SAMS

FLUENT ENTITY FRAMEWORK

REBECCA M. RIORDAN

SAMS

Fluent Entity Framework

Rebecca M. Riordan

Associate Publisher
Greg Wiegand

Signing Editor
Neil Rowe

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Indexer
Cheryl Lenser

Proofreader
Karen Gill

Technical Editor
Craig Lee

Publishing Coordinator
Cindy Teeters

Cover Designer
Gary Adair

Composition
Rebecca Riordan

ISBN-13: 9780672335921

ISBN-10: 0672335921

Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

First Printing February 2013

ACKNOWLEDGEMENTS

I know it says "Rebecca M. Riordan" on the cover, but that's almost a lie of omission. Without the assistance of some amazing people, this book would never have made it out of my head and into your hands. In order of appearance, I'd like to thank:

Neil Rowe, my long-suffering editor, for patience above and beyond the call of duty. My technical reviewer, Craig Lee, stepped in when I was having a crisis of confidence and made the book much better than it would have been otherwise. Once again, Karen Gill caught the typos, infelicities and malapropisms and provided wonderfully unexpected moral support. (Any errors these two wonderful people missed are, of course, mine and mine alone.)

Jake von Slatt of The Steampunk Workshop (steampunkworkshop.com) and Samantha Wright (samantha-wright.deviantart.com) were both gracious enough to allow me to use their images. These are seriously talented artists, folks. I can't urge you strongly enough to go visit their sites.

The applesauce bread recipe is adapted from King Arthur Flour 200th Anniversary Cookbook. The challah recipe is adapted from Peter Reinhardt's Bread Bakers Apprentice. (The remaining 2 recipes are the author's.)

GETTING STARTED

Find out how the book works, what sorts of problems Entity Framework can help you solve, and get a taste of how it works.

THE DESIGNER

Learn how to use the Entity Framework Designer to build models of your data and create the code you need to work with it.

Contents

Explore the Entity Framework code model, and learn how to create models without the Designer and before you have a database.

Only you can decide what you need to do with your data, but this section will introduce you to the tools that Entity Framework gives you for queying and manipulating it.

Put all you've learned to good use by building a complete data application.

TELL US WHAT YOU THINK!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As a Executive Editor for Sams, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of email I receive, I might not be able to reply to every message.

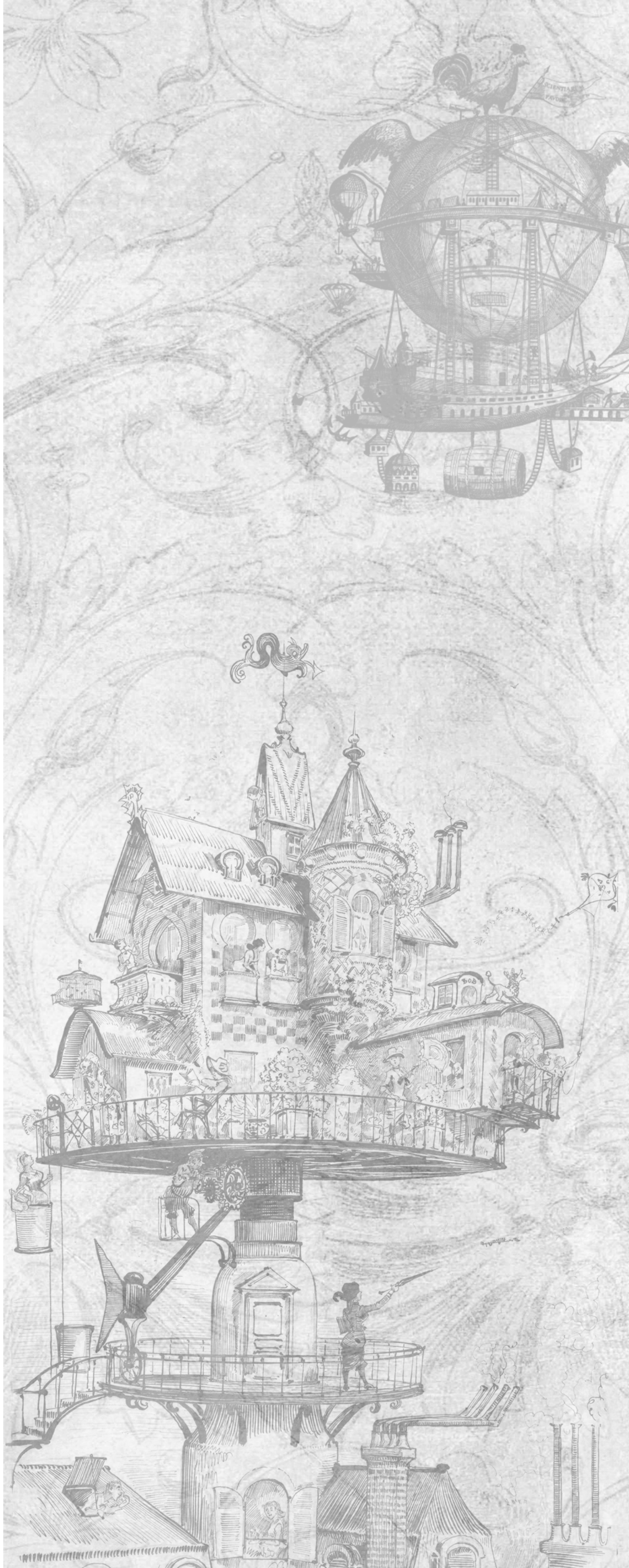
When you write, please be sure to include this book's title and author as well as your name and email address, phone, or fax number. I will carefully review your comments and share them with the author and editors who worked on this book.

Email: feedback@samspublishing.com

Fax: 317-428-3310

Mail: Neil Rowe, Executive Editor
Sams Publishing
800 East 96th Street

Indianapolis, IN 46240 USA



USING THE DESIGNER

3

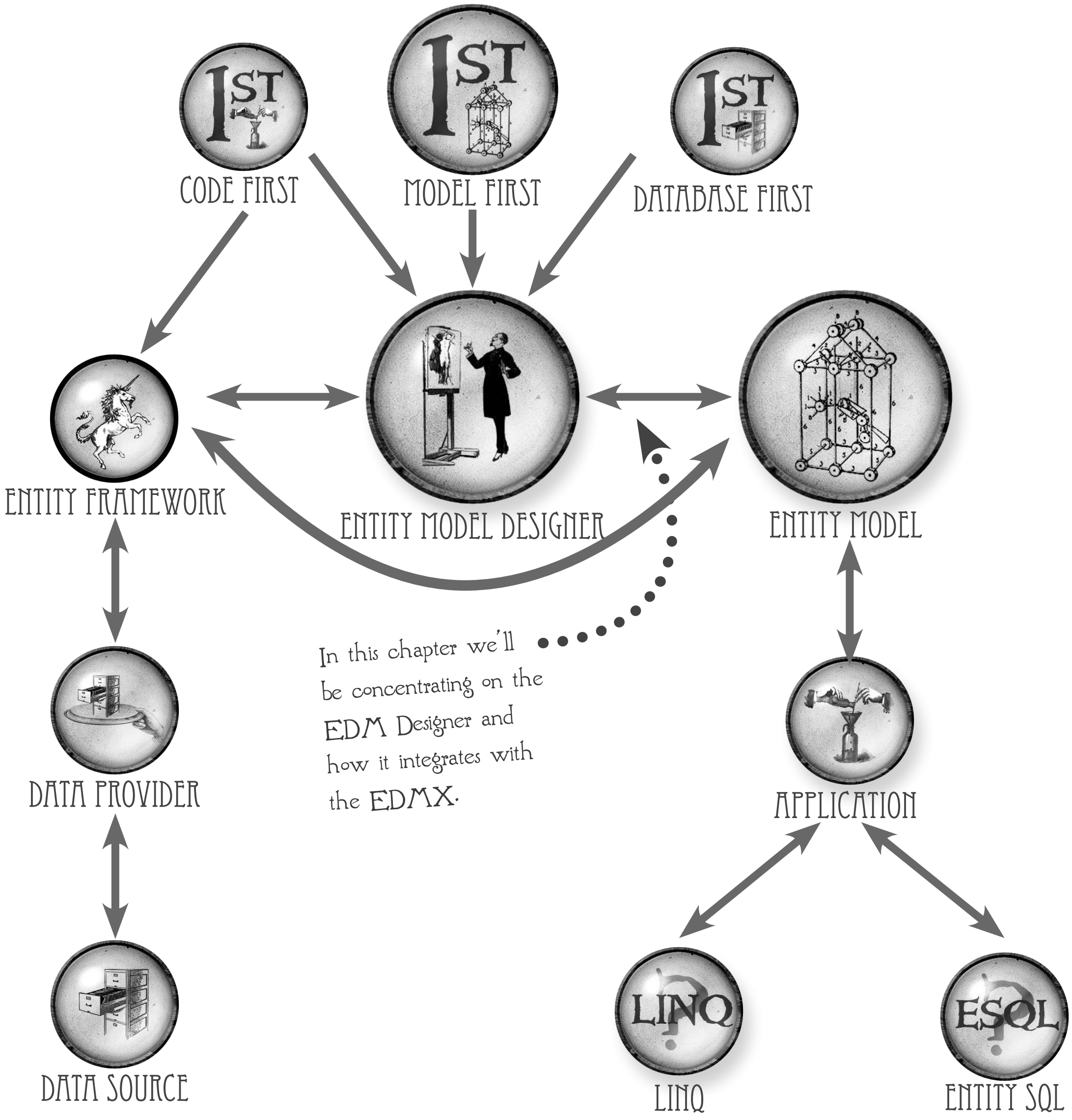
Congratulations. You've now written a real Entity Framework application. A pretty simple one, I grant you, and you're unlikely to build many applications that only need a couple of loops and some Console.Writeline() statements by way of UI, but the skills you've already gained will get you through a surprising number of situations, particularly when you have a preexisting database that's in reasonably good shape.

