# The Relational Model and Integrity Constraints

## Table of contents


Today's outline . . . 1
Relational model concepts . . . 1
Question — Schema and Instance . . . 3
Integrity constraints . . . 3
Question — Key . . . 4
Question — Key (multiple candidate keys) . . . 4
Question — Integrity Constraints (password) . . . 5
Question — Integrity Constraints (which is violated?) . . . 6
Constraints and operations . . . 6
The transaction concept . . . 7


Module 2, part 1. See relational-mapping-notation[1] for the schema notation this lecture (and 2026-03-16-er-to-relational-mapping[2]) uses.

### Today's outline

- Relational model concepts — relations, attributes, domains, tuples
- Integrity constraints
- The transaction concept

### Relational model concepts

Introduced by E.F. Codd in 1970. Many DBMS products are based on this model — a sound theoretical foundation with a simple, uniform data structure called a **relation**. Four basic concepts: **relations**, **attributes**, **domains**, **tuples**.

[1] courses/infs1200/relational-mapping-notation.pdf
[2] courses/infs1200/2026-03-16-er-to-relational-mapping.pdf

### Relations

A relation is the main construct for representing data — informally, a set of records, similar to a table with columns and rows. The term *table* is used interchangeably with *relation*, but **every relation is a table; not every table is a relation** — relations have specific properties based on mathematical set theory (e.g. a "pivoted" table with merged header cells, like a quarterly sales report broken down by region/suburb, is a table but not a valid relation).

### Domains

A **domain** `D` is a set of atomic values — an atomic value is indivisible as far as the relational model is concerned. Each domain has a data type/format: integers, numbers/currency, fixed/variable-length character strings, date/timestamp, a sub-range of a data type (e.g. `1 ≤ grade ≤ 7`), or an enumerated type (e.g. `Gender ∈ {Male, Female, Other}`) — including format-constrained domains like Australian phone numbers (`61` + 9 digits) or car registrations (6 alphanumeric characters, no `Q`).

### Attributes

Each **attribute** `A` is the name of a role played by some domain `D` in a relation `R`. The number of attributes in `R` is its **degree**. Same-named attributes across relations don't necessarily share a domain (`Department.id` and `Employee.id` can be different domains despite the shared name `id`), and differently-named attributes can share a domain (`Employee.id` and `Employee.managerId` can both be drawn from the same "employee id" domain, despite the different names).

### Tuples

Each **tuple** `t` is an ordered list of $n$ values `t = <v1, v2, …, vn>`, where each value belongs to the corresponding attribute's domain, or is the special value `NULL`. `t` is called an $n$-tuple.

### Relation schema and instance

- **Relation schema**: `R [A1, A2, …, An]` — a relation name `R` and its list of attributes; $n$ is the **degree of the relation**. E.g. `Employee [id, name, sex, salary, department]` is a schema of degree 5.
- **Relation instance**: `r(R)`, a set of $n$-tuples `r = {t1, t2, …, tm}` — the actual data, at a point in time.

## Question — Schema and Instance

*Match each characteristic to Schema or Instance: "data in the database", "specified during database design", "data describing the data", "created through data update operations".*

> **i** Solution
>
> **Instance**: "data in the database", "created through data update operations" (inserts/updates/deletes change the instance, not the schema). **Schema**: "specified during database design", "data describing the data" (the schema is metadata — it describes the shape data must take, not the data itself).

### Ordering of tuples and values

Relations are *sets* of tuples — mathematically, a set has no implied order. Semantically (e.g. when writing queries), tuple order is irrelevant. Physically, tuples reside on blocks of secondary storage with some ordering, but two instances with the same tuples in a different order are still the *same* relation. Likewise, an $n$-tuple is syntactically an ordered list — every tuple in one relation must list values in the same attribute order — but semantically, which order is chosen doesn't matter, as long as the attribute/value correspondence is maintained consistently.

## Integrity constraints

Integrity constraints are rules that enforce the *correctness* of a database — they must hold on every instance of the schema. Five kinds: **domain**, **key**, **entity integrity**, **referential integrity**, **semantic**.

### Domain constraint

A domain constraint violation occurs when an attribute's value doesn't appear in its corresponding domain — e.g. `Employee.id = "LOL"` when `id` is meant to be a 4-digit integer.

### Key

A **key** is a minimal set of attributes that uniquely identifies tuples in a relation — *minimal* meaning no redundant attributes, not necessarily the smallest possible set. A schema can have more than one key (each a **candidate key**); the one chosen as the relation's main key is the **primary key** (conventionally underlined). A **key constraint** violation occurs when a tuple is inserted/modified to share a key value with an existing tuple.

### Question — Key

*Assuming department IDs are unique, which of the following is a key for* ***`Department [id, name, manager]`****: (A)* ***`(id)`****, (B)* ***`(id, name)`****, (C)* ***`(id, manager)`****, (D) all of the above?*

> **i** Solution
>
> **(A)** `(id)` — since IDs are already unique on their own, `(id)` is the *minimal* key; `(id, name)` and `(id, manager)` both contain a redundant extra attribute (adding anything to an already-unique set isn't minimal).

