Industry best practices.Repository Design Pattern, Model View Controller design pattern, Kendo Controls, Version Control System, and Entity Framework

Prepared, written and edited by Jinan Kordab

In repository design pattern, Unit Of Work, which is responsible for database Insert, Update, and Delete operations, updates, (including delete) and inserts records into tables of database of your choice using Transactional Approach, meaning either all transaction commits, or if something happens, the whole transaction rolls back, and no table is affected. For example, you have three tables. Table A, table B, and table C. Table C has two foreign keys from table A and B. In other words, we have one to many relationships, A to C, and another one to many relationship from B to C. I want to add a new (for the sake of simplicity for this example) record in A , and another record into C.
With regular database operations, I would do this:

1-InsertAndCommitToA();
2-InsertAndCommitToC();

And thus, we will have one record in A and another related record in C. The fact of the matter, is that those are two separate sole, with time difference, encapsulated operations. Each one of them has a beginning, and an end:

1-Begin, End
2-Begin, End

By using UnitOfWork in repository model, which in its turn (UnitOfWork), one can extend its functionality to an interface, adding additional functionality , those two operations listed above are performed AS IF it is one operation, and if something goes wrong, all operation rolls back, and no change is commited to the database.

Ex:
tableA.Add(record);
tableC.Add(record);
UnitOfWork.Save();

This approach helps us prevent having inconsistent records in some of our tables in database.

Another thing worth mentioning here, is that when only updating a record in a database, and to avoid duplication of records, check if the changed record exists in ALL OTHER rows in a table, EXCLUDING the row that is being edited. This minor details is rarely noticed, because not much unit testing is done to all parts of an application, generally speaking, concerning the speed of returned records, which is in milliseconds. But the difference, is very much noticed with slow internet connection.
Since we are still on the topic of databases, lets consider entity framework as well, a little bit. We have a DataModel, and in case of Telerik's DataModel, it is represented with .rlinq extension. This datamodel has many useful operations, where with two clicks of a mouse, you can update all your model drom database, as well as the other way around, update databse from your model. One of your team mates adds tables to the database. Best practice is to update the model from the database, and not get latest from your solutions's source control, after your team mate checks in his latest updates and commits, because each DataModel has application and environment specific and independent settings that would be overriden, if one gets latest (in some cases), especially with table mappings. To summarize, it is always best practice to update your data .rlinq Model from database directly.

Another best practice concerning databases, is related to same information that is needed to be present, and related to each single entity in a database, such as dateCreated, dateInserted,

nameOfPersonWhoInserted. Never add those fields to ALL database tables ! Because it will break normalization rules for the tables, and it will increase the size of the database very much ! Best practice is to create a separate group of related tables, normalized, that will store all this information.
As for Kendo UI ASP.NET MVC Razor controls, and especially with Kendo Grid Control, best practice is to make sure one has defined and passed the ID or primary key of your model in (Model, View, Controller) to the Kendo Grid, as it does not find and bind automatically.
Another best practice to mention, is when all the needed data is passed to Kendo Grid, including complex data types, and not only primitive arrays, when passing the data to Edit or Add Action Result in Controller to perform some operations on, all data should and must be SERIALIZED ! It can be done using JavaScript.

Thank you,

Jinan Kordab
softuniversum.com