But of course that isn't always going to be true, and there's a lot more to learn about working with Entity Framework. (Otherwise this would be a very short book!) You might, for example, decide to start your application with the EDM and build the database from it (Model-First), or you might decide to forego a model entirely and do everything in code (Code First). We'll look at both of these options in later chapters. Even when you are starting from a database, you may need to make more substantial changes than the simple ones we looked at in the last chapter.

In this chapter, we'll start exploring some of the nooks & crannies of the Entity Framework by taking a closer look at the Entity Framework Designer and some of the advanced capabilities it offers.

Fitting It In

Here's how this chapter fits in to the book as a whole...



TASK LIST

In this chapter we'll explore the Entity Framework designer and the tools it provides for manipulating the EDMX.

THE DESIGNER & THE EDMX

We'll start this chapter by exploring how the Entity Framework designer translates the conceptual model in the EDMX into the class diagram you can manipulate on the design surface and through the Properties window.

UPDATING THE MODEL

EDMs are just as likely to change as any other part of an application. (You knew that, right?) Fortunately, the Entity Data Model Wizard makes it just as easy to update a model as it was to build it in the first place. We'll find out how in the second section of this chapter.

MAPPING DETAILS

After we've used the primary designer window to explore the conceptual layer of the EDMX, we'll look at the Mapping Details window, which is the designer's way of letting you view and control the way the conceptual model maps data to the database schema.

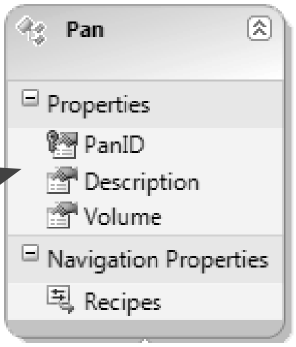
THE MODEL BROWSER

Finally, we'll turn our attention to the Model Browser, which provides a hierarchical view of all three layers of the EDMX. In addition to general poking around (more useful than you might think), you'll mostly use the Model Browser to explore stored procedures that don't map neatly to database operations, and we'll learn how to do that at the end of this chapter.

THE DESIGNER & THE EDMX

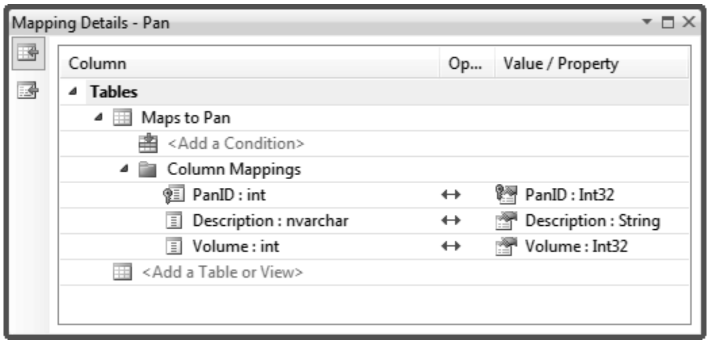
You may have worked with the Class Designer in Visual Studio, which provides a graphic view of a class diagram. The Entity Model Designer plays a similar role, but it works directly with the underlying EDMX. Here's how it works:

The EntityType definitions in the EDMX are represented as entity classes on the primary designer surface.



```
<EntityType Name="Pan">
  <Key>
    <PropertyRef Name="PanID" />
  </Key>
    <Property Name="PanID" Type="int" Nullable="false"
      StoreGeneratedPattern="Identity" />
    <Property Name="Description" Type="nvarchar" Nullable="false"
      MaxLength="50" />
    <Property Name="Volume" Type="int" />
</EntityType>
```

The Mapping Details window represents the content of the Mappings section of the EDMX. We'll look at the Mapping window in a few pages.

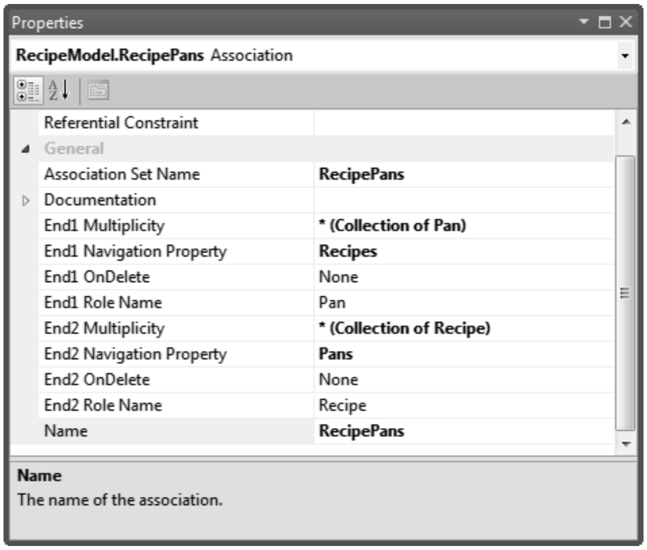


Mapping Details - Pan

Column	Op...	Value / Property
Tables		
Maps to Pan		
<Add a Condition>		
Column Mappings		
PanID : int	↔	PanID : Int32
Description : nvarchar	↔	Description : String
Volume : int	↔	Volume : Int32
<Add a Table or View>		

```
<EntitySetMapping Name="Pans">
  <EntityTypeMapping TypeName="RecipeModel.Pan">
    <MappingFragment StoreEntitySet="Pan">
      <ScalarProperty Name="PanID" ColumnName="PanID" />
      <ScalarProperty Name="Description" ColumnName="Description" />
      <ScalarProperty Name="Volume" ColumnName="Volume" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

```
<Association Name="FK_RecipePans_Pans">
 <End Role="Pan" Type="RecipeModel.Store.Pan" Multiplicity="1" />
 <End Role="RecipePans" Type="RecipeModel.Store.RecipePans" Multiplicity="*" />
 <ReferentialConstraint>
  <Principal Role="Pan">
   <PropertyRef Name="PanID" />
  </Principal>
  <Dependent Role="RecipePans">
   <PropertyRef Name="PanID" />
  </Dependent>
 </ReferentialConstraint>
</Association>
```



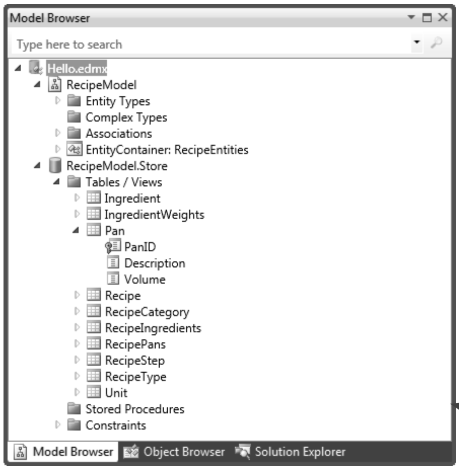
Properties

RecipeModel.RecipePans Association

Referential Constraint	
General	
Association Set Name	**RecipePans**
Documentation	
End1 Multiplicity	*** (Collection of Pan)**
End1 Navigation Property	**Recipes**
End1 OnDelete	None
End1 Role Name	Pan
End2 Multiplicity	*** (Collection of Recipe)**
End2 Navigation Property	**Pans**
End2 OnDelete	None
End2 Role Name	Recipe
Name	**RecipePans**

Name
The name of the association.

Details of Associations can be seen in the Properties window when you click on them in the primary designer window.



```
Model Browser
Type here to search
Hello.edmx
  RecipeModel
    Entity Types
    Complex Types
    Associations
    EntityContainer: RecipeEntities
  RecipeModel.Store
    Tables / Views
      Ingredient
      IngredientWeights
      Pan
        PanID
        Description
        Volume
      Recipe
      RecipeCategory
      RecipeIngredients
      RecipePans
      RecipeStep
      RecipeType
      Unit
    Stored Procedures
    Constraints
Model Browser | Object Browser | Solution Explorer
```

