

# ENTITY FRAMEWORK

---

## Seminar 6.1

**Software Engineering**  
ETS Software Engineering  
DSIC – UPV

**DOCENCIA VIRTUAL**

**Finalidad:**  
Prestación del servicio Público de educación superior (art. 1 LOU)

**Responsable:**  
Universitat Politècnica de València.

**Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:**  
<http://www.upv.es/contenidos/DPD/>

**Propiedad Intelectual:**  
Uso exclusivo en el entorno de aula virtual.  
Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.  
La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil

 **UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA**



# Goals

- To know the persistence model of Entity Framework
- To learn the application of the code-first approach
- To develop an example App based on Entity Framework code-first approach

# Contents

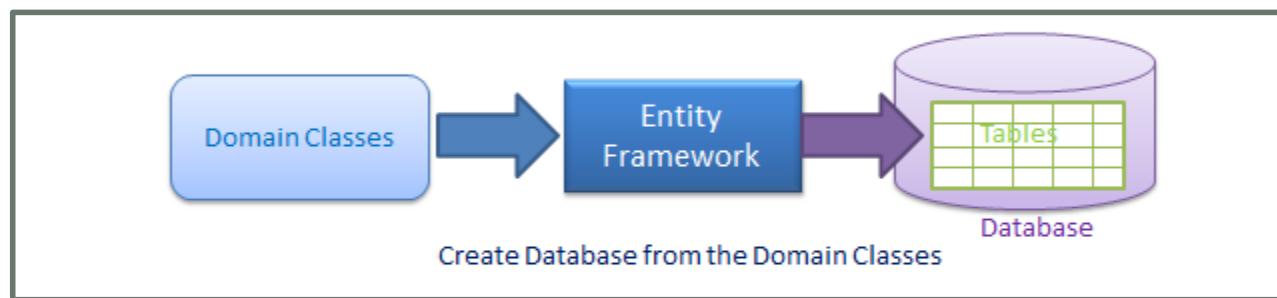
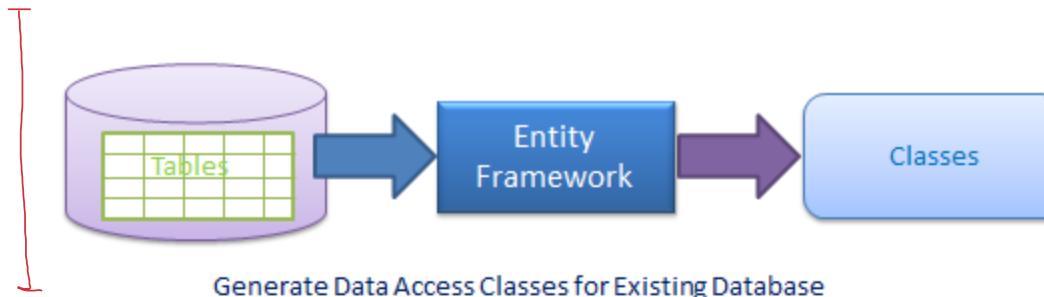
1. Introduction
2. EF DBContext
3. Code-First Conventions
4. Data Annotations
5. DB Initialization
6. DB Operations
7. Loading Strategies
8. Conclusions

# Introduction

- EF is an **Object/Relational Mapping (O/RM)** framework
  - Keep our database design separate from our domain class design.
  - Automate standard CRUD operations (Create, Read, Update & Delete) so that the developer doesn't need to write them manually.
- EF supports three development approaches:
  - **Database-first:** you already have existing database or you want to design your database ahead of other parts of the application
  - **Code-first:** you want to focus on your domain classes and then create the database from your domain classes
  - **Model-first:** you want to design your database schema on the visual designer and then create the database and classes

# Introduction *3 approaches*

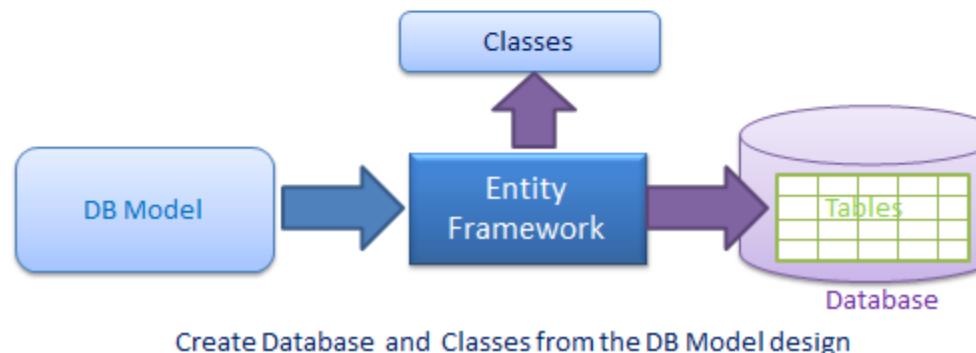
For legacy application.  
You give E.F the database  
and E.F performs the  
classes and relations



## Code-first

Give E.F the C# classes  
and E.F will perform  
the rest

You draw the diagram  
and the entity  
framework performs  
all the classes, relations  
and methods

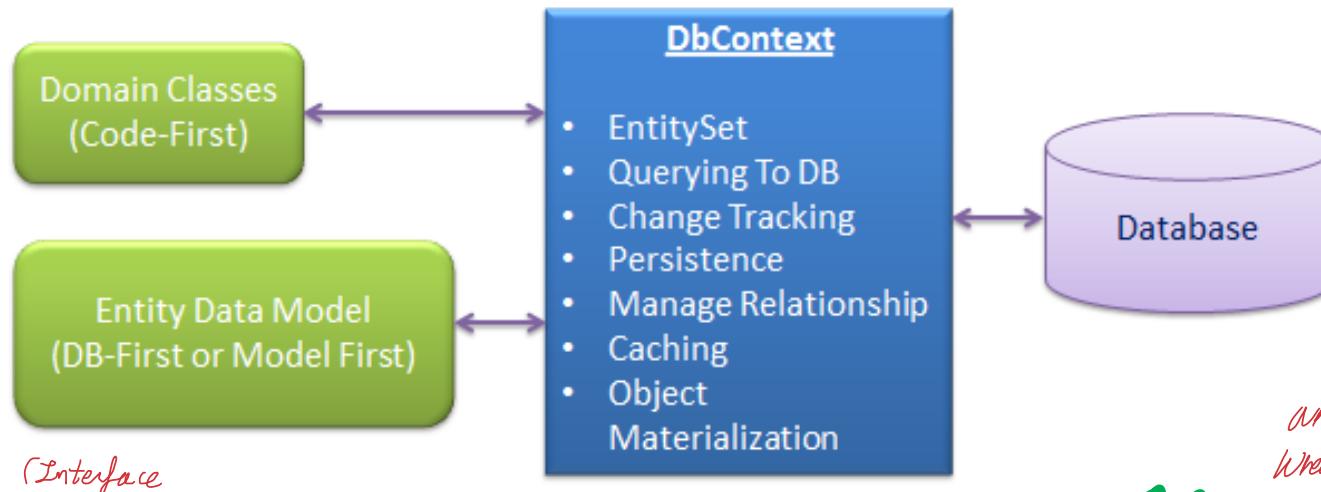


# The Context: DbContext Class

↳ Implements <sup>BOTH</sup> Repository pattern  
 ↳ Unit of Work pattern

- DbContext is an important part of Entity Framework. It is a bridge between your domain or entity classes and the database.

