Attribute Closure and Finding Keys

Table of contents

Definition . . . 1
The algorithm . . . 2
Worked examples . . . 2
Key vocabulary . . . 3
Finding all candidate keys . . . 4
Where closures get used . . . 4
See also . . . 5

Attribute closure is the one algorithm the whole of Module 4 runs on. Every later question — is this a key? is this relation in BCNF? is this FD redundant? does this FD survive a decomposition? — is answered by computing a closure. Introduced in 2026-04-27-database-design-guidelines-and-functional-dependencies[1].

Definition

Given a relation `R`, a set of FDs[2] `F`, and a set of attributes `X` `R`, the **closure of X** (written `X`) is the set of all attributes functionally determined by `X` under `F`:

$$X^{+} = \{\, A \in R \;:\; F \models X \to A \,\}$$

Computing `X` is the practical substitute for computing `F` — which would need Armstrong's Axioms and is explicitly out of scope for this course (see functional-dependencies[3]) — and it answers the only question we ever actually ask of `F` .

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

The algorithm

Start with `X` itself, then repeatedly apply any FD whose LHS is already fully contained in the set, until nothing more can be added:

```
X+ := X;
repeat
    old X+ := X+;
    for each FD Y → Z in F:
        if Y   X+ then X+ := X+   Z;
until (old X+ = X+);
```

Two things to watch:

- The LHS must be **entirely** inside `X` before the FD can fire — a partial match does nothing. `{A, D} = {A, D}` under `{A,B} → E, C → {B,E}, {E,D} → F`, because none of the three LHSs is a subset of `{A, D}`, so the closure never grows past its starting point.
- It's a **fixpoint** loop, not a single pass: an FD that couldn't fire earlier may fire once a later FD has added its LHS. Keep sweeping `F` until a full pass adds nothing.

Worked examples

`Employee (id, level, salary, name)` with `F = {id → level, level → salary, id → name}`:

Step	`{id}`	Using
0	`{id}`	— (start)
1	`{id, level}`	`id → level`
2	`{id, level, salary}`	`level → salary`
3	`{id, level, salary, name}`	`id → name`

`{id}` covers every attribute, so `id` is a superkey — and since it's a single attribute it's trivially minimal, so it's a key.

A chained closure — `R [pNumber, pName, pLocation, dNum, dName, mgrSSN, mgrStartDate]` with `F = {pNumber → {pName, pLocation, dNum}, dNum → {dName, mgrSSN, mgrStartDate}}`:

```
{pNumber}+ = {pNumber}
{pNumber}+ = {pNumber, pName, pLocation, dNum}                      using FD1
{pNumber}+ = {pNumber, pName, pLocation, dNum, dName, mgrSSN,
              mgrStartDate}                                         using FD2
```

The second FD only became applicable *after* the first added `dNum` — the fixpoint loop in action. The closure is all of `R`, so `pNumber` is a key.

A composite LHS — showing `{sName, pNum}` is a superkey for `SupplierPart (sName, city, status, pNum, pName, qty)` with `F = {sName → city, city → status, pNum → pName, {sName, pNum} → qty}`:

```
{sName, pNum}+ = {sName, pNum}
{sName, pNum}+ = {sName, pNum, city}                          using sName → city
{sName, pNum}+ = {sName, pNum, city, status}                  using city → status
{sName, pNum}+ = {sName, pNum, city, status, pName}           using pNum → pName
{sName, pNum}+ = {sName, pNum, city, status, pName, qty}      using {sName,pNum} → qty
```

Showing it is a **key** rather than merely a superkey takes a second half: every proper subset must fail. `{sName} = {sName, city, status}` and `{pNum} = {pNum, pName}`, neither of which is all of `SupplierPart`, so nothing smaller works and `{sName, pNum}` is minimal. "Cover all attributes, then show each proper subset doesn't" is the standard two-part answer whenever a question says *key* rather than *superkey*.

Key vocabulary

Everything below is defined *relative to a set of FDs*, and each is decided by a closure computation. See relational-keys[4] for how the same terms are used in the relational model generally.

