Structured query language (SQL)

SQL (Structured Query Language) is a powerful tool used for managing and manipulating relational databases. It allows you to retrieve, insert, update, and delete data and create and modify database structures.

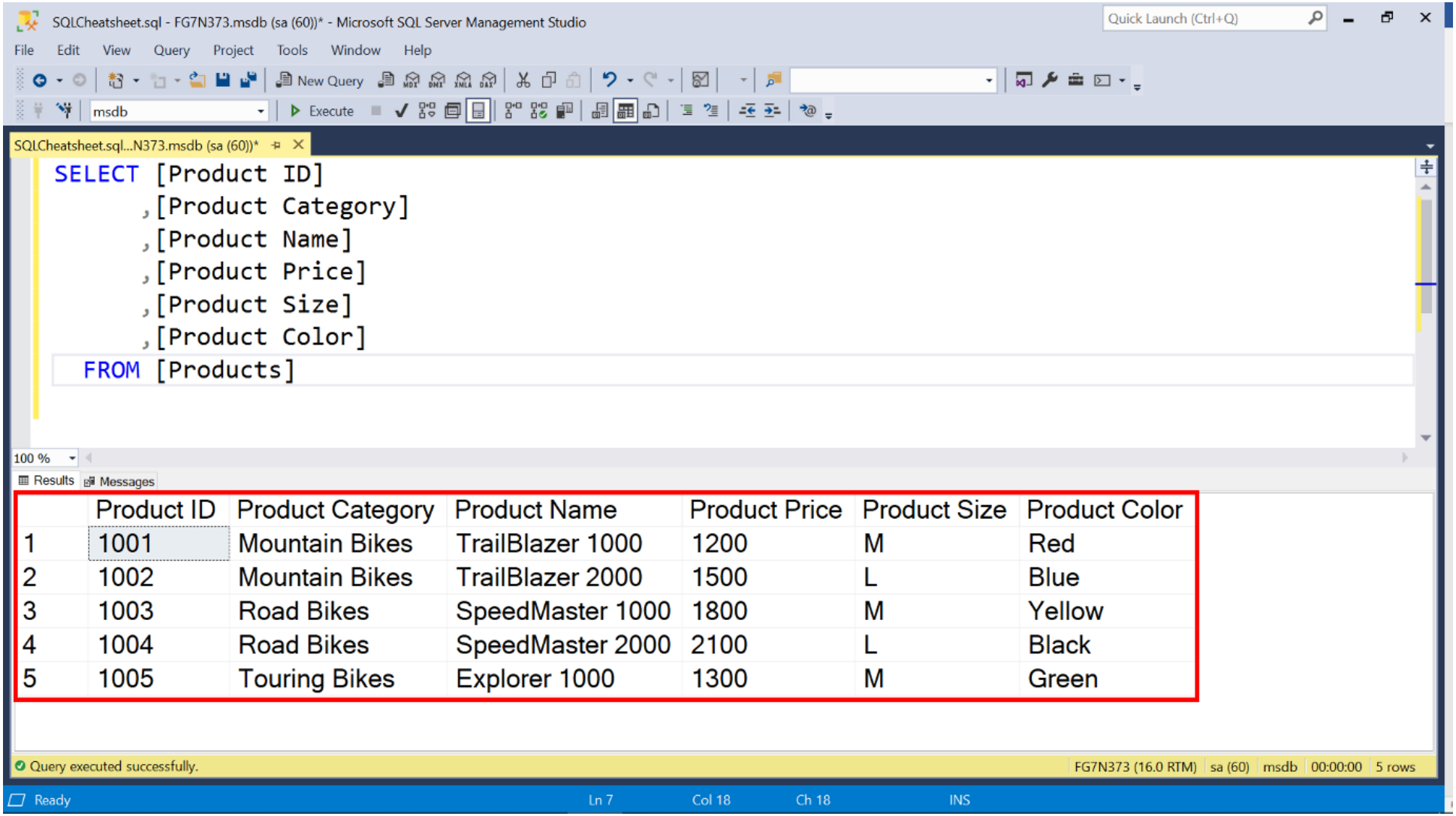
This reading structures the basic SQL commands into three separate categories:

Data Retrieval: These commands focus on querying and retrieving specific information from a database.

Data Manipulation: This section covers commands for performing calculations, combining data, and applying conditional logic.

Data Modification: These commands can be used for adding, updating, and deleting records in a database.

To assist you in understanding the functionality of all these commands, a simple **Product** table is used as an example. The table contains five product rows, as seen below:



We will identify how each of these commands affects the query result in the window.

