Integrity Constraints

Table of contents

The five kinds 1
Domain constraint 3
Key constraint 3
Entity integrity constraint 3
Referential integrity constraint 4
Semantic (business-rule) constraint 4
Naming an example of each 5
Diagnosing a violation 5

Integrity constraints are the rules that enforce the *correctness* of a database: conditions stated on the schema that must hold on **every** instance of it. A DBMS's job is to reject any operation that would leave them unsatisfied. Introduced in 2026-03-09-the-relational-model-and-integrity-constraints[1] and practised in week4-tutorial-applied-class-3-relational-integrity[2]; these are constraints on a *relational* schema, distinct from the relationship-constraints[3] placed on an ER diagram.

The five kinds

Constraint	Violated when	Enforced by
Domain	An attribute's value isn't a member of its domain	The column's declared type/format
Key	Two tuples share the same value for a key	The declared key
Entity integrity	Any part of a tuple's primary key is `NULL`	The declared primary key

[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/relationship-constraints.pdf

Constraint	Violated when	Enforced by
Referential integrity	A non-null foreign key value matches no primary key value in the referenced relation	The declared foreign key
Semantic	A business rule that isn't derivable from the schema's structure is broken	Application-level rules — triggers, assertions, `CHECK`

The first four are **structural**: they follow from the schema itself, so the DBMS enforces them for free once the types and keys are declared. The fifth is not — nothing in the schema's shape implies it, so it has to be stated separately.

Enforcement is unconditional and the default outcome is refusal: any insert, modification or deletion submitted to the DBMS that would violate integrity — that would leave the database in an inconsistent state — **must be rejected**. The only constraint whose violation has other sanctioned outcomes is referential integrity on a delete, and only because the foreign key was declared with a policy saying so (see update-operations-and-constraint-violations[4]).

Which operation can break which constraint is not symmetric, and the asymmetry is examinable:

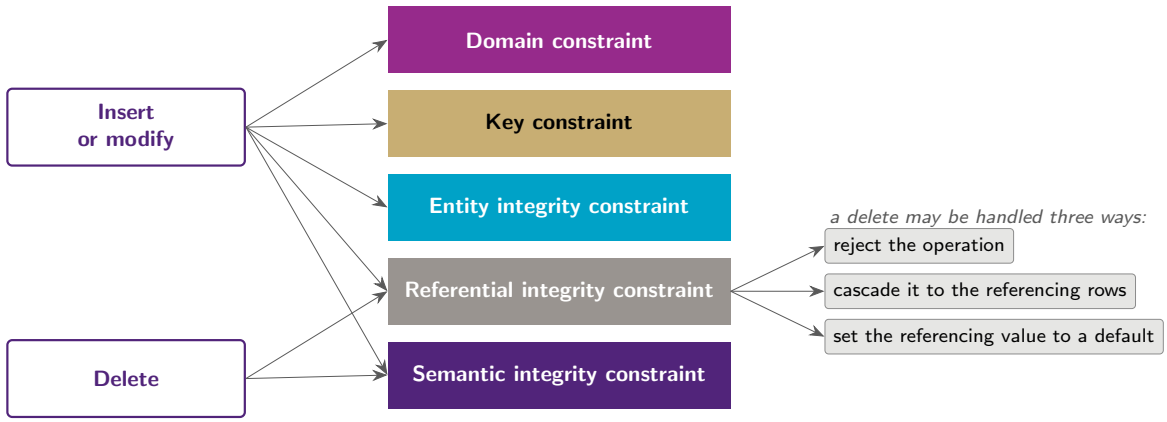


Figure 1: Insert and modify can violate all five constraints; delete can only violate the last two, and a referential violation on delete has three sanctioned outcomes rather than one. The banner colours are the course's fixed fills for the five constraint types.

A delete cannot break a domain, key or entity-integrity constraint because it adds no value and duplicates no key — it can only strand something that was pointing at the row it removed, or break a business rule about what must continue to exist.

[4] courses/infs1200/update-operations-and-constraint-violations.pdf

Domain constraint

The domain is the set of legal atomic values for an attribute (see relational-model[5]), so a domain violation is simply a value that isn't in that set. `Employee.id = "LOL"` violates it when `id` is a four-digit integer.

The domain need not be a plain data type — a format counts. A bank rejecting the password `password123` with "your password must contain at least one capital letter and a number" is enforcing a **domain** constraint: the domain of `password` is defined as the strings matching that format, and `password123` isn't a member of it. Likewise a course code declared as four letters followed by four digits, or an enumerated domain: inserting `VehiclePurpose = 'Personal'` when the column is `ENUM('Private', 'Business')` is a domain violation, not a semantic one.

The commonest exam version of a domain violation is subtler — a value of the wrong *kind* in the right position. Inserting `(2644, James, Smith, 1)` into `Employee [id, name, salary, department]` breaks the domain constraint because the third value lands in `salary`, whose domain is integer or float, not a surname.

Key constraint

No two tuples may share a value for a key. Violated by inserting a tuple whose key duplicates an existing one, or by modifying an existing tuple into a duplicate. See relational-keys[6] for what qualifies as a key in the first place.

Entity integrity constraint

No part of a primary key may be `NULL` — for a composite primary key, *no component* may be null, not merely "not all of them". Inserting `(677678989, null, 40.0)` into `WorksOn [essn, pNo, hours]` violates entity integrity because `pNo` is half of the primary key.

The rationale is that the primary key is the relation's identity: a tuple with an unknown key value cannot be distinguished from, or referred to by, anything. This is the one place a `NULL` is categorically banned — sql-nulls[7] has the table of which declarations do and don't permit one.

[5] courses/infs1200/relational-model.pdf
[6] courses/infs1200/relational-keys.pdf
[7] courses/infs1200/sql-nulls.pdf

Referential integrity constraint

A non-null foreign key value must match some existing primary key value in the referenced relation. Inserting an `Employee` with `department = 5` when no `Department` has `id = 5` violates it; so does deleting `Department 5` while employees still reference it, and so does updating a foreign key to a value that doesn't exist. `NULL` is always allowed — see foreign-keys[8] for the formal statement.

Semantic (business-rule) constraint

Semantic constraints are **defined by the business or organisation during client consultation** — they come from the organisation, not from the data structure:

- "An employee's salary must not exceed their supervisor's."
- "The maximum number of hours an employee can work on a project is 56."
- "A department cannot have more than three active projects at a time."
- "A customer must be at least 18 years old."

Take the first and third. Inserting Grace Chan on 58000 with `superSSN = 888665555` — James Borg, on 55000 — violates the salary rule and nothing else; inserting a *fourth* project for department 5, which already has `ProductX`, `ProductY` and `ProductZ`, violates the three-project rule and nothing else. In both cases every structural check passes, which is exactly the signal that the answer is "semantic".

Nothing about the relations' shape rules these out, so they're expressed in a **constraint specification language** — in SQL's case, the mechanisms are `CHECK` clauses, **assertions** and **triggers**. This course files the latter two under DCL, alongside transaction control and privilege management. Only `CHECK` is developed and examined (see sql-ddl[9] for the `CHECK` and named-constraint syntax); assertions and triggers are the general-purpose fallback and are not defined further, so "enforced by an assertion or a trigger" is as far as the course goes. This is also the category to fall back on when asking "which constraint does this break?" and none of the structural four apply.

The converse matters just as much: a rule that was never *stated* isn't a constraint. Updating `Department (2, Development, 1751)` to `(2, Development, 2034)` where employee `2034` works in a different department violates nothing, unless the business rule "a manager must work in the department they manage" has actually been specified.

[8] courses/infs1200/foreign-keys.pdf
[9] courses/infs1200/sql-ddl.pdf

Naming an example of each

The other exam shape is the reverse: given a schema, point at one instance of each kind of constraint. For the enrolment schema

```
Student [stID, name, email]

Enrolment [stID, cCode, sem, year]
Enrolment.stID references Student.stID
Enrolment.cCode references Course.cCode

Course [cCode, title, units]
```

a **key** is `stID` in `Student`, a **foreign key** is `stID` in `Enrolment`, and a **domain constraint** is `cCode` being four letters followed by four digits. `stID` supplies both of the first two answers depending on which relation it is read in — the same attribute name is a key where it identifies and a foreign key where it refers.

Diagnosing a violation

Given an operation and a schema, work down the five in a fixed order. The order isn't significant in itself; checking all five every time is, since one operation can break several at once.

1. **Domain** — is every value of the right type/format for its column?
2. **Key** — does this key value already exist?
3. **Entity integrity** — is any part of the primary key null?
4. **Referential integrity** — does every non-null foreign key point at a row that exists? And, for a delete, does anything still point *at* the row being removed?
5. **Semantic** — does it break a stated business rule?

A single operation can violate more than one at once. Inserting `('ProductA', 'A', 'Bellaire', 2)` into `Project [pName, pNumber, pLocation, dNum]` breaks **both** the domain constraint (`pNumber` is an integer, not `'A'`) and referential integrity (no department `2`). Inserting `('Production', 4, 943775543, …)` into `Department` breaks the key constraint (`dNumber = 4` exists) *and* referential integrity (`mgrSSN` isn't a real employee). Don't stop at the first one found.

And the operations that violate nothing

Three of the applied class's ten answers are "No", and they are the ones worth rehearsing, because each looks like a violation and isn't:

Operation on the company schema	Why it is legal
Insert `('Robert', 'F', 'Scott', 943775543, '1942-06-21', '2365 Newcastle Rd, Bellaire, TX', 'M', 50000, 888665555, 1)` into `EMPLOYEE`	Fresh `ssn`, every value in its domain, `superSSN` 888665555 and `dNo` 1 both exist, and 50000 is under the supervisor's 55000
Delete the `WORKS_ON` tuples with `essn = 333445555`	Nothing references `WORKS_ON`, so no reference is stranded
Modify `DEPARTMENT`'s `mgrSSN`/`mgrStartDate` for `dNumber = 5` to 123456789 and `'1988-10-01'`	123456789 is a real employee (John Smith), so the foreign key still resolves, and no other constraint touches the manager column

Compare that last row with the case where the new manager works in a *different* department: it still violates nothing. A manager who works elsewhere offends intuition, not any declared constraint.

See update-operations-and-constraint-violations[10] for which constraints each kind of operation can possibly violate, and what a DBMS does about it; sql-dml[11] shows the same violations as the error messages MySQL actually returns.

[10] courses/infs1200/update-operations-and-constraint-violations.pdf
[11] courses/infs1200/sql-dml.pdf