The Relational Model

Table of contents

The four basic concepts 1
Relation schema vs relation instance 3
Properties of relations 4
Where the relational model sits 4

The data model underlying almost every commercial DBMS, introduced by E.F. Codd in 1970: a sound theoretical foundation built on set theory, with a single simple, uniform data structure called a **relation**. Introduced in 2026-03-09-the-relational-model-and-integrity-constraints[1]; see relational-mapping-notation[2] for how schemas are written down in this course, and er-to-relational-mapping[3] for how a conceptual ER design becomes one.

The four basic concepts

Concept	Definition
Domain	A set of atomic values, with a data type/format
Attribute	The name of a role played by some domain in a relation
Tuple	An ordered list of values, one per attribute
Relation	A set of tuples — informally, a table of rows and columns

All four are visible at once in the anatomy figure, which is worth reading before the definitions below:

[1] courses/infs1200/2026-03-09-the-relational-model-and-integrity-constraints.pdf
[2] courses/infs1200/relational-mapping-notation.pdf
[3] courses/infs1200/er-to-relational-mapping.pdf

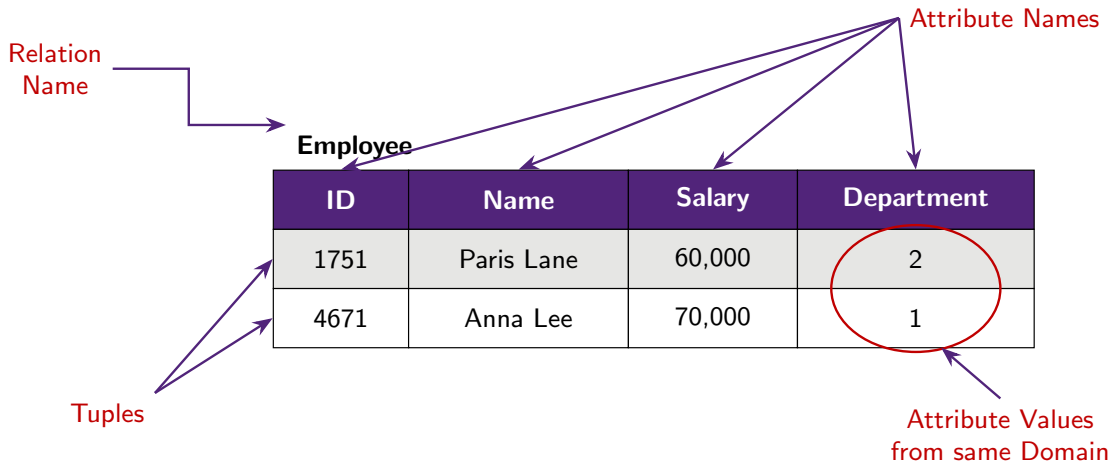


Figure 1: The four components of a relation, on an annotated `Employee` table. The relation *name* sits above the table; the *attribute names* are the header row; each body row is a *tuple*; and every value down one column — here `Department` — is drawn from that attribute's single *domain*.

Domains

A domain D is a set of **atomic** values — atomic meaning indivisible as far as the relational model is concerned. Each domain carries a data type or format: integers, numbers/currency, fixed- or variable-length character strings, dates/timestamps, a sub-range of a data type (e.g. $1 \leq \text{grade} \leq 7$), or an enumerated type (e.g. $\text{Gender} \in \{\text{Male}, \text{Female}, \text{Other}\}$). Format-constrained domains count too — an Australian phone number (`61` + 9 digits), or a car registration (6 alphanumeric characters, no `Q`).

The domain is what makes a value legal or illegal, which is why the **domain constraint** is the first of the five integrity-constraints[4]: a value that isn't a member of its attribute's domain simply cannot be stored.

Attributes

Each attribute A names a role that some domain plays in a relation R. Name and domain are independent in *both* directions:

- Same-named attributes in different relations need not share a domain — `Department.id` and `Employee.id` can be drawn from entirely different domains despite the shared name `id`.

[4] courses/infs1200/integrity-constraints.pdf

- Differently-named attributes *can* share a domain — `Employee.id` and `Employee.managerId` are both drawn from the same "employee id" domain, which is precisely what makes a self-referencing foreign key possible (see foreign-keys[5]).

The number of attributes in R is its **degree** — `Employee [id, name, sex, salary, department]` is of degree 5, and "degree of the relation" and "degree of the relation schema" are used interchangeably for the n in `R [A1, …, An]`. Degree is the only size measure Module 2 names. Unqualified "cardinality" in this course almost always means a relationship's cardinality ratio on an ER diagram (relationship-constraints[6]) or the "many cardinality" side of a relationship in the mapping rules; a relation's *row* count only acquires a notation later, when Module 3 writes it as $|R|$ to size a Cartesian product ($|R_1 \times R_2| = m \cdot n$) — see infs1200-diagram-notation[7].

Tuples

Each tuple t is an ordered list of n values $t = \langle v_1, v_2, \ldots, v_n \rangle$, where each v_i either belongs to the domain of the corresponding attribute or is the special value `NULL`. A tuple with n values is an n-tuple.

Relation schema vs relation instance

The single most-tested distinction in this part of the course — the schema is *metadata*, the instance is *data*.

