MySQL UNION

**Summary**: Here, you will learn how to use MySQL UNION operator to combine two or more result sets from multiple SELECT statements into a single result set.

## **MySQL UNION operator**

MySQL UNION operator allows you to combine two or more result sets of queries into a single result set. The following illustrates the syntax of the UNION operator:

SELECT column\_list

UNION [DISTINCT | ALL]

SELECT column\_list

UNION [DISTINCT | ALL]

SELECT column\_list

...

To combine result set of two or more queries using the UNION operator, these are the basic rules that you must follow:

* First, the number and the orders of columns that appear in all SELECT statements must be the same.
* Second, the data types of columns must be the same or compatible.

By default, the UNION operator removes duplicate rows even if you don’t specify the DISTINCT operator explicitly.

Let’s see the following sample tables: t1 and t2:

DROP TABLE IF EXISTS t1;

DROP TABLE IF EXISTS t2;

CREATE TABLE t1 (

id INT PRIMARY KEY

);

CREATE TABLE t2 (

id INT PRIMARY KEY

);

INSERT INTO t1 VALUES (1),(2),(3);

INSERT INTO t2 VALUES (2),(3),(4);

The following statement combines result sets returned from t1 and t2 tables:

SELECT id

FROM t1

UNION

SELECT id

FROM t2;

The final result set contains the distinct values from separate result sets returned by the queries:

| id |

+----+

| 1 |

| 2 |

| 3 |

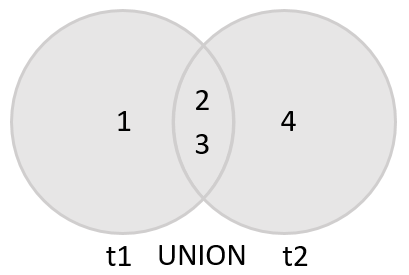
| 4 |

+----+

4 rows in set (0.00 sec)

Because the rows with value 2 and 3 are duplicates, the UNION removed them and kept only unique values.

The following Venn diagram illustrates the union of two result sets that come from t1 and t2 tables:



If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. Because UNION ALL does not need to handle duplicates, it performs faster than UNION DISTINCT .

SELECT id

FROM t1

UNION ALL

SELECT id

FROM t2;

+----+

| id |

+----+

| 1 |

| 2 |

| 3 |

| 2 |

| 3 |

| 4 |

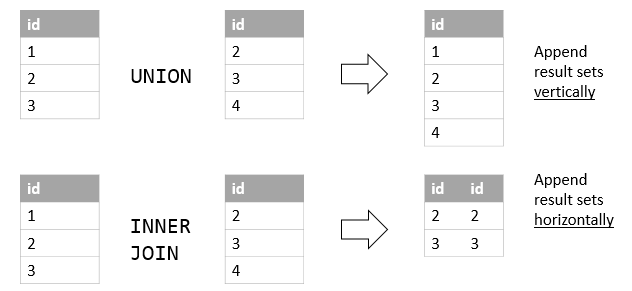
+----+

6 rows in set (0.00 sec)

As you can see, the duplicates appear in the combined result set because of the UNION ALL operation.

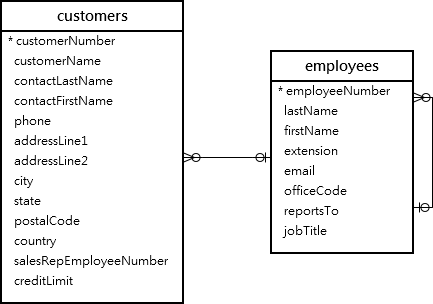
## **UNION vs. JOIN**

A JOIN combines result sets horizontally, a UNION appends result set vertically. The following picture illustrates the difference between UNION and JOIN:



## **MySQL UNION and column alias examples**

We’ll use the customers and employees tables in the sample database for the demonstration:



Suppose that you want to combine the first name and last name of employees and customers into a single result set, you can use the UNION operator as follows:

SELECT

firstName,

lastName

FROM

employees

UNION

SELECT

contactFirstName,

contactLastName

FROM

customers;



As you can see from the output, the MySQL UNION uses the column names of the first SELECT statement for the column headings of the output.

