Modeling applications exist in abundance, but each application has a specific focus such as process data

flow or describing objects. The following lists some of the modeling application types and their primary

focus.

*ERM (Entity Relationship Model)*: Used with databases, it is a way to represent

logical relationships of entities (objects) in order to create a database.

*UML (Unified Modeling Language)*: A standard modeling language that is used to

describe objects.

*ORM (Object-Relational Mapping)*: Method of mapping relational databases and

object-oriented programming languages.

*DFD (Data Flow Diagram)*: A graphical representation of the flow of data between

processes and between systems.

Entity framework aims both to have object to table relationship and also encorporate the power of UML and other modeling diagrams, i.e both easy tension free coding, developer need not worry about changing table field names etc, also provides easy graphical representation. Compared to its previous competitors like nhibernate, sprint.net etc

<https://www.simple-talk.com/dotnet/.net-framework/entity-framework-the-cribsheet/>

<https://www.simple-talk.com/dotnet/.net-framework/developing-your-data-access-layer-with-ado.net-entity-framework-4/>

<http://nerddinnerbook.s3.amazonaws.com/Part3.htm> (this explains entity framework with Repository pattern)

It can provide an excellent interface between the Object-oriented model and the relational. The Entity Framework allows developers to work with data in the form of objects and properties without having to concern themselves directly with either the stored procedures or functions of the defined interface, or the underlying database tables and columns where this data is stored.

The Entity Data Model consists of the storage schema, the conceptual schema, the mapping schema and the entity classes.

It is meant to accomplished by raising the abstraction level to a point that lets developers query entities and relationships in the conceptual model,all while letting the Entity Framework translate the query operations to data source–specific commands.

It allows applications to be written against conceptual models and not directly against the database. By

doing so, the gap between object-oriented programming and databases has been closed, letting

developers focus on the task of developing applications, without concerning themselves about the

database (structure or otherwise) or data access. The Entity Framework divides the data model into three separate models: conceptual, logical, and physical.

# Model-Driven

With the Entity Framework, you do not have to worry about the database. Rather, you simply code and

query against a set of objects (entities) that reflects the business model. Results are returned as objects,

and unlike other data access options, the developer does not have to spend time (code) figuring out rows

and columns in the returned results just to bind them to objects. Since results are returned as objects,

this work is automatically done.

Entity Data Model (EDM). The EDM is the foundation of the Entity Framework and is comprised of the three models mentioned earlier:

the conceptual model, the logical model, and the storage model. Think of the EDM as an “enhanced”

version of an ERM.

The Entity Data Model and database model, are distinct and separate, but the goal of the EF is to reform or reorganize the database objects in such a way that your EDM matches the goal of your business layer instead of an exact match of your database schema.

Entities are like objects, for example:

Entities have a known type.

Entities have properties, and these properties can hold scalar values.

Entity properties can hold references to other entities.

Each entity has a distinct identity.

Entities also differ from objects, for example:

Entities live within a collection.

Each entity has associations with other entities.

Entities have primary keys that uniquely identify the entity.

Equally important, entities also have some similarities to relational data

Entities do not have physical storage knowledge.

Entities can have relationships between them, and those relationships can be

defined directly in the EDM Designer.

One of the many nice features of the Entity Framework is the ability to retrieve what is called a

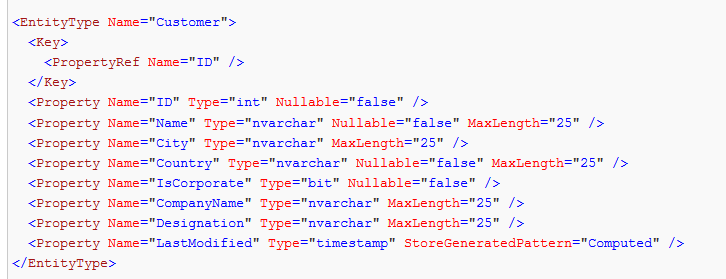
graph. This does not refer to the pie/bar chart type of graph. An Entity Framework graph is the ability to

return shaped data. For example, in a single result I can return data such as a salesperson and all of that

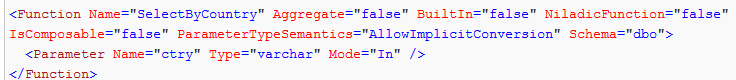
person’s contact details along with it.