► Only 1 DbContext  
will carry all  
Repositories



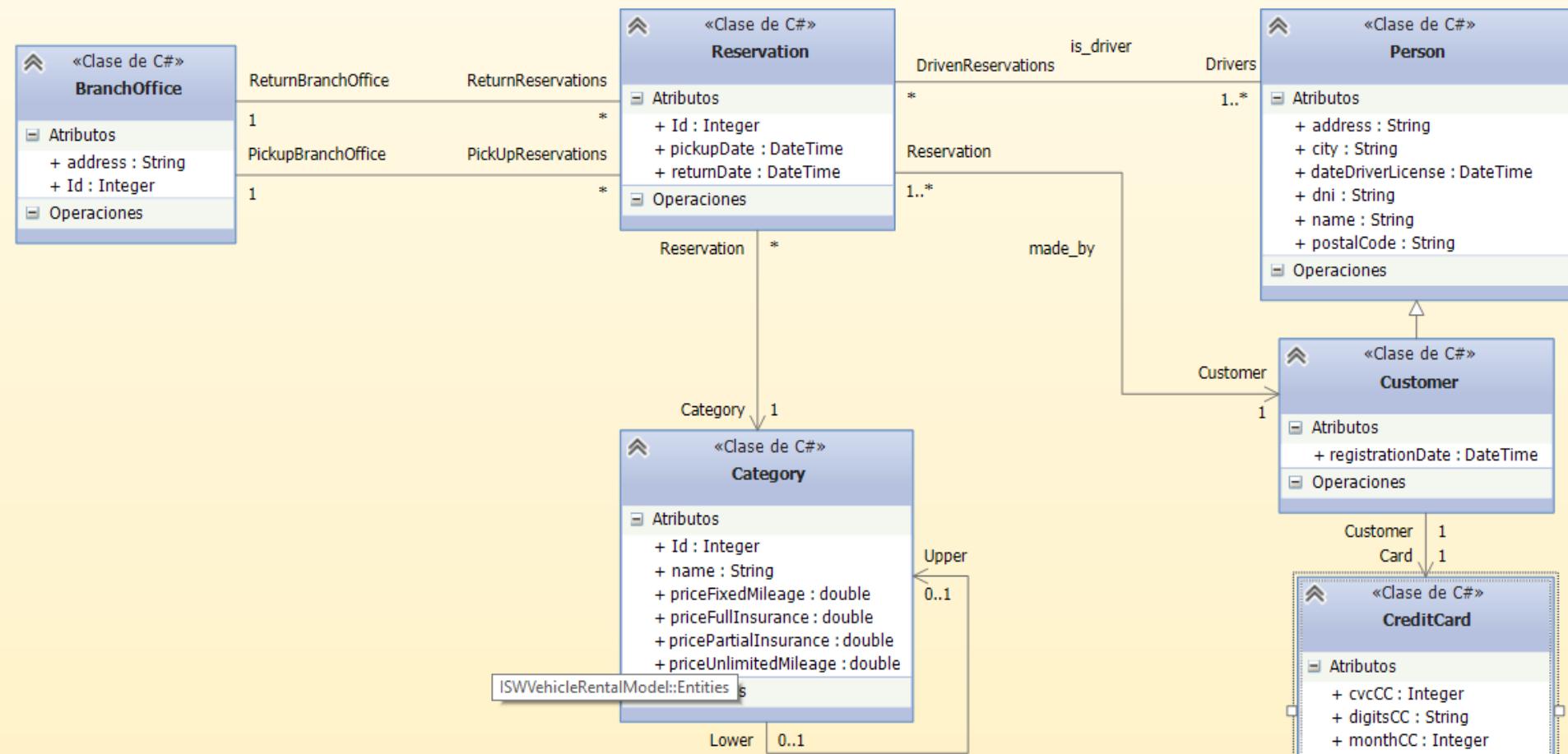
**THE REPOSITORIES) ARE IN DbContext !!**

Where is DbContext?  
Where is DAO?  
!! are !!

# EF DbContext Functionality

- **EntitySet:** DbContext contains entity sets for all the entities which are mapped to DB tables (`DbSet< TEntity >`)
- **Querying:** DbContext converts LINQ-to-Entities queries to SQL queries and sends them to the database.
- **Change Tracking:** It keeps track of changes that occurred in the entities after they have been retrieved from the database.
- **Persisting Data:** It also performs the Insert, Update and Delete operations to the database, based on what the entity states.
- **Caching:** It does first level caching by default. It stores the entities which have been retrieved during its life time.
- **Managing Relationships:** DbContext also manages relationships using fluent API in Code-First approach. *(Manages that objects are correctly connected)*
- **Object Materialization:** DbContext converts raw table data into entity objects.

# Example Domain Model

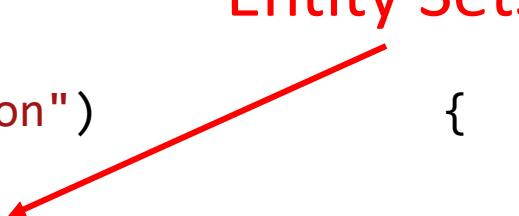


# Example DbContext

```
internal class VehicleRentalDbContext : DbContext
{
    public VehicleRentalDbContext() :
        base("Name=VehicleRentalDbConnection")
}

public IDbSet<BranchOffice> BranchOffices { get; set; }
public IDbSet<Reservation> Reservations { get; set; }
public IDbSet<Category> Categories { get; set; }
public IDbSet<Person> People { get; set; }
public IDbSet<Customer> Customers { get; set; }
public IDbSet<CreditCard> CreditCards { get; set; }
```

**Entity Sets**



# IDbSet & DbSet

What operations are we expecting in IDbSet?  
↳ Repository operations

- IDbSet< TEntity > represents the collection of all the entities of a given type that may be managed in persistent repository (DB).
- DbSet< TEntity > is a concrete implementation of IDbSet.
- DbSet objects are created from a DbContext object using the method  
`DbContext.Set< TEntity >()`