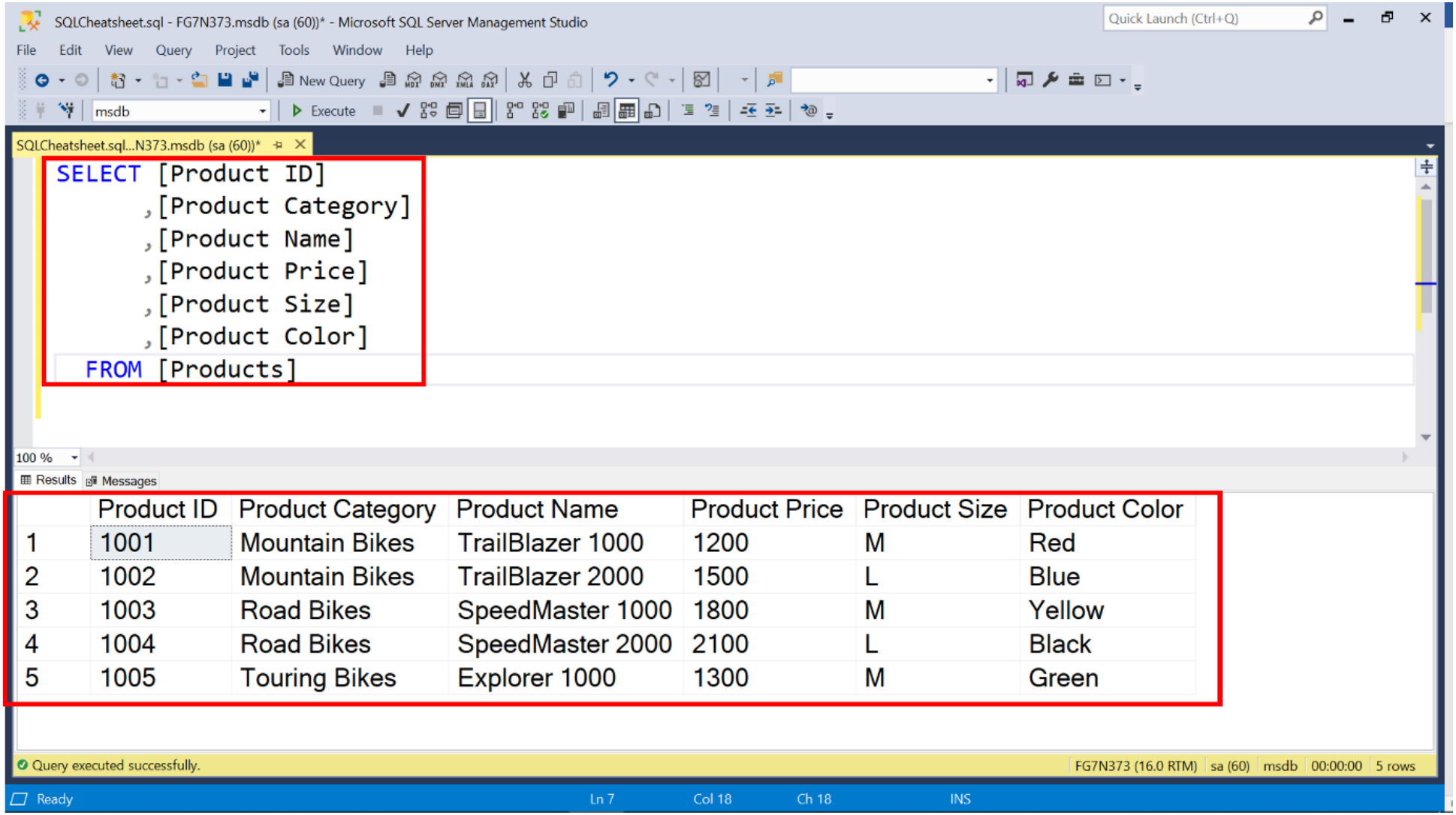
Data retrieval

You'll often have to retrieve data from databases. You can query and retrieve specific information from a database using several commands. These commands are explored in detail below.

SELECT query

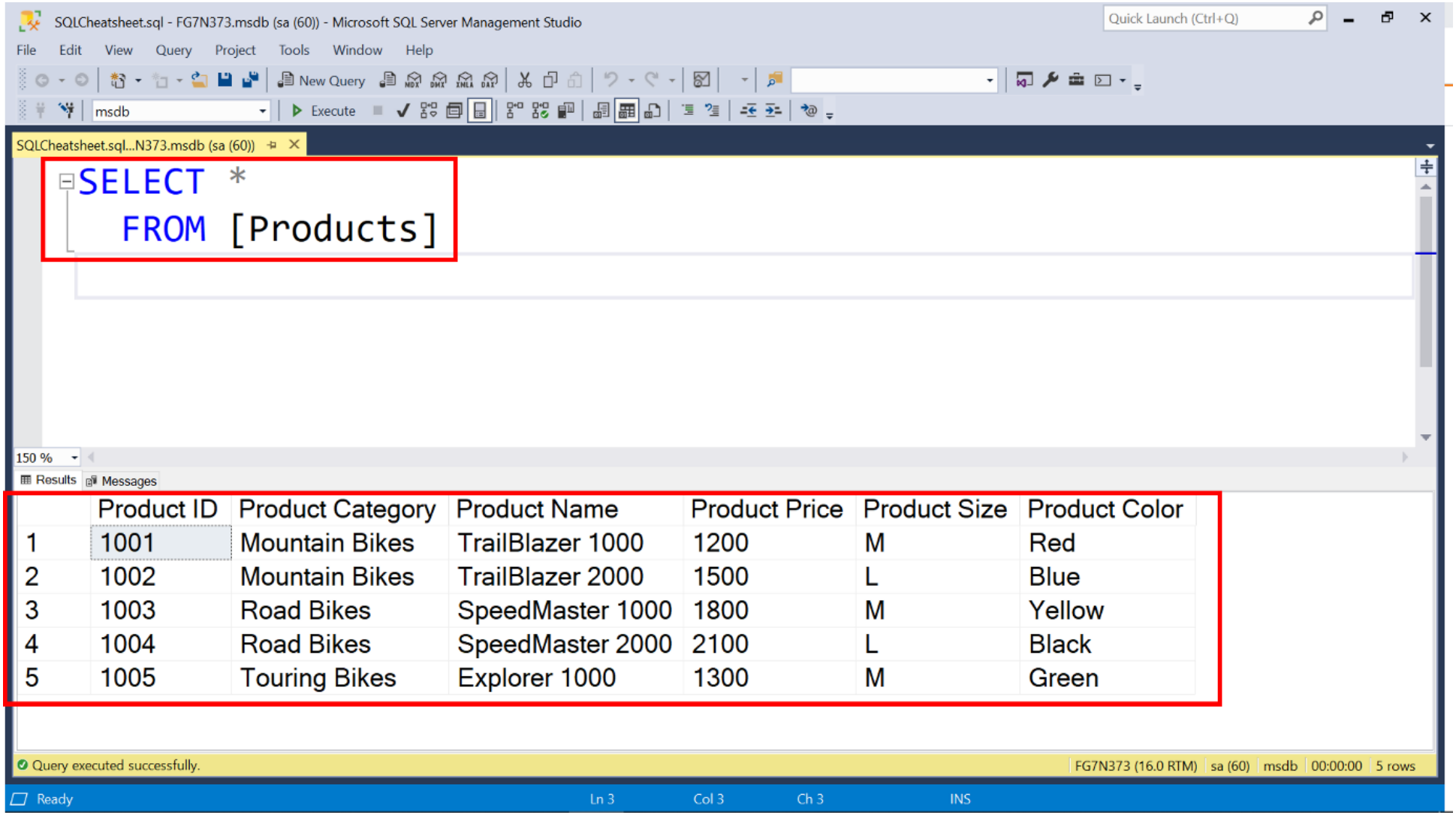
Data is retrieved from a database using a SQL **SELECT** query. The query must specify the data to be extracted, like column names. A **FROM** clause indicates what table it must be extracted from.

The following example extracts all data from the **Product** table.