## Storage Schema Definition (SSDL)

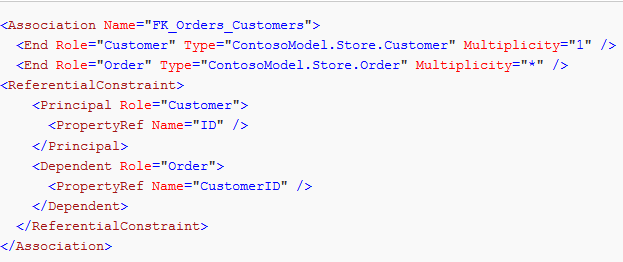
The storage schema describes, in an XML definition, the schema of the data store. This is the exact schema of your data store. The storage schema definition can be generated directly from the data store. The Storage schema describes the schema of a database table as an entity type. The following XML fragment shows the definition for a Customer table:



The storage schema also refers to any stored procedures in the data store. The following XML definition defines an existing stored procedure named **SelectByCountry** that accepts a single parameter named **ctry:**



The storage schema also describes relationships between tables as an association. The following definition for an association shows that the Customer table has a relationship with an Order table:

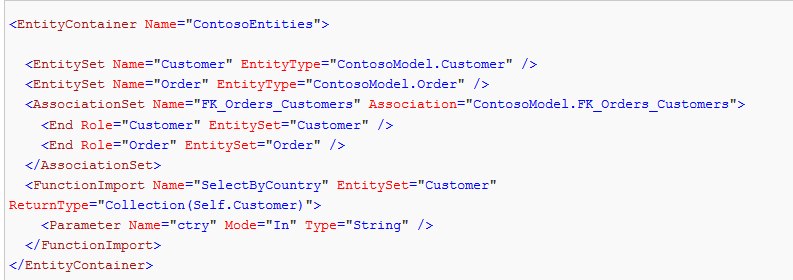


## Conceptual Schema Definition (CSDL)

The conceptual schema describes the conceptual entities as entity types and is defined in XML. This is created to describe the schema of the conceptual entities. Conceptual entity types are .NET classes. Though this schema can be auto-generated, most of the times the application developer would alter the conceptual schema to reflect the model used for the conceptual entities. If the conceptual schema is auto-generated it would simply map a conceptual entity type to a table. The following shows the schema for two conceptual entities named Customer and **CorporateCustomer**. The **CorporateCustomer** entity type inherits from the Customer entity type.



Finally all the defined conceptual entity types are grouped into entity sets and associations are grouped into association sets. An entity container is defined to include the entity sets, association sets and any function definitions that can be mapped to stored procedures:



The mapping schema definition is the glue that binds the conceptual model and the data store model. This XML definition contains information on how the conceptual entities, functions and associations are mapped to the storage schema. The following shows the mapping definition to map the Customer and **CorporateCustomer** conceptual entities defined in the conceptual schema to the database table named **Customer** defined in the storage schema:

POCO Support

One of the more powerful new features of the Entity Framework is the ability to add and use your own

custom data classes in conjunction with your data model. This is accomplished by using CLR objects,

commonly known as “POCO” (Plain Old CLR Objects).