```
<EntitySet Name="Pan"
 EntityType="RecipeModel.Store.Pan"
 store:Type="Tables" Schema="dbo" />
```

The Store Schema, which you'll recall is the EDMX representation of the underlying database, is visible in the Model Browser. We'll be looking at it in detail later in the chapter, as well.

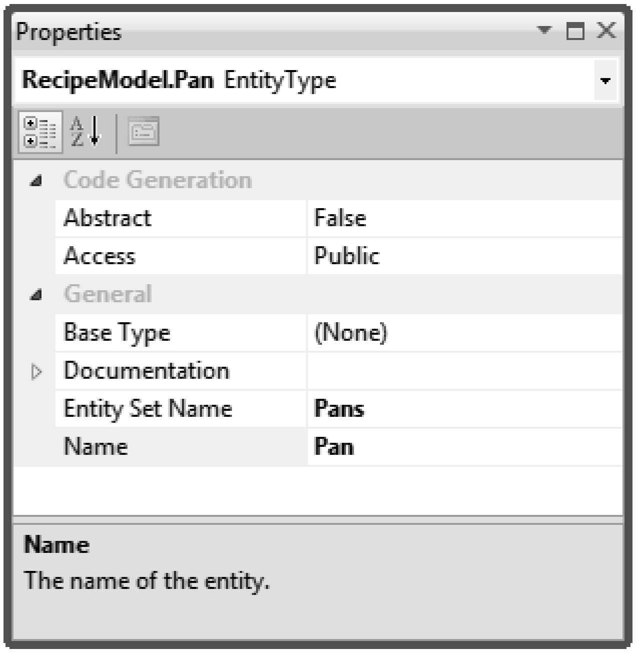
```
 <Designer xmlns="http://schemas.microsoft.com/ado/2008/10/edmx">
....
   <Diagrams>
  <Diagram Name="Hello" ZoomLevel="100">
   <EntityTypeShape EntityType="RecipeModel.Ingredient"
    Width="1.5" PointX="5" PointY="6.625"
    Height="1.9802864583333335" IsExpanded="true" />
...
```

Even the layout of the diagram is represented in the EDMX, in a special section at the bottom.

UPDATING THE MODEL

Because the designer is so closely linked to the underlying EDMX, changes you make in the designer will update the EDMX. (And vice versa, of course.) The designer itself works as you would expect if you've worked with other designers in Visual Studio. We'll look at some complex manipulations in later chapters, but here are the basics:

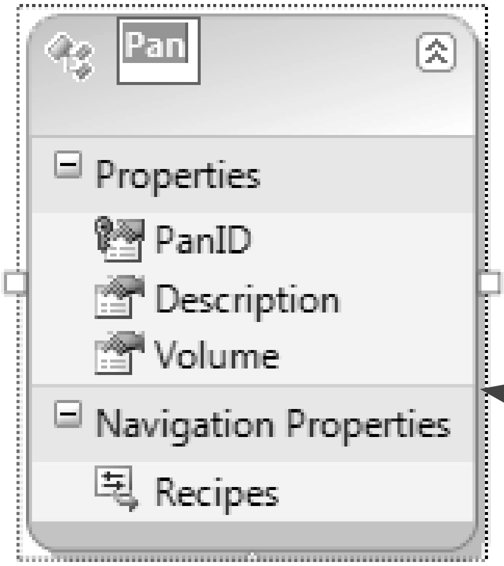
You can select an association, an entity, or one of the entity's members and change its properties in the Properties window.


Properties

RecipeModel.Pan EntityType

Code Generation	
Abstract	False
Access	Public
General	
Base Type	(None)
Documentation	
Entity Set Name	**Pans**
Name	**Pan**

Name
The name of the entity.


Pan

Properties
- PanID
- Description
- Volume

Navigation Properties
- Recipes

If you double-click an entity or one of its members, you can change the name directly on the diagram.

Put on your thinking hat

Can you figure out how to perform the following tasks in the designer?

How would you check the data type of an entity property?

You know that a relationship in a database can be one-to-one or one-to-many. The "one" or "many" is the relationships MULTIPLICITY. The multiplicity of an association in an EDM can also be many-to-many. How can you determine the multiplicity of an association in the designer?

An ENTITY KEY in an EDM is like the primary key of a table. Like a primary key, it must be unique, and like a primary key, it can be composed of multiple entity properties. How can you find out if a given entity member participates in the entity key?

In the Entity Framework, the model itself has properties. How do you display the properties of the model in the designer?

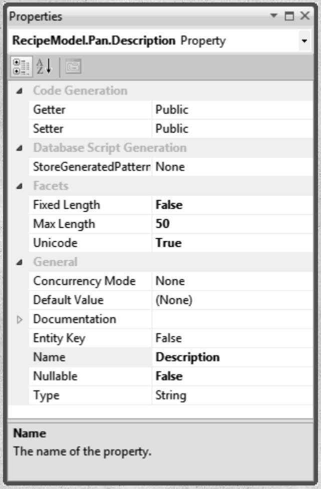
Put On Your Thinking Hat

How'd you do?

How would you check the data type of an entity property?

This is an easy one: Just select the property on the designer surface and its type will be displayed in the Properties window.

But there's another way that I haven't shown you: Right-click on the design surface and choose Scalar Property Format. Did you find that one? Try it now.



Properties

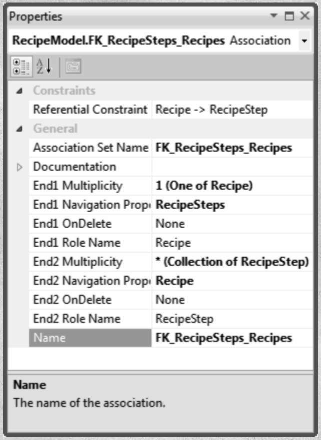
RecipeModel.Pan.Description Property

Code Generation	
Getter	Public
Setter	Public
Database Script Generation	
StoreGeneratedPatterr	None
Facets	
Fixed Length	**False**
Max Length	**50**
Unicode	**True**
General	
Concurrency Mode	None
Default Value	(None)
Documentation	
Entity Key	False
Name	**Description**
Nullable	**False**
Type	String

Name
The name of the property.

You know that a relationship in a database can be one-to-one or one-to-many. The "one" or "many" is the relationships MULTIPLICITY. The multiplicity of an association in an EDM can also be many-to-many. How can you determine the multiplicity of an association in the designer?

It's shown in the Properties window if you select the association, and also directly on the diagram.

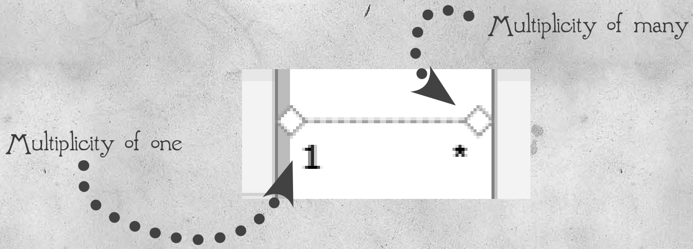


Properties

RecipeModel.FK_RecipeSteps_Recipes Association

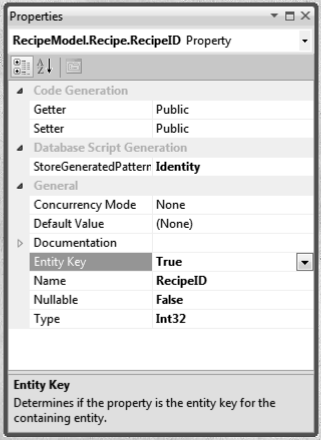
Constraints	
Referential Constraint	Recipe -> RecipeStep
General	
Association Set Name	**FK_RecipeSteps_Recipes**
Documentation	
End1 Multiplicity	**1 (One of Recipe)**
End1 Navigation Prop	**RecipeSteps**
End1 OnDelete	None
End1 Role Name	Recipe
End2 Multiplicity	*** (Collection of RecipeStep)**
End2 Navigation Prop	**Recipe**
End2 OnDelete	None
End2 Role Name	RecipeStep
Name	**FK_RecipeSteps_Recipes**

Name
The name of the association.



