal definition

An FD `X → Y` holds on relation `R` if, for every legal instance `r` of `R` and every pair of tuples $t_1, t_2 \in r$:

$$t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$$

[1] courses/infs1200/2026-04-27-database-design-guidelines-and-functional-dependencies.pdf
[2] courses/infs1200/update-anomalies.pdf

i.e. any two tuples that agree on `X` must also agree on `Y`. `X` is the **left-hand side (LHS)** or *determinant*, `Y` the **right-hand side (RHS)**. Both are *sets* of attributes, so `{sName, pNum} → qty` is a perfectly ordinary FD; single-attribute sets are written without braces.

An FD is a **constraint between two attribute sets** — it restricts which tuples may legally appear together in an instance of `R`, exactly as a key or referential-integrity constraint does (see integrity-constraints[3]). A key is in fact a special case: `X` is a superkey of `R` precisely when `X → R` holds, which is how relational-keys[4]'s "minimal set that uniquely identifies" becomes something you can *compute* rather than assert — see attribute-closure[5].

An FD is a statement about *all* instances

This is the single most important thing about FDs, and the thing exam questions probe most often:

> You can check whether a given instance **violates** an FD, but you can never **prove** an FD holds by looking at one instance.

FDs come from the semantics of the application — the Universe of Discourse and its business rules — not from reverse-engineering a data sample. A sample can only ever rule FDs *out*.

Ruling one out: given `R [A, B, C, D]` with rows `(1,2,3,4)`, `(2,3,4,6)`, `(6,7,8,9)`, `(1,3,4,5)` — `B → D` **cannot** be true, since rows 2 and 4 both have `B = 3` but disagree on `D` (`6` vs `5`). `B → C` and `D → B` are still *possible* — meaning only "not yet contradicted", never "proven".

Reading candidate FDs off a sample instance

Case-study questions do ask you to *identify* the FDs a sample table suggests. The mechanical procedure is: for each candidate `X → Y`, group the rows by their `X` value; if any group disagrees on `Y`, the FD is dead, otherwise it survives as a **potential** FD. Two traps:

- An `X` value that never repeats makes `X → Y` **trivially survive** — there is no pair of rows to contradict it. In week10-tutorial-applied-class-9-functional-dependencies[6] Section A, `A → B`, `A → C` and `{A, B} → C` all "hold" only because `A` is distinct in every row. That's evidence of nothing.

[3] courses/infs1200/integrity-constraints.pdf
[4] courses/infs1200/relational-keys.pdf
[5] courses/infs1200/attribute-closure.pdf
[6] courses/infs1200/week10-tutorial-applied-class-9-functional-dependencies.pdf

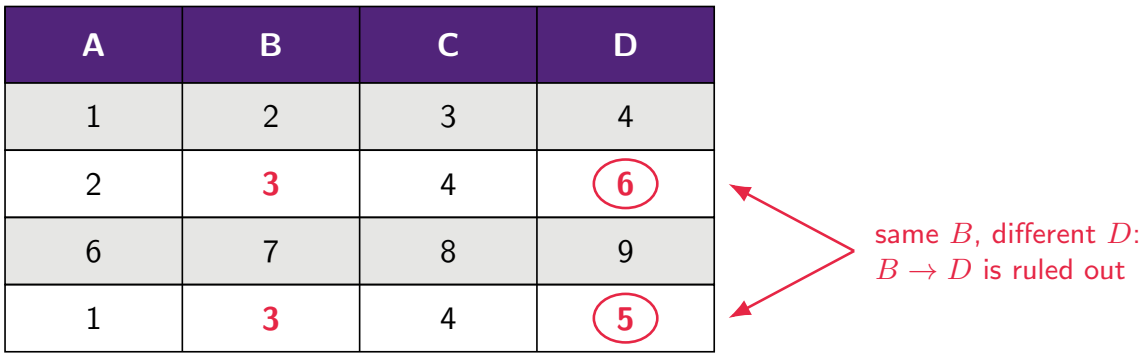


Figure 1: The only thing an instance can do to an FD. Rows 2 and 4 agree on `B` and disagree on `D`, so `B → D` is refuted outright; nothing in the table can establish `B → C` or `D → B`, which merely survive un-contradicted. The crimson ellipses mark the two `D` values that clash.

