Mapping Stored Procedure Output to POCO Class

John F. Gnazzo

Introduction

A Stored Procedure is a group of SQL statements that form a logical unit and perform a particular task, and they are used to encapsulate a set of operations or queries to execute on a database server.

Contemporary business applications rely on using Stored Procedures to efficiently manage data from a database.

Recently, while working on a Asp.Net 5, Entity Frameworks 7, .Net Core 1.0 project, I had to digest the results of a stored procedure in my application. Unfortunately, the functionality to execute a stored procedure directly has not been implemented in Entity Frameworks 7 at this time.

This blog will discuss a work around to execute a stored procedure and also to easily map a result set from a stored procedure into a class.

Discussion

After Googling the night away, I came to the realization that ADO.NET database operations using DataSets and DataTables did not work either. Finally, I was able to cobble together workable solution.

The following code uses a generic method **ExecuteStoredProcedure**, which takes a stored procedure name, and a list of parameters, as arguments. This method calls another generic method **DataReaderMapToList** to perform the actual mapping.

See the actual code for both methods below:

The Code

```
private readonly MyDbContext _context;

public DatabaseServices(MyDbContext context)
{
    _context = context;
}

public List<T> ExecuteStoredProcedure<T>(string storedProcedure,
      List <SqlParameter> parameters ) where T : new()
{
      using (var cmd =
         _context.Database.GetDbConnection().CreateCommand())
     {
        cmd.CommandText = storedProcedure;
        cmd.CommandType = CommandType.StoredProcedure;

        // set some parameters of the stored procedure
        foreach (var parameter in parameters)
        {
```

```
                    cmd.Parameters.Add(parameter);
                }

                if (cmd.Connection.State != ConnectionState.Open)
                    cmd.Connection.Open();

                using (var dataReader = cmd.ExecuteReader())
                {
                    var test = DataReaderMapToList<T>(dataReader);
                    return test;
                }
            }
        }

        private static List<T> DataReaderMapToList<T>(DbDataReader dr)
        {
            List<T> list = new List<T>();
            while (dr.Read())
            {
                var obj = Activator.CreateInstance<T>();
                foreach (PropertyInfo prop in obj.GetType().GetProperties())
                {
                    if (!Equals(dr[prop.Name], DBNull.Value))
                    {
                        prop.SetValue(obj, dr[prop.Name], null);
                    }
                }
                list.Add(obj);
            }
            return list;
        }
    }
```

Usage

The code can simply do the mapping as follows:

```
            var Id = 1;
            var databaseServices = new DatabaseServices();
            var sqlParameters = new List<SqlParameter>
            {
                new SqlParameter("@ID",
                    SqlDbType.Int) {Value = Id}
            };

            var resultSet = databaseServices
              .ExecuteStoreProcedure<MyPocoClass>
              ("dbo.ClassificationSettingsCustomerList", sqlParameters);
```

The resultSet object above contains a list of data matching the MyPocoClass. (i.e. List<MyPocoClass>)

Conclusion

Contemporary business applications rely on using Stored Procedures to efficiently manage data from a database.

This blog has discussed a work around to execute Stored Procedures and also to easily map a result set from a Stored Procedure into a class, as the direct execution of Stored Procedures has not yet been implemented in Entity Frameworks 7.0.