Microsoft Entity Framework Overview

.NET is a framework that utilizes Object oriented programming languages such as C# and VB.NET. One of the greatest challenges object oriented languages is persisting and saving object state. In general the most widely utilized storage mechanism for .NET applications is the Microsoft SQL Server Database, which is a relational database system. By their very nature, objects created from object oriented programming do not easily save into relational databases without a developer creating logic to translate between the object application code and relational database storage structures.

Traditionally a developer would have to manually create the same basic translating logic over and over again to communicate between the object application code level and the SQL database level. This logic is seen as very boiler plate, time consuming and often consists of very basic SQL Create, Read, and Update Delete (CRUD) commands to populate, update, add or delete data for object state.

In the past the only way to perform theses communications was to use the ADO.NET .NET class libraries to communicate with backend databases. Although ADO.NET is very robust and provides many ways to communicate with backend databases, it is still up to the developer to create the database structures and the logic to translate between the two layers. Because it is up to the developer on how to develop the above translation and structures, each application’s level of abstraction, relational database models and object code differed greatly due to the skill set of each developer.

As .NET has neared over 11 years old the pool of .NET developer skills and abilities and range greatly. Some developers may have started out developing relational databases and have very strong SQL querying and relational database design skills and later picked up .NET object oriented programming. Other developers may have started developing in .NET object oriented programming and have never programming against a relational database or written any SQL. The point being is that this wide range of skillsets produced .NET applications database access layers that varied wildly and were very inconsistent because there was no standard framework or guidance on how to communicate between the application code level and the backend database level.

The introduction of 3rd party ORM tools about half way into the life of .NET attempted to provide a standardized way for application code to communicate with databases in a very object oriented manor. One of the primary features of ORM tools is for the developer to focus primarily in the application level without needing to worry too much about developing at the relational database level. Microsoft eventually released its own ORM named entity framework late in the game. The first versions of Entity Framework were deemed unusable and riddled with so many issues that the development community gave it a vote of no confidence. Eventually with the release of version 2 the development community approved of Entity Framework and began utilizing it.

Microsoft’s Entity framework is an ORM tool which is tightly integrated with Visual Studio. Within visual Studio the Entity framework allows the developers three conceptual and visual views of how application and databases communicate and co-exist with each other. The basic conceptual and visual views are the entity ORM generated classes within the .NET code; the entity ORM Visual Studio diagrams of the entity object (which are based on XML configuration files) and the entity ORM generated database schema and tables.

Using the three Entity ORM conceptual and visual views Microsoft has provided great flexibility on how to develop and automatically generated the data access layer to communicate between the application and database.

Developers can begin creating the data access layer at any of the three conceptual and visual views.

The first way is by using “code first”. If a developer is most comfortable or desires to start working with application code, they can develop their own objects which inherit from an entity base object. By inheriting from the entity base objects the framework will generate the entity xml configuration file. The xml configuration file will then be used by the framework to automatically create the SQL CRUD logic and actually create tables in the database.

The second way to develop the data access layer visually. If a developer wants to visually layout and design the objects in a UML like manor before coding or creating the database. Developer can visually manipulate the entity nodes in the Visual Studio Entity ORM visual diagram tool which will generate the entity configuration XML file. This file will then generate the entity ORM application objects and the database schema and tables.

The third way is to develop the database first. If the database already exists or if the developer desires to create it first, the entity framework also allows this option. After the database is created the entity framework wizard in Visual Studio will read the database schema, generate the entity ORM XML definition file and then generate the application objects. With this option it is up to the developer to select whether one or all of the database tables in schema will be converted. This is an important note because a developer may not want to convert the entire database or it may not convert due to legacy design constraints.

The above three conceptual Entity Framework methods provides an automated and powerful easy to use tool to create data access layers. Furthermore a developer can come in and make modifications to any of the three conceptual levels and sync them between the object code, entity diagram and database schema. This flexibility allows for developers of different skills and strengths to work on existing or new data access layers with great ease. The only learning curve a developer will need is to transpose his or her skills to the entity framework. A developer could focus on one of the above conceptual layers which match his or her skillset and be as productive as one that knows another.

The entity framework provided a centralized and decoupled conceptual methodology to developing data access layers which can utilize an eclectic pool of developer talents and skillsets. Furthermore because the entity framework is shielded from the database by a layer, you can in theory swap out any type of database or data storage mechanism that is compatible with the entity framework. Overall the entity framework allow for great flexibility in development and maintenance of existing and future applications