NULLs in SQL

Table of contents

Which constraints permit a null 1
Testing for a null 2
The limits of what the course specifies 3
Where nulls come from 3
Nulls as a design smell 4

`NULL` is the special value a relational attribute takes when there is no ordinary value to store — it represents an **inapplicable, unknown or missing** value rather than zero or the empty string. Because a null is not a value in the attribute's domain, it needs its own handling everywhere it appears: in constraints, in `WHERE` conditions, and in the results a query produces. Defined in 2026-02-23-conceptual-database-design-and-the-er-model[1], constrained in 2026-03-09-the-relational-model-and-integrity-constraints[2], and handled in SQL from 2026-03-30-basic-sql-ddl-and-dml[3] onwards.

Which constraints permit a null

Declaration	Nulls allowed?
`PRIMARY KEY`	**No** — a primary key value is unique *and* not null; for a composite key, no part of it may be null (entity integrity)
`FOREIGN KEY`	**Yes** — a foreign key value must either be null, or match an existing row of the referenced relation (referential integrity)
`UNIQUE`	**Yes** — the value must be unique *or* null; this is the one difference from a primary key

[1] courses/infs1200/2026-02-23-conceptual-database-design-and-the-er-model.pdf
[2] courses/infs1200/2026-03-09-the-relational-model-and-integrity-constraints.pdf
[3] courses/infs1200/2026-03-30-basic-sql-ddl-and-dml.pdf

Declaration	Nulls allowed?
`NOT NULL`	**No** — a value must be supplied

See integrity-constraints[4] for the underlying entity and referential integrity rules, relational-keys[5] and foreign-keys[6] for what the keys themselves are, and sql-ddl[7] for how each is declared.

The nullable foreign key is what makes several mappings work at all: an employee with no manager, or a department with no manager assigned yet, is represented by a null foreign key rather than by a missing row. It is also why er-to-relational-mapping[8] prefers to hang a 1:1 foreign key off the *total*-participation side — the other side would be null for almost every row.

Testing for a null

`NULL` is tested with `IS`, never with `=`. `IS` sits alongside `LIKE`, `IN` and `BETWEEN` in the course's catalogue of things a `WHERE` search condition may contain (sql-select-statement[9]):

```
SELECT * FROM Employee WHERE dNum IS NULL;
SELECT * FROM Employee WHERE dNum IS NOT NULL;
```

This shows up inside subqueries as well as at the top level — here, restricting the result to employees whose department currently has a manager assigned:

```
SELECT name, dNum
FROM   Employee e
WHERE  EXISTS (
    SELECT * FROM Department d
    WHERE  d.dNumber = e.dNum AND d.mgrSSN IS NOT NULL
);
```

An explicit `IS NOT NULL` is also the usual guard before grouping or building a derived value out of several columns, so that rows with missing components are excluded rather than silently contributing:

[4] courses/infs1200/integrity-constraints.pdf
[5] courses/infs1200/relational-keys.pdf
[6] courses/infs1200/foreign-keys.pdf
[7] courses/infs1200/sql-ddl.pdf
[8] courses/infs1200/er-to-relational-mapping.pdf
[9] courses/infs1200/sql-select-statement.pdf

```sql
-- Last names shared by more than one employee, ignoring unknown names
SELECT   DISTINCT LastName FROM Employee
WHERE    LastName IS NOT NULL
GROUP BY LastName
HAVING   COUNT(FirstName) > 1;

-- Build FirstName-LastName-yyyy only where all three parts exist
SELECT CONCAT(FirstName, '-', LastName, '-', YEAR(DOB))
FROM   Employee
WHERE  FirstName IS NOT NULL
  AND  LastName  IS NOT NULL
  AND  DOB       IS NOT NULL;
```

The limits of what the course specifies

Module 3 treats nulls **operationally** rather than logically. It gives the `IS` operator, the constraint rules above, and the places a null can come from — and it stops there. The lectures do not develop a three-valued logic of `TRUE/FALSE/UNKNOWN`, do not state how an aggregate function treats the nulls inside a group, and do not cover what happens when a `NOT IN` subquery returns a null among its values. Those questions are simply not settled by the material, so don't reason from a rule the course never gave you.

What the course *does* teach is the defensive habit: write the `IS NOT NULL` guard explicitly wherever a null could reach a comparison, a grouping, or a derived value, rather than relying on the DBMS to do something sensible with it. Every example above follows that habit.

Where nulls come from

Not every null is data someone failed to enter — several are created by the schema or the query itself:

Source	Why
A genuinely inapplicable/unknown/missing value	The original meaning of a null
`ALTER TABLE … ADD <column>`	Every existing row gets a null in the new column — which is exactly why the new column cannot be declared `NOT NULL` (sql-ddl[10])

[10] courses/infs1200/sql-ddl.pdf

Source	Why
`ON DELETE SET NULL`	The referential action deliberately clears the link instead of deleting the dependent rows (update-operations-and-constraint-violations[11])
An **outer join**	Unmatched rows are padded with nulls in the columns of the table that had no match (sql-joins[12])
Access-control masking	A cell above a user's clearance is presented to them as null rather than withheld (database-access-control[13])

The last two matter when reading results: a null in the output of a `LEFT JOIN` means "no matching row existed", which is information about the *relationship*, not about the row's stored data.

```
-- Departments with no manager still appear, with NULL for the name
SELECT    D.dName, E.name
FROM      Department AS D
LEFT JOIN Employee AS E ON D.mgrSSN = E.ssn;
```

Nulls as a design smell

A schema that produces many nulls is usually telling you something. Reducing the number of null values in tuples is guideline 3 of the informal design guidelines behind normalisation (update-anomalies[14]) — an attribute that is null for most rows of a relation often belongs in a separate relation instead. See 2026-04-27-database-design-guidelines-and-functional-dependencies[15] and normal-forms[16].

[11] courses/infs1200/update-operations-and-constraint-violations.pdf
[12] courses/infs1200/sql-joins.pdf
[13] courses/infs1200/database-access-control.pdf
[14] courses/infs1200/update-anomalies.pdf
[15] courses/infs1200/2026-04-27-database-design-guidelines-and-functional-dependencies.pdf
[16] courses/infs1200/normal-forms.pdf