- The surviving FDs still have to make *sense*. In week10-tutorial-case-study-8-dirt-road-driving[7], the FD worth reporting on `EmployeeHistory` is `seniorityLevel → baseSalary, securityLevel` — every `Junior` row has `{70,000, Limited}` and every `Executive` row has `{80,000, Full Access}` — because seniority genuinely determines pay band in the UoD, not merely in the sample.

Trivial and non-trivial FDs

- A **trivial FD** holds regardless of `F`, because the LHS already contains the RHS — e.g. `A → A`, `{A, B, C} → {A, B}`.
- A **non-trivial FD** depends on the specific `F` — e.g. `A → B`.

Every normal-form test is stated over *non-trivial* FDs only, since a trivial FD can never be a design flaw.

Explicit and implicit FDs; the closure of F

Given a set `F` of *explicit* FDs, further *implicit* FDs follow: from `id → level` and `level → salary` we can infer `id → salary`. Implicit FDs are also called **inferred** FDs. The notation

$$F \models X \to Y$$

[7] courses/infs1200/week10-tutorial-case-study-8-dirt-road-driving.pdf

means `X → Y` can be inferred from `F` — equivalently, "`Y` can be derived from `X` under `F`", or "`X` implies `Y` under `F`".

The **closure of F**, written `F` , is the set of *all* FDs — trivial and non-trivial — implied by `F`.

Armstrong's axioms and what is actually examinable

`F` can be computed by applying **Armstrong's Axioms** as inference rules, but doing so is *difficult and time consuming*, and it is **outside the scope of this course**. The individual rules — reflexivity, augmentation, transitivity, and the derived union, decomposition and pseudotransitivity rules — are outside it too, so there is nothing to memorise and no question can ask you to name or apply one.

What replaces them is attribute-closure[8] (`X`), which answers the only question the rest of the module ever asks of `F` : given `X`, which attributes are determined? Every inference you actually need — the transitive chain `id → level → salary`, say — falls out of one closure computation, and *that* is the examinable skill.

`F` still matters conceptually, because every normal-form test is phrased "for every non-trivial `X → A` **in** `F` " — the implied FDs count too. Missing an implicit FD is the classic way to declare a relation BCNF-compliant when it isn't; in practice you compute the closures first, spot that `F  A → C`, add `A → C` to `F`, and *then* run the algorithm. See decomposition[9].

Why FDs create redundancy

FDs are also the precise account of *where redundancy comes from*. In `R [A, B, C]` with no FDs at all, no value constrains any other and there is no redundancy. Add `A → B`, and any two tuples sharing an `A` value are now *forced* to repeat the same `B` value. That forced repetition is exactly what normalisation identifies and removes — which is why the normal-form tests in normal-forms[10] are all conditions on the LHS of an FD relative to the keys.

A running instance makes the mechanism concrete. `Employee [ID, name, level, salary]` with `level → salary` stores `Developer, 60,000` twice and `Manager, 700,000` twice — not by accident but *because* the FD forbids anything else. Project the relation onto `[ID, name, level]` and `[level, salary]` and each level's salary is recorded exactly once, with nothing lost: the natural join on `level` rebuilds the original rows. The two halves are named `StaffAppointment [ID, name, level]` and `StaffIncome [level, salary]` in the 3NF treatment; see normal-forms[11] and decomposition[12].

[8] courses/infs1200/attribute-closure.pdf
[9] courses/infs1200/decomposition.pdf
[10] courses/infs1200/normal-forms.pdf
[11] courses/infs1200/normal-forms.pdf
[12] courses/infs1200/decomposition.pdf