IRepository  $\longleftrightarrow$  IDbSet  
||  
IDAL

# Code-First Conventions

- Code First APIs create the database and map domain classes with the database using default code-first conventions
  - **Type Discovery** Convention
  - **Primary Key** Convention
  - **Relationship** Convention
  - **Foreign key** Convention
  - **Inheritance** Convention
- A convention is a set of default rules to automatically configure a conceptual model based on domain class definitions

# Type-Discovery Convention

- Code-First will create tables for classes included as DbSet properties
- Code-First also includes any referenced types included in these classes

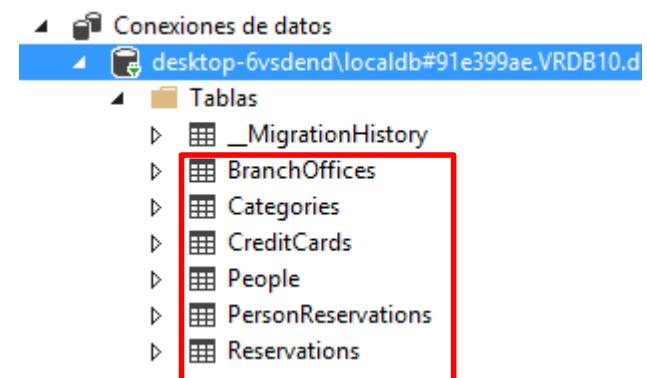
# Example Type-Discovery Convention

- EF automatically generates a table for each DbSet Entity

```
public IDbSet<BranchOffice> BranchOffices { get; set; }
public IDbSet<Reservation> Reservations { get; set; }
public IDbSet<Category> Categories { get; set; }
public IDbSet<Person> People { get; set; }
public IDbSet<Customer> Customers { get; set; }
public IDbSet<CreditCard> CreditCards { get; set; }
```

Names of tables in Plural. E.g.  
People instead of Person

Additional tables for many-to-many relationships. E.g.  
PersonReservations



# Primary-Key Convention

- Code-First will create a primary key for a property if the property name is Id or <class name>Id
- The type of a primary key property can be anything, but if the type of the primary key property is numeric or GUID, it will be configured as an identity column

# Example Primary-Key Convention

The screenshot illustrates the implementation of a primary-key convention across three different modeling and execution layers:

- Database Layer (left):** Shows the `Categories` table structure. The primary key column `Id` is highlighted with a red box.
- Object Layer (center):** Shows the `Category` class from a UML class diagram. The primary key attribute `+ Id : Integer` is highlighted with a red box.
- Code Layer (bottom):** Shows the T-SQL `CREATE TABLE` statement for the `Categories` table. The primary key column `[Id]` is highlighted with a red box.

The UML class diagram also includes other attributes: `+ name : String`, `+ priceFixedMileage : double`, `+ priceFullInsurance : double`, `+ pricePartialInsurance : double`, and `+ priceUnlimitedMileage : double`. The relationship between the database table and the UML class is shown with multiplicity 1 at the class side and 0..1 at the table side.

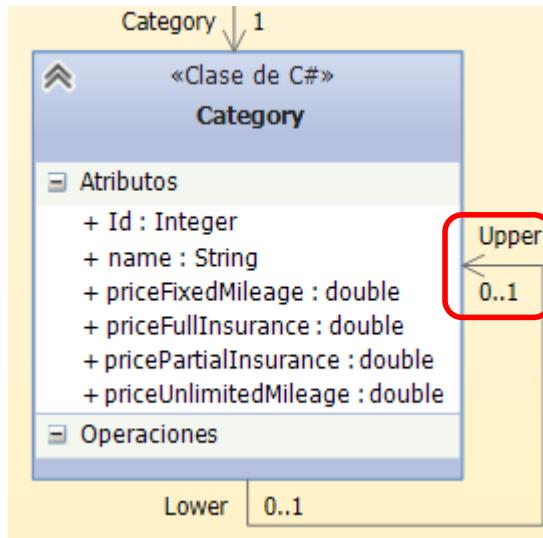
```

CREATE TABLE [dbo].[Categories] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [name] NVARCHAR (MAX) NULL,
    [priceUnlimitedMileage] FLOAT (53) NOT NULL,
    [priceFixedMileage] FLOAT (53) NOT NULL,
    [priceAdditionalKm] FLOAT (53) NOT NULL,
    [priceFullInsurance] FLOAT (53) NOT NULL,
)
  
```

# Relationship Conventions

- If your classes include two reference properties, Code First will assume a **one-to-one relationship**
- If your classes contain a reference and a collection navigation property, Code First assumes a **one-to-many relationship**.
- If your classes include two collection properties, Code First will use a **many-to-many relationship**.

# One-to-One Relationship Example



```
public partial class Category
{
    public double priceUnlimitedMileage...
    public double priceFixedMileage...
    public string name...
    public double priceFullInsurance...
    public double pricePartialInsurance...
    public int Id
    {
        get;
        set;
    }

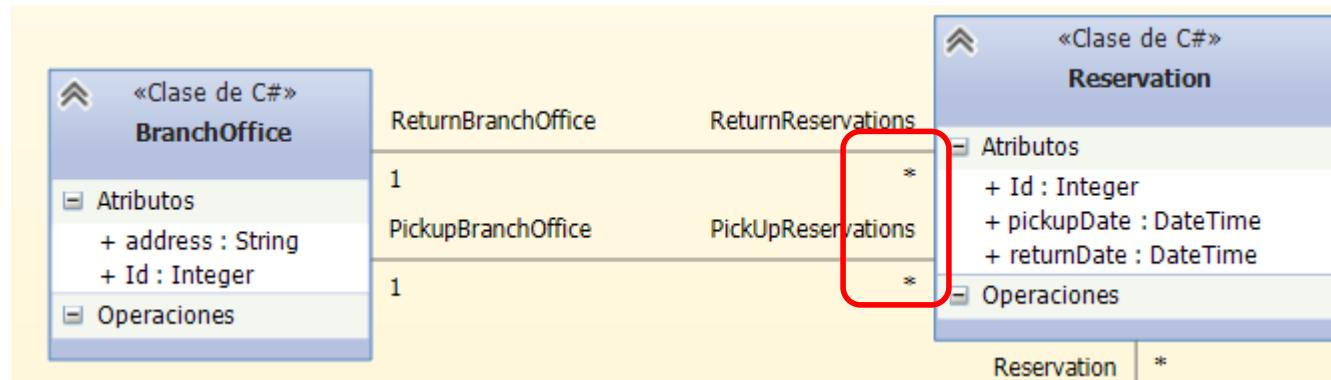
    public virtual Category Upper
    {
        get;
        set;
    }
}
```

