770 - Database-driven Applications - Task Set 3

Introduction

This task set uses a DVLA database based on the requirements specified in course 120, which contains cars, drivers, cameras and the resulting fines.

You will now create a C# application to display and modify the data in that database, using a tiered model.

All interactions with the database should be via the Entity Framework.

Task 1 – Create the Projects

**Solution/project setup**

1. Create a new Windows Forms application – this will be the presentation tier.
2. Add a new Class Library project to the solution to hold the Entity Framework model.
3. Add another Class Library to contain any additional models / business classes you may need.
4. Add a database project to the solution, and import the Initialise Database.sql script from the Task Set 3 Scripts task folder:
   1. Right-click on the database project and select **Import -> Script**.
   2. Click the **Next** button on the welcome page.
   3. Click the **Browse** button and select the Initialise Database.sql script.
   4. Click the **Finish** button to import the table and constraint definitions into the project.
   5. You should now be able to view all of your database tables, views, stored procedures etc within the database project. When selecting a script you should see a split designer/code view. When making changes you can use whichever you prefer and the other will update automatically.

*See the* [*article in the knowledgebase*](https://intranet.dorsetsoftware.com/Lists/Knowledge%20Base/Threaded.aspx?RootFolder=%2fLists%2fKnowledge%20Base%2fDatabase%20projects%20in%20Visual%20Studio%202010) *for details of how database projects work in Visual Studio 2010. They have a few more quirks than 2017 which we are using here. One difference worth noting here (also covered in the database programming standards) is that in Visual Studio 2017 database projects we are back to needing to set permissions within our scripts. These moved in 2010 but have since moved back again.)*

1. Publish the database project to your local SQL instance:
   1. Right click on your database project and select **Publish…**
   2. Click the **Edit...** button and select the **Browse** tab.
   3. Expand the **Local** option and select your local SQL instance (MSSQLLocalDb).
   4. In the Database Name textbox enter a name for your database (DVLA), and click the **OK** button.
   5. Click the **Publish** button. This will create a new database based on the schema defined in your database project.
2. You will likely want to delete the default classes from some of your projects and set appropriate namespaces. Add references between the relevant projects (nothing needs to reference the database project).

**Entity Framework Setup**

1. Right click on your Entity Framework project and choose **Add -> New Item** then select **Data -> ADO.NET Entity Data Model** choosing a sensible name (e.g. DvlaEntities).
2. On the next screen select **EF Designer from database** (note here that there are other ways to use the Entity Framework - we are going to generate a model from an existing database, but equally you can write code first and use that to generate a database schema).
3. Select the database published in step 5 on the next screen and continue. If given the option choose the most recent version of Entity Framework.
4. Lastly, select the objects you want to include in your model. For now we will only include the Car table. Once the import has completed you will be able to find the generated classes under the newly added model (DvlaEntities.edmx), similarly to Task Set 1.
5. Right-click on the Solution and choose **Manage NuGet Packages for Solution,** select **Installed Packages,** and press **Manage** next to the Entity Framework**.** This will give you the option to add the package to your other projects, which you should do.
6. Copy the connection string from your Entity Framework projects’s App.Config to that in your presentation project.

Task 2 – Create View Car Screen

Create a new screen in the presentation tier with relevant controls to display all the Cars in the system. Then create a new screen to display all the properties of a Car and populate the controls from these properties. Link these screens together so that selecting a Car from the list opens up the full details in another window.

Task 3 – Enable Editing of Cars

Write code in an appropriate place (or places) that will take the values from the car details screen and update the relevant car in the database.

Extend your application so that it is also possible to insert new cars. You should find that you can reuse a lot of your existing code.

At this point, having only included the Car table in the Entity Framework model you are probably limited to having the user manually enter the ID of things such as the Make and Model.

Task 4 – User Friendly Editing

In order to let the user edit a car without needing to remember database identifiers we first need to be able to represent the related entities in our code as objects. This is achieved by updating the Entity Framework model to include the relevant database tables.

Double click on the Entity Framework model file to open the entity relationship diagram. Right click on an empty space here and select **Update Model from Database**. On the **Add** tab of the dialog, select any tables you need (probably Make, Model and Owner) and click finish. Note that there are now auto-generated classes for each of these tables, and importantly that they have navigation properties for each of their relationships (e.g. Car should now have a Model property, and Model should have a Cars property).

Alter your car details form so that the user can change the car’s make, model and owner using sensible controls, and without having to ever see database identifiers.

**Hint:** A search form might provide a much nicer way to select a car’s owner than built-in controls.

Task 5 – Editing Owners and Cameras

Add a screen to your application that allows the user to edit the details of a person. They should also be able to edit the person’s address, which may require you to add further tables to your Entity Framework Model.

Notice that the class representing a person contains a “navigation property” that will return all the Cars owned by that person.

Add to this screen a table that shows summary details of each car owned by the person. Make it so that clicking a row in this grid opens the relevant car details screen.

Follow the same steps as in Task 4 to add the Camera table to your Entity Framework model, along with any others that you will require to enable creation and editing of cameras.

Create screens to enable viewing and editing of cameras.

Task 6 – More Navigation Properties (Sightings)

Use the navigation properties provided by the Entity Framework to populate a grid on the car screen with sightings of that car, including some details of any fine that has been raised.

Similarly add a grid to the relevant camera screen showing sightings it has made, including the same fine information as on the car screen.

You may need to update your model again to include some more tables depending on your database structure (e.g. Sighting or CameraCar, Fine)

It should now be possible to link the various screens together – clicking on a row in a grid opens the relevant screen to see more information about that item.

Note that using the Entity Framework you don’t need to explicitly provide methods to list related entities, the auto generated navigation properties give this to you for free.

Add a simple main menu screen that allows you to open any of the car list, search person and camera list screens. Bear in mind that when selecting a person from your search screen the desired behaviour will not always be the same (open the person’s details vs select them as the owner of a car). Think about how you can achieve this in an extensible fashion.

Task 7 – Concurrency with the Entity Framework

Run two instances of the application (you can do this by right clicking the project and selecting debug -> Start New Instance). Test that using both of these windows to update the same Owner record causes concurrency issues – the first update may be undone by the second.

Using the database project in visual studio (not SQL Server Management Studio), add a RowVersion column to your Owner table.

In order to apply this change to the database, right click on the database project and select **Publish.** Select your database in the dialog and click **Publish.** If you wish you can now connect to your database in SQL Server Management Studio and verify the change has been applied.

Right click the Entity Framework model and select **Update Model from Database**. This time just click **Finish** without selecting anything.

Once this is complete, select the newly added RowVersion column in the Entity Framework model diagram, and use the properties window to set **Concurrency Mode: Fixed**. This will ensure that any time Entity Framework performs an update or delete on a Customer, it will include the RowVersion in the where clause of the SQL it generates.

**Important:** If you are creating a new database context to perform your save operation, when you load the target record to update you will also be loading the **new** rowversion. This will bypass the desired concurrency check. In this case you need to load the record using **.AsNoTracking()**, manually change the rowversion to the **original** value, and **Attach()** the entity to the context again. Ask if you are unsure on this point.

If this concurrency check fails, an appropriate exception will be thrown depending on the operation attempted. For our purposes here **DbUpdateConcurrencyException** is the relevant one. Handle this and display an appropriate message to the user.