NOTReallyORM

Framework

User Documentation

Version 0.6

www.notreallyorm.com

1. Introduction

"It's time to become a real modern software company. We should begin to use an ORM framework in our new projects, maybe the latest version of the Microsoft Entity Framework. Everybody uses ORM nowadays, Microsoft pushes it, why we don't?" That's what I told the colleagues in my team some months ago.

It was the time when I began to read books about Microsoft Entity Framework and set up some experimental applications. I was fed up with writing boilerplate code for simple CRUD operations and hoped for some magic that was promised when you use these Entity Objects with data binding. Unfortunately the members of my team were not that enthusiastic about my ideas. You must know: We do a lot with SQL in our development, and we never made bad experiences with these "old school" handmade queries.

After some discussions I thought about my "modern" point of view and asked Mr. Google about some critical writing on ORM. One of the most inspiring texts for me was Kenneth Downs' essay "Why I do not use ORM" in his blog. There you also find a newer version with some other aspects. Well, this was only the begin of the journey. But it was one of many impacts that led me to start this blog and to try an alternative with the NotReallyORM framework.

From my blog, posted at (here you also find the referenced links):

http://www.notreallyorm.com/93/why-i-do-not-use-orm-or-the-beginning/

The NotReallyORM framework tries an approach that is near to the database: You should always know what happens between your framework and the database. But it also tries to provide some convenience especially for the always repeating, sometimes boring day by day tasks: Retrieve data from a database, fill a form, validate the input, write the form back to the database ..... and so on.

Try it out, find bugs, improve it. And if you have some time spared I would be happy for some feedback on my blog.

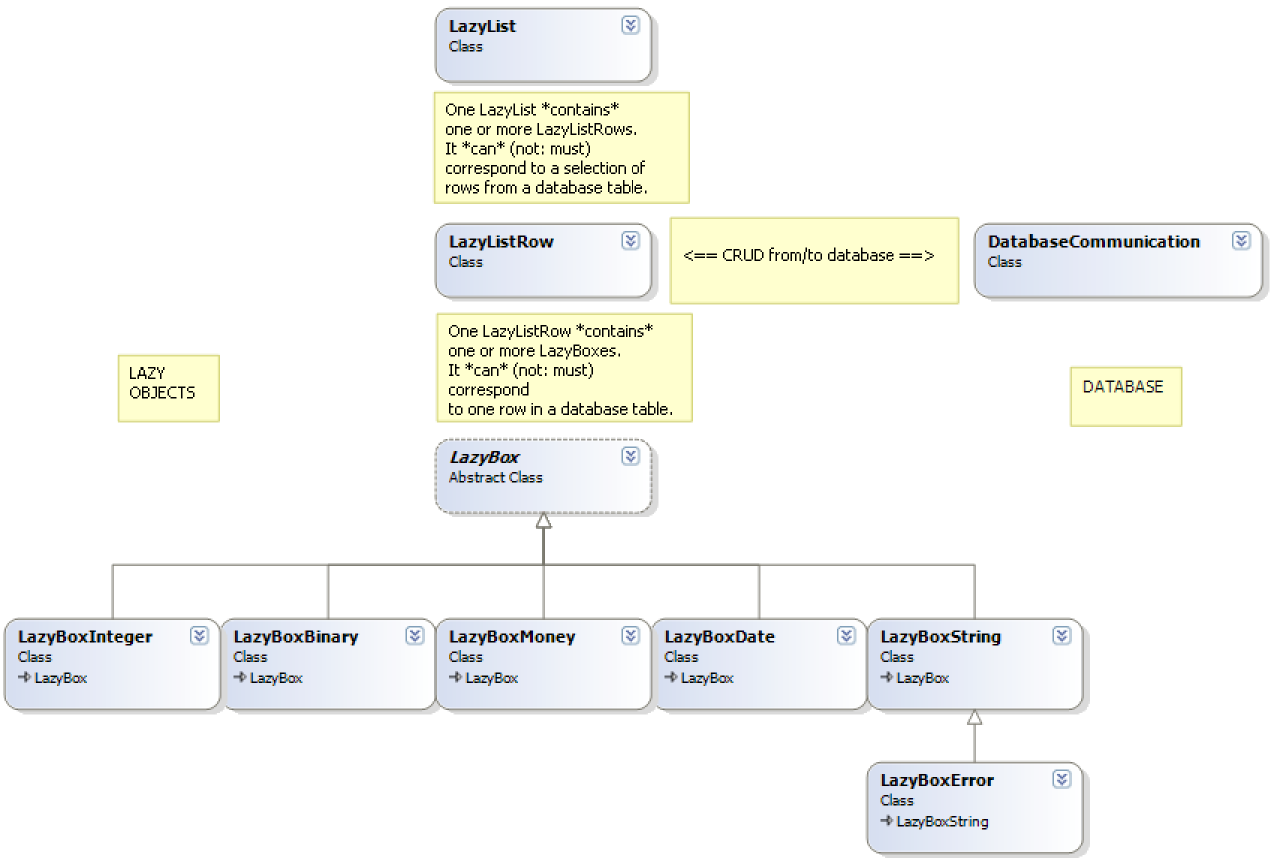
2. License

Like the DotNetOdf library the NotReallyORM framework is distributed under the *Apache 2.0 license*. You are encouraged to use it in both free or commercial applications.

3. Installation

For usage you should have installed at least version 3.5 of the .NET framework. Unzip the NotReallyORM.zip package into your path. In your project set a reference to NotReallyORM.dll and import the namespaces NotReallyORM and NOTReallyORM.LazyObjects. Make sure that the DLL is found by your application at runtime (for example in your setup program).