# One-to-One Relationship Example

The screenshot shows the SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer displays a database structure with tables like `BranchOffices`, `Categories`, and `People`. The `Categories` table is selected, and its properties are shown in the center pane. The `Upper_Id` column is highlighted with a red box. On the right, the `Claves` (Keys) section shows a primary key constraint `PK_dbo.Categories` on the `Id` column and a foreign key constraint `FK_dbo.Categories_dbo.Categories_Upper_Id` on the `Upper_Id` column. Below the table properties, the `Diseñar` (Design) tab is active, showing the T-SQL definition of the table:

```
[priceFullInsurance] FLOAT (53) NOT NULL,  
[pricePartialInsurance] FLOAT (53) NOT NULL,  
[Upper_Id] INT NULL,  
CONSTRAINT [PK_dbo.Categories] PRIMARY KEY CLUSTERED ([Id] ASC),  
CONSTRAINT [FK_dbo.Categories_dbo.Categories_Upper_Id] FOREIGN KEY ([Upper_Id])
```

# One-to-Many Relationship Example



```
public partial class BranchOffice
{
    public string address...
    public int Id...
    public virtual ICollection<Reservation> PickUpReservations
    {
        get;
        set;
    }

    public virtual ICollection<Reservation> ReturnReservations
    {
        get;
        set;
    }
}
```

# One-to-Many Relationship Example



```
public partial class Reservation
{
    public int Id...
    public DateTime pickupDate...
    public DateTime returnDate...
    public virtual category Category...
    [InverseProperty("PickUpReservations")]
    public virtual BranchOffice PickupBranchOffice
    {
        get;
        set;
    }
    [InverseProperty("ReturnReservations")]
    public virtual BranchOffice ReturnBranchOffice
    {
        get;
        set;
    }
}
```

# One-to-Many Relationship Example

Diagram illustrating a One-to-Many relationship between Reservations and BranchOffices.

**Object Explorer (Left):**

- Connexiones de datos
  - desktop-6vsdend\localdb#91e399ae.VRDB10.db
- Tablas
  - \_MigrationHistory
  - BranchOffices
  - Categories
  - CreditCards
  - People
  - PersonReservations
  - Reservations** (Selected)
- Vistas
- Procedimientos almacenados
- Funciones
- Sinónimos
- Tipos
- Ensamblados

**Server Explorer (Top):**

Shows the structure of the Reservations table in the VRDB10 database.

	Nombre	Tipo de datos	Permitir valores NU
Id	int	<input type="checkbox"/>	
pickupDate	datetime	<input type="checkbox"/>	
returnDate	datetime	<input type="checkbox"/>	
Category_Id	int	<input checked="" type="checkbox"/>	
Customer_dni	nvarchar(128)	<input checked="" type="checkbox"/>	
PickupBranchOffice_Id	int	<input checked="" type="checkbox"/>	
ReturnBranchOffice_Id	int	<input checked="" type="checkbox"/>	

**Properties (Right):**

- Claves (1)**: PK\_dbo.Reservations (Clave principal, Clustered: Id)
- Restricciones CHECK (0)**
- Índices (4)**: IX\_Category\_Id (Category\_Id), IX\_Customer\_dni (Customer\_dni), IX\_PickupBranchOffice\_Id (PickupBranchOffice\_Id), IX\_ReturnBranchOffice\_Id (ReturnBranchOffice\_Id)
- Claves externas (4)**: FK\_dbo.Reservations\_dbo.Categories\_Category\_Id (Id), FK\_dbo.Reservations\_dbo.People\_Customer\_dni (dni), FK\_dbo.Reservations\_dbo.BranchOffices\_PickupBranchOffice\_Id (Id), FK\_dbo.Reservations\_dbo.BranchOffices\_ReturnBranchOffice\_Id (Id)

**Script (Bottom):**

```

CREATE TABLE [dbo].[Reservations] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [pickupDate] DATETIME NOT NULL,
    [returnDate] DATETIME NOT NULL,
    [Category_Id] INT NULL,
    [Customer_dni] NVARCHAR (128) NULL,
    [PickupBranchOffice_Id] INT NULL,
    [ReturnBranchOffice_Id] INT NULL,
    CONSTRAINT [PK_dbo.Reservations] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_dbo.Reservations_dbo.Categories_Category_Id] FOREIGN KEY ([Category_Id]) REFERENCES [dbo].Categories ([Category_Id]),
    CONSTRAINT [FK_dbo.Reservations_dbo.People_Customer_dni] FOREIGN KEY ([Customer_dni]) REFERENCES [dbo].People ([Customer_dni]),
    CONSTRAINT [FK_dbo.Reservations_dbo.BranchOffices_PickupBranchOffice_Id] FOREIGN KEY ([PickupBranchOffice_Id]) REFERENCES [dbo].BranchOffices ([Id]),
    CONSTRAINT [FK_dbo.Reservations_dbo.BranchOffices_ReturnBranchOffice_Id] FOREIGN KEY ([ReturnBranchOffice_Id]) REFERENCES [dbo].BranchOffices ([Id])
)

```

# Many-to-Many Relationships Example



```

public partial class Reservation
{
    public int Id{...}
    public DateTime pickupDate{...}
    public DateTime returnDate{...}
    public virtual Category Category{...}
    [InverseProperty("PickUpReservations")]
    public virtual BranchOffice PickupBranchOffice{...}
    [InverseProperty("ReturnReservations")]
    public virtual BranchOffice ReturnBranchOffice{...}
    public virtual ICollection<Person> Drivers
    {
        get;
        set;
    }
}

```

```

public partial class Person
{
    [Key]
    public string dni{...}
    public string name{...}
    public string address{...}
    public string city{...}
    public string postalCode{...}
    public DateTime dateDriverLicense{...}
    public virtual ICollection<Reservation> DrivenReservations{...}
}

```

# Many-to-Many Relationships Example

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer tree shows a database structure with tables like PersonReservations, People, and Reservations. A red box highlights the PersonReservations table under the People node. In the center, the 'Diseñar' (Design) tab of the Object Explorer is selected, showing the columns Person\_dni (nvarchar(128)) and Reservation\_Id (int). A red box highlights these two columns. A green callout box labeled 'New table with composed primary key' points to the columns. To the right, the properties pane shows:

- Claves (1)**: PK\_dbo.PersonReservations (Clave principal, Clustered: Person\_dni, Reservation\_Id)
- Restricciones CHECK (0)**
- Índices (2)**: IX\_Person\_dni (Person\_dni), IX\_Reservation\_Id (Reservation\_Id)
- Claves externas (2)**: FK\_dbo.PersonReservations\_dbo.People\_Person\_dni (dni), FK\_dbo.PersonReservations\_dbo.Reservations\_Reservation\_Id (Id)
- Desencadenadores (0)**

