|  |
| --- |
|  |
| Entity Framework Migrations |
| Fluent Schema Changes |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

Migrations

Objectives

After completing this lab, you should understand how to do the following:

* Enable code-first database migrations
* Create migration scripts
* Customize migration scripts

# Part I – First Migration

1. Open the EmployeeTimeCards solution in the before directory of this lab.
2. **Run** the application to ensure the project is in working order.
3. Open Employee.cs from the .Core project.
4. Add a public virtual property to the Employee to store the employee HireDate.

public class Employee

{

public virtual int Id { get; set; }

[Required]

public virtual string FirstName { get; set; }

[Required]

public virtual string LastName { get; set; }

**public virtual DateTime HireDate { get; set; }**

public virtual ICollection<TimeCard> TimeCards { get; set; }

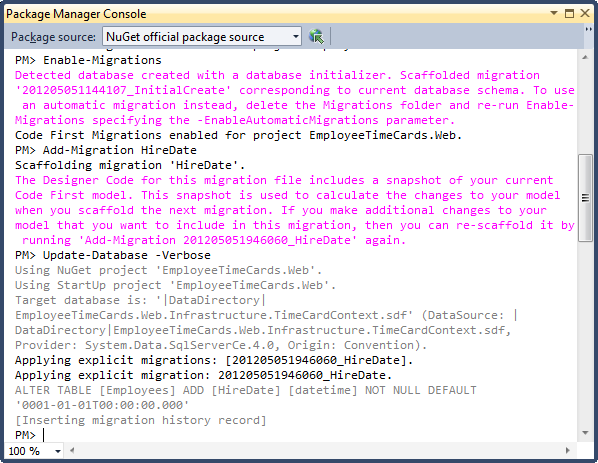
}

1. Run the application – you should see an error because of a bad column name.
2. Open the Package Manager Console window (View -> Other Windows -> Package Manager Console).
3. Type “Enable-Migrations”

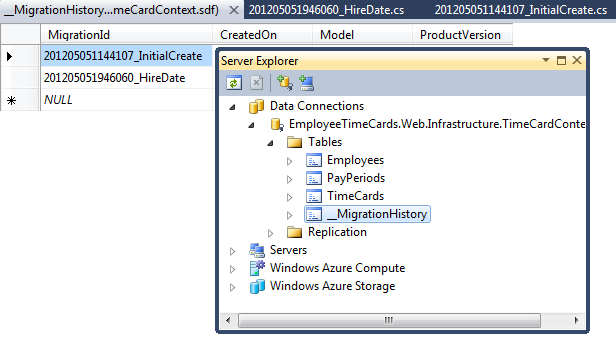
*You might see a message about detecting a database initializer – don’t panic – it’s normal.*

1. Type “Add-Migration HireDate”
2. In the Migrations folder, open the .cs file ending with “HireDate”. You should see code inside of the Up method to add a column for HireDate.
3. In the console window, type “Update-Database –Verbose”

*You should see an ALTER TABLE command scroll by in the output.*



1. Run the application again – it should be working now!
2. Click the .Web project and make sure the “Show All Files” button is selected in the Solution Explorer toolbar at the top of the Solution Explorer window.
3. Double click the .sdf file that should appear in the App\_Data folder (click Refresh in the Solution Explorer toolbar if it isn’t appearing).
4. In the Server Explorer window, drill into the .sdf file, right-click the \_\_MigrationHistory table and select Show Table Data.



1. Note the history of schema migrations.

## Part II – Customizing Migrations

1. Before continuing, make sure you have at least one employee in the Employees table.
2. Open Employee.cs in the .Core project.
3. Add a new string property to store an employee email address.

public virtual string EmailAddress { get; set; }

1. Build the project, then in the package manager console, type “Add-Migration”.
2. When prompted for a name, enter “Employee.EmailAddress”
3. Open the newly generated migration file from the Migrations folder.
4. Change the maxLength of the EmailAddress field from 4000 to 40.
5. Add a CreateIndex command in Up to add a unique index on the Employees table covering the EmailAddress column.

public override void Up()

{

AddColumn("Employees", "EmailAddress", c =>

c.String(maxLength: **40**));

**CreateIndex("Employees", "EmailAddress", unique:true);**

}

*In order for the unique index to work, we’ll have to make sure all the existing employee records have a unique value for the email address. This will require a “data migration” which is in the next step.*

1. Use the Sql method in the Up migration to set the EmailAddress for existing records to [FirstName.LastName@whatever.com](mailto:FirstName.LastName@whatever.com). Add this line of code in the Up method **before** the call to **CreateIndex** but **after** the call to **AddColumn**.

Sql(@"UPDATE Employees

SET EmailAddress = FirstName + '.' + LastName + '@otc.com'");

1. In the console window, type “Update-Database -Verbose”.
2. Verify every employee now has an email address using the Server Explorer window and checking the Employees table, or by modifying the Views/Home/Index.cshtml file to display the email address in the application.

## Part III: Scripts for the DBA

1. Type “Update-Database -Script -SourceMigration:$InitialDatabase” into the package manager console.
2. Observe the generated .sql script in the editor.