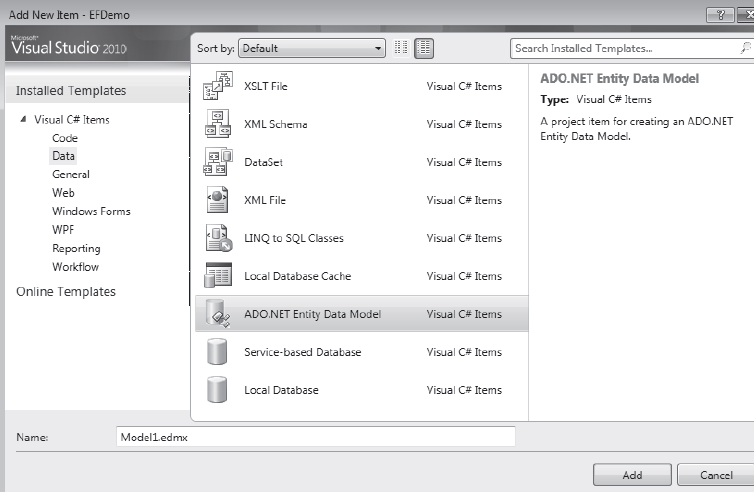
Database first:First it allowed only creating a Model based on database it now also gives the following option:

Model-first: Allows you to start with an empty model, define your model and then

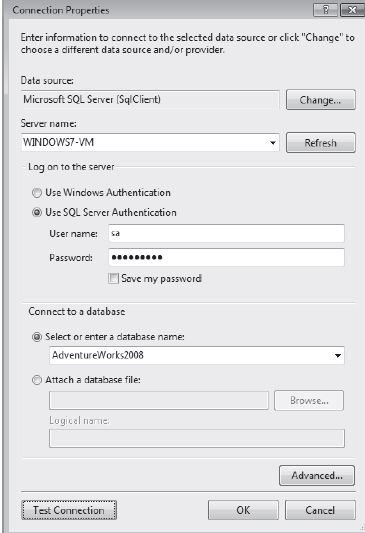
generate the database, mappings, and classes from the defined model.

• Code-only: Allows you to use the Entity Framework using Plain Old CLR Objects

(POCO) entities and without an EDMX file.



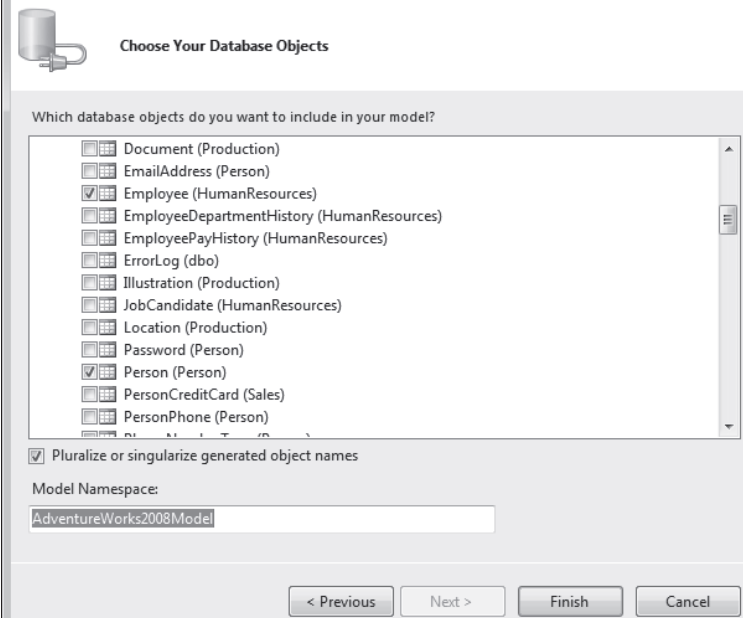
After choosing above project it gives u options to connect to an existing database (oracle , Sql server and many other types depending on the providers installed)



The next step of the wizard allows you to select the database objects to include in the EDM. On this

page, shown in Figure 2-6, you can select tables, views, and stored procedures. Also included in the list of

stored procedures are scalar-valued functions



The wizard applies English-language rules for singulars and plurals by doing

the following:

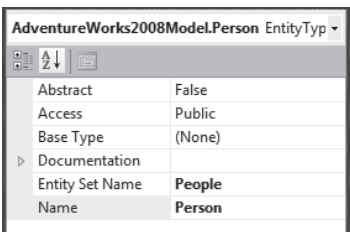
Making all EntityType names singular

Making all Entity Set names plural

For example, take a look at Figure 2-7. When we mapped the Person table a few moments ago, it

pluralized the name Person to People. It left the Entity Name as Person but pluralized that name to

People for the Entity Set Name property.



