**The lambda expression** defines an anonymous method that the method can call back.

The correct syntax for using a lambda with a method would look something like this:

methName(par1, par2, par3, (result) =>

{

return sth + result;

});

Setup Database Connection: Establish a connection to– your database. This usually involves creating a connection object with your database connection string.

Create a Command Object: This object is used to execute the stored procedure.

Set Command Type and Name: Specify that the command is a stored procedure and set the name of the stored procedure.

Add Parameters to the Command Object: If your stored procedure expects parameters, you'll need to add them to the command object. In your case, the parameter is likely the DataTable you've created.

Execute the Command: Run the stored procedure using the command object. Depending on what the stored procedure returns, you might execute it as a non-query, scalar, or reader.

Handle the Results: Process any results returned by the stored procedure. This could involve reading data, getting output parameters, or just confirming successful execution.

A screenshot of a computer program

Description automatically generated

––––A screenshot of a computer

Description automatically generated

**Func<IDataReader, T> reader**: This is a delegate (function pointer) to a method that takes an **IDataReader** as input and returns an object of type **T**. The purpose of this ––––parameter is to provide a way to process the data returned by the stored procedure. When you call **ExecuteReader**, you must provide a function that knows how to read the data from the **IDataReader** and convert it into the type **T**. This is where the flexibility comes in: you can define any function that adheres to this signature to process different kinds of data in different ways.

* In your call example: **(IDataReader reader) => {...}** is a lambda expression that defines this function. It means you are providing an anonymous function that takes an **IDataReader** (**reader**) as input, and inside the braces **{...}**, you would write the code to process the data from **reader**.

A close-up of a computer code

Description automatically generated

**Delegates as Parameters**: In .NET, delegates (like **Func<...>**) are used to pass methods as arguments. It's a powerful feature that allows methods to be more dynamic and adaptable. In your case, the **Func<IDataReader, T>** delegate allows your **ExecuteReader<T>** method to work with any processing logic that can read from an **IDataReader** and return an object of type **T**.

A screen shot of a computer

Description automatically generated

**For upserting a single row, using a User-Defined Table Type (UDTT) might indeed be overkill a**

**UDTTs is more evident in bulk operations.**

**ExecuteNonQuery : returns an integer indicating the number of rows affected by the SQL statement, but it does not return any data rows from the database.**

**ExecuteNonQuery**: Use this if your upsert operation does not need to return any data. For example, if you're simply inserting or updating a record and don't need any feedback from the database about the operation, other than the number of rows affected.

**ExecuteScalar**: Use this if you need a single piece of information back from the upsert operation, such as a new ID generated for an inserted record.

**ExecuteReader**: This is less common for upsert operations, but you might use it if your upsert procedure is designed to return a data row or