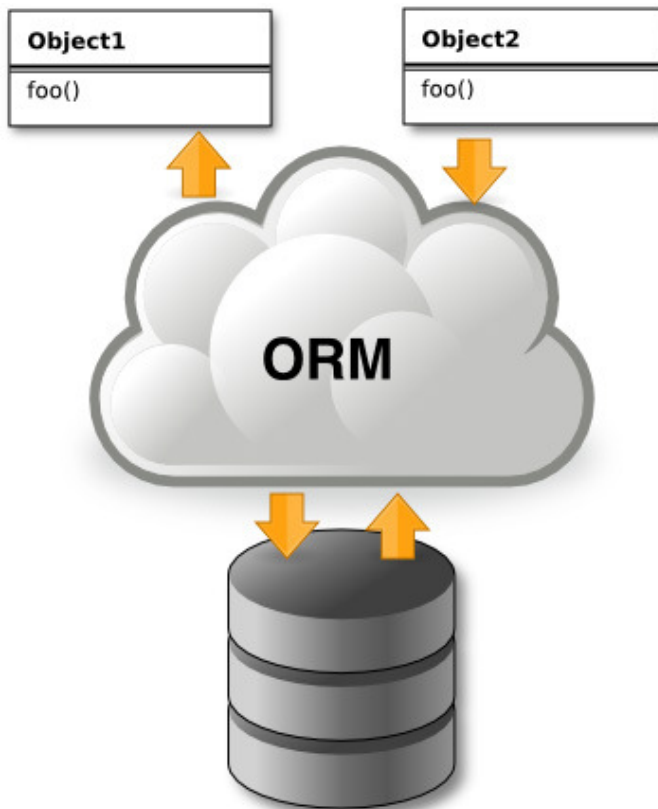


Module 01 – Entity Framework (partie 1)

1.1.	O/RM.....	2
1.2.	Modes de développement.....	3
1.3.	Création du Modèle Entity Framework.....	4
1.4.	Explorer le modèle Entity Framework.....	7
1.4.1.	Cardinalités.....	7
1.4.2.	Propriétés de navigation.....	9
1.4.3.	L'explorateur de modèle.....	10
1.4.4.	L'onglet propriété.....	11
1.4.5.	Mappage de table.....	12
1.5.	Le code généré par EF5.....	13
1.5.1.	Classes POCO.....	13
1.5.2.	DbContext.....	14
1.6.	Opérations CRUD avec DbContext.....	15
1.7.	DataAnnotations avec EF.....	16
1.7.1.	Classes Partielles.....	16
1.7.2.	MetaData.....	17

1.1. O/RM

- Une des problématiques auxquelles les programmeurs doivent faire face est que les **applications sont conçues avec un modèle objet et que les données sont conservées sur un système qui n'est pas objet** (notamment les bases de données relationnelles).
- Lorsqu'on prête attention à la quantité de code qui doit être écrit pour gérer le différent d'impédance entre les différentes représentations (par exemple objet versus bd relationnelle), il est clair qu'il y a place à l'amélioration.
- La solution consiste à utiliser un système de mappage objet-relationnel (en anglais *object relational mapping* O/RM). **L'objectif d'un O/RM est de donner un accès purement objet à un système de base de données relationnel.**



- Un des objectifs d'Entity Framework (EF) est de monter le niveau d'abstraction de la programmation de données, pour éliminer le différent d'impédance mentionné ci-haut. Les bénéfices sont:
 - Les applications peuvent travailler en termes d'un modèle conceptuel centré sur l'application, où on retrouve l'héritage, les relations et les membres complexes.
 - Les applications sont libérées de dépendances directes d'un moteur de base de données spécifique.
 - Les jointures établissant le lien entre les données du modèle conceptuel et les champs de données de la base de données peuvent être établies (et conséquemment changées) sans avoir à modifier le code de l'application.
 - Comme les développeurs de l'application font appel à un modèle conceptuel plutôt qu'à un moteur de base de données directement, le

système de gestion de données peut être changé sans modifier l'application.