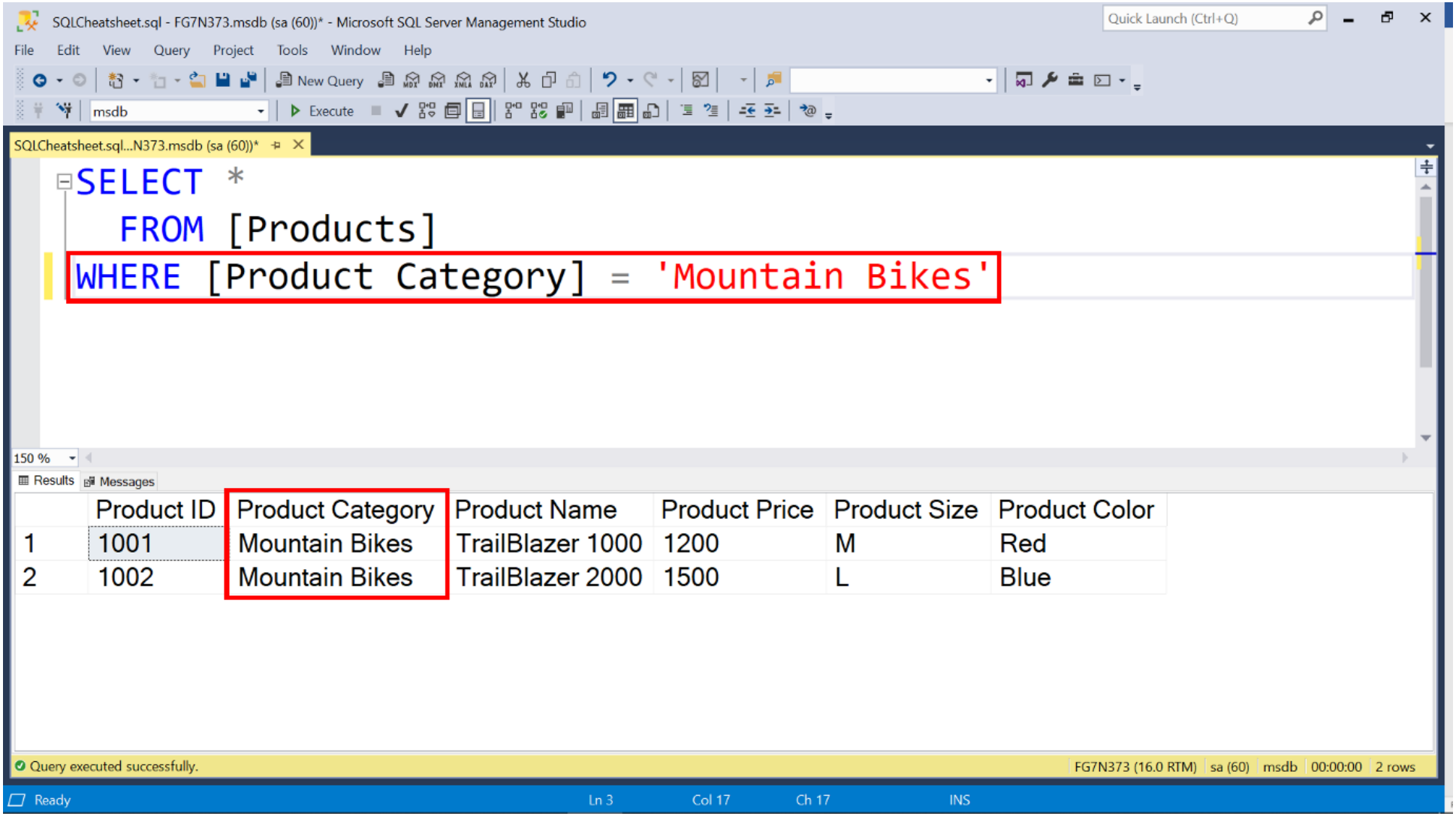
	Product ID	Product Category	Product Name	Product Price	Product Size	Product Color
1	1001	Mountain Bikes	TrailBlazer 1000	1200	M	Red
2	1002	Mountain Bikes	TrailBlazer 2000	1500	L	Blue
3	1003	Road Bikes	SpeedMaster 1000	1800	M	Yellow
4	1004	Road Bikes	SpeedMaster 2000	2100	L	Black
5	1005	Touring Bikes	Explorer 1000	1300	M	Green

Instead of listing all column names, you can also use an asterisk to retrieve all columns from a table, like in the following example:



Filtering clauses

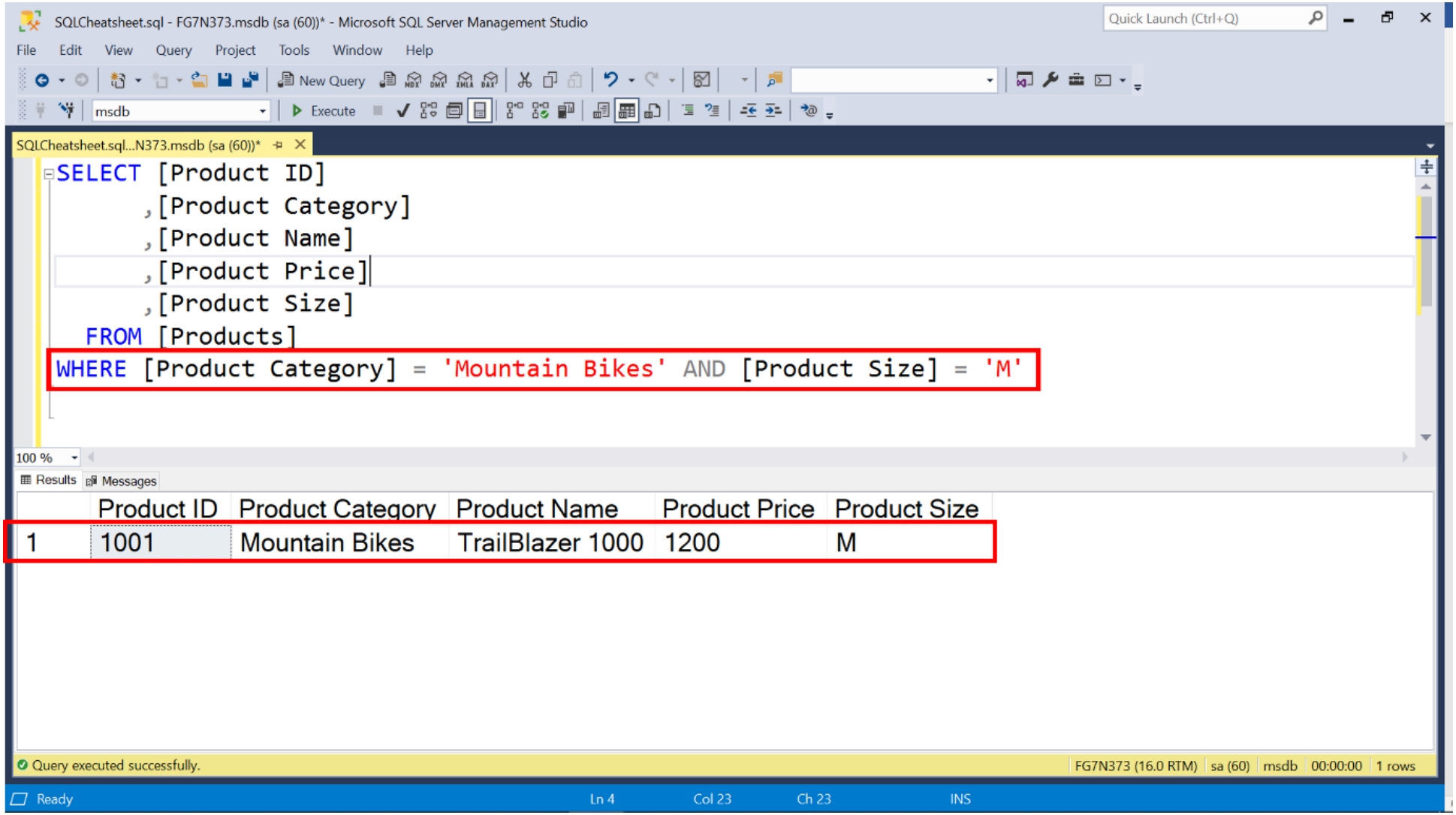
You can also apply filters to your **SELECT** queries so that only specific information is retrieved from the table instead of all records. A common method of applying a filter condition is using the **WHERE** clause.
In the following example, the **SELECT** query uses a **WHERE** clause to specify that only product information related to the value **Mountain Bikes** should be retrieved from the table.