Term	Definition	Test
Superkey	A set of attributes for which no two tuples can share the same values	`S = R`
(Candidate) key	A *minimal* superkey — removing any attribute breaks the superkey property	`S = R` **and** no proper subset `S'` of `S` has `S' = R`
Primary key	One candidate key, chosen as *the* key (only one per relation)	— (a design choice)
Prime attribute	An attribute belonging to *any* candidate key — not just the primary key	Appears in at least one candidate key
Non-prime attribute	Belongs to no candidate key	—

[4] courses/infs1200/relational-keys.pdf

Minimal shortest: a relation can have several candidate keys of different sizes, and every superkey contains at least one candidate key. Prime/non-prime is the distinction the 2NF and 3NF tests turn on, so it has to be computed against *all* candidate keys, not the primary one — see normal-forms[5].

Finding all candidate keys

For `n` attributes there are `2` subsets in the worst case, but three observations cut that down to something you can do by hand:

1. **An attribute that never appears on the RHS of any FD must be in every key.** Nothing can ever derive it, so no set that omits it can have full closure. This is usually the fastest way in.
2. **Once `S` is known to be a key, skip every superset of `S`.** Those are superkeys, not candidate keys.
3. **Expect more than one key.** Stop only when you've checked every minimal combination, not at the first hit.

Worked example — `R [A, B, C, D, E, F]` with the FDs `A → {B, C}`, `{C,D} → E`, `{A,C} → E`, `B → D`, `E → {A, B}` (from week10-tutorial-applied-class-9-functional-dependencies[6], Section C). Note the attribute named `F` here has nothing to do with the usual name for the FD set:

The attribute `F` never appears on any RHS, so by tip 1 it belongs to every key. Among the remaining attributes the FDs form a tightly coupled cluster: `A` alone reaches `B, C`, then `D` (via `B → D`), then `E`; `E` alone reaches `A, B` and so everything `A` does; `{B, C}` reaches `D` then `E` then `A`; `{C, D}` reaches `E` then `A, B`. So there are four minimal ways to cover the cluster, each needing `F` added:

Candidate keys: `{A, F}`, `{E, F}`, `{B, C, F}`, `{C, D, F}`.

A negative example — `R [A, B, C, D, E, F]` with `{A,B} → E`, `C → {B, E}`, `{E,D} → F`. Is `{A, B}` a key? `{A,B} = {A, B, E}` — misses `C, D, F`, so it isn't even a superkey. Is `{A, B, C}`? `{A,B,C} = {A, B, C, E}` — still misses `D, F`. The key is `{A, C, D}`: `C → {B, E}` adds `B, E`, then `{E,D} → F` adds `F`, giving all six attributes, and no proper subset gets there.

Superkey key — with `B → C`, `C → B`, `D → {A, B, C}` on `R [A, B, C, D]`, `{B, D}` has full closure but is *not* a candidate key, because `D` alone already does (`D = R`). Every superset of `D` is a superkey; only `D` itself is the key.

Where closures get used

[5] courses/infs1200/normal-forms.pdf
[6] courses/infs1200/week10-tutorial-applied-class-9-functional-dependencies.pdf

Task	Closure computed
Is `S` a superkey?	`S` $=$ `R`?
Is `S` a candidate key?	`S` $=$ `R` and `S'` $\not=$ `R` for every `S'` $\subset$ `S`
Does `R` satisfy BCNF for `X → A`?	Is `X` $=$ `R`? (see normal-forms[7])
Is an LHS attribute `B` redundant in `X → A`?	`(X - {B})` $=$ `X` ? (see minimal-cover[8])
Is the whole FD `X → A` redundant?	`X` computed *without* it still contains `A`?
Does `X → Y` hold in a decomposed relation `S`?	`X` $\rightarrow$ `Y` $\subseteq$ `S` and `Y` $\not\rightarrow$ `X` (see decomposition[9])

See also

- functional-dependencies[10] · normal-forms[11] · minimal-cover[12] · decomposition[13] · 3nf-synthesis[14]
- 2026-04-27-database-design-guidelines-and-functional-dependencies[15] — Module 4, part 1
- week10-tutorial-applied-class-9-functional-dependencies[16] — Sections B and C are pure closure/candidate-key drill

[7] courses/infs1200/normal-forms.pdf
[8] courses/infs1200/minimal-cover.pdf
[9] courses/infs1200/decomposition.pdf
[10] courses/infs1200/functional-dependencies.pdf
[11] courses/infs1200/normal-forms.pdf
[12] courses/infs1200/minimal-cover.pdf
[13] courses/infs1200/decomposition.pdf
[14] courses/infs1200/3nf-synthesis.pdf
[15] courses/infs1200/2026-04-27-database-design-guidelines-and-functional-dependencies.pdf
[16] courses/infs1200/week10-tutorial-applied-class-9-functional-dependencies.pdf