At the bottom, the T-SQL pane displays the CREATE TABLE statement:

```
CREATE TABLE [dbo].[PersonReservations] (
    [Person_dni] NVARCHAR (128) NOT NULL,
    [Reservation_Id] INT NOT NULL,
    CONSTRAINT [PK_dbo.PersonReservations] PRIMARY KEY CLUSTERED ([Person_dni] ASC, [Reservation_Id] ASC),
    CONSTRAINT [FK_dbo.PersonReservations_dbo.People_Person_dni] FOREIGN KEY ([Person_dni]) REFERENCES [dbo].[People],
    CONSTRAINT [FK_dbo.PersonReservations_dbo.Reservations_Reservation_Id] FOREIGN KEY ([Reservation_Id]) REFERENCES
);
```

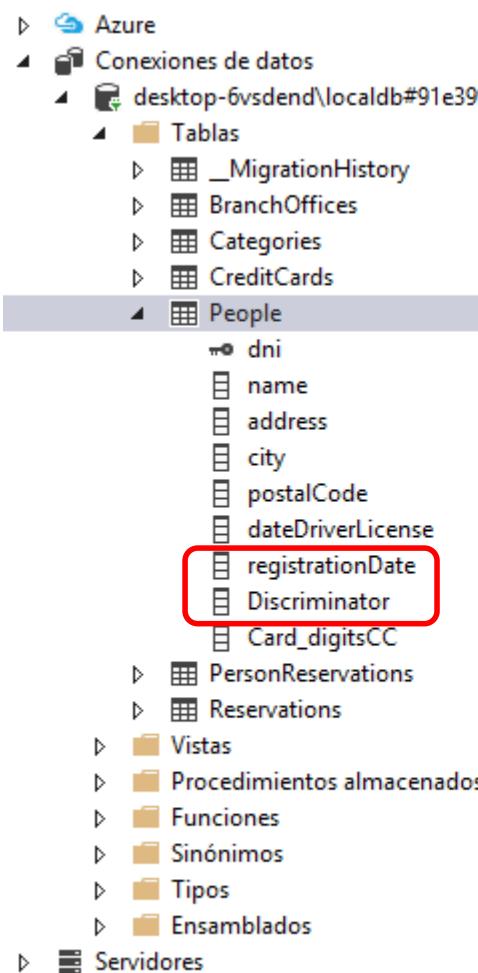
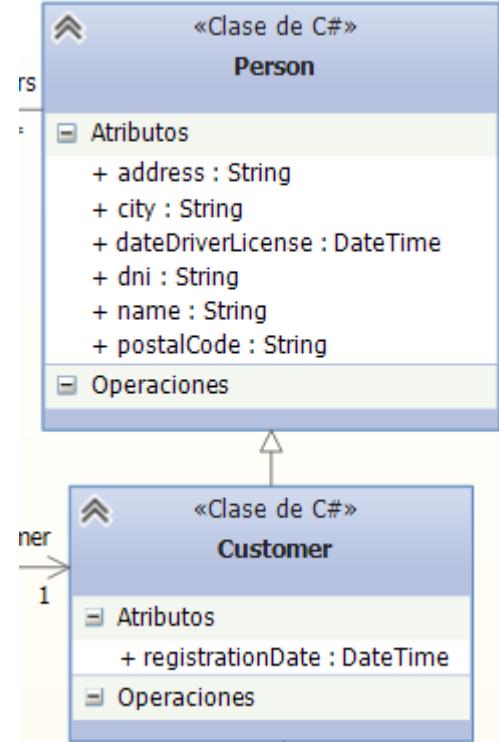
A red box highlights the primary key and foreign key constraints in the T-SQL code. A green callout box labeled 'Foreign keys' points to the foreign key declarations.

Foreign keys

# Inheritance Convention

- **Table per Hierarchy (TPH):** This approach suggests one table for the entire class inheritance hierarchy.
  - Table includes discriminator column which distinguishes between inheritance classes.
  - Default inheritance mapping strategy in Entity Framework.
- **Table per Type (TPT):** This approach suggests a separate table for each domain class.
- **Table per Concrete class (TPC):** This approach suggests one table for one concrete class, but not for the abstract class.
  - The properties of the abstract class will be part of each table of the concrete classes.

# Table per Hierarchy Example



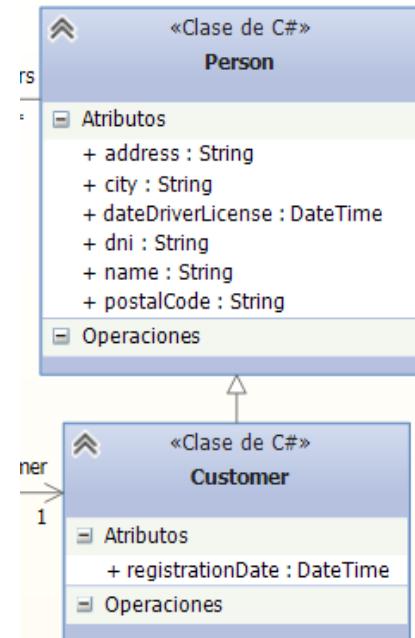
	Nombre	Tipo de datos	Permitir valores NULL	Prede
•	dni	nvarchar(128)	<input type="checkbox"/>	
	name	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	address	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	city	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	postalCode	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	dateDriverLicense	datetime	<input type="checkbox"/>	
	registrationDate	datetime	<input checked="" type="checkbox"/>	
	Discriminator	nvarchar(128)	<input type="checkbox"/>	
	Card_digitsCC	nvarchar(128)	<input checked="" type="checkbox"/>	

The screenshot shows the SQL Server Object Explorer with the 'T-SQL' tab selected. It displays the `CREATE TABLE` statement for the **dbo].[People]** table. The attributes dni, name, address, city, postalCode, dateDriverLicense, registrationDate, and Discriminator are defined. The attributes registrationDate and Discriminator are highlighted with a red rectangle.

```

CREATE TABLE [dbo].[People] (
    [dni] NVARCHAR (128) NOT NULL,
    [name] NVARCHAR (MAX) NULL,
    [address] NVARCHAR (MAX) NULL,
    [city] NVARCHAR (MAX) NULL,
    [postalCode] NVARCHAR (MAX) NULL,
    [dateDriverLicense] DATETIME NOT NULL,
    [registrationDate] DATETIME NULL,
    [Discriminator] NVARCHAR (128) NOT NULL,
    [card_digitsCC] NVARCHAR (128) NULL,
    CONSTRAINT [PK_dbo.People] PRIMARY KEY CLUSTERED
    CONSTRAINT [FK_dbo.People_dbo.CreditCards_Card]
  
