SQL Set Operations

Table of contents

The three operators 1
Union compatibility 2
Duplicates: set vs multiset semantics 2
Algebraic properties 3
Join or set operation? 3
Dialect support, and the two rewrites 4

A relation in the relational-model[1] is a *set* of tuples — no duplicates, no order — so the ordinary set operators apply directly to whole query results. These are the one corner of SQL that keeps that set discipline by default, where plain `SELECT` returns a bag (sql-select-statement[2]). Where a join (sql-joins[3]) combines tables into wider **rows**, a set operation stacks two result sets into more **rows** of the same shape. Introduced in 2026-04-06-aggregation-grouping-and-multiple-relation-queries[4]; practised in week8-tutorial-applied-class-7-multiple-relation-queries[5].

The three operators

Operator	Relational algebra	Result
`UNION`	$\cup$	Tuples appearing only in `R1`, only in `R2`, or in both
`INTERSECT`	$\cap$	Tuples appearing in both `R1` and `R2`
`EXCEPT` (a.k.a. `MINUS`)	$-$	Tuples appearing in `R1` but not in `R2`

[1] courses/infs1200/relational-model.pdf
[2] courses/infs1200/sql-select-statement.pdf
[3] courses/infs1200/sql-joins.pdf
[4] courses/infs1200/2026-04-06-aggregation-grouping-and-multiple-relation-queries.pdf
[5] courses/infs1200/week8-tutorial-applied-class-7-multiple-relation-queries.pdf

```
SELECT ...
UNION [ALL] SELECT ...
[UNION [ALL] SELECT ...]
```

Union compatibility

Both operands must be **union compatible**: the same number of columns, with pair-wise compatible domains. The result takes its column names from the first `SELECT`, so the second query's names are irrelevant — only its column *order* matters.

```
-- Error: 3 columns UNION 2 columns
SELECT * FROM Table1
UNION
SELECT a, b FROM Table1 WHERE b = 'X';
```

A mismatch is an error, not a silently odd result — which makes it one of the easier SQL mistakes to catch.

Duplicates: set vs multiset semantics

Each operator eliminates duplicates by default. Appending `ALL` keeps multiset semantics instead. If a tuple occurs m times in r and n times in s, it occurs:

Expression	Occurrences
`r UNION ALL s`	$m + n$
`r INTERSECT ALL s`	$\min(m, n)$
`r EXCEPT ALL s`	$\max(0,\ m - n)$

Deduplication is often the *reason* to reach for `UNION`: three independent ways of qualifying as a co-founder, unioned together, yield each person once even if they qualify twice.

```
SELECT E.EmployeeID, E.FirstName, E.LastName
FROM   Employee E
JOIN   AdministrativeEmployee AE ON AE.EmployeeID = E.EmployeeID
WHERE  AE.Type = 'LegacyEngineer'
UNION
SELECT E.EmployeeID, E.FirstName, E.LastName
FROM   Employee E
JOIN   RoleGranting RG ON RG.EmployeeID = E.EmployeeID
JOIN   Role R ON R.RoleID = RG.RoleID
WHERE  R.Name = 'FinancialPerformanceOverview';
```

Algebraic properties

$$A \cup B = B \cup A \qquad (A \cup B) \cup C = A \cup (B \cup C)$$
$$A \cap B = B \cap A \qquad (A \cap B) \cap C = A \cap (B \cap C)$$
$$A - B \neq B - A \qquad (A - B) - C \neq A - (B - C)$$

Union and intersection are commutative and associative; difference is **neither** — so with `EXCEPT` the order of the operands, and the bracketing of a chain of them, is part of the meaning.

Join or set operation?

	Join	Set operation
Combines	Rows from many tables into new **columns**	Result sets of `SELECT`s into new **rows**
Column count	May differ between tables	Must match
Column types	Can differ	Must be compatible
Duplicates	Kept by default	Removed by default

A set operation earns its place when the two halves need *different* grouping or filtering that cannot be expressed in one pass — for example "granted at least 2 roles" **and** "can edit at least 4 distinct websites", which are two different `COUNT`s over two different groupings:

```
SELECT   EmployeeID
FROM     RoleGranting
GROUP BY EmployeeID
HAVING   COUNT(RoleID) >= 2
INTERSECT
SELECT   EmployeeID
FROM     RoleGranting RG
JOIN     Permission P ON P.RoleID = RG.RoleID
WHERE    P.GrantType = 'Edit'
GROUP BY EmployeeID
HAVING   COUNT(DISTINCT WebsiteURI) >= 4;
```

Conversely, a simple `UNION` of two queries differing only in a `WHERE` condition is usually just an `OR` written the long way:

```sql
-- These two are equivalent
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID
WHERE  year = 1944 OR year = 1974;

SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1944
UNION
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1974;
```

Note that `OR` and `UNION` coincide here only because the condition is about a *single* row's year. "Stars who were in a 1944 movie **and** a 1974 movie" cannot be an `AND` over one row — no single row has two years — which is exactly the case `INTERSECT` handles.

Dialect support, and the two rewrites

The course is precise about what is missing where, and the two claims are different in scope:

Feature	What the course says
`INTERSECT`, `EXCEPT/MINUS`	**Some SQL dialects do not support** intersection and difference — a portability caveat about dialects generally, not about MySQL specifically
`FULL OUTER JOIN`	**Not implemented in MySQL** (sql-joins[6])

`UNION` is the only one of the three set operators you can rely on everywhere. For the other two the course supplies a rewrite each, and they are different rewrites: an `INTERSECT` becomes a **join**, an `EXCEPT` becomes a **nested query**.

`INTERSECT` becomes a self-join that requires both conditions to hold for the same key:

```sql
-- Stars in a movie from BOTH 1944 and 1974
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1944
INTERSECT
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1974;

-- Rewritten without INTERSECT, joining StarsIn to itself on starID
SELECT DISTINCT S1.starID
FROM   Movie M1
JOIN   StarsIn S1 ON M1.movieID = S1.movieID
JOIN   StarsIn S2 ON S1.starID = S2.starID
JOIN   Movie M2 ON M2.movieID = S2.movieID
WHERE  M1.year = 1944 AND M2.year = 1974;
```

[6] courses/infs1200/sql-joins.pdf

`EXCEPT` queries can likewise always be implemented using nested queries — "in the first result but not the second" becomes a `NOT IN` or `NOT EXISTS` subquery (sql-nested-queries[7]):

```sql
-- Stars in a movie from 1944 but NOT 1974
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1944
EXCEPT
SELECT starID FROM Movie M JOIN StarsIn S ON M.movieID = S.movieID WHERE year = 1974;

-- Rewritten without EXCEPT
SELECT DISTINCT S.starID
FROM   Movie M JOIN StarsIn S ON M.movieID = S.movieID
WHERE  M.year = 1944
  AND  S.starID NOT IN (
      SELECT S2.starID
      FROM   Movie M2 JOIN StarsIn S2 ON M2.movieID = S2.movieID
      WHERE  M2.year = 1974);
```

The rewrite is not just a workaround — `NOT IN`/`NOT EXISTS` is also the building block of the double-negation form of relational-division[8], so it is the pattern worth being fluent in regardless of dialect.

[7] courses/infs1200/sql-nested-queries.pdf
[8] courses/infs1200/relational-division.pdf