An ENTITY KEY in an EDM is like the primary key of a table. Like a primary key, it must be unique, and like a primary key, it can be composed of multiple entity properties. How can you find out if a given entity member participates in the entity key?

It's shown in the Properties window when you select the property, but notice that it doesn't tell you if this is the only property that participates in the key, so you might have to check several properties to be sure. We'll see another way to check the Entity Key when we look at the Model Browser later in this chapter.


Properties

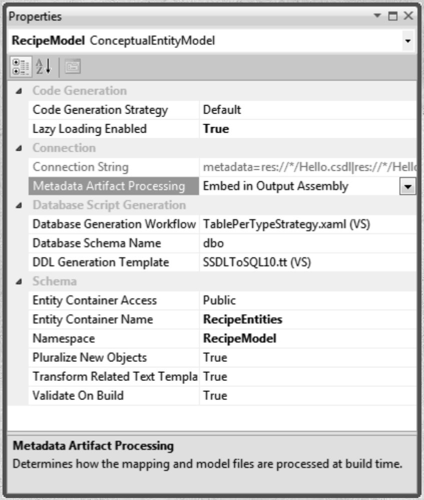
RecipeModel.Recipe.RecipeID Property

Code Generation	
Getter	Public
Setter	Public
Database Script Generation	
StoreGeneratedPatterr	**Identity**
General	
Concurrency Mode	None
Default Value	(None)
Documentation	
Entity Key	**True**
Name	**RecipeID**
Nullable	**False**
Type	**Int32**

Entity Key
Determines if the property is the entity key for the containing entity.

In the Entity Framework, the model itself has properties. A model is called an `EntityContainer` in the EDMX and a `ConceptualEntityModel` in the designer. How do you display the properties of the model in the designer?

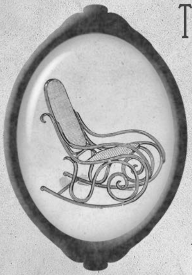
To show the properties of the model itself, just click on a blank area of the designer surface.


Properties

RecipeModel ConceptualEntityModel

Code Generation	
Code Generation Strategy	Default
Lazy Loading Enabled	**True**
Connection	
Connection String	metadata=res://*/Hello.csdl|res://*/Hell
Metadata Artifact Processing	Embed in Output Assembly
Database Script Generation	
Database Generation Workflow	TablePerTypeStrategy.xaml (VS)
Database Schema Name	dbo
DDL Generation Template	SSDLToSQL10.tt (VS)
Schema	
Entity Container Access	Public
Entity Container Name	**RecipeEntities**
Namespace	**RecipeModel**
Pluralize New Objects	True
Transform Related Text Templa	True
Validate On Build	True

Metadata Artifact Processing
Determines how the mapping and model files are processed at build time.



TAKE A BREAK

Why don't you take a break before completing the review and we move on to updating the model?

REVIEW

Based on what you've learned so far, do you think the following statements are true or false?

TRUE FALSE By default, database tables become entity classes in the EDM.

TRUE FALSE The Entity Model Designer is a visual representation of the classes in the .designer.cs or .designer.vb file.

TRUE FALSE One-to-many relationships in the database are called associations in the EDM.

TRUE FALSE Changes that you make in the designer will update the EDMX when you save them.

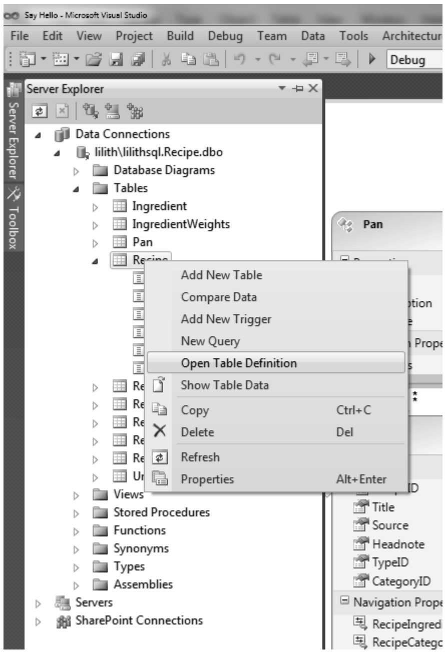
TRUE FALSE The Entity Model Designer is the only way to view the EDMX.

TRUE FALSE Selecting an entity property in the designer shows the entity key in the Properties Window.

TRUE FALSE Selecting an entity property in the designer shows whether the property participates in the entity's entity key.

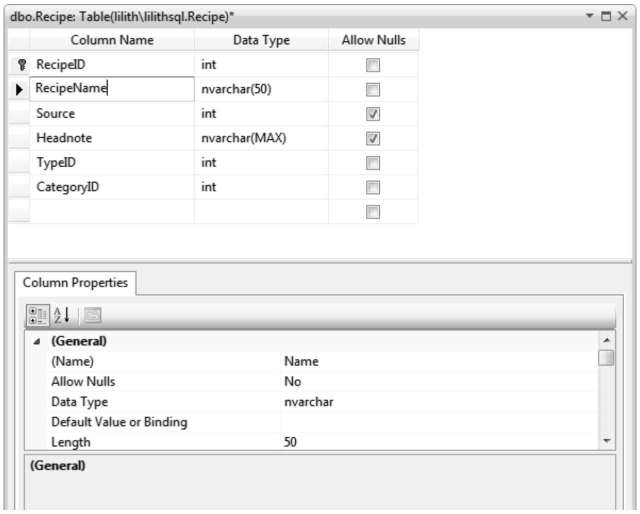
UPDATING THE MODEL

Stuff changes. It's a basic fact of our profession, and you've learned to expect and plan for that, right? Right? Well, even if you haven't, the designers at Microsoft have, and they've built the Entity Model Wizard to allow you to be able to update the model when the database schema changes, or when you need to add additional database objects to your model. To see how that works, let's start by making a minor change to the database:



In the Server Explorer (choose Server Explorer from the Windows menu if it's not visible), expand the connection to the Recipe database that Visual Studio created for you. Expand the Tables node and then right-click the Recipe table and choose Open Table Definition.

Change the name of the Title field to RecipeName, save the change, and then close the tab.


dbo.Recipe: Table(lilith\lilithsql.Recipe)*

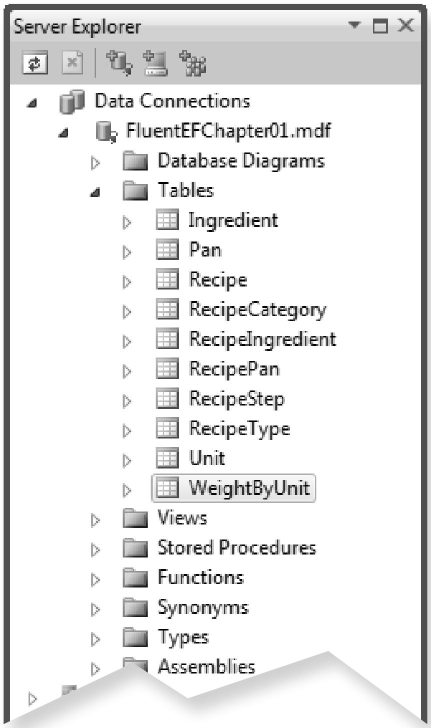
Column Name	Data Type	Allow Nulls
RecipeID	int	☐
RecipeName	nvarchar(50)	☐
Source	int	✓
Headnote	nvarchar(MAX)	✓
TypeID	int	☐
CategoryID	int	☐
		☐

Column Properties

(General)

(Name)	Name
Allow Nulls	No
Data Type	nvarchar
Default Value or Binding	
Length	50

(General)



Server Explorer

- Data Connections
  - FluentEFChapter01.mdf
    - Database Diagrams
    - Tables
      - Ingredient
      - Pan
      - Recipe
      - RecipeCategory
      - RecipeIngredient
      - RecipePan
      - RecipeStep
      - RecipeType
      - Unit
      - WeightByUnit