4. Principles


LazyList
Class
One LazyList *contains* one or more LazyListRows. It *can* (not: must) correspond to a selection of rows from a database table.
LazyListRow
Class
<== CRUD from/to database ==>
DatabaseCommunication
Class
One LazyListRow *contains* one or more LazyBoxes. It *can* (not: must) correspond to one row in a database table.
LAZY OBJECTS
DATABASE
LazyBox
Abstract Class
LazyBoxInteger
Class
LazyBox
LazyBoxBinary
Class
LazyBox
LazyBoxMoney
Class
LazyBox
LazyBoxDate
Class
LazyBox
LazyBoxString
Class
LazyBox
LazyBoxError
Class
LazyBoxString

The first thing you must know is that the NotReallyORM framework has two sides:

- The **LazyObjects** (LazyList, LazyListRow, LazyBox and subclasses): These objects store the data, they know some information about the stored database fields (is it integer or string or..., is it a primary key etc.), the table and the database they came from. They even know how to create SQL statements for standard the CRUD (Create, Read, Update, Delete) operations.
  But they *don't know how to communicate with the database*. Consequently they *do not depend on namespaces like System.Data.SQLClient*. And not to forget: They *do not track any changes made in the database tables they came from* after they are instantiated. You also can instantiate them without any reference to a database.
- The **DatabaseCommunication** Class: These objects know how to connect to the database, they can create LazyObjects from a given SQL string, they know how to perform CRUD operations for LazyObjects, they provide a special transactional context ("LazyTransaction") and they have helper functions to create datatables from a given SQL.
  In a multi-threaded environment (for example on a webserver) you can create one DC object per thread and you do not have to worry about mixing up different LazyTransactions.
  The recent version only supports MS SQLServer, but it is possible to expand the class to use MySQL or other databases. Remember: the LazyObjects will not care about the database server, as long as you use standard SQL.

5. A Simple Example

Now you need an SQL Server, a database and a table with one or more primary keys. Let's say you want to save a number of binary stored pictures with some additional data, and your table "Documents" in the database "Test" has the fields:

- documentID (Integer, Primary Key)
- createdTime (smalldatetime)
- price (money)
- name (varchar(255))
- data (varbinary(max))

Open your project in Visual Studio and add a reference to "NotReallyORM.dll". Import the namespaces "NotReallyORM" and "NotReallyORM.LazyObjects".

Now define an SQL string to retrieve all rows:

```
String mySelect = "SELECT documentID, createdTime, price, name, data
FROM Test.dbo.Documents";
```

Retrieve a DatabaseCommunication object:

```
DatabaseCommunication dbContext = new
DatabaseCommunication(@"Server=SIMON-PC\SQLEXPRESS;User
ID=sa;Password=martin");
```

Build a template for a LazyListRow from the select string. You will need this later to perform easy CRUD operations:

```
LazyListRow myTemplateRow = dbContext.buildEmptyLazyListRow(mySelect);
```

This LazyListRow contains a list with different LazyBoxes, according to the field types in the database: LazyBoxInteger (for DocumentID), LazyBoxDate (for createdTime), LazyBoxMoney (for price), LazyBoxName (for the name), and LazyBoxBinary (for the data). The factory function knows to instantiate the appropriate subclasses of LazyBox by evaluating the schema information in the database.

Now you have to tell the template additional information about the database table (maybe in later Version this will be done automatically):

```
var primaryKeys = new List<String>() { "First_ID", "Second_ID" };
myTemplateRow.updateAutoCRUDInfo(myQualifiedTableName: "dbo.Copies",
myDatabaseName: "LazyTest", primaryKeyList:primaryKeys);
```

It's to retrieve some real data. Here we select it by the primary key (in a real life project you will have functions to built those strings more convieniently).

```
String myPrimSelect = mySelect + " WHERE DocumentID = 3";
LazyListRow myCurrentRow = dbContext.buildLazyListRow(myPrimSelect);
```

If there is no row with DocumentID = 3, myCurrentRow will be null. Check it before you proceed. Copy the necessary information for later CRUD operations from your template row:

```
myCurrentRow.updateAutoCRUDInfo(myTemplateRow);
```

We now want to change some data and restore it in the database. You get the boxes in a LazyListRow by its database field name:

```
LazyBox myCurrentBox = myCurrentRow.box("price");
Boolean parseOK = myCurrentBox.TryParseStringToData("19.00");
```

The function TryParseStringToData ensures the right input format (from a textbox control for example). Alternatively you can set the data directly (check the myCurrentBox.MyFlavour property first):

```
myCurrentBox.setData(new Decimal("19.00"));
```

With a LazyBoxBinary this is the only way to change the data, TryParseStringToData will throw an exception.

Store myCurrentRow together with the changed price back to the database:

```
dbContext.updateLazyListRow(myCurrentRow);
```

When you check the function LazyListRow.getUpdateSql() you can see that all fields are updated - not just "price". As mentioned above the LazyObjects do not track the changes. This is less comfortable than in "really" ORM frameworks, but there is also much less overhead. And there is much more control: You always know what happens between your LazyObjects and the database (when you know the .NET Entity Framework you know what I mean...).

As you can see the Auto-CRUD operations only refer to a LazyListRow. There is no way to do it directly with a LazyBox. If you want to do this you have to build a LazyListRow with just one LazyBox in it. This reminds to the corresponding structure in the database. There you also deal always with complete rows - even if you just select one field.

###

To learn more have a look at NOTReallyORM_TestWinforms.Copies.

If you want to learn more about binary fields and about LazyTransactions have a look at NOTReallyORM_TestWinforms.Form1.

Try out the LazyListRow.toDictionary() function together with the DotNetOdf reporting tool (see the example in the User Manual there).

More documentation to be expected...