Multiple Conditions

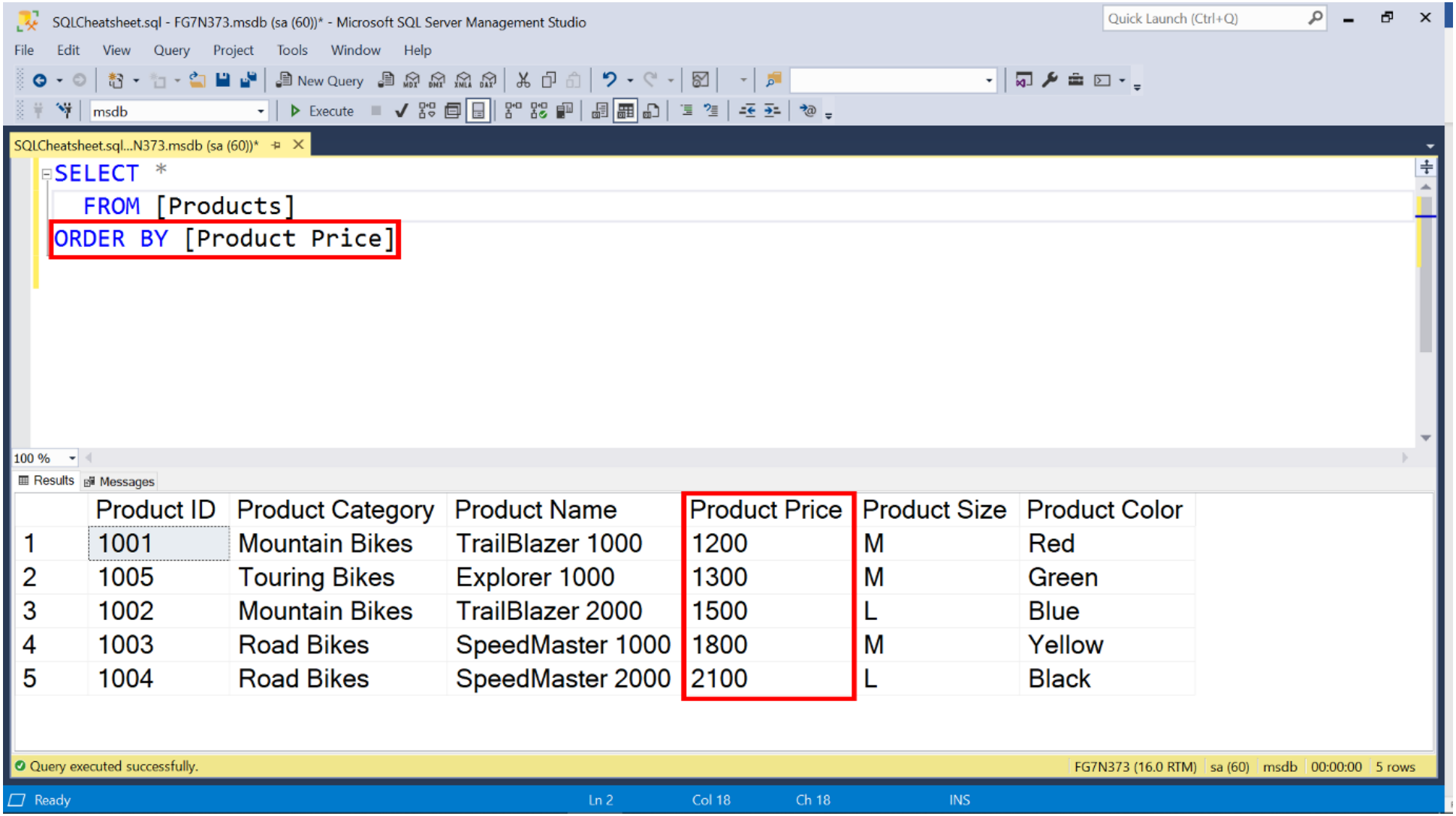
You can extend the use of the **WHERE** clause by adding certain conditions through the **AND** or **OR** commands. These commands are used to specify further conditions for filtering.

In the following example, the **WHERE** clause must filter all records with the value **Mountain Bike** from the **Product Category** table **AND** a value of **Medium** from the **Product Size** column.



Ordering results

You can use the **ORDER BY** command to sort your retrieved data in a specific order. In the example below, the records are sorted by price in ascending order. **ORDER BY** returns records in ascending order by default. To return records in descending order, use the **ORDER BY DESC** command.