    - Views
    - Stored Procedures
    - Functions
    - Synonyms
    - Types
    - Assemblies

Let's make one more change: Select the WeightByUnit table in the Server Explorer and press the Delete key to delete it from the database. Visual Studio will ask you to confirm the change. Click OK.

ON YOUR OWN

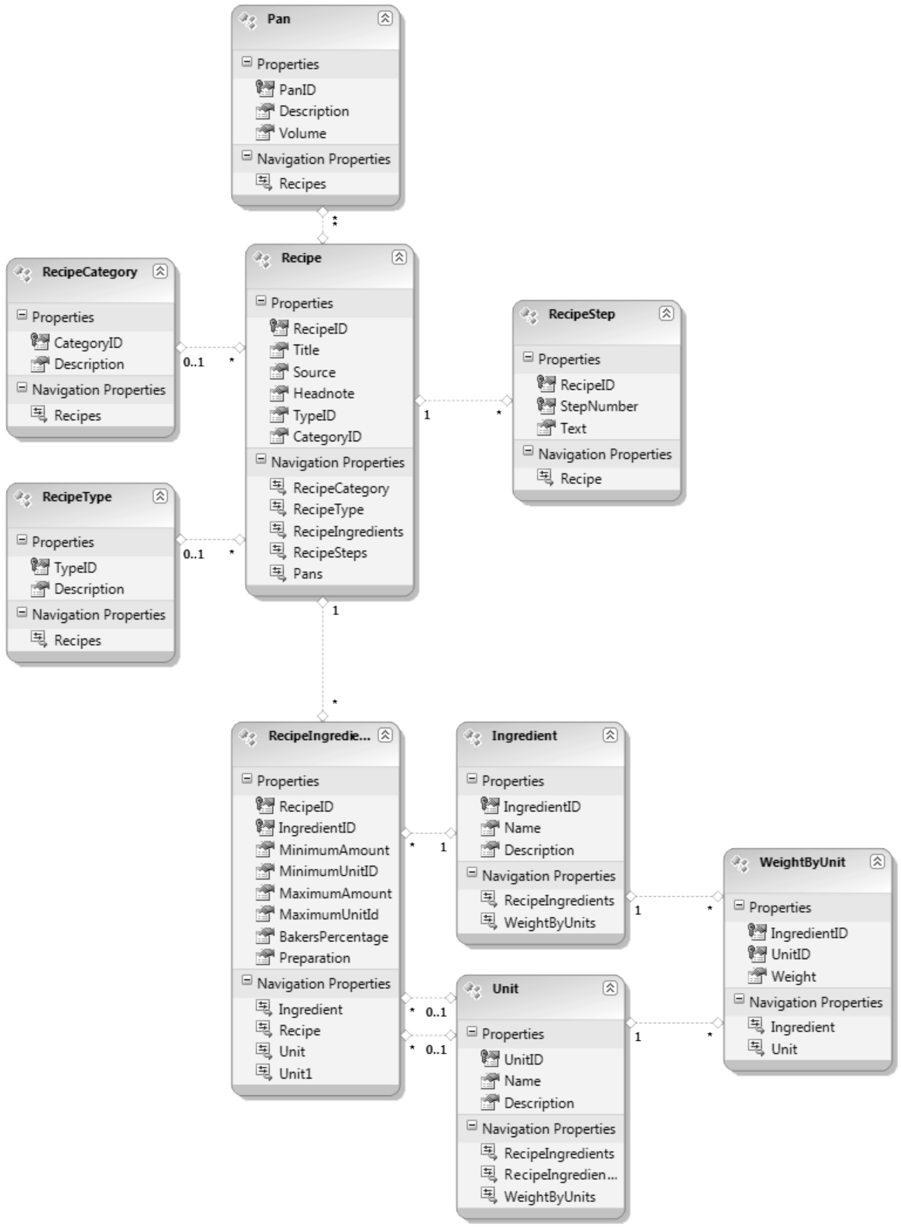
When you right-click on a blank area of the designer window, one of the options is "Update Model from Database..." What do you think will happen if you choose it?

We changed the name of a field in the **Recipe** table. Do you expect the name to change in the model? (Remember that we changed the names of the association properties in the **RecipeIngredient** entity. What do you think will happen to those?)

We deleted a table from the database. What do you expect to happen to it in the model?

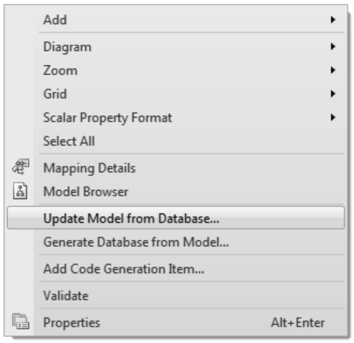
UPDATE MODEL WIZARD

You've changed the database, but you haven't updated the EDMX, so the designer is still showing "Title" as the name of the member. Let's fix that.



Update the Model

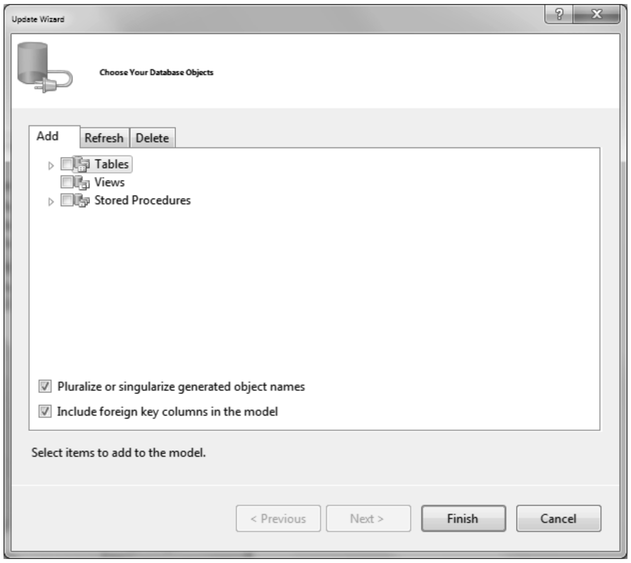
Have you thought about what you expect to happen? Let's try it out and see if you were right:



In the Entity Model Designer, right-click on a blank area of the design surface and choose Update Model from Database...

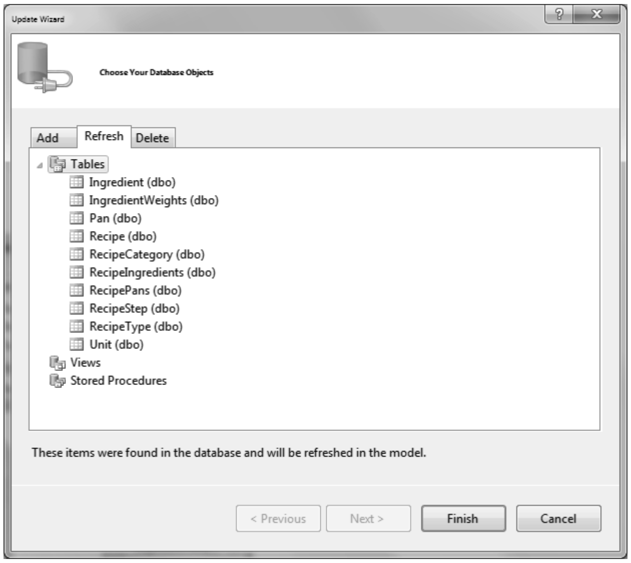
The wizard will open on a screen with three tabs, and the Add tab will be displayed. You can use this tab to add database objects to your model after it has been created.

We'll do that in the next section of this chapter, but not right now, so select the Refresh tab.



[✓] Pluralize or singularize generated object names
[✓] Include foreign key columns in the model

Select items to add to the model.



- Tables
  - Ingredient (dbo)
  - IngredientWeights (dbo)
  - Pan (dbo)
  - Recipe (dbo)
  - RecipeCategory (dbo)
  - RecipeIngredients (dbo)
  - RecipePans (dbo)
  - RecipeStep (dbo)
  - RecipeType (dbo)
  - Unit (dbo)
- Views
- Stored Procedures

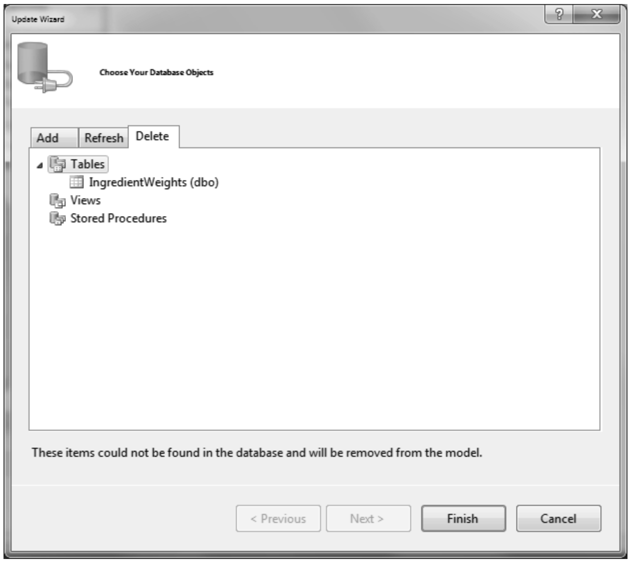
These items were found in the database and will be refreshed in the model.

You can't make changes on the Refresh tab. Visual Studio is going to update every object that has changed in the database (assuming that you've already included the object in the model).

Are you surprised at the number of tables to be updated, even though we only updated one? That's because of the way all the tables are related. When we made a change to the `Recipe` table, Visual Studio decided that every table that's related (directly or indirectly) to it needs to be updated.

You can't make changes on the Delete tab either, but the `IngredientWeights` table that we deleted from the database is listed here.

Click Finish to exit the wizard and update the model.

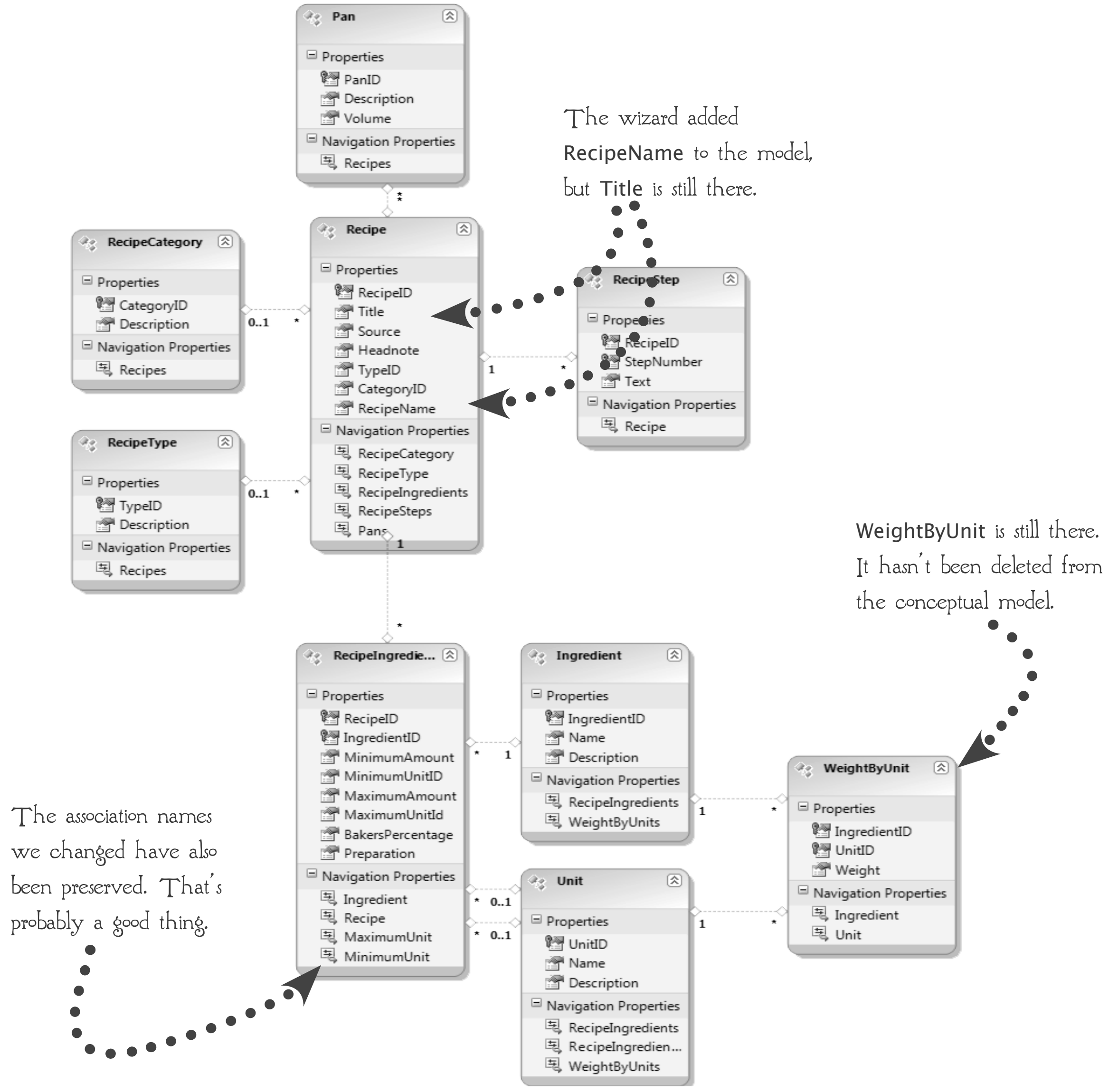


- Tables
  - IngredientWeights (dbo)
- Views
- Stored Procedures

These items could not be found in the database and will be removed from the model.

HEY, WHAT HAPPENED?

When you click Finish on the wizard (the button is available on every tab), the wizard will update the EDMX and redisplay the model. Is the display what you expected? Probably not. The wizard didn't rename the `Title` property; it just added a new `RecipeName` property, and the `WeightByUnit` table is still there. Actually, Visual Studio has just been a little smarter about things than we expected. The secret is the Mapping window, and by a strange coincidence, we'll look at that next.



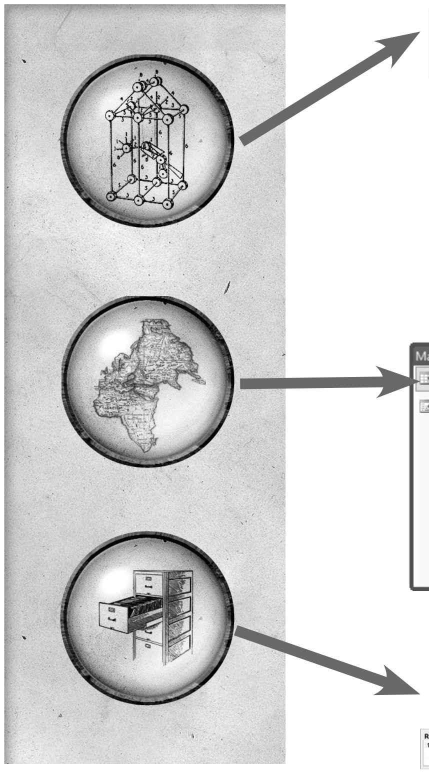
The wizard added **RecipeName** to the model, but **Title** is still there.

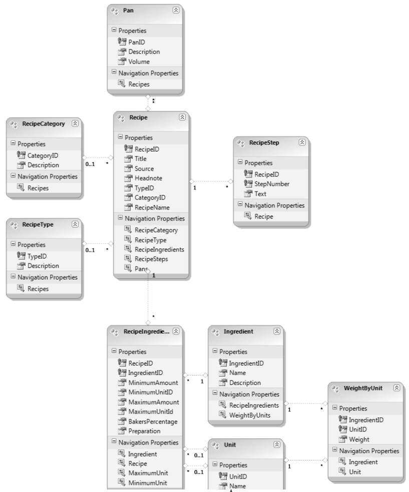
WeightByUnit is still there. It hasn't been deleted from the conceptual model.

The association names we changed have also been preserved. That's probably a good thing.

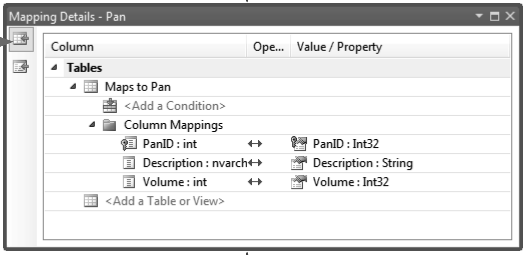
THE MAPPING WINDOW

In order to understand what the Update Model Wizard did (and didn't) do, we need to look at the relationship between the sections of the EDMX, the database, and the designer, and how they fit together.





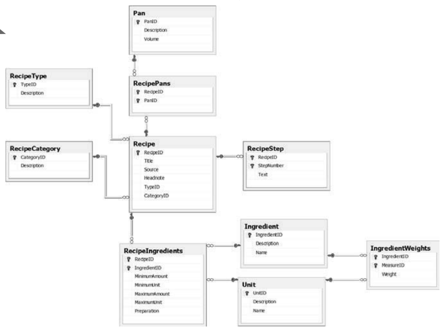
The CSDL, or Conceptual Schema Definition Language, represents the conceptual model. It's displayed in the primary designer window. These are the objects you'll work with in code.



Mapping Details - Pan