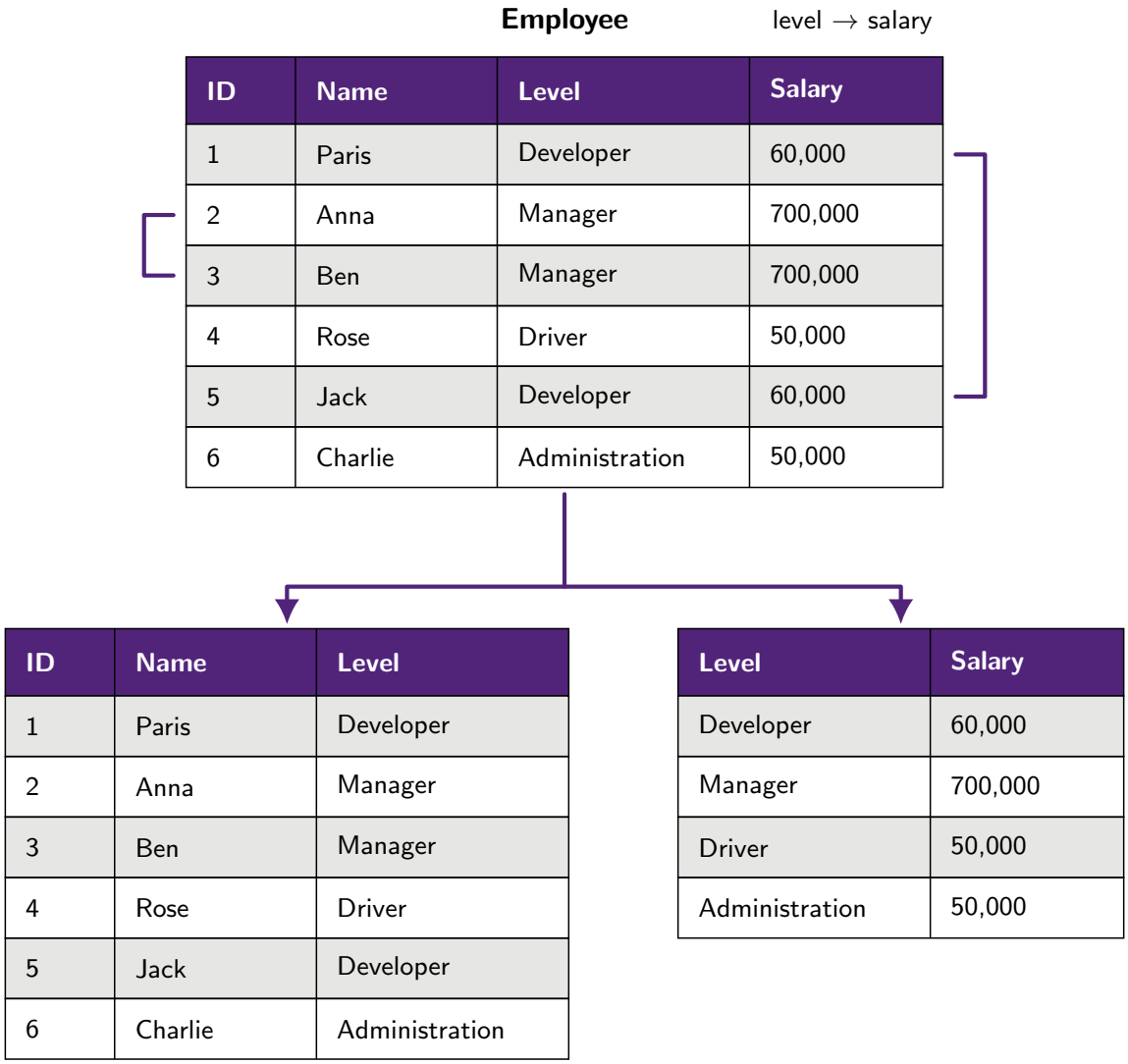


Figure 2: An FD is a redundancy generator, and the split is the cure. `level → salary` forces every Developer row to carry `60,000` and every Manager row `700,000` — the purple brackets mark the two groups the FD ties together. Splitting off a `[level, salary]` table records each level's salary once; the natural join on `level` rebuilds the six original rows.

The three anomalies that repetition causes — and which the split fixes — are catalogued in update-anomalies[13].

See also

- attribute-closure[14] · normal-forms[15] · minimal-cover[16] · decomposition[17]
- update-anomalies[18] — the symptoms FDs explain
- 2026-04-27-database-design-guidelines-and-functional-dependencies[19] — Module 4, part 1
- week10-tutorial-applied-class-9-functional-dependencies[20] — FD identification and closure practice

[13] courses/infs1200/update-anomalies.pdf
[14] courses/infs1200/attribute-closure.pdf
[15] courses/infs1200/normal-forms.pdf
[16] courses/infs1200/minimal-cover.pdf
[17] courses/infs1200/decomposition.pdf
[18] courses/infs1200/update-anomalies.pdf
[19] courses/infs1200/2026-04-27-database-design-guidelines-and-functional-dependencies.pdf
[20] courses/infs1200/week10-tutorial-applied-class-9-functional-dependencies.pdf