# Repository Design Pattern in C#

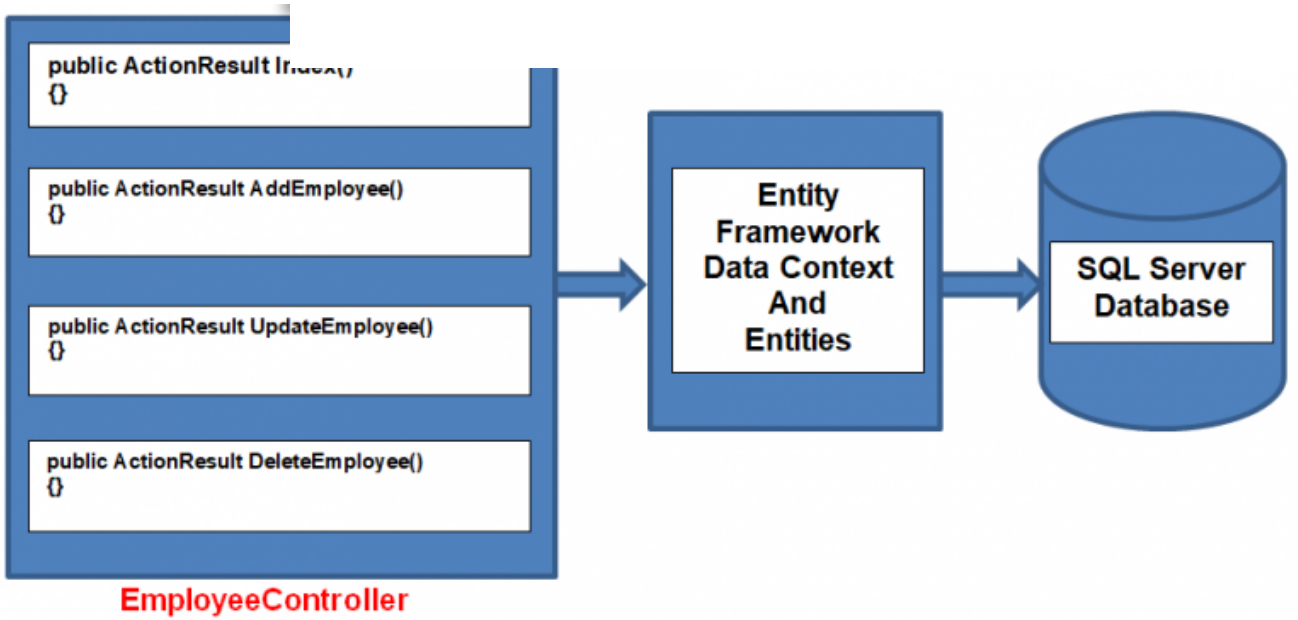
Back to: Dot Net Design Patterns With Real-Time Examples



## Introduction to Repository Design Pattern in C#

In this article, I am going to discuss the basics of the **Repository design pattern in C#** from the context of **Entity Framework** and ASP.NET MVC application. The **Repository Design Pattern in C#** is one of the most used design patterns in the real-time application.

Nowadays, most of the data-driven applications need to access the data residing in one or more other data sources. The easiest or simplest approach is to write all the data access related code in the main application itself. For example, if you have an ASP.NET MVC controller let say EmployeeController. Then the Employee controller class may have many action methods that can perform the typical CRUD (Create, Read, Update and Delete) operations against the underlying database. Let's further assume that you are using Entity Framework for doing all these database related operations. In that case, your application would do something like as shown in the below diagram.




As you can see in the above diagram, the action methods of the Employee controller are directly interacting with the Entity Framework data context class and execute the queries to retrieve the data from the database. They also perform the INSERT, UPDATE, and DELETE operations using the data context and DbSet. The Entity Framework in turn talks with the underlying SQL Server database.

## The drawback of Above Implementation:

The above implementation works as expected. But it suffers from the drawback that the database access code (i.e. creating the data context object, writing the queries, manipulating the data, persisting the changes to the database, etc.) is embedded directly inside the controller action methods. This design or implementation can cause code duplication and further, we need to change the controller even if we do a small change in the data access logic.