```
Column                      Ope...  Value / Property
▲ Tables
  ▲ Maps to Pan
      <Add a Condition>
    ▲ Column Mappings
        PanID : int           ↔     PanID : Int32
        Description : nvarch  ↔     Description : String
        Volume : int          ↔     Volume : Int32
    <Add a Table or View>
```

The MSL, or Mapping Schema Language, controls the relationship between the CSDL and the SSDL. It's displayed in the Mapping Details window.



The SSDL, or Store Schema Definition Language, represents the database. It's visible in the Model Browser, which we'll discuss in the next section.

So What Happened?

When we changed the database and then updated the model, the wizard didn't do what you probably expected it to do. (It certainly came as a surprise to me when I was learning the Entity Framework, but you're probably smarter than I am.)

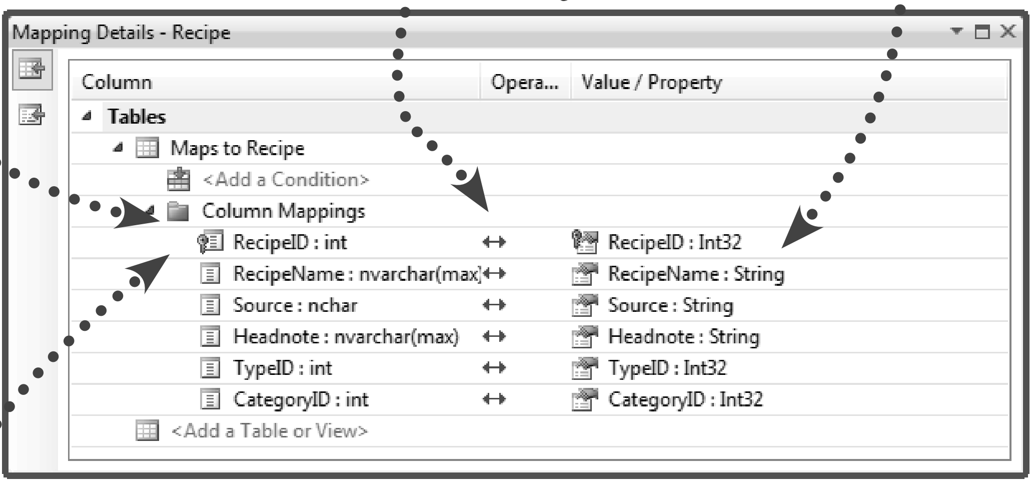
What the wizard did was update the SSDL to reflect the changes in the database and update the MSL so that nothing in the conceptual model was mapped to a nonexistent database fields, but it otherwise maintained the conceptual model as we'd designed it.

Right-click on a blank area of the primary design surface and choose Mapping Details from the context menu. By default, the Mapping Details window will appear below the primary design surface. If you select the **Recipe** entity, you can see what's happened:

This column shows the properties of the entity.

This column indicates how the mapping is performed. The double-headed arrow indicates a direct mapping.

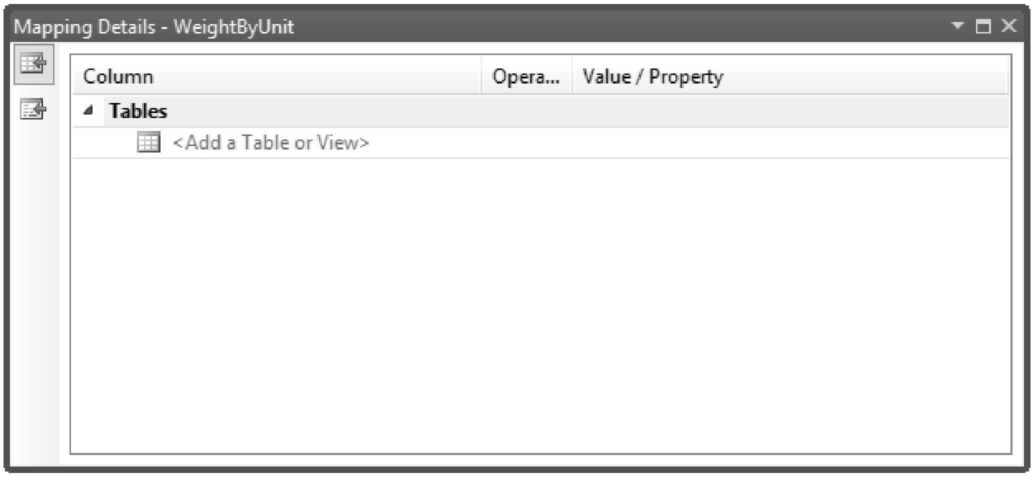
This column shows what the property is mapped to--usually (but not always, as we'll see) a field in the database.



Column	Opera...	Value / Property
Tables		
Maps to Recipe		
<Add a Condition>		
Column Mappings		
RecipeID : int	↔	RecipeID : Int32
RecipeName : nvarchar(max)	↔	RecipeName : String
Source : nchar	↔	Source : String
Headnote : nvarchar(max)	↔	Headnote : String
TypeID : int	↔	TypeID : Int32
CategoryID : int	↔	CategoryID : Int32
<Add a Table or View>		

Do you see what's happened? The new entity property, **RecipeName**, is mapped to the **RecipeName** field. The **Title** property, which still exists in the entity, isn't in the list at all because it's no longer mapped to anything.

If you select the WeightByUnit entity, you'll see that the Mapping Details window is completely empty. None of the properties of this entity are mapped to the database any longer:



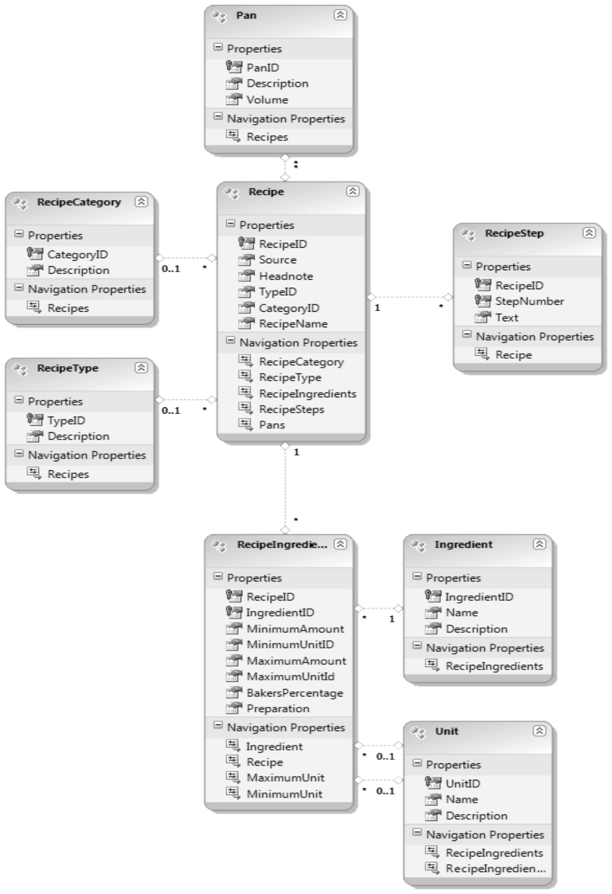
My Opinion

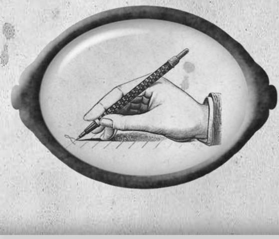
The wizard does what it does, and there's not much to be done about that. You can choose not to use it, of course, if you really don't like the way it behaves.

But before you throw your hands up in disgust and decide that the Entity Framework team made a terrible decision, consider this: There is no way (or at least no practical way) for the wizard to know which of the changes you've made to the conceptual model you want to keep. It makes the changes it can—to the schema definition and the mapping layer—and leaves the decisions it can't make to the person who can (you). Personally, I'd much rather fiddle around in the Mapping Details Window for a minute or two than spend hours manually updating the EDMX.

BEFORE WE MOVE ON...

Go ahead and make the changes to the conceptual model. Simply select the `Title` property of the `Recipe` entity and either choose `Delete` from the context menu or press the delete key. Do the same thing with the entire `WeightByUnit` entity. Your model should look like this:





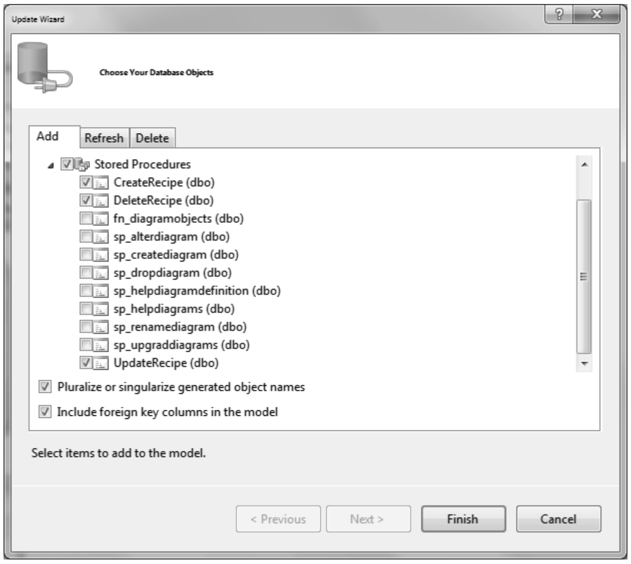
MAKE A NOTE

You'll probably want to change the code we wrote to reflect the change of name. Otherwise, you'll get build errors if you try to rerun the application.

MAPPING FUNCTIONS

So far all our entity properties are mapped directly to database fields, and the Entity Framework is generating the code to insert, update and delete values. But as you probably know, many database administrators don't allow this kind of direct access. For very good reasons having to do with maintaining the integrity of the data for which they're responsible, they require you to perform these operations through stored procedures. The Entity Framework treats stored procedures as functions. You add them using the Update Wizard and connect them to the conceptual model in the database file. Let's give it a whirl:

The first step is easy. Run the wizard the same way you did before, by right-clicking on a blank area of the primary designer window and choosing Update Model from Database.



Stored Procedures
- [✓] CreateRecipe (dbo)
- [✓] DeleteRecipe (dbo)
- [ ] fn_diagramobjects (dbo)
- [ ] sp_alterdiagram (dbo)
- [ ] sp_creatediagram (dbo)
- [ ] sp_dropdiagram (dbo)
- [ ] sp_helpdiagramdefinition (dbo)
- [ ] sp_helpdiagrams (dbo)
- [ ] sp_renamediagram (dbo)
- [ ] sp_upgraddiagrams (dbo)
- [✓] UpdateRecipe (dbo)

[✓] Pluralize or singularize generated object names

[✓] Include foreign key columns in the model

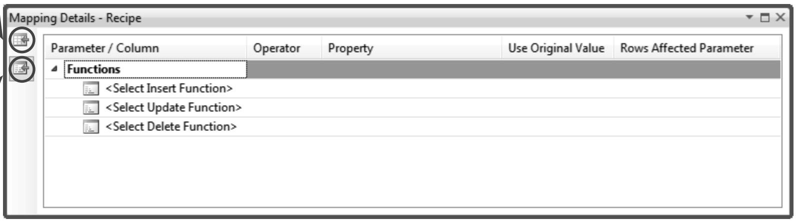
Select items to add to the model.

On the Add tab of the wizard, select the CreateRecipe, DeleteRecipe and UpdateRecipe stored procedures, as shown. (The other stored procedures that the wizard lists were added by Visual Studio and the SQL Server Management Studio. You can ignore them.)

Click Finish. Once again, the wizard will update the SSDL and MSL but leave your conceptual model alone, so you won't see any changes.

Make sure the Recipe entity is still selected on the primary designer surface, and then click the second button on the left side of the Mapping Details Window to display the Map Entity to Functions Pane.

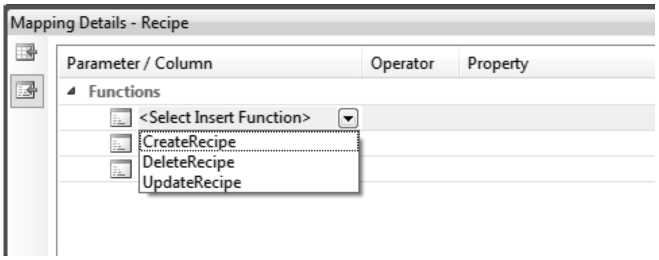
This button shows the default view, Map Entity to Tables/Views.



Mapping Details - Recipe

Parameter / Column	Operator	Property	Use Original Value	Rows Affected Parameter
Functions				
<Select Insert Function>				
