Relational Keys

Table of contents

Minimality is the whole point 1
The vocabulary 2
Common misconceptions 4
Worked example — electoral roll 5
What keys enforce 5

A **key** is a *minimal* set of attributes that uniquely identifies the tuples of a relation. Everything else in this note is a refinement of that one sentence. Introduced in 2026-03-09-the-relational-model-and-integrity-constraints[1] and drilled in week4-tutorial-applied-class-3-relational-integrity[2]; see relational-mapping-notation[3] for how keys are written down (the primary key is underlined) and foreign-keys[4] for the cross-relation case.

Minimality is the whole point

Uniqueness alone isn't enough — a key must also be **minimal**, meaning it contains no redundant attributes. Minimal does *not* mean "the shortest set anyone could find": it means that removing any attribute from the set destroys the uniqueness property.

Given `Department [id, name, manager]` where department IDs are unique:

Candidate set	Unique?	Minimal?	Key?
`(id)`	Yes	Yes	**Yes**
`(id, name)`	Yes	No — `id` alone already suffices	No
`(id, manager)`	Yes	No — same reason	No

[1] courses/infs1200/2026-03-09-the-relational-model-and-integrity-constraints.pdf
[2] courses/infs1200/week4-tutorial-applied-class-3-relational-integrity.pdf
[3] courses/infs1200/relational-mapping-notation.pdf
[4] courses/infs1200/foreign-keys.pdf

Adding anything to an already-unique set keeps it unique but destroys minimality, so `(id, name)` is never a key while `(id)` is one.

Ask the same key question twice on one instance, changing only the assumptions, and the pair is the whole lesson:

Only ID is assumed unique
Department

ID	Name	Manager
1	Marketing	4671
2	Development	1751
4	Human Resources	1023

(ID) unique, minimal ⇒ key

(ID, Name) unique, *not* minimal ⇒ not a key

Dropping Name still identifies a department, so Name is redundant. Same for (ID, Manager).

ID unique and (Name, Manager) unique
Department

ID	Name	Manager
1	Marketing	4671
2	Development	1751
4	Human Resources	1023

(ID) candidate key

(Name, Manager) candidate key

Two candidate keys; exactly one is nominated primary key. (ID, Name) is still not a key.

Figure 1: Two key questions on one instance. Left: only ID is assumed unique, so (ID) is the sole key and (ID, Name) fails minimality. Right: (Name, Manager) is also assumed unique, which adds a second candidate key without making (ID, Name) one.

Nothing in the *data* changed between the two panels — only what the question tells you to assume. Keys are a property of the schema and its stated uniqueness assumptions, never something you can read off one instance.

The vocabulary

Term	Meaning
Superkey	Any set of attributes that uniquely identifies tuples — not necessarily minimal
Candidate key	A *minimal* superkey; a relation may have several
Primary key	The one candidate key chosen as *the* key for the relation; exactly one per relation
Composite key	A key made of more than one attribute

Every key is a superkey; the reverse doesn't hold. In the table above, `(id, name)` is a superkey but not a candidate key. The relational-model material works with *key*, *candidate key* and *primary key* only — **superkey** enters later, with design theory, where the formal treatment of it lives: attribute closure, and how to *find* all the candidate keys of a relation from its functional dependencies (see functional-dependencies[5] and attribute-closure[6]).

Textbook terms the course never uses are worth *not* importing: there is no **alternate key**, **secondary key** or **surrogate key** anywhere in INFS1200. A candidate key that lost the nomination is just "another candidate key" and becomes an ordinary attribute in the relational schema; an artificial identifier is just a key that was chosen. Answers should stay inside the four terms above.

Several candidate keys

If department IDs are unique **and** the combination of `name` and `manager` is also unique per department, then `Department` has two candidate keys: `(id)` and `(name, manager)`. Both are minimal and both are unique; `(id, name)` still isn't a key because `(id)` alone already suffices. Exactly one of the candidate keys is then nominated as the primary key — a *design* choice, not something the data dictates.