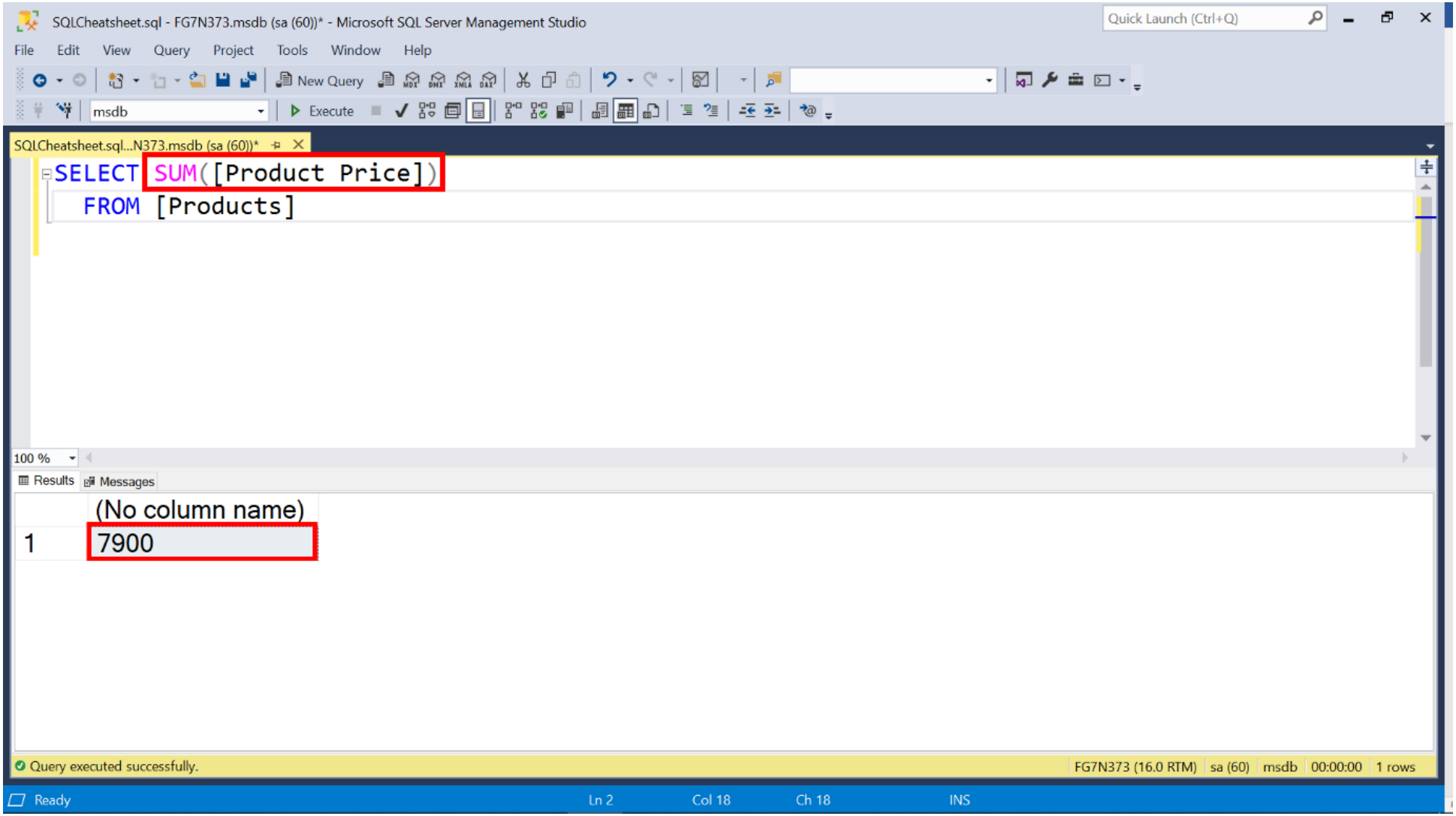
Data manipulation

SQL allows data analysts to change or manipulate the data in the database. You can use techniques like aggregation functions, aliases, conditional logic, and **NULL** values to manipulate your data as required.

Aggregation Functions

Aggregation functions are used to calculate values to return a single result. Examples of aggregating functions include **SUM**, **COUNT**, **AVERAGE**, **MAX** and **MIN**.

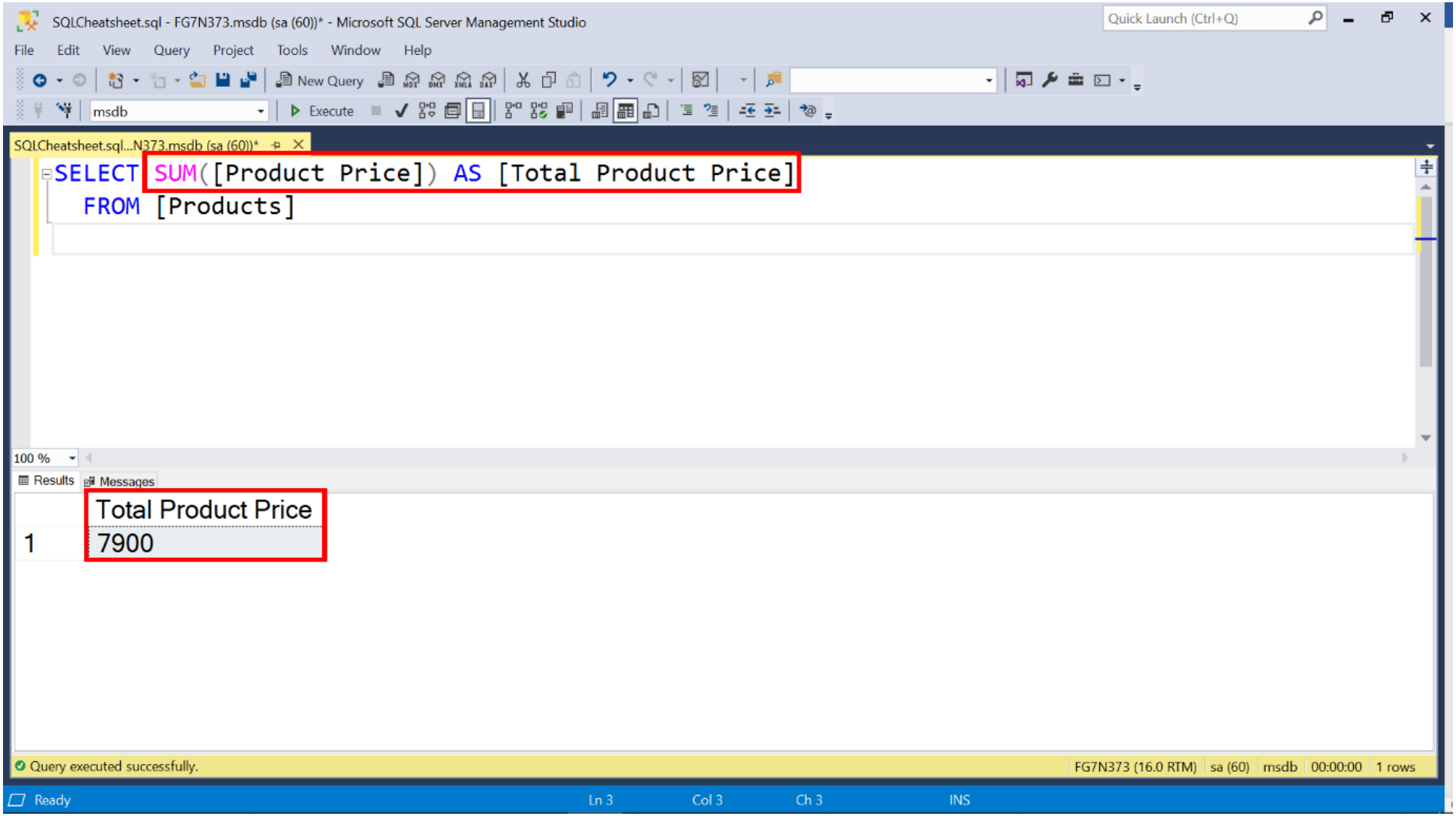
In the following example, the **SUM** aggregation function calculates the total of all prices from the **Product Price** column.