```

# Table per Hierarchy Example



dni	name	address	city	postalCode	dateDriverLice...	registrationDate	Discriminator
1 11111111A	asdf Javier Jaen	sdf Camino de Vera...	asdf Valencia	dsf 46960	16/03/2016 5:56.. 23/05/2014 0:00..	16/03/2016 6:04... 11/12/2015 0:00...	Customer Customer
2 22222222B	asdf Vicente Nacher	sdf C/ Goya, 13	asdf Valencia	dsf 46023	16/03/2016 5:56.. 15/03/2016 17:2..	NULL 15/03/2016 17:2...	Person Customer
3 33333333C	asdf Patricia Pons	sdf C/Goya 16	asdf Valencia	dsf 46960	16/03/2016 5:56.. 15/03/2016 17:2..	NULL 15/03/2016 17:2...	Person Customer
44444444D	asdf	asdf	asd	asdf	15/03/2016 17:3..	15/03/2016 17:3...	Customer
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

# Conventions Key Facts

Default Convention For	Description
Table Name	<Entity Class Name> + 's' EF will create DB table with entity class name suffixed by 's'
Primary key Name	1) Id 2) <Entity Class Name> + "Id" (case insensitive)  EF will create primary key column for the property named Id or <Entity Class Name> + "Id" (case insensitive)
Foreign key property Name	By default EF will look for foreign key property with the same name as principal entity primary key name. If foreign key property does not exists then EF will create FK column in Db table with <Dependent Navigation Property Name> + "_" + <Principal Entity Primary Key Property Name> e.g. EF will create Standard_StandardId foreign key column into Students table if Student entity does not contain foreignkey property for Standard where Standard contains StandardId
Null column	EF creates null column for all reference type properties and nullable primitive properties.
Not Null Column	EF creates NotNull columns for PrimaryKey properties and non-nullable value type properties.
DB Columns order	EF will create DB columns same as order of properties in an entity class. However, primary key columns would be moved first.
Properties mapping to DB	By default all properties will map to database. Use [NotMapped] attribute to exclude property or class from DB mapping.
Cascade delete	Enabled By default for all types of relationships.

# Domain Classes Configuration

- To **override** the previous conventions by configuring your domain classes to provide EF with the information it needs
- Two ways to configure your domain classes
  - **DataAnnotations:** Attribute based configuration, that may be applied to domain classes and their properties
  - **Fluent API:** An advanced way of specifying model configuration that covers everything that data annotations can do in addition to some more advanced configuration not possible with data annotations

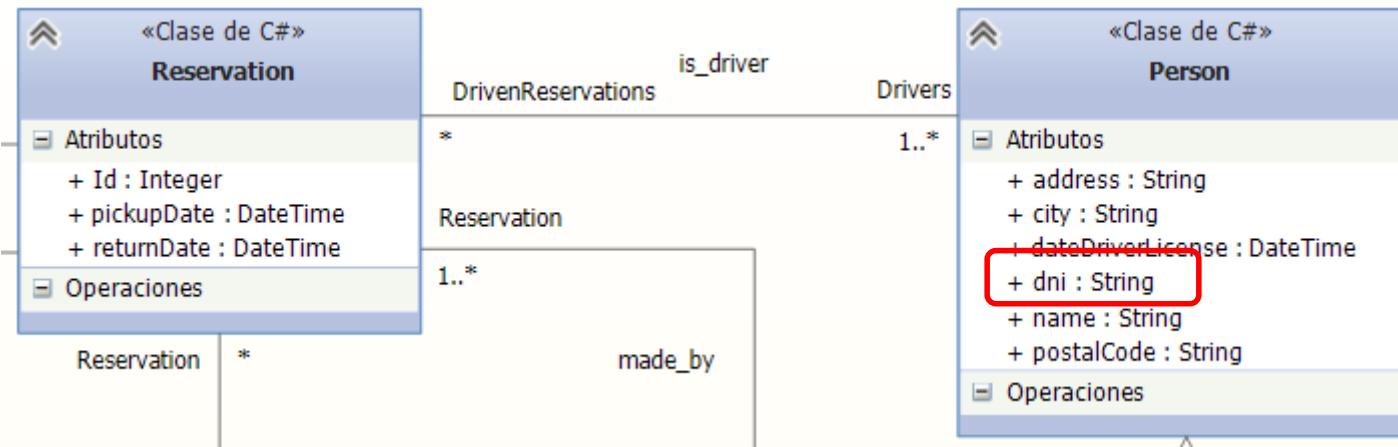
# Data Annotations

- **System.ComponentModel.DataAnnotations** includes the following attributes that impacts the nullability or size of the column.
  - Key, DatabaseGeneratedAttribute  
(DatabaseGeneratedOption.None)
  - Timestamp
  - ConcurrencyCheck
  - Required
  - MinLength
  - MaxLength
  - StringLength

# Data Annotations

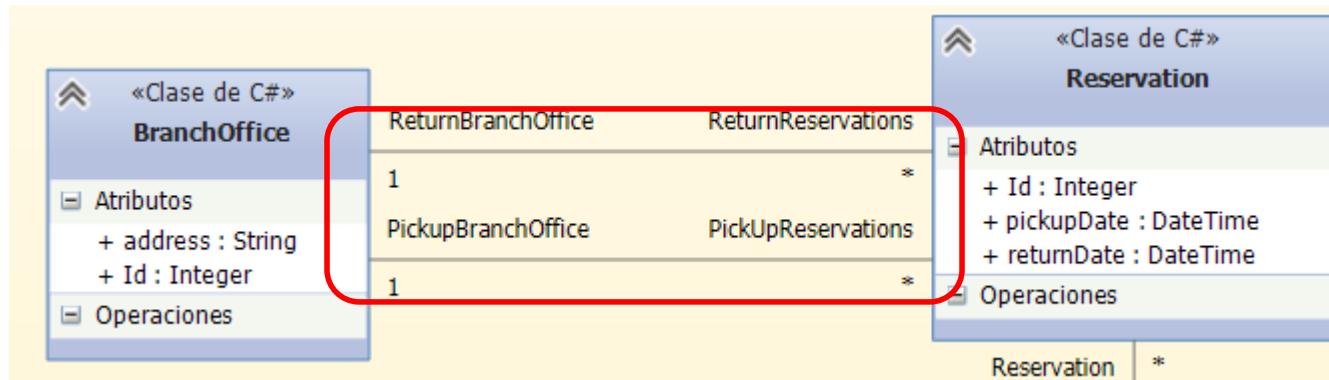
- **System.ComponentModel.DataAnnotations.Schema.Schema** namespace includes the following attributes that impacts the schema of the database.
  - Table
  - Column
  - Index
  - ForeignKey
  - NotMapped
  - InverseProperty