The same is true for the opposite scenario when the Entity Name is pluralized. For example, if the

Entity Name property had a value of People, the EDM wizard would have set the Entity Set Name

property to Person. Just think how much less confusing naming will be. Now when you code against the

model, the names of objects will be logical.

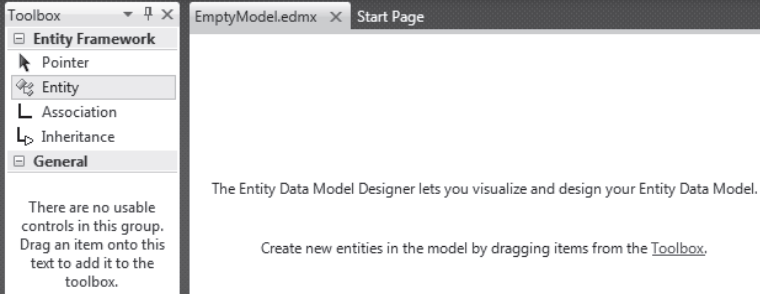
Taking a Model-First Approach

A new and welcomed feature to the EF 4.0 is the ability to create a conceptual model first and then derive

the storage model, database, and mappings from that. This section will walk you through each step of

generating a conceptual model, and will explain all the different properties and components involved in

the model-first approach.



Also shown in Figure 2-9 is the Toolbox. Inside the toolbox are the objects you can drag into your

model in the Designer. There are three:

*Entity*: Allows you to design and create an Entity.

*Association*: Lets you create an association (or relationship) between two entities.

*Inheritance*: Lets you create an Inheritance relationship between two entities.

**Managing Table Inheritance**

Table inheritance is a concept that will come in handy while developing databases from entity model, with the Entity Framework. You

have two options for how table inheritance is implemented within your model. They are

• Table-per-type: Uses a separate table in storage to maintain data for each type in

the inheritance hierarchy.

• Table-per-hierarchy: Uses one table in storage to maintain data for all the types in

an inheritance hierarchy.

|  |  |
| --- | --- |
| **Scalar property** | A property of an entity that maps to a single field in the storage model. |
|  | Taking a Code-Only Approach So far we talked about two different ways to create your Entity Data Model: database-first and modelfirst. There is a third method, as was mentioned earlier, which allows developers to create their model using POCO (Plain Old CLR Objects) classes. This method is called code-only because the model that is created is done only through code. The code-only approach allows developers to write domain classes without ever looking at or touching a designer or dealing with XML. |
|  | *Entity type*: This represents a particular type of data, such as Employee, Order, or  Product. Entity types are highly structured records with a key.  *Entity set*: This is a logical container for entities of a single type.  **Entities**, **therefore, are instances of entity types**, and entities can be grouped into entity sets. The  model in Figure 3-3 illustrates this concept, which includes three entity types (*SalesPerson*,  *SalesOrderHeader*, and *SalesOrderDetail*), two entity sets (*SalesPerson* and *SalesOrder*), and relationships between the three entity types. |

Complex Types Defined

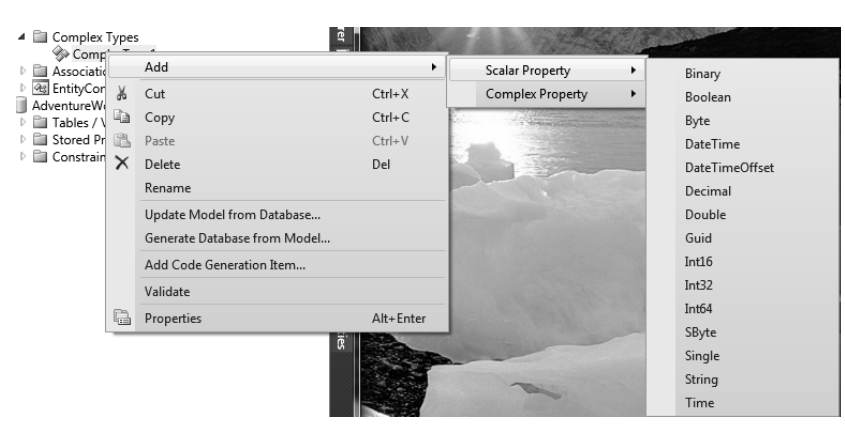
Complex types provide a handy mechanism for storing and encapsulating properties related to one or

