# MySQL vs PostgreSQL: Functions and Return Types

In database systems, both MySQL and PostgreSQL allow the creation of functions to return values. However, the way they handle return types varies between these two database systems. This document compares MySQL and PostgreSQL functions, focusing on how they return single or multiple values.

## MySQL Function Return Types

In MySQL, functions generally return a single scalar value. The most common return types are numeric, string, or date values. Below is an example of a simple function that returns a scalar value in MySQL:

### Example: MySQL Scalar Function

MySQL Function to return the sum of two numbers:

CREATE FUNCTION sum\_two\_numbers(a INT, b INT)  
RETURNS INT  
DETERMINISTIC  
BEGIN  
 RETURN a + b;  
END;

This function sum\_two\_numbers returns a scalar value (an integer). It performs the sum of two input values.

## PostgreSQL Function Return Types

In PostgreSQL, functions can return multiple values, unlike MySQL where only scalar values are returned. PostgreSQL provides more flexibility with return types such as set-returning functions, arrays, and composite types. Below are a few examples of PostgreSQL functions returning different types of values.

### Set-Returning Functions (SRF)

In PostgreSQL, a Set-Returning Function (SRF) can return a set of rows, similar to a query result. The function returns a result set with one or more columns, which can be used in queries just like a table. Here is an example of a set-returning function in PostgreSQL:

#### Example: PostgreSQL Set-Returning Function

PostgreSQL Function to return a table of values:

CREATE FUNCTION get\_multiple\_values()  
RETURNS TABLE(id INT, name TEXT) AS $$  
BEGIN  
 RETURN QUERY  
 SELECT id, name FROM my\_table;  
END;  
$$ LANGUAGE plpgsql;

The function get\_multiple\_values returns a set of rows from the table my\_table. Each row contains an id and a name.

### Returning Arrays

In PostgreSQL, a function can also return an array. An array can store multiple values of the same type in a single column. Here’s an example where a function returns an array of integer values:

#### Example: PostgreSQL Array Return

PostgreSQL Function to return an array of integer values:

CREATE FUNCTION get\_array\_of\_values()  
RETURNS INTEGER[] AS $$  
BEGIN  
 RETURN ARRAY(SELECT id FROM my\_table);  
END;  
$$ LANGUAGE plpgsql;

The function get\_array\_of\_values returns an array of integers, each representing an id from the table my\_table.

### Returning Composite Types

PostgreSQL also supports composite types, which are custom types that can hold multiple values in a single return object. Here is an example where a composite type is returned by a function:

#### Example: PostgreSQL Composite Type Return

First, define a composite type:

CREATE TYPE my\_composite\_type AS (id INT, name TEXT);

Then create a function that returns a set of this composite type:

CREATE FUNCTION get\_composite\_values()  
RETURNS SETOF my\_composite\_type AS $$  
BEGIN  
 RETURN QUERY  
 SELECT id, name FROM my\_table;  
END;  
$$ LANGUAGE plpgsql;

The function get\_composite\_values returns multiple rows, each with an id and name, encapsulated in a composite type.

## Conclusion

In conclusion, MySQL functions are limited to returning a single scalar value, whereas PostgreSQL provides much more flexibility. PostgreSQL supports returning multiple rows (Set-Returning Functions), arrays, and composite types, making it more versatile for complex queries and data retrieval.