# Data Annotations Example



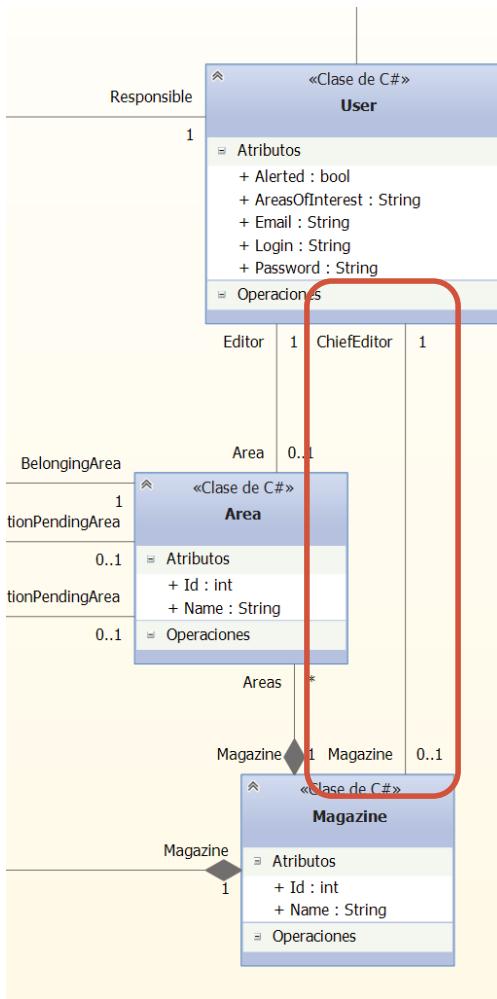
```
public partial class Person
{
    [Key]
    public string dni...
    public string name...
    public string address...
    public string city...
    public string postalCode...
    public DateTime dateDriverLicense...
    public virtual ICollection<Reservation> DrivenReservations...
}
```

# Data Annotations Example



```
public partial class Reservation
{
    public int Id...
    public DateTime pickupDate...
    public DateTime returnDate...
    public virtual Category Category...
    [InverseProperty("PickUpReservations")]
    public virtual BranchOffice PickupBranchOffice...
    [InverseProperty("ReturnReservations")]
    public virtual BranchOffice ReturnBranchOffice...
    public virtual ICollection<Person> Drivers
    {
        get;
        set;
    }
}
```

# Data Annotations Example



```

public partial class User : Person{
    public string Email{get;set;}
    public string Login{get;set;}
    public string Password{get;set;}
    public string AreasOfInterest{get;set;}
    public bool Alerted{get;set;}
    public virtual ICollection<Paper> PapersResponsible{get;set;}
    public virtual Area Area{get;set;}
    public virtual Magazine Magazine{get;set;}
}

public partial class Magazine{
    public int Id{get;set;}
    public string Name{get;set;}
    public virtual ICollection<Area> Areas{get;set;}
    public virtual ICollection<Issue> Issues{get;set;}
}
  
```

[Required] → When cardinality = 1

```

public virtual User ChiefEditor{get;set;}
}
  
```

# Tasks

- Understand the meaning of the different data annotations:

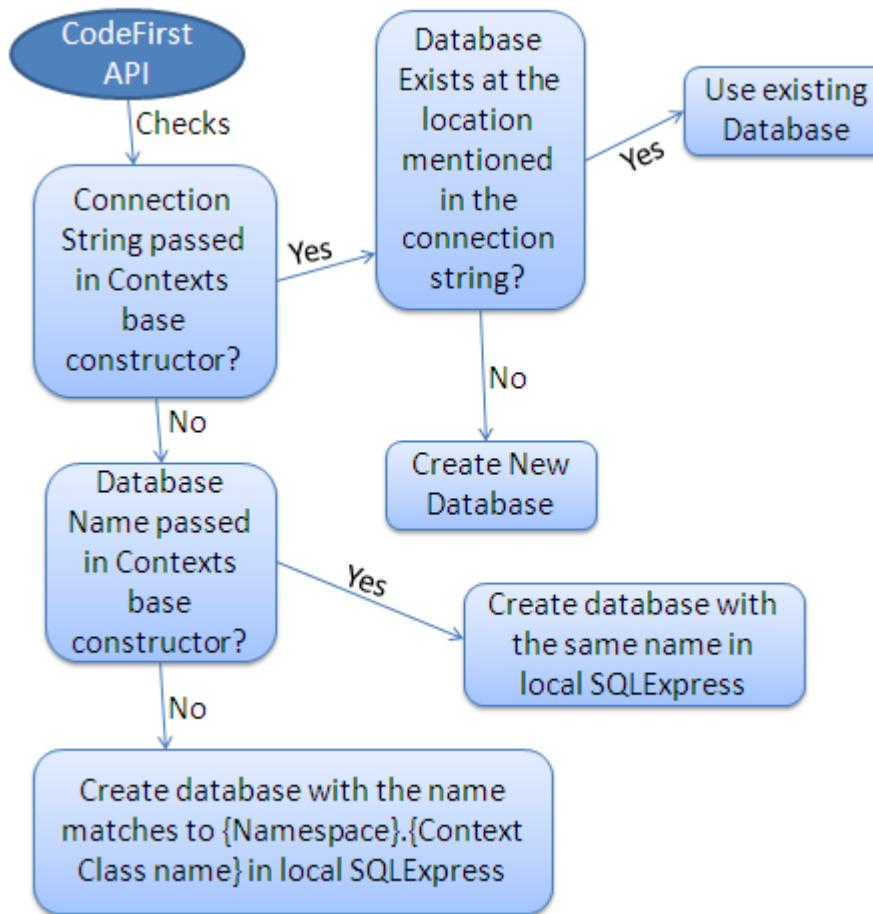
[http://www.tutorialspoint.com/entity\\_framework/entity\\_framework\\_data\\_annotations.htm](http://www.tutorialspoint.com/entity_framework/entity_framework_data_annotations.htm)

- Advanced Task. Understand how Fluent API Works

[http://www.tutorialspoint.com/entity\\_framework/entity\\_framework\\_fluent\\_api.htm](http://www.tutorialspoint.com/entity_framework/entity_framework_fluent_api.htm)

