**Lab 4: Set Operations and Aggregate Functions**

## MySQL SET Operations

## 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:

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](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statements must be the same.
* Second, the [data types](http://www.mysqltutorial.org/mysql-data-types.aspx) of columns must be the same or compatible.

By default, the UNION operator removes [duplicate rows](http://www.mysqltutorial.org/mysql-find-duplicate-values/) even if you don’t specify the [DISTINCT](http://www.mysqltutorial.org/mysql-distinct.aspx) operator explicitly.

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](http://www.mysqltutorial.org/mysql-alias/) 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;

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](http://www.mysqltutorial.org/mysql-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;

**Intersect and Minus Operations**

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);

**MySQL Intersect Operator**

SELECT id FROM t1

INTERSECT

SELECT id FROM t2;

*Note that MySQL did not support the INTERSECT operator before version 8.0.31. Though we could emulate the INTERSECT operator in MySQL using join clauses.*

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

### Emulate INTERSECT using DISTINCT and INNER JOIN clause

The following statement uses [DISTINCT](http://www.mysqltutorial.org/mysql-distinct.aspx) operator and [INNER JOIN](http://www.mysqltutorial.org/mysql-inner-join.aspx) clause to return the distinct rows in both tables:

SELECT DISTINCT

   id

FROM t1

   INNER JOIN t2 USING(id);

### Emulate INTERSECT using IN and subquery

The following statement uses the [IN](http://www.mysqltutorial.org/sql-in.aspx) operator and a [subquery](http://www.mysqltutorial.org/mysql-subquery/) to return the intersection of the two result sets.

SELECT DISTINCT id

FROM t1

WHERE id IN (SELECT id FROM t2);

**MySQL MINUS Operator**

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.

In mysql, MINUS operator is defined by EXCEPT.

SELECT id FROM t1

EXCEPT

SELECT id FROM t2;

## MySQL MINUS operator emulation

*Note that MySQL did not support the MINUS/EXCEPT operator before version 8.0.31. Though we could emulate the MINUS operator in MySQL using join clauses.*

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

SELECT

    id

FROM

    t1

LEFT JOIN

    t2 USING (id)

WHERE

    t2.id IS NULL;

**Using MySQL Aggregate Operations**

**Count Function**

The COUNT function returns the number of the rows in a table.

For example, you can use the COUNT function to get the number of products in the products table as the following query:

SELECT COUNT(\*) AS Total

FROM products;

Following statement uses the COUNT() function with the [GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx) clause to return the number of products in each product line:

SELECT

    productLine,

    COUNT(\*)

FROM

    products

GROUP BY productLine;

**Using Distinct with COUNT()**

The following query returns total of 110 since it counts all.

SELECT COUNT(productVendor) AS Total

FROM products;

However, if we use the distinct clause like below it returns 13.

SELECT COUNT(DISTINCT productVendor) AS Total

FROM products;

**COUNT with group by clause**

Similarly, following example uses the COUNT(\*) function to find the number of products supplied by each vendor:

SELECT

    productVendor,

    COUNT(\*)

FROM

    products

GROUP BY productVendor

ORDER BY COUNT(\*) DESC;

To find vendors who supply at least 9 products, you use the COUNT(\*) function in the [HAVING](http://www.mysqltutorial.org/mysql-having.aspx) clause as shown in the following query:

SELECT

    productVendor,

    COUNT(\*)

FROM

    products

GROUP BY productVendor

HAVING COUNT(\*) >= 9

ORDER BY COUNT(\*) DESC;

**SUM Function**

The SUM function returns the sum of a set of values. The SUM function ignores NULL values. If no matching row found, the SUM function returns a NULL value.

To get the total sales of each product, you can use the SUM function in conjunction with the GROUP BY clause as follows:

SELECT productCode,sum(priceEach \* quantityOrdered) total

FROM orderdetails

GROUP by productCode;

**AVG Function**

The AVG function calculates the average value of a set of values. It ignores NULL values in the calculation.

You can use the AVG function to calculate the average buy price of all products in the products table by using the following query:

SELECT AVG(buyPrice) average\_buy\_price

FROM products;

The following  example uses the AVG() function to calculate the average buy price of products in the product line Classic Cars

SELECT

    AVG(buyprice) 'Average Classic Cars Price'

FROM

    products

WHERE

    productline = 'Classic Cars';

**MAX Function**

The MAX function returns the maximum value in a set of values

For example, you can use the MAX function to get the most expensive product in the products table as the following query:

SELECT MAX(buyPrice) highest\_price

FROM Products;

**MIN Function**

The MIN function returns the minimum value in a set of values.

For example, the following query uses the MIN function to find the product with the lowest price in the products table:

SELECT MIN(buyPrice) lowest\_price

FROM Products