If you want to use other column headings, you need to use column aliases explicitly in the first SELECT statement as shown in the following example:

SELECT

CONCAT(firstName,' ',lastName) fullname

FROM

employees

UNION SELECT

CONCAT(contactFirstName,' ',contactLastName)

FROM

customers;

## **MySQL UNION with column alias example**

This example uses the column heading of the first query for the output. It uses the CONCAT() function to concatenate first name, space, and last name into a full name.

## **MySQL UNION and ORDER BY**

If you want to sort the result set of a union, you use an ORDER BY clause in the last SELECT statement as shown in the following example:

SELECT

concat(firstName,' ',lastName) fullname

FROM

employees

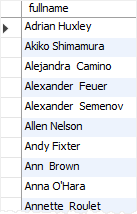
UNION SELECT

concat(contactFirstName,' ',contactLastName)

FROM

customers

ORDER BY fullname;



Notice that if you place the ORDER BY clause in each SELECT statement, it will not affect the order of the rows in the final result set.

To differentiate between employees and customers, you can add a column as shown in the following query:

SELECT

CONCAT(firstName, ' ', lastName) fullname,

'Employee' as contactType

FROM

employees

UNION SELECT

CONCAT(contactFirstName, ' ', contactLastName),

'Customer' as contactType

FROM

customers

ORDER BY

fullname;



MySQL also provides you with an alternative option to sort a result set based on column position using ORDER BY clause as follows:

SELECT

CONCAT(firstName,' ',lastName) fullname

FROM

employees

UNION SELECT

CONCAT(contactFirstName,' ',contactLastName)

FROM

customers

ORDER BY 1;

However, it is not a good practice to sort the result set by column position.

In this tutorial, you have learned how to use MySQL UNION statement to combine data from multiple queries into a single result set.

# MySQL MINUS

**Here**, you will learn how about SQL MINUS operator and how to simulate MINUS in MySQL using join.

Note that MySQL does not support the MINUS operator. This tutorial shows you to how to emulate the MINUS operator in MySQL using join clauses.

## **Introduction to SQL MINUS operator**

The MINUS operator is one of three set operators in the SQL standard that includes UNION, INTERSECT, and MINUS.

The MINUS compares the results of two queries and returns distinct rows from the result set of the first query that does not appear in the result set of the second query.

The following illustrates the syntax of the MINUS operator:

SELECT select\_list1

FROM table\_name1

MINUS

SELECT select\_list2

FROM table\_name2;

The basic rules for a query that uses MINUS operator are the following:

* The number and order of columns in both select\_list1 and select\_list2 must be the same.
* The data types of the corresponding columns in both queries must be compatible.

Suppose that we have two tables t1 and t2 with the following structure and data:

CREATE TABLE t1 (

id INT PRIMARY KEY

);

CREATE TABLE t2 (

id INT PRIMARY KEY

);

INSERT INTO t1 VALUES (1),(2),(3);

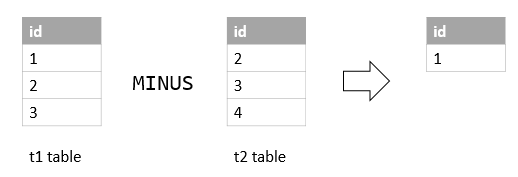
INSERT INTO t2 VALUES (2),(3),(4);

The following query returns distinct values from the query of the t1 table that is not found in the result of the query of the t2 table.

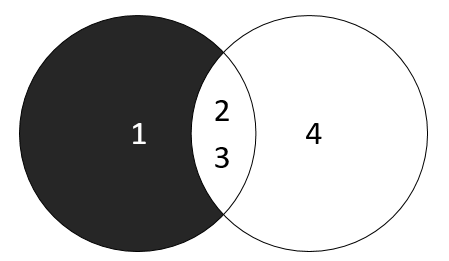
SELECT id FROM t1

MINUS

SELECT id FROM t2;



The following Venn diagram illustrates the MINUS operation:



Note that some database systems e.g., Microsoft SQL Server and PostgreSQL use the EXCEPT instead of MINUS. They have the same function.

