Creating a stored procedure with one parameter

The following query returns a product list from the products table in the [sample database](https://www.sqlservertutorial.net/sql-server-sample-database/):

SELECT

product\_name,

list\_price

FROM

production.products

ORDER BY

list\_price;

Code language: SQL (Structured Query Language) (sql)

You can create a stored procedure that wraps this query using the CREATE PROCEDURE statement:

CREATE PROCEDURE uspFindProducts

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

ORDER BY

list\_price;

END;

Code language: SQL (Structured Query Language) (sql)

However, this time we can add a parameter to the stored procedure to find the products whose list prices are greater than an input price:

ALTER PROCEDURE uspFindProducts(@min\_list\_price AS DECIMAL)

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

WHERE

list\_price >= @min\_list\_price

ORDER BY

list\_price;

END;

Code language: SQL (Structured Query Language) (sql)

In this example:

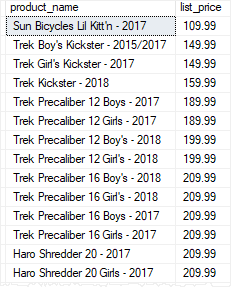
* First, we added a parameter named @min\_list\_price to the uspFindProducts stored procedure. Every parameter must start with the @ sign. The AS DECIMAL keywords specify the data type of the @min\_list\_price parameter. The parameter must be surrounded by the opening and closing brackets.
* Second, we used @min\_list\_price parameter in the WHERE clause of the SELECT statement to filter only the products whose list prices are greater than or equal to the @min\_list\_price.

Executing a stored procedure with one parameter

To execute the uspFindProducts stored procedure, you pass an argument to it as follows:

EXEC uspFindProducts 100;

Code language: SQL (Structured Query Language) (sql)

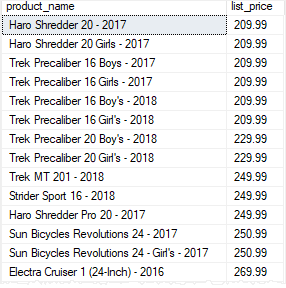


The stored procedure returns all products whose list prices are greater than or equal to 100.

If you change the argument to 200, you will get a different result set:

EXEC uspFindProducts 200;

Code language: SQL (Structured Query Language) (sql)



Creating a stored procedure with multiple parameters

Stored procedures can take one or more parameters. The parameters are separated by commas.

The following statement modifies the uspFindProducts stored procedure by adding one more parameter named @max\_list\_price to it:

ALTER PROCEDURE uspFindProducts(

@min\_list\_price AS DECIMAL

,@max\_list\_price AS DECIMAL

)

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

WHERE

list\_price >= @min\_list\_price AND

list\_price <= @max\_list\_price

ORDER BY

list\_price;

END;

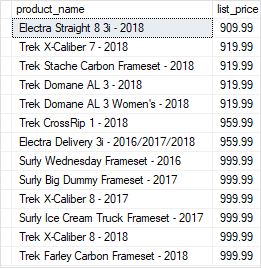
Code language: SQL (Structured Query Language) (sql)

Once the stored procedure is modified successfully, you can execute it by passing two arguments, one for @min\_list\_price and the other for @max\_list\_price:

EXECUTE uspFindProducts 900, 1000;

Code language: SQL (Structured Query Language) (sql)

The following shows the output:



Using named parameters

In case stored procedures have multiple parameters, it is better and more clear to execute the stored procedures using named parameters.

For example, the following statement executes the uspFindProducts stored procedure using the named parameters @min\_list\_priceand @max\_list\_price:

EXECUTE uspFindProducts

@min\_list\_price = 900,

@max\_list\_price = 1000;

Code language: SQL (Structured Query Language) (sql)

The result of the stored procedure is the same however the statement is more obvious.

Creating text parameters

The following statement adds the @name parameter as a character string parameter to the stored procedure.

ALTER PROCEDURE uspFindProducts(

@min\_list\_price AS DECIMAL

,@max\_list\_price AS DECIMAL

,@name AS VARCHAR(max)

)

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

WHERE

list\_price >= @min\_list\_price AND

list\_price <= @max\_list\_price AND

product\_name LIKE '%' + @name + '%'

ORDER BY

list\_price;

END;

Code language: SQL (Structured Query Language) (sql)

In the [WHERE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/) clause of the [SELECT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) statement, we added the following condition:

product\_name LIKE '%' + @name + '%'

Code language: SQL (Structured Query Language) (sql)

By doing this, the stored procedure returns the products whose list prices are in the range of min and max list prices and the product names also contain a piece of text that you pass in.

Once the stored procedure is altered successfully, you can execute it as follows:

EXECUTE uspFindProducts

@min\_list\_price = 900,

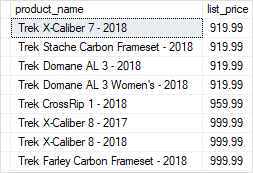
@max\_list\_price = 1000,

@name = 'Trek';

Code language: SQL (Structured Query Language) (sql)

In this statement, we used the uspFindProducts stored procedure to find the product whose list prices are in the range of 900 and 1,000 and their names contain the word Trek.

The following picture shows the output:



Creating optional parameters

When you execute the uspFindProducts stored procedure, you must pass all three arguments corresponding to the three parameters.

SQL Server allows you to specify default values for parameters so that when you call stored procedures, you can skip the parameters with default values.

See the following stored procedure:

ALTER PROCEDURE uspFindProducts(

@min\_list\_price AS DECIMAL = 0

,@max\_list\_price AS DECIMAL = 999999

,@name AS VARCHAR(max)

)

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

WHERE

list\_price >= @min\_list\_price AND

list\_price <= @max\_list\_price AND

product\_name LIKE '%' + @name + '%'

ORDER BY

list\_price;

END;

Code language: SQL (Structured Query Language) (sql)

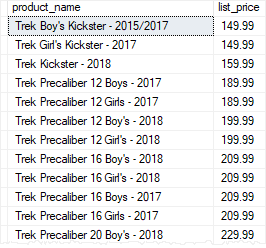
In this stored procedure, we assigned 0 as the default value for the @min\_list\_price parameter and 999,999 as the default value for the @max\_list\_price parameter.

Once the stored procedure is compiled, you can execute it without passing the arguments to @min\_list\_price and @max\_list\_price parameters:

EXECUTE uspFindProducts

@name = 'Trek';

Code language: SQL (Structured Query Language) (sql)



In this case, the stored procedure used 0 for @min\_list\_price parameter and 999,999 for the @max\_list\_price parameter when it executed the query.

The @min\_list\_price and @max\_list\_price parameters are called optional parameters.

Of course, you can also pass the arguments to the optional parameters. For example, the following statement returns all products whose list prices are greater or equal to 6,000 and the names contain the word Trek:

EXECUTE uspFindProducts

@min\_list\_price = 6000,

@name = 'Trek';

Code language: SQL (Structured Query Language) (sql)



Using NULL as the default value

In the uspFindProducts stored procedure, we used 999,999 as the default maximum list price. This is not robust because in the future you may have products with the list prices that are greater than that.

A typical technique to avoid this is to use NULL as the default value for the parameters:

ALTER PROCEDURE uspFindProducts(

@min\_list\_price AS DECIMAL = 0

,@max\_list\_price AS DECIMAL = NULL

,@name AS VARCHAR(max)

)

AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production.products

WHERE

list\_price >= @min\_list\_price AND

(@max\_list\_price IS NULL OR list\_price <= @max\_list\_price) AND

product\_name LIKE '%' + @name + '%'

ORDER BY

list\_price;

END;

Code language: SQL (Structured Query Language) (sql)

In the WHERE clause, we changed the condition to handle NULL value for the @max\_list\_price parameter:

(@max\_list\_price IS NULL OR list\_price <= @max\_list\_price)

Code language: SQL (Structured Query Language) (sql)

The following statement executes the uspFindProducts stored procedure to find the product whose list prices are greater or equal to 500 and names contain the word Haro.

EXECUTE uspFindProducts

@min\_list\_price = 500,

@name = 'Haro';

Code language: SQL (Structured Query Language) (sql)