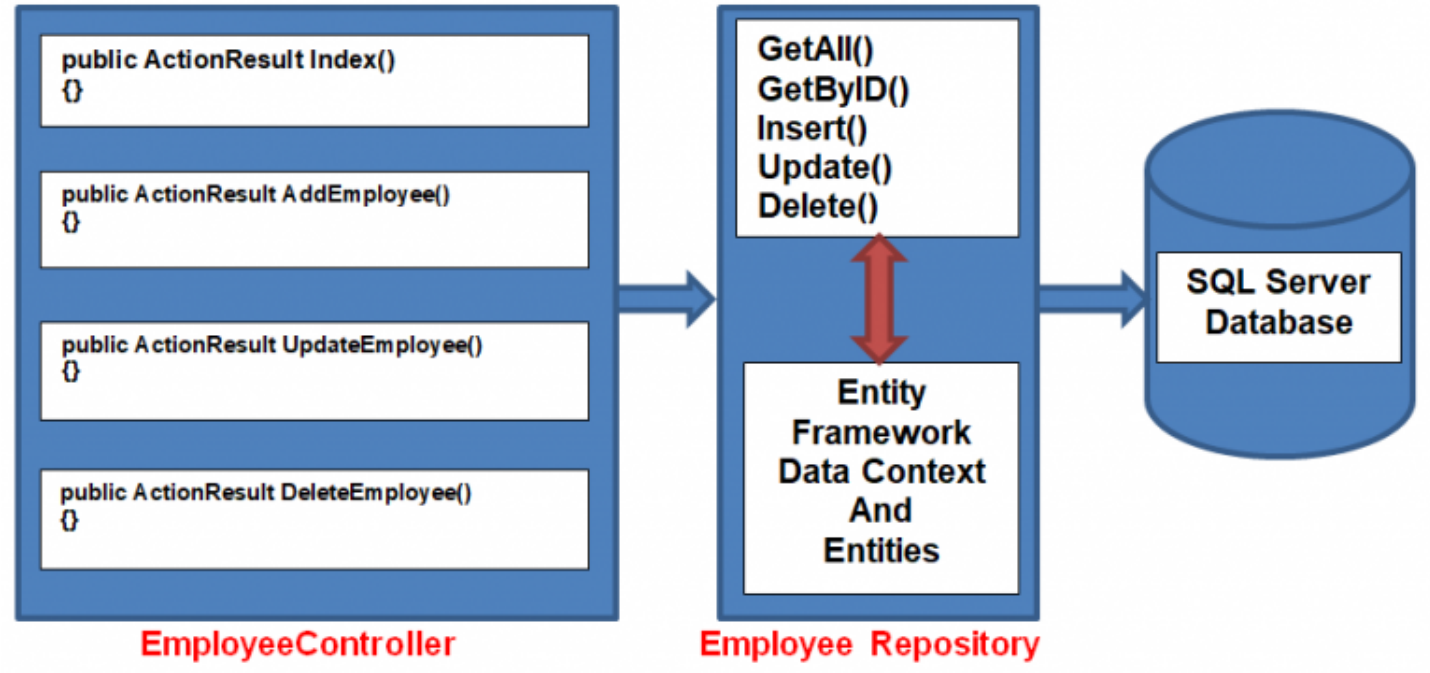
For example, if the application is modifying the employee information from two controllers, then each controller will repeat the same data access code. And future any modifications also need to be done at two places i.e. the two controllers where we write the same data access code.



## What is the Repository Design Pa

The **Repository Design Pattern in C#** Mediat
domain objects.

In other words, we can say that a **Repository Design Pattern** acts as a middleman or middle layer between the rest of the application and the data access logic. That means a repository pattern isolates all the data access code from the rest of the application. The advantage of doing so is that, if you need to do any change then you need to do in one place. Another benefit is that testing your controllers becomes easy because the testing framework need not run against the actual database access code. With a repository design pattern introduced, the above figure can be changed to:




In the above design, now the Employee controller won't talk with the Entity Framework data context class directly. Also, now there is no queries or any other data access code in the action methods of the Employee Controller. All these operations (i.e. CRUD operations) are wrapped by the Employee repository. The Employee repository uses the
diagram, now the Employee repository has met
Typical CRUD operations against underlying dat

## Why we need the Repository Design P

As we already discussed, nowadays, most of the data-driven applications need to access the data residing in one or more other data sources. Most of the time data source will be a database. Again, these data-driven applications need to have a good and secure strategy for data access to perform the CRUD operations against the underlying database. One of the most important aspects of this strategy is the separation between the actual database, queries and other data access logic from the rest of the application. In our example, we need to separate the data access logic from the Employee Controller. The **Repository Design Pattern** is one of the most popular design patterns to achieve such separation between the actual database, queries and other data access logic from the rest of the application.

In the next article, I will discuss **how to implement repository pattern in ASP.NET MVC application** with a real-time example.

### SUMMARY

In this article, I try to explain the basics of the **Repository Design Pattern in C#**. I hope this article will help you with your need. I would like to have your feedback. Please post your feedback, question, or comments about this article.

## Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Post Comment



**SINGLETON DESIGN PATTERN**

- Singleton Design Pattern in C#
- Why Singleton Class sealed in C#
- Thread-safe Singleton Design Pattern in C

- Lazy Loading and Eager loading in Single
- Singleton VS Static class in C#
- Singleton Design Pattern Real Time Example in C#

**DEPENDENCY INJECTION DESIGN PATTERN**

- Dependency Injection in C#
- Property and Method Dependency Injection in C#
- Dependency Injection using Unity Container in MVC

**REPOSITORY DESIGN PATTERN**

- **Repository Design Pattern in C#**
- How to implement Repository Design Pattern in C#
- Generic Repository Pattern in C#
- Repository Pattern Implementation Guidelines in c#
- Unit of Work using Repository Design Pattern

**Factory Design Pattern**

- Factory Design Pattern in C#
- Factory Method Design Pattern in C#
- Abstract Factory Design Pattern in C#

**INVERSION OF CONTROL**

- Introduction to Inversion of Control
- Inversion of Control Using Factory Pattern in C#
- Inversion of Control Using Dependency Inversion Principle
- Inversion of Control Using Dependency Injection Design pattern
- Inversion of Control Containers in C#