Aliases

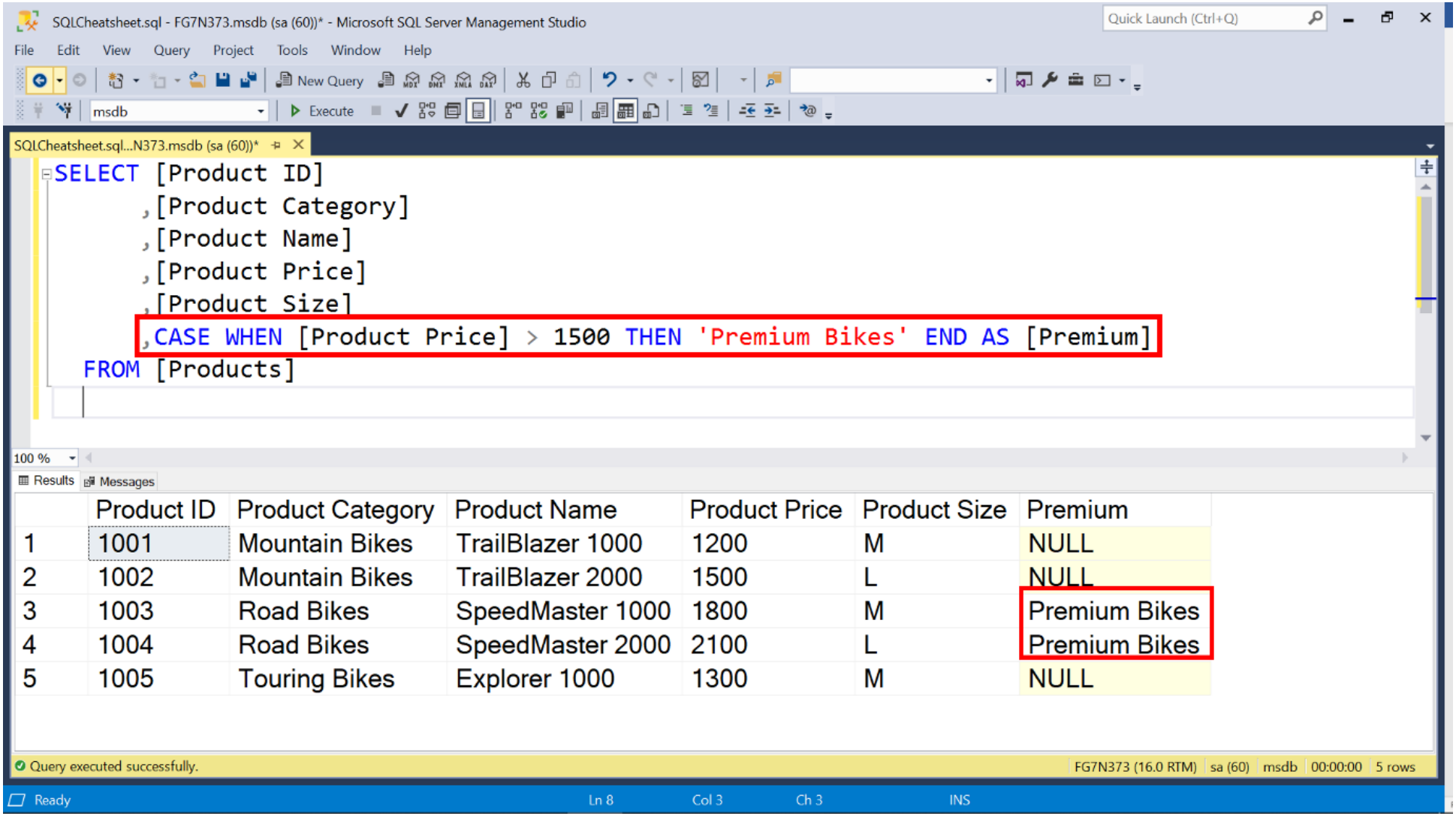
It is often useful to return your results as a new column. With SQL, you can use an alias to provide a name for your new column. Write the **AS** command followed by the name of your new column in square brackets.

The following example returns the total price of all products as a new column called **Total Product Price**.



Conditional Logic

You might often have to create complicated queries to return specific results from a table. In these instances, you can use conditional logic clauses like **CASE WHEN** to provide the required instructions. **CASE WHEN** is a conditional expression that lets you set a value on a field based on multiple conditions. The example below instructs SQL to label all values from the **Product Price** column with a value greater than **1500** as **Premium Bikes**. This is demonstrated in the example SQL statement below. In a **CASE** or instance, **WHEN** SQL encounters a value greater than **1500**, it must be labeled **Premium Bikes**.

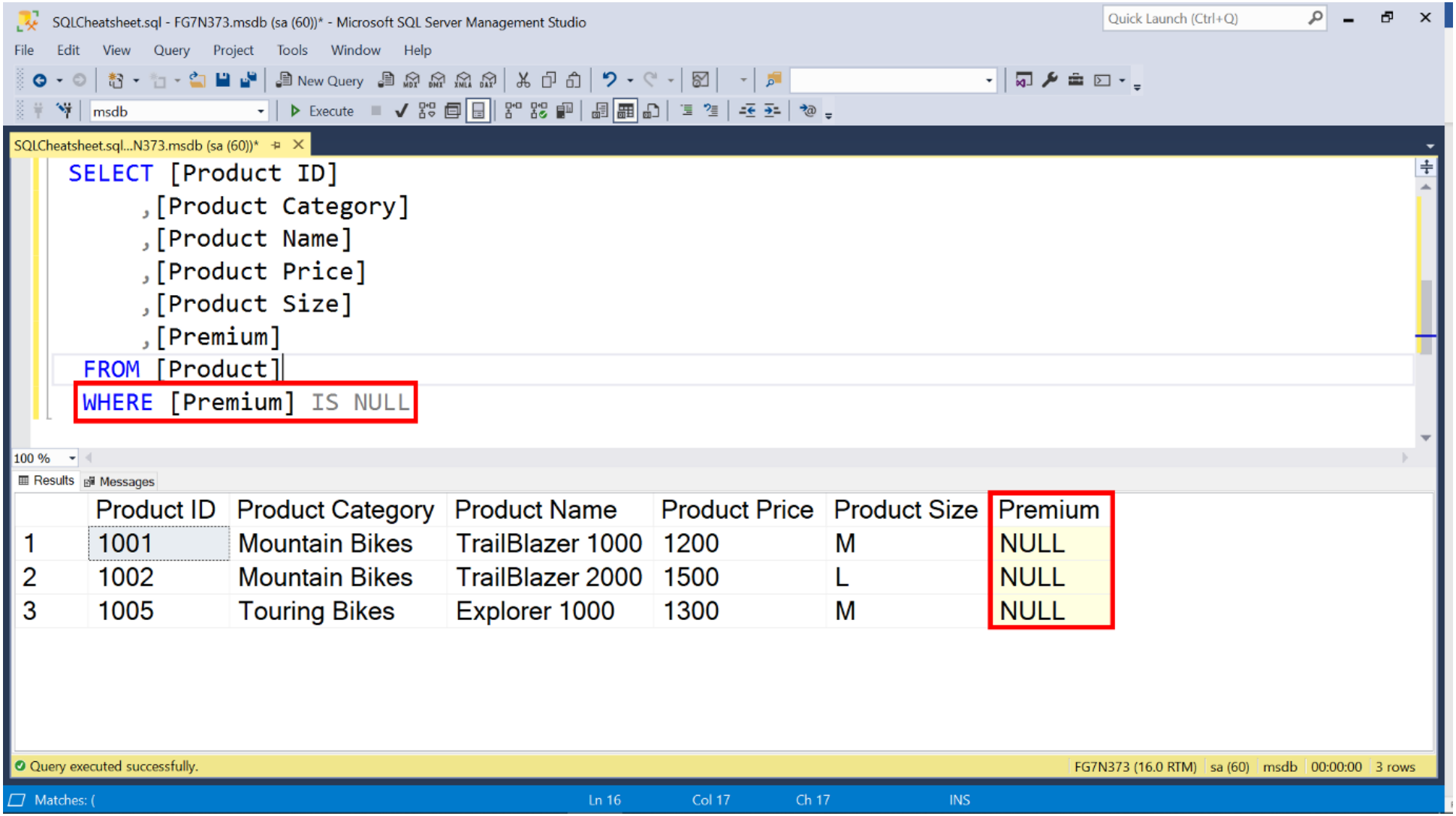


	Product ID	Product Category	Product Name	Product Price	Product Size	Premium
1	1001	Mountain Bikes	TrailBlazer 1000	1200	M	NULL
2	1002	Mountain Bikes	TrailBlazer 2000	1500	L	NULL
3	1003	Road Bikes	SpeedMaster 1000	1800	M	Premium Bikes
4	1004	Road Bikes	SpeedMaster 2000	2100	L	Premium Bikes
5	1005	Touring Bikes	Explorer 1000	1300	M	NULL

NULL values

In the previous example, the rows that didn't meet the condition were marked with the **NULL** value. This is theblank value in SQL. To retrieve these rows, you can use a **WHERE** clause with the **IS NULL** condition.