<Select Update Function>				
<Select Delete Function>				

This button (selected in the screen shot) shows the Map Entity to Functions pane.



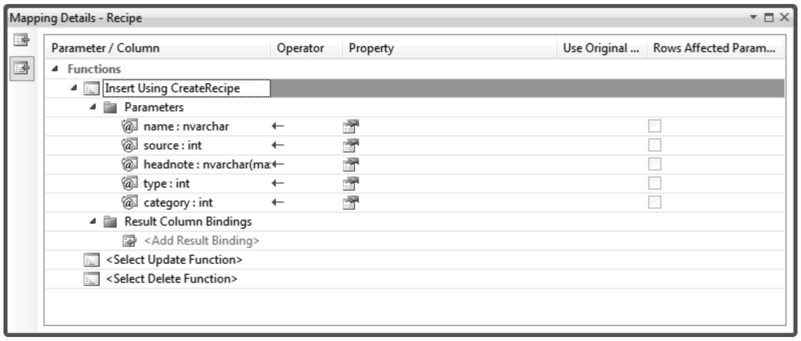
Mapping Details - Recipe

Parameter / Column	Operator	Property
Functions		
<Select Insert Function>		

CreateRecipe
DeleteRecipe
UpdateRecipe

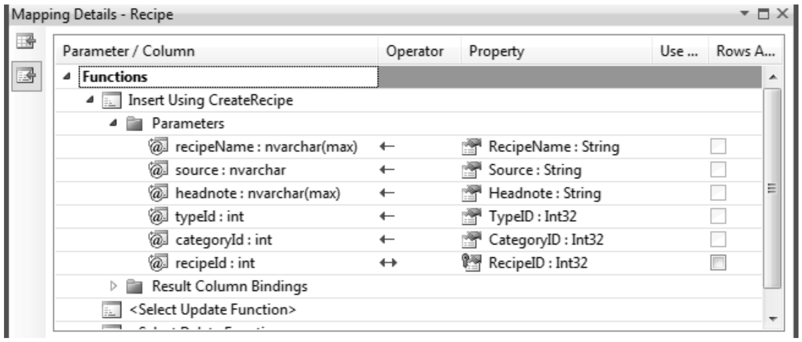
Click in the <Select Insert Function> cell, and a list of the stored procedures we've imported into the model will be displayed. Choose CreateRecipe.

After you choose the stored procedure (if you choose the wrong one, just choose a different one from the list), the Mapping Details Window will display a list of the parameters that were defined when the stored procedure was created.



Mapping Details - Recipe

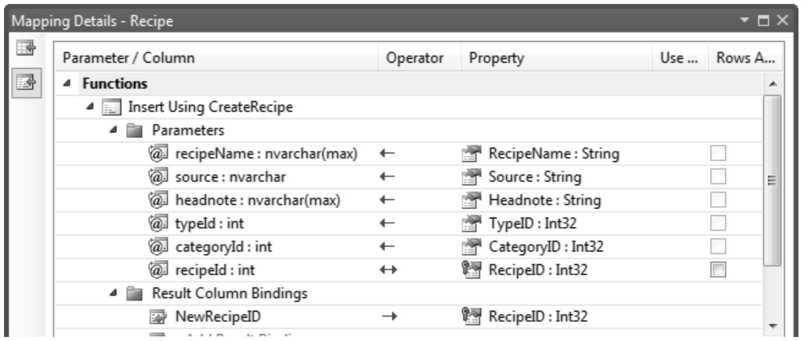
Parameter / Column	Operator	Property	Use Original ...	Rows Affected Param...
Functions				
Insert Using CreateRecipe				
Parameters				
name : nvarchar	←			☐
source : int	←			☐
headnote : nvarchar(ma	←			☐
type : int	←			☐
category : int	←			☐
Result Column Bindings				
<Add Result Binding>				
<Select Update Function>				
<Select Delete Function>				



Mapping Details - Recipe

Parameter / Column	Operator	Property	Use ...	Rows A...
Functions				
Insert Using CreateRecipe				
Parameters				
recipeName : nvarchar(max)	←	RecipeName : String	☐	
source : nvarchar	←	Source : String	☐	
headnote : nvarchar(max)	←	Headnote : String	☐	
typeId : int	←	TypeID : Int32	☐	
categoryId : int	←	CategoryID : Int32	☐	
recipeId : int	↔	RecipeID : Int32	☐	
Result Column Bindings				
<Select Update Function>				

We need to tell Entity Framework how to map the stored procedure parameters to the entity properties. When you click in the Property column, the Mapping Details Window will display a list of properties for you to choose from. Go ahead and fill it out now, using the screenshot as an example.



Mapping Details - Recipe

Parameter / Column	Operator	Property	Use ...	Rows A...
Functions				
Insert Using CreateRecipe				
Parameters				
recipeName : nvarchar(max)	←	RecipeName : String	☐	
source : nvarchar	←	Source : String	☐	
headnote : nvarchar(max)	←	Headnote : String	☐	
typeId : int	←	TypeID : Int32	☐	
categoryId : int	←	CategoryID : Int32	☐	
recipeId : int	↔	RecipeID : Int32	☐	
Result Column Bindings				
NewRecipeID	→	RecipeID : Int32		

One last step. The RecipeID field is an identity field, which means the value is generated by the database. We need to store the generated value in the entity instance to make sure our in-memory data matches up with the rows of the table. The stored procedure returns that value as an output parameter called NewRecipeID, so all we have to do is tell the Entity Framework about it. Type NewRecipeID in the cell labeled <Add Result Binding> and then press the Tab key. The Mapping Details Window will add RecipeID for you, since it's the entity key for the Recipe entity.

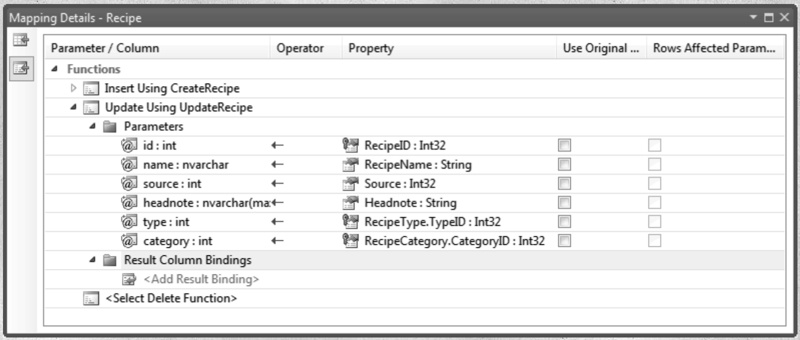
ON YOUR OWN

The UpdateRecipe stored procedure needs to be mapped to the Update function. It doesn't return any values (although the corresponding procedure in a production database might return the number of rows affected).

Try adding it now.

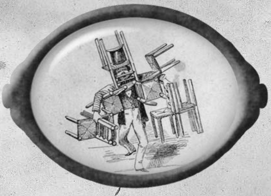
THINKING HAT?

How'd you do? Here's what the Mapping Details Window should look when you're finished:



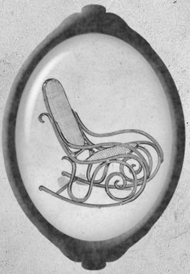
Mapping Details - Recipe

Parameter / Column	Operator	Property	Use Original ...	Rows Affected Param...
Functions				
Insert Using CreateRecipe				
Update Using UpdateRecipe				
Parameters				
id : int	←	RecipeID : Int32	☐	☐
name : nvarchar	←	RecipeName : String	☐	☐
source : int	←	Source : Int32	☐	☐
headnote : nvarchar(ma	←	Headnote : String	☐	☐
type : int	←	RecipeType.TypeID : Int32	☐	☐
category : int	←	RecipeCategory.CategoryID : Int32	☐	☐
Result Column Bindings				
<Add Result Binding>				
<Select Delete Function>				



ON YOUR OWN

It isn't necessary to map every operation to a stored procedure. Sometimes you can't delete a row at all, for example. But our database does have stored procedures for the full set of operations, so now that you're an expert at this, why don't you go ahead and add the DeleteRecipe function to the Mapping Details Window. Like the UpdateRecipe stored procedure, it doesn't have an output value, and it only has one input value (since only the key is required to identify the row to be deleted).



TAKE A BREAK

Once you've completed the On Your Own exercise, why don't you take a break before you complete the Review and we move on to the Model Browser Window?

REVIEW

Based on what you've learned in this section, can you answer the following questions?

How do you trigger the Update Database Wizard?

Which layers of the EDMX does the wizard change when a change is made to the database schema?

How do you add new database objects to the conceptual model?

What do the two little buttons on the left side of the Mapping Details Window mean?

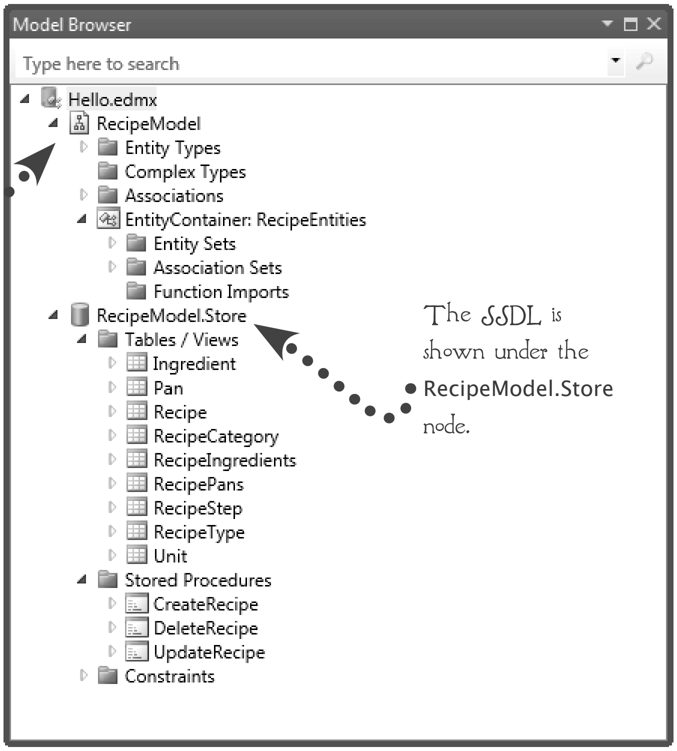
Is it necessary to map all the functions if you map one?

The Model Browser

So far we've explored the primary designer surface that lets us manipulate the CDSL and the Mapping Details window that lets us manipulate the MSL. You can't control the SSDL directly in the Entity Model Designer–you have to do that in the Server Explorer or a tool like SQL Server Management Studio–but you can view it using the last major component of the designer, the Model Browser. The Model Browser also shows you the structure of your conceptual model. Let's see how it works.

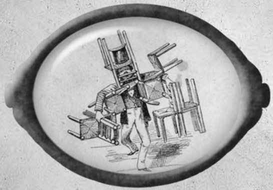
You display the Model Browser the same way you display the Mapping Details Window: by right-clicking on a blank area of the primary designer surface. (But of course, this time you choose Model Browser from the menu.) It displays the CSDL and SSDL as a TreeView:

The conceptual model is shown under the RecipeModel node.



