

Set Operations and LIMIT – Key Points

- JOIN creates a new table with more columns
 - new rows are pairwise combinations of rows from the original relations
 - UNION creates a new table with more rows
 - new rows are the rows from either of the original relations
 - removes duplicate rows; use UNION ALL to keep duplicates
 - INTERSECT, EXCEPT have only been available in MySQL since 8.0.31 (late 2022)
-
- LIMIT *n* returns the first *n* rows
 - usually combined with ORDER BY since otherwise row order is arbitrary

UNION

(SELECT ...)	(SELECT ...)
UNION [DISTINCT]	UNION ALL
(SELECT ...)	(SELECT ...)

- result consists of the rows from both result sets
 - A UNION [DISTINCT] B does *not* include duplicate rows
 - one copy of a row if it appears at all in either A or B
 - A UNION ALL B includes duplicate rows
 - number of copies is sum of number of copies in each of A and B

Notes.

- columns need to match in number, order
- columns should match in data type

INTERSECT

(SELECT ...)	(SELECT ...)
INTERSECT [DISTINCT]	INTERSECT ALL
(SELECT ...)	(SELECT ...)

- result consists of only the rows contained in both result sets
 - A INTERSECT [DISTINCT] B does *not* include duplicate rows
 - one copy of a row if both A and B have at least one copy each
 - A INTERSECT ALL B includes duplicate rows
 - number of copies is the number present in both A and B (minimum of the number of copies in A and the number of copies in B)

Notes.

- columns need to match in number, order
- columns should match in data type

EXCEPT

(SELECT ...)	(SELECT ...)
EXCEPT [DISTINCT]	EXCEPT ALL
(SELECT ...)	(SELECT ...)

- result consists of only the rows contained in the left set and not the right set
 - A EXCEPT [DISTINCT] B does *not* include duplicate rows
 - one copy in the rest only if there's at least one copy in A and none in B
 - no copies if B has any, even if A has more
 - A EXCEPT ALL B includes duplicate rows
 - number of copies is the number in A minus the number in B

Notes.

- columns need to match in number, order
- columns should match in data type

SELECT mgrssn AS ssn FROM DEPARTMENT UNION SELECT superssn FROM EMPLOYEE	employees (SSNs) who are managers or supervisors
SELECT mgrssn AS ssn FROM DEPARTMENT UNION ALL SELECT superssn FROM EMPLOYEE	employees (SSNs) who are managers or supervisors, with duplicates if there are employees who manage multiple departments, supervise multiple other employees, and/or both manage and supervise
SELECT mgrssn FROM DEPARTMENT INTERSECT SELECT superssn FROM EMPLOYEE	employees (SSNs) who are both managers and supervisors
SELECT superssn FROM EMPLOYEE EXCEPT SELECT mgrssn FROM DEPARTMENT	employees (SSNs) who are supervisors but not managers

Examples

- List the SSNs of employees who either work more than 40 hours on some project or have more than two dependents.
- List the SSNs of employees who either earn more than \$80,000 or are department managers.
- Find employees who are both supervisors and parents of dependents.
- Find project numbers that currently have no employees working on them.
- Find supervisor SSNs that don't correspond to any actual employee's SSN.

Best Practices

- prefer a set operation when –
 - the same kind of value is being pooled or compared from independently-produced sources
 - e.g. find every city where the company has either a department office or a project
 - there's not a shared join key between the two sides
 - the phrase matches the English directly
 - e.g. "employees who are X or Y" → UNION, "employees who are both X and Y" → INTERSECT, "employees who are X but not Y" → EXCEPT
- prefer a join when –
 - you're relating two different kinds of things via a shared relationship
 - e.g. departments and their locations
 - you need columns from both sides in the output
- prefer a subquery (IN/EXISTS) when –
 - you only need columns from one side and are just filtering by membership
 - e.g. employees who are both a supervisor and a parent
 - you're using a database version/engine that doesn't support the necessary set operators
 - performance profiling shows a meaningful difference
 - without ALL, set ops involve deduplication which a subquery may be able to avoid
- be wary of NOT IN when the subquery might contain NULLs
 - prefer NOT EXISTS, a LEFT JOIN ... IS NULL anti-join, or EXCEPT