How to use SQLHydra

SQLHydra is a library that write you perform basic SQL queries, using a fluent interface.

In order to use this library, you need a class that contains your database connection string and wrapper methods for executing queries. You'll also need to either have your data access classes inherit from this base class, or use this class.

Here is an example of how I've used this library.

All my data access classes inherit from this BaseAccessor class

BaseAccessor.cs

```
using SQLHydra;
using SQLHydra.Interfaces;

public class BaseAccessor
{
    private const Enums.DBTypes _databaseType = Enums.DBTypes.MSSQL2008;

    protected static ICanSetColumnValueToInsert BuildInsertQueryForTable(string 
tableName)
    {
        return 
QueryEngine.ForDatabaseType(_databaseType).BuildInsertQueryForTable(tableName);
    }

    protected static ICanSelectAnyColumnType BuildSelectQueryForTable(string tableName)
    {
        return 
QueryEngine.ForDatabaseType(_databaseType).BuildSelectQueryForTable(tableName);
    }

    protected static ICanSetColumnValueToUpdateOrAddWhereCondition 
BuildUpdateQueryForTable(string tableName)
    {
        return 
QueryEngine.ForDatabaseType(_databaseType).BuildUpdateQueryForTable(tableName);
    }

    protected static ICanApplyToAllRowsOrAddWhereCondition 
BuildDeleteQueryForTable(string tableName)
    {
        return 
QueryEngine.ForDatabaseType(_databaseType).BuildDeleteQueryForTable(tableName);
    }
}
```

I also have a static class, with extension methods for the final methods in the fluent interface.

Persistence.cs

```
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.Threading.Tasks;

using SQLHydra.Interfaces;

internal static class Persistence
{
    private const int COMMAND_TIMEOUT_IN_SECONDS = 300;
    private static readonly string _connectionString;

    static Persistence()
    {
        _connectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["SQLServer"].ConnectionString
;
    }

    internal static void ExecuteNonQuery(this ICanGetSQLCommandObject
sqlCommandObjectCreator)
    {
        SqlCommand command = sqlCommandObjectCreator.GetSQLCommandObject();

        using(SqlConnection connection = new SqlConnection(_connectionString))
        {
            command.Connection = connection;
            command.CommandTimeout = COMMAND_TIMEOUT_IN_SECONDS;
            connection.Open();

            command.ExecuteNonQuery();
        }
    }

    internal static void AsyncExecuteNonQuery(this ICanGetSQLCommandObject
sqlCommandObjectCreator)
    {
        using(Task task = new Task(() => ExecuteNonQuery(sqlCommandObjectCreator)))
        {
            task.Start();
        }
    }

    internal static object ExecuteScalarQuery(this ICanGetSQLCommandObject
sqlCommandObjectCreator)
    {
        SqlCommand command = sqlCommandObjectCreator.GetSQLCommandObject();
        object result;

        using(SqlConnection connection = new SqlConnection(_connectionString))
        {
            command.Connection = connection;
            command.CommandTimeout = COMMAND_TIMEOUT_IN_SECONDS;
            connection.Open();

            result = command.ExecuteScalar();
        }
        return result;
    }

    internal static DataSet ExecuteQuery(this ICanGetSQLCommandObject
sqlCommandObjectCreator)
    {
        SqlCommand command = sqlCommandObjectCreator.GetSQLCommandObject();

        using(SqlConnection connection = new SqlConnection(_connectionString))
        {
            command.Connection = connection;
            command.CommandTimeout = COMMAND_TIMEOUT_IN_SECONDS;

            using(DataSet results = new DataSet())
            {
                results.Locale = CultureInfo.CurrentCulture;
                using(SqlDataAdapter adapter = new SqlDataAdapter(command))
                {
                    adapter.Fill(results);

                    return results;
                }
            }
        }
    }
}
```

Then, in the code for my data accessors, I have methods like this:

```
using System;

public class ContactAccessor : BaseAccessor
{
    public void SendMessage(string name, string emailAddress, string message)
    {
        BuildInsertQueryForTable(Constants.TableName.USER_MESSAGE)
            .SetColumnToValue("ID", Guid.NewGuid())
            .SetColumnToValue("Name", name)
            .SetColumnToValue("EmailAddress", emailAddress)
            .SetColumnToValue("MessageText", message)
            .SetColumnToValue("CreatedOn", DateTime.Now)
            .ExecuteNonQuery();
    }
}
```

FUTURE POSSIBILE UPDATES

- Change Where method from one that accepts the column name, comparator, and value, into a more fluent style, such as “Where(columnName).IsEqualTo(value)”
- Make a similar change for SetColumnToValue, break it into “SetColumn(columnName).To(value)”
- Add in more complex Where conditions – ORs, parentheses, correlated sub-queries.