1.2. Modes de développement.

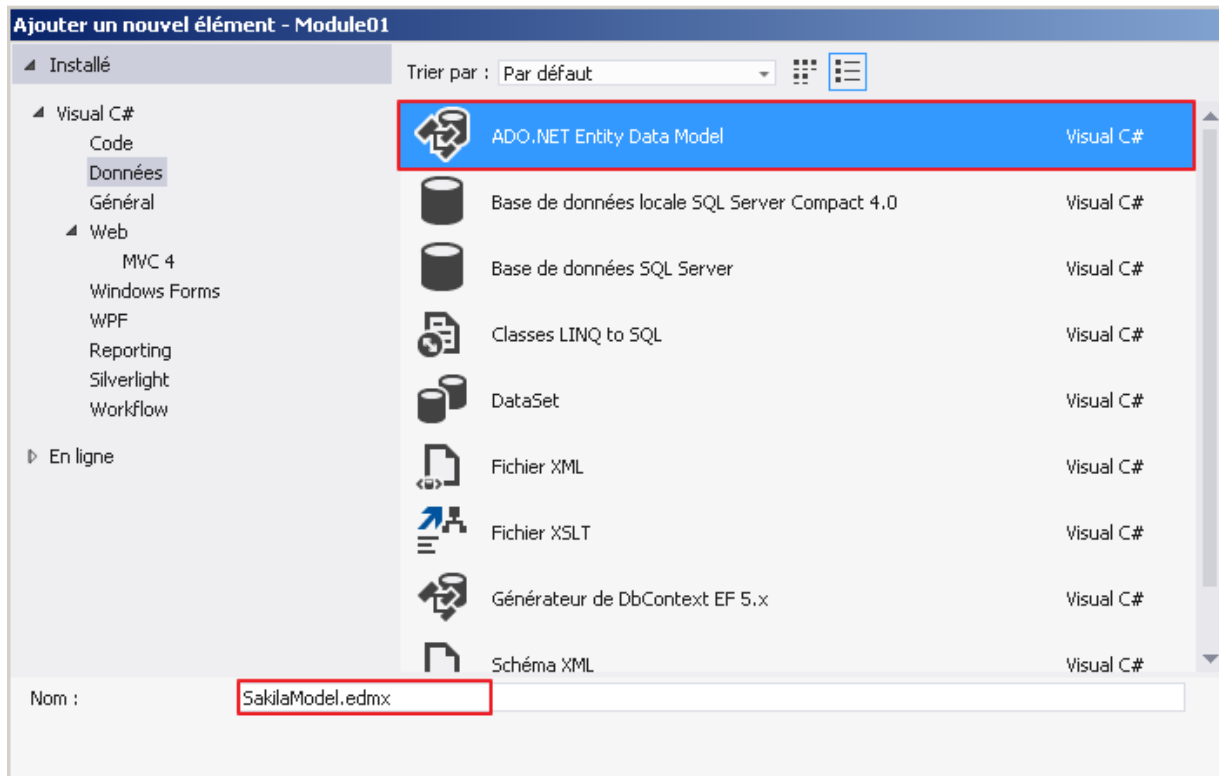
- EF5 permet 4 modes distincts de développement. Le programmeur choisi son approche selon deux facteurs:
 - La base de données existe (ou pas) lors de la création du projet.
 - Le programmeur préfère utiliser un concepteur graphique ou le code pour gérer le modèle.



- Ce document utilise l'approche Database First / avec utilisation du concepteur graphique (EF Designer).

1.3. Création du Modèle Entity Framework

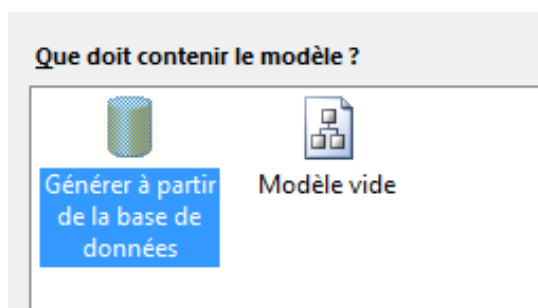
- Dossier Models → Ajouter un nouvel élément → Onglet Données → ADO.NET Entity Data Model



- Après avoir sélectionné le nom, l'assistant de création démarre et nous permet de créer le modèle.



Choisir le contenu du modèle



- Créez vous une nouvelle connexion (ou utilisez celle que vous avez créée au préalable dans l'explorateur de serveur).

Assistant EDM

Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

sv54\sqlexpress.420D73-Sakila.dbo [Nouvelle connexion...]

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☒ Oui, inclure les données sensibles dans la chaîne de connexion.

Chaîne de connexion de l'entité :

```
metadata=res://*/Models.SakilaModel.csdl|res://*/Models.SakilaModel.ssdl|
res://*/Models.SakilaModel.msl;provider=System.Data.SqlClient;provider connection
string="data source=sv54.cmaisonneuve.qc.ca\SQLEXPRESS,1433;initial catalog=420D73-
```

☒ Enregistrer les paramètres de connexion de l'entité dans Web.Config en tant que :

SakilaEntities

< Précédent Suivant > Terminer Annuler

- On doit ensuite sélectionner les objets tables, vues ou procédures