Repository pattern, should it be 1:1?

Asked 5 years, 4 months ago Active 5 years, 4 months ago Viewed 226 times

0

I'm currently working on a project in which I'm using the Repository Pattern. Currently for each table in my db, I'm building out a repository. As the db grows this is becoming a bit tedious, and i'm wondering if I need to do is for each table. The users have requested the ability to edit all tables, including those which are just used in dropdowns. For example, we have a SubContractor table, which has a link to a WorkLocation table. SubContractors also have a link to a TypeOfWork table, that section alone has me building out 3 repositories. I initially thought of only building a SubContractor one, but again the users would like to edit any of them, so we built each one a repository. Is this common?

c# asp.net design-patterns

asked Sep 22 '14 at 13:59
Paritosh
3,492 6 31 65

2 Don't you have some sort of a generic base repo for basic CRUD operations? – Groo Sep 22 '14 at 14:01

2 Generics are your friend! You can create a `Repository<T>` where `T` is an entity type (i.e., an object that represents a table entry). – Roy Dictus Sep 22 '14 at 14:06

2 This article might help if you're using EF: "Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application" (you might need to jump down to the "Implement a Generic Repository and a Unit of Work Class" chapter). – Groo Sep 22 '14 at 14:12

1 DBContext's DBSets *are* the repositories. Do not wrap them for the sake of wrapping. Repository pattern was useful in the good ol days when ORM frameworks didn't exist. – Mrchief Sep 22 '14 at 14:17

1 @Mrchief I'm not agree. Repository pattern is still required because you don't want to couple your domain with a specific OR/M. The OR/M (the data mapper) is what repository encapsulates. – Matías Fidemraizer Sep 22 '14 at 14:22

4

Actually you shouldn't create a repository-per-table, because a repository is pattern to encapsulate how domain objects get translated to another data representation (for example, a relational database), and also translates them again into domain objects.

First of all, 1 table may or may not be 1 domain object. OR/M frameworks like Entity Framework or NHibernate are more than just mapping objects to tables.

A domain object might have associations with other domain objects of the same domain, and this means that a domain object might be persisted to one or more tables depending on the relational design behind the repository.

Also, there's a common practice where you implement repositories for *root* entities only. For example, there's an `Company` domain object which has many `Employee`. You should design a `ICompanyRepository` and code an implementation on top of your favourite OR/M, and `Employee` creation would be done by adding employees to `Company.Employees` *1-n* association:

```
ICompanyRepository repo = new CompanyRepositoryImplementation();
Company myCompany = repo.GetById(839984);
myCompany.Employees.Add(new Employee { FullName = "Matias Fidemraizer" });
```

A good full OR/M like Entity Framework or NHibernate will persist associations and the simple act of adding an employee to `Employees` association will issue an `INSERT` to create the whole employee, and you'll be able to query and obtain that employee using LINQ (or any other object query approach).

Finally, you should try to implement a generic repository where `GetById`, `GetByCriteria` (custom criteria using an expression tree/LINQ), `Add` and `Remove` (*no* `Update` please, this is also handled by OR/M transactions when you modify an object or collection) that both can be work *as is* or it can be derived in order to provide specific domain requirements.

Generic repository signature would look like the following sample:

```
public interface IGenericRepository<TDomainObject>
   where TDomainObject : DomainObject
```

answered Sep 22 '14 at 14:39

Matías Fidemraizer
54.6k 13 95 167