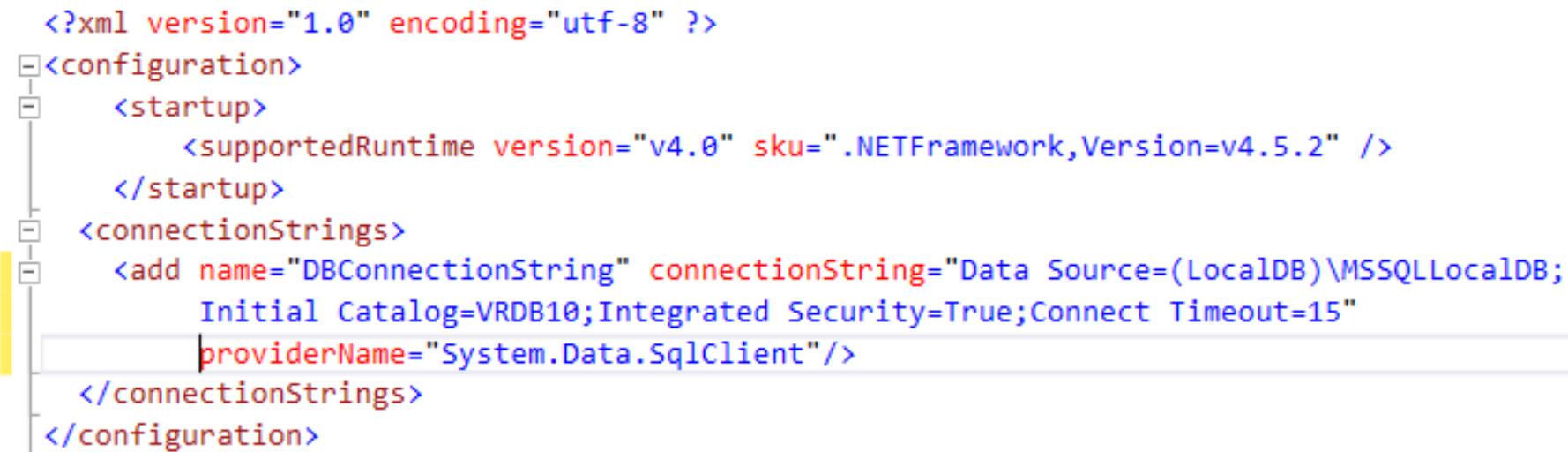
# Database Initialization

- Code First creates a database automatically according to the following workflow



# Database Initialization

- Define connection string in App.config or web.config and specify connection string name starting with "name=" in the base constructor of the context class



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <connectionStrings>
    <add name="DBConnectionString" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
      Initial Catalog=VRDB10;Integrated Security=True;Connect Timeout=15"
      providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

A screenshot of a code editor showing the configuration section of an application's configuration file. The XML code defines a connection string named 'DBConnectionString' with specific connection details for a LocalDB database. The code is color-coded, and the configuration tree on the left shows the hierarchy of the configuration file.

# Database Operations: Using DbContext

- IDbSet offers methods to add, remove and search objects
- Enables to express and execute queries
- Takes query results from the database and transforms them into instances of our model classes
- Can keep track of changes to entities, including adding and deleting, and then triggers the creation of insert, update and delete statements that are sent to the database on demand

# Example of operations in EF

```
VehicleRentalDbContext context = new  
VehicleRentalDbContext();  
  
context.Categories.Add(new Category("luxury",  
23, 12, 2, 32, 14));  
context.People.Add(new Person("22222222A", ...));  
Person p = context.People.Find("12345678A");  
context.People.Remove(p);  
context.SaveChanges();  
  
context.People.Where(person => person.Id ==  
"123456789A")
```

# Task

- Explore VehicleRentalApp and identify the different EF elements
- What data Access pattern has been implemented and where is it implemented in the project?
- What are the benefits of the proposed architecture and of the selected EF technology?
- What are the benefits of having the IDAL interface

# Conclusions

- EF is an **Object/Relational Mapping (O/RM)** framework
- Automate standard CRUD operations (Create, Read, Update & Delete) so that the **developer doesn't need to write them manually**
- **Code-first:** you want to focus on your domain classes and then create the database from your domain classes
- Code First APIs create the database and map domain classes with the database using default code-first conventions
- DbContext enables to express and execute queries, keeps changes tracking and materializes objects

# References

- Hirani, Z., et al. Entity Framework 6 Recipes 2013.
- Entity Framework Documentation (MSDN)
  - [Entity Framework Code First Conventions](#)
  - [Entity Framework Code First Data Annotations](#)
  - [Entity Framework Fluent API – Relationships](#)
  - [Entity Framework Fluent API - Configuring and Mapping Properties and Types](#)
  - [Entity Framework Loading Related Entities](#)
- Tutoriales Online
  - [<http://www.entityframeworktutorial.net>](#)
  - [\[http://www.tutorialspoint.com/entity\\\_framework/\]\(http://www.tutorialspoint.com/entity\_framework/\)](#)

# LOADING STRATEGIES

---

# Entities Loading Strategies

- **Eager loading:** a query for one type of entity also loads related entities as part of the query.
  - Achieved by the use of the **Include()** method.
- **Lazy loading:** An entity or collection of entities are automatically loaded from the database the first time that a property referring to the entity/entities is accessed.
  - Delaying the loading of related data, until requested.
  - Achieved by creating instances of derived proxy types and then overriding virtual properties to add the loading hook.
  - Default loading mechanism
- **Explicit loading:** if disabled the lazy loading, it is still possible to lazily load related entities with an explicit call.
  - No ambiguity or possibility of confusion regarding when a query is run.
  - Use the **Load()** method on the related entity's entry.

# Eager Loading Example

```
class Program {  
  
    static void Main(string[] args) {  
  
        using (var context = new UniContextEntities()) {  
            // Load all students and related enrollments  
            var students = context.Students  
                .Include(s => s.Enrollments).ToList();  
  
            foreach (var student in students) {  
                string name = student.FirstMidName + " " + student.LastName;  
                Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);  
  
                foreach (var enrollment in student.Enrollments) {  
                    Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",  
                        enrollment.EnrollmentID, enrollment.CourseID);  
                }  
            }  
  
            Console.ReadKey();  
        }  
    }  
}
```

# Explicit Loading Example

```
class Program {

    static void Main(string[] args) {

        using (var context = new UniContextEntities()) {

            context.Configuration.LazyLoadingEnabled = false;

            var student = (from s in context.Students where s.FirstMidName ==
                "Ali" select s).FirstOrDefault<Student>();

            string name = student.FirstMidName + " " + student.LastName;
            Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);

            foreach (var enrollment in student.Enrollments) {
                Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
                    enrollment.EnrollmentID, enrollment.CourseID);
            }

            Console.WriteLine();
            Console.WriteLine("Explicitly loaded Enrollments");
            Console.WriteLine();

            context.Entry(student).Collection(s => s.Enrollments).Load();
            Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);

            foreach (var enrollment in student.Enrollments) {
                Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
                    enrollment.EnrollmentID, enrollment.CourseID);
            }

            Console.ReadKey();
        }
    }
}
```