In the following example, SQL retrieves all records from the **Product** table with a value of **NULL** in the **Premium** column.



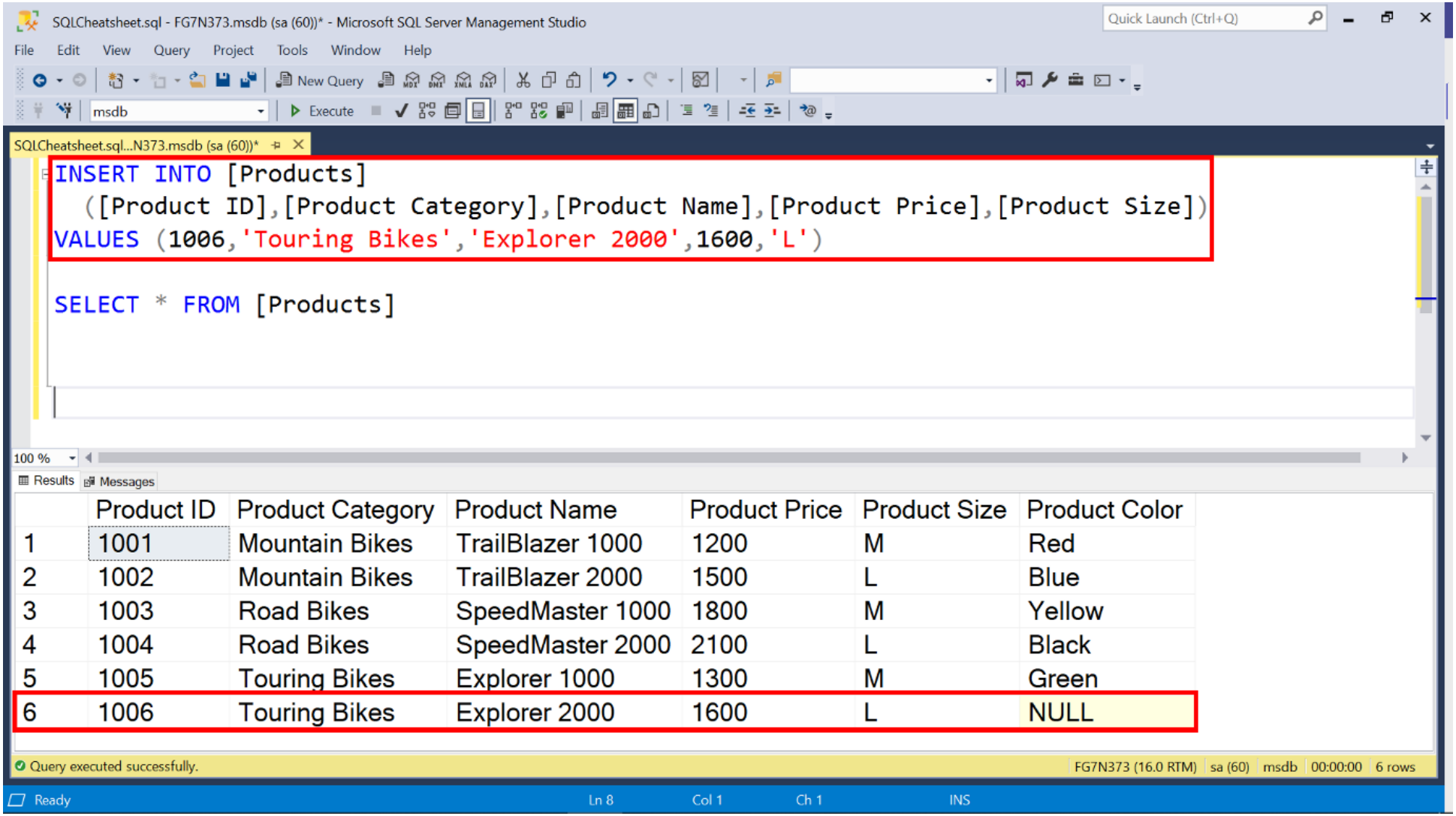
Data modification

With SQL, it is also possible to modify the results of your queries using data modification functions. Examples of these functions include the **INSERT**, **UPDATE**, and **DELETE** clauses. You can learn more about clauses in the examples that follow.

The INSERT clause

You can add new data to your SQL tables using the **INSERT** or **INSERT INTO** clause. When using this clause, specify all columns to which values must be assigned. Ensure you assign values in the correct order. If you omit a value, then the column remains blank.

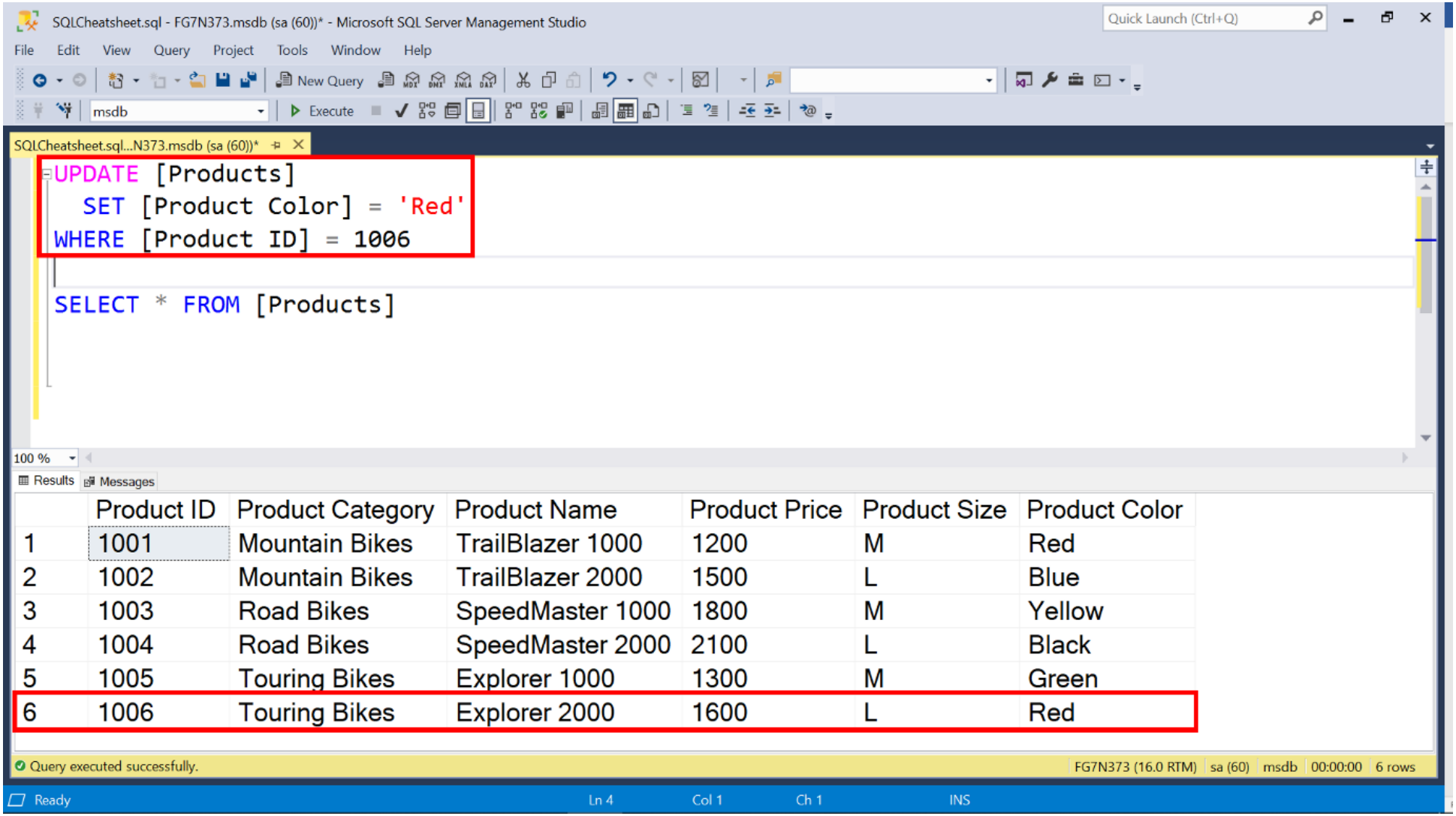
In the example below, several values are added for a new product called **Explorer 2000**. However, no value is specified for the **Product Color** column, so it's left blank or **NULL**.



The UPDATE clause

You can update values in a column using the **UPDATE** clause. However, it must be used with the **WHERE** clause to specify which rows the update should occur within.

In the following example, the value of **Red** is added to the **Product Color** column for the new product with a **Product ID** of **1006**.

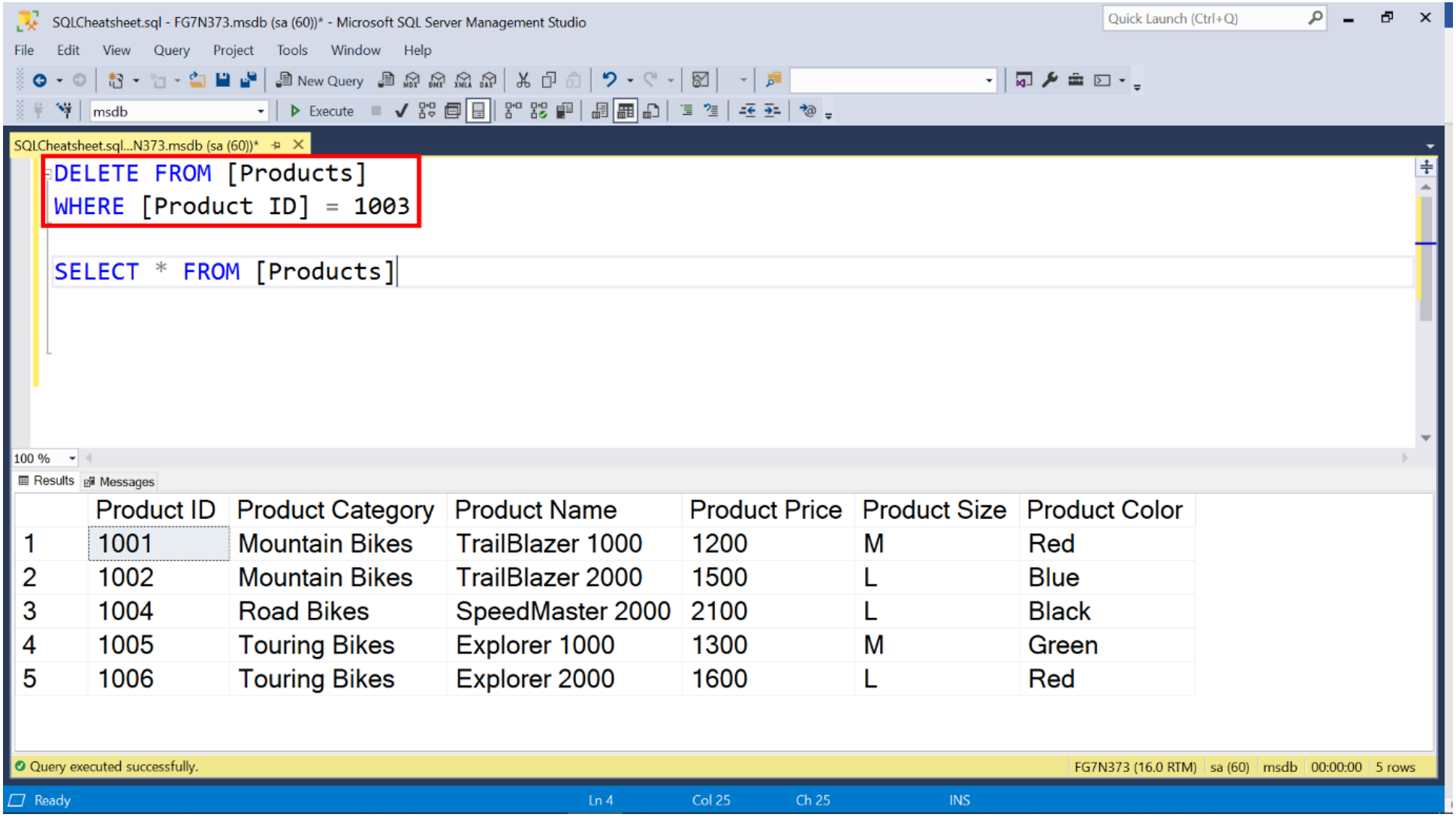


The DELETE FROM command

You can remove or delete data from a SQL table using the **DELETE FROM** command. Like with **UPDATE**, you must also use the **WHERE** clause to identify the rows that will be affected.

Be careful when using **DELETE FROM** in a SQL database. There's no undo button, so any changes you make are permanent.

In the example below, the **DELETE FROM** command instructs SQL to **DELETE** all records related to **Product ID 1003 FROM** the **Products** table. Note that **1003** is no longer in the results set.


SQLCheatsheet.sql - FG7N373.msdb (sa (60))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Query Project Tools Window Help
New Query
msdb
Execute
SQLCheatsheet.sql...N373.msdb (sa (60))*
DELETE FROM [Products]
WHERE [Product ID] = 1003
SELECT * FROM [Products]
100 %
Results
Messages
Product ID Product Category Product Name Product Price Product Size Product Color
1 1001 Mountain Bikes TrailBlazer 1000 1200 M Red
2 1002 Mountain Bikes TrailBlazer 2000 1500 L Blue
3 1004 Road Bikes SpeedMaster 2000 2100 L Black
4 1005 Touring Bikes Explorer 1000 1300 M Green
5 1006 Touring Bikes Explorer 2000 1600 L Red
Query executed successfully.
FG7N373 (16.0 RTM) sa (60) msdb 00:00:00 5 rows
Ready
Ln 4 Col 25 Ch 25 INS