	Relation schema	Relation instance
Written	`R [A1, A2, …, An]`	`r(R) = {t1, t2, …, tm}`
Is	The relation's name and attribute list	The actual set of tuples, at one point in time
Also called	Metadata — "data describing the data"	The data in the database
Comes from	Database design	Insert/update/delete operations
Changes	Rarely (a design decision)	Constantly

`Employee [id, name, sex, salary, department]` is a *schema* of degree 5; the rows currently sitting in that table are its *instance*. Because the schema is fixed at design time and the instance changes with every update, an integrity constraint is a property of the *schema* that must hold on **every** instance of it.

[5] courses/infs1200/foreign-keys.pdf
[6] courses/infs1200/relationship-constraints.pdf
[7] courses/infs1200/infs1200-diagram-notation.pdf

Properties of relations

A relation is a mathematical set of tuples, and the consequences of that are what separate a relation from an arbitrary table.

- **Tuple order is irrelevant.** A set has no implied order. Two instances holding the same tuples in a different sequence are the *same* relation. Physically, tuples do sit in some order on blocks of secondary storage, but nothing semantic (queries included) may depend on it.
- **Value order is a syntactic convention only.** An n-tuple is syntactically an ordered list, so every tuple in one relation must list its values in the same attribute order — but *which* order is chosen carries no meaning, as long as the attribute-to-value correspondence is maintained consistently.
- **Values are atomic.** Each cell holds one indivisible value from its domain (or `NULL`) — no repeating groups and no nested structure, which is why a multivalued attribute has to become a relation of its own during mapping.
- **Every relation is a table; not every table is a relation.** A "pivoted" presentation table — a quarterly sales report with merged header cells, broken down by region and suburb — is a perfectly good table and not a valid relation at all. The word *table* is used interchangeably with *relation* in practice, but only relations obey the properties above.

One caveat for later: SQL does **not** enforce the set property. A `SELECT` returns a *bag*, duplicates and all, unless you ask for `DISTINCT` — see sql-select-statement[8]. The properties above describe the model, not what a query result happens to look like.

Where the relational model sits

The relational model is the **logical** layer: what a DBMS actually implements, and the target of the mapping algorithm in er-to-relational-mapping[9] (which is also where the conceptual/logical split is spelled out). The relational model is the middle box below: still a description of *what* is stored, not of how the bytes sit on disk (dbms-fundamentals[10] has the three-schema architecture the right-hand column belongs to).

What that algorithm emits is relations plus their primary[11] and foreign key[12] constraints — from there, sql-ddl[13] turns the schema into real `CREATE TABLE` statements, and normal-

[8] courses/infs1200/sql-select-statement.pdf
[9] courses/infs1200/er-to-relational-mapping.pdf
[10] courses/infs1200/dbms-fundamentals.pdf
[11] courses/infs1200/relational-keys.pdf
[12] courses/infs1200/foreign-keys.pdf
[13] courses/infs1200/sql-ddl.pdf

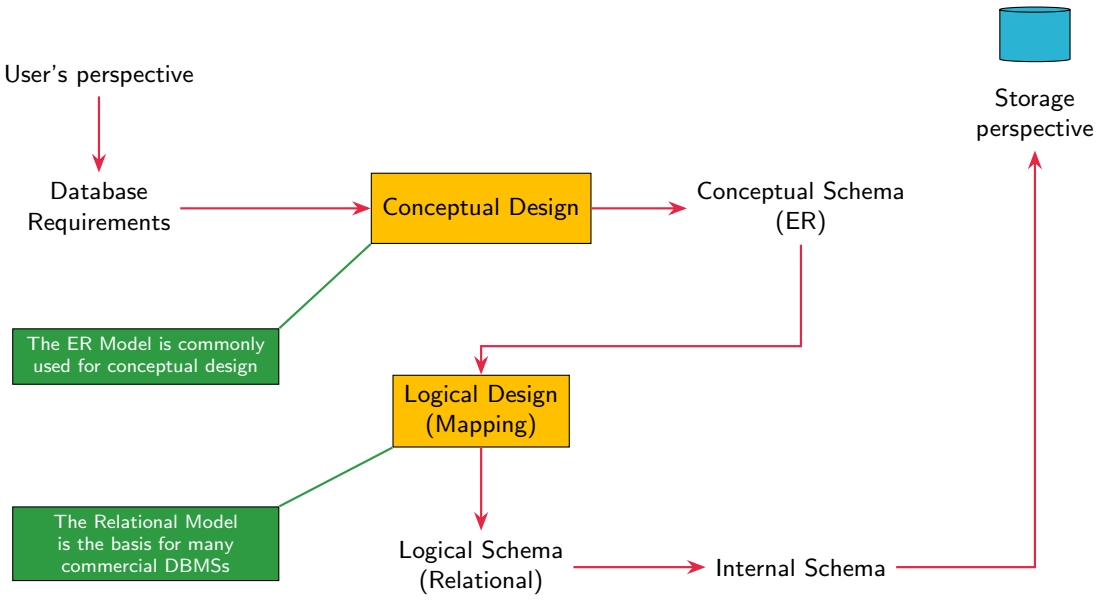


Figure 2: Where the relational model sits. Two design steps (gold) turn requirements into two schemas: conceptual design produces an ER schema, logical design — the mapping — produces a relational one. The green notes say what each model is for: the ER model is what conceptual design is done in, the relational model is what commercial DBMSs implement.

isation (functional-dependencies[14], normal-forms[15]) checks that the resulting relations are well-designed.

[14] courses/infs1200/functional-dependencies.pdf
[15] courses/infs1200/normal-forms.pdf