## **MySQL MINUS operator emulation**

Unfortunately, MySQL does not support MINUS operator. However, you can use join to emulate it.

To emulate the MINUS of two queries, you use the following syntax:

SELECT

select\_list

FROM

table1

LEFT JOIN table2

ON join\_predicate

WHERE

table2.column\_name IS NULL;

For example, the following query uses the LEFT JOIN clause to return the same result as the MINUS operator:

SELECT

id

FROM

t1

LEFT JOIN

t2 USING (id)

WHERE

t2.id IS NULL;

Here, you have learned about the SQL MINUS operator and how to emulate MINUS operator in MySQL using LEFT JOIN clause.

# MySQL INTERSECT

**Summary**: Here, we will introduce you to the INTERSECT operator and show you how to emulate the MySQL INTERSECT operator.

Note that MySQL does not support the INTERSECT operator. This tutorial introduces you to how to emulate the INTERSECT operator in MySQL using join clauses.

## **Introduction to the INTERSECT operator**

The INTERSECT operator is a set operator that returns only distinct rows of two queries or more queries.

The following illustrates the syntax of the INTERSECT operator.

(SELECT column\_list

FROM table\_1)

INTERSECT

(SELECT column\_list

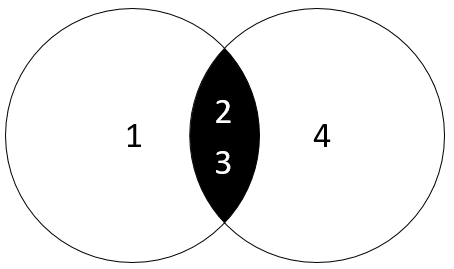
FROM table\_2);

The INTERSECT operator compares the result sets of two queries and returns the distinct rows that are output by both queries.

To use the INTERSECT operator for two queries, you follow these rules:

1. The order and the number of columns in the select list of the queries must be the same.
2. The data types of the corresponding columns must be compatible.

The following diagram illustrates the INTERSECT operator.



The left query produces a result set of (1,2,3).

The right query returns a result set of (2,3,4).

The INTERSECT operator returns the distinct rows of both result sets which include (2,3).

Unlike the UNION operator, the INTERSECT operator returns the intersection between two circles.

Note that the SQL standard has three set operators that include UNION, INTERSECT, and MINUS.

## **Emulating INTERSECT in MySQL**

Unfortunately, MySQL does not support the INTERSECT operator. However, you can emulate the INTERSECT operator.

### **Setting up sample tables**

The following statements create tables t1 and t2, and then insert data into both tables.

CREATE TABLE t1 (

id INT PRIMARY KEY

);

CREATE TABLE t2 LIKE t1;

INSERT INTO t1(id) VALUES(1),(2),(3);

INSERT INTO t2(id) VALUES(2),(3),(4);

The following query returns rows from the t1 table.

SELECT id FROM t1;

id

----

1

2

3

The following query returns the rows from the t2 table:

SELECT id

FROM t2;

id

---

2

3

4

### **1) Emulate INTERSECT using DISTINCT and INNER JOIN clause**

The following statement uses DISTINCT operator and INNER JOIN clause to return the distinct rows in both tables:

SELECT DISTINCT

id

FROM t1

INNER JOIN t2 USING(id);

id

----

2

3

How it works.

1. The INNER JOIN clause returns rows from both left and right tables.
2. The DISTINCT operator removes the duplicate rows.

### **2) Emulate INTERSECT using IN and subquery**

The following statement uses the IN operator and a subquery to return the intersection of the two result sets.

SELECT DISTINCT id

FROM t1

WHERE id IN (SELECT id FROM t2);

id

----

2

3

How it works.

1. The subquery returns the first result set.
2. The outer query uses the IN operator to select only values that exist in the first result set. The DISTINCT operator ensures that only distinct values are selected.

Here, you have learned a couple of ways to simulate the INTERSECT operator in MySQL.