```
Model Browser
Type here to search
◢ Hello.edmx
  ◢ RecipeModel
    ▷ Entity Types
      Complex Types
    ▷ Associations
    ◢ EntityContainer: RecipeEntities
      ▷ Entity Sets
      ▷ Association Sets
        Function Imports
  ◢ RecipeModel.Store
    ◢ Tables / Views
      ▷ Ingredient
      ▷ Pan
      ▷ Recipe
      ▷ RecipeCategory
      ▷ RecipeIngredients
      ▷ RecipePans
      ▷ RecipeStep
      ▷ RecipeType
      ▷ Unit
    ◢ Stored Procedures
      ▷ CreateRecipe
      ▷ DeleteRecipe
      ▷ UpdateRecipe
    ▷ Constraints
```

The SSDL is shown under the RecipeModel.Store node.



On Your Own

You can use the Model Browser for more than just inspecting the EDMX, but we'll look at that more advanced functionality in the next chapter. For right now, why don't you explore the basic display and see if you can answer these questions?

How many entity sets are in our model?

What properties comprise the entity key of the RecipeIngredient entity? How can you tell?

How does inspecting entity keys in the Model Browser differ from selecting the properties individually on the primary designer surface?

What's the data type of the RecipeName field as defined in the database? (Hint: check the Properties window.)

Is the table we deleted from the database (IngredientWeights) shown in the Model Browser?

There are two things that we haven't yet discussed shown in the Model Browser. One is a node in RecipeModel, the other a node in the EntityContainers: RecipeEntities. What are they? What do you think they do?

REVIEW

There are three primary windows in the Entity Model Designer. What are they? How is each used?

Why does the Update Model Wizard try to preserve the conceptual model?

What window would you use to map a stored procedure to the delete entity function?

How do you delete an entity from the model?

How do you change the name of an entity property?

What is the relationship between two tables in the database called in the conceptual model?

Congratulations! You've finished the chapter. Take a minute to think about what you've accomplished before you move on to the next one...

List three things you learned in this chapter:

1.

2.

3.

Why do you think you need to know these things in order to work with Entity Framework?

Is there anything in this chapter that you think you need to understand in more detail? If so, what are you going to do about that?

INDEX