Composite keys

A key can perfectly well span several attributes. Composite primary keys turn up constantly, and mostly for structural reasons rather than arbitrary ones:

- A weak entity's relation is keyed on its owner's primary key plus its own partial key, e.g. `Dependent [ssn, depName, …]`.
- An M:N relationship's relation is keyed on the pair of foreign keys it carries, e.g. `WorksOn [ssn, pNo, hours]`.
- A multivalued attribute's relation is keyed on the owner's key plus the value itself, e.g. `DeptLocs [dNumber, location]`.

All three come out of er-to-relational-mapping[7]. In this course's notation a composite primary key is written as one continuous underline spanning all of its attributes — one key, not several.

[5] courses/infs1200/functional-dependencies.pdf
[6] courses/infs1200/attribute-closure.pdf
[7] courses/infs1200/er-to-relational-mapping.pdf

Common misconceptions

From the true/false drill in week4-tutorial-applied-class-3-relational-integrity[8]:

Statement	Verdict
A key is a unique identifier of a tuple in the relation	**True** — that's the definition
The key constraint (or uniqueness constraint) implies that no two tuples in a relation can have the same values for *all* attributes	**True** *(as the course marks it)*
A primary key is a candidate key chosen as the main key for a relation	**True**
A key cannot have multiple attributes	**False** — composite keys exist
An entity may have more than one primary key	**False** — several *candidate* keys, but only one is the primary key

The second row is worth pausing on, because the obvious objection is a good one: no two tuples can agree on *all* attributes in any relation, declared key or not, since a relation is a set. The course still marks it true, reading it as "the key constraint implies uniqueness of tuples" — so answer it the course's way, and keep the set-semantics point as the reason the statement is weak rather than as a reason to call it false.

Matching the four terms

Matching definitions to names is a compact statement of all four:

Definition	Term
Any minimal set of attributes that can uniquely identify tuples in a relation	**Candidate key**
A candidate key that is chosen to identify tuples in a relation	**Primary key**
Defines the structure of a relation; specified during database design	**Relational schema**
Data in a relation, modified via create, update and delete operations	**Relational instance**

[8] courses/infs1200/week4-tutorial-applied-class-3-relational-integrity.pdf

Worked example — electoral roll

```
Region [prefix, suffix]

Member [enrolmentId, passportNum, regionPrefix, regionSuffix]
Member.{regionPrefix, regionSuffix} references Region.{prefix, suffix}

Candidate [enrolmentId, name, regionPrefix, regionSuffix]
Candidate.enrolmentId references Member.enrolmentId
Candidate.{regionPrefix, regionSuffix} references Region.{prefix, suffix}
```

- **Candidate keys**: `Member.enrolmentId`, `Member.passportNum`, `Candidate.enrolmentId`, `Region.{prefix, suffix}` — `Member` has two, since a passport number identifies a member just as well as an enrolment ID does.
- **Primary keys**: `Member.enrolmentId`, `Candidate.enrolmentId`, `Region.{prefix, suffix}` — one per relation, `passportNum` losing out to `enrolmentId` as a design choice.
- **Composite primary key**: `Region.{prefix, suffix}`.
- **Foreign keys**: `Member.{regionPrefix, regionSuffix}`, `Candidate.{regionPrefix, regionSuffix}`, `Candidate.enrolmentId` — note the last one is simultaneously `Candidate`'s primary key *and* a foreign key, which is exactly what a subclass relation looks like.

What keys enforce

Two of the five integrity-constraints[9] exist to protect keys:

- The **key constraint** — no two tuples may share a value for a key. Violated by inserting or modifying a tuple so that it duplicates an existing key value.
- The **entity integrity constraint** — no part of a primary key may be `NULL`. For a composite primary key this means *every* component must be non-null; a single null component is already a violation.

Both are declarative: the DBMS enforces them automatically once the key is declared, without any application code (see sql-ddl[10] for the `PRIMARY KEY` / `UNIQUE` syntax that declares them).

[9] courses/infs1200/integrity-constraints.pdf
[10] courses/infs1200/sql-ddl.pdf