### Question — Key (multiple candidate keys)

*Assuming department IDs are unique* ***and*** *the combination of Name and Manager is also unique per department, which of the following is a key(s): (A)* ***`(id)`****, (B)* ***`(id, name)`****, (C)* ***`(name, manager)`****, (D) both A and C?*

> **i** Solution
>
> **(D) Both A and C.** `(id)` is still minimal and unique on its own; now `(name, manager)` is *also* minimal and unique, making it a second candidate key. `(id, name)` is still not minimal since `(id)` alone already suffices.

#### Entity integrity constraint

An entity integrity constraint violation occurs when a tuple is inserted/modified such that (any part of) its primary key is `NULL` — for a composite primary key, *no part* of it can be null.

#### Foreign keys

A **foreign key** is a set of attributes in one relation that links it to another relation's primary key. Formally: let `FK` be attributes in `R1` and `PK` the primary key of `R2`. `FK` in `R1` is a foreign key referencing `PK` in `R2` if `FK`/`PK` share a domain, and for every tuple `t1` in `R1`, either `t1[FK]` is `NULL`, or some tuple `t2` in `R2` has `t1[FK] = t2[PK]`.

```
Department [id, name, manager]
Department.manager references Employee.id

Employee [id, name, sex, salary, department]
Employee.department references Department.id
```

**Self-referencing relations** are also possible — a table can reference itself, e.g. `Employee.managerId references Employee.id`. **Composite foreign keys** are possible too, referencing a multi-attribute primary key:

```
Student [sid, name]

Course [cid, department]

Enrolment [sid, cid, department, grade]
Enrolment.sid references Student.sid
Enrolment.{cid, department} references Course.{cid, department}
```

### Referential integrity constraint

A referential integrity constraint violation occurs when a foreign key value doesn't match any existing primary key value in the referenced relation (and isn't `NULL`) — e.g. inserting an `Employee` tuple with `department = 5` when no `Department` with `id = 5` exists.

### Semantic (business-rule) constraint

Semantic constraints are generally defined by the business/organisation (not derivable from the schema's structure alone) — e.g. "an employee's salary must not exceed their supervisor's", or "an employee can work at most 56 hours across all projects". Often implemented via a constraint specification language (SQL triggers/assertions).

## Question — Integrity Constraints (password)

*Opening an online bank account, you enter your usual password "password123", but get: "your password must contain at least one capital letter and a number". What kind of constraint is this?*

> **i** Solution
>
> **Domain constraint** — the domain of "password" is defined as strings matching a particular format ( 1 capital,  1 digit); "password123" simply isn't a member of that domain.

## Question — Integrity Constraints (which is violated?)

*Given* `Department [id, name, manager]` *and* `Employee [id, name, salary, department]`*, with* `Department` *rows* `(1, Marketing, 4671)`*,* `(2, Development, 1751)`*, and* `Employee` *rows* `(1751, Paris Lane, 60000, 2)`*,* `(4671, Anna Lee, 70000, 1)`*,* `(2670, Grace Mills, 50000, 2)`*,* `(2034, Jack Smith, 40000, 1)` *— which constraint is violated by each of the following?*

1. *Inserting* `(2670, James Smith, 40000, 1)` *into* `Employee`*.*
2. *Inserting* `(2644, James, Smith, 1)` *into* `Employee` (note: only 4 values given for a 4-attribute relation, but shifted).
3. *Inserting* `(2644, James Smith, 40000, 3)` *into* `Employee`*.*
4. *Deleting* `(2, Development, 1751)` *from* `Department`*.*
5. *Updating* `(2, Development, 1751)` *to* `(2, Development, 2034)` *in* `Department`*.*

**i** Solution

1. **Key constraint** — `id = 2670` already exists in `Employee`.
2. **Domain constraint** — the shift means `salary` receives a non-numeric value (`"Smith"`), which isn't a member of `salary`'s domain.
3. **Referential integrity constraint** — no `Department` with `id = 3` exists.
4. **Referential integrity constraint** — `Employee` rows with `department = 2` still exist (Grace Mills), so deleting `Department 2` would leave a dangling foreign key.
5. **None of the above (by the *stated* constraints)** — `Employee` with `id = 2034` doesn't actually work in `Department 1`, but there's no *declared* constraint stopping a department's manager from being an employee of a different department; this would only be a semantic constraint if the business rule "a manager must work in the department they manage" were explicitly specified.

## Constraints and operations

Enforcing integrity constraints keeps the database consistent — insert, modify, and delete operations must not leave the database in an inconsistent state; a DBMS should reject any update that would violate integrity.

- **Insertion/modification** can violate any of the five constraint types (domain, key, entity integrity, referential integrity, semantic).
- **Deletion** can only violate **referential** or **semantic** constraints. A referential integrity violation on delete can be handled by rejecting the delete, cascading it (deleting dependent rows too), or setting the referencing value to a default/`NULL`.

## The transaction concept

A **transaction** is an executing program that includes database operations (reads, inserts, deletes, updates). At the end of a transaction, the database must be left in a valid/consistent state satisfying all constraints — but constraint violations are allowed at *intermediate* steps within the transaction.

Example: given `Department [id, name, manager]` and `Employee [id, name, sex, salary, department]`, with the business rules "every department must have 1 employee" and "every employee must work for a department" — neither a new department (no employees yet) nor a new employee (no department yet) can be inserted alone without violating one of these rules. A **transaction** that inserts both the new department and its first employee together resolves this: the intermediate state (after just one insert) is inconsistent, but the final state (after both inserts) is valid.