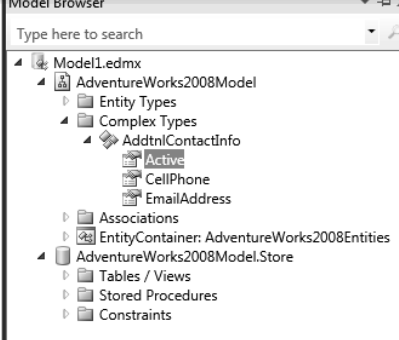
more entities. For example, you may have more than one entity that needs to store phone and email

information. Complex types can also be used to add additional structure to your entities. Regardless of

how you use them, they are very useful. As you will see shortly, complex types are made up of scalar

properties as well as additional complex types.

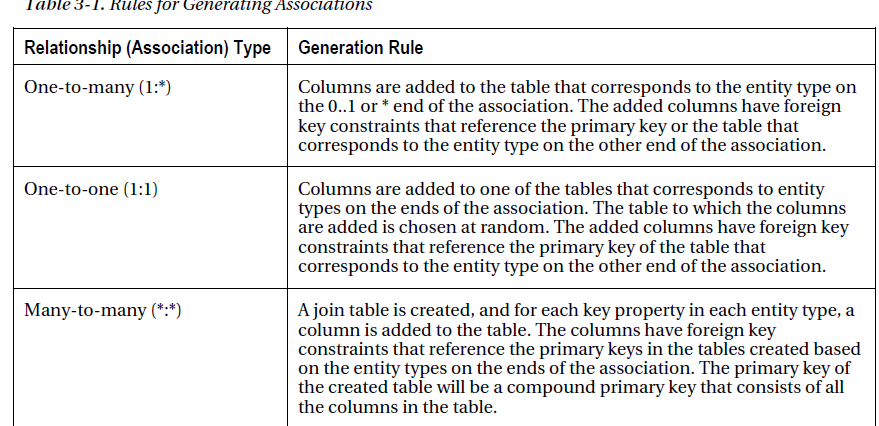
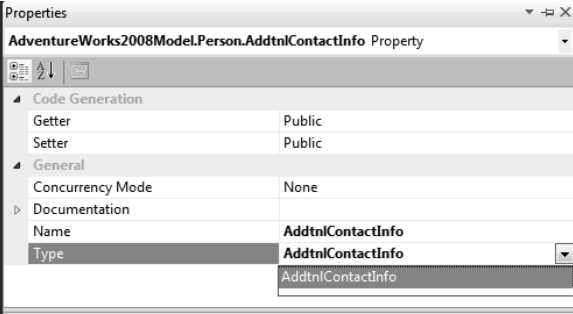
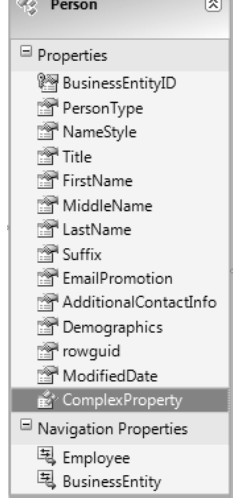




Throughout this exercise we used the terms complex type and complex property. These terms are

not interchangeable. As we have discussed, complex types are made up of scalar properties and other

complex types. A complex property is what is added to an entity and based on the complex type.



Navigation Properties

Figure 3-15 shows the EDM created earlier, highlighting the entities contained in the EDM. In the

figure, the Employee navigation property in the Person entity is highlighted. The properties of that

navigation property are shown on the right in the Properties pane. In this figure you can see that the

navigation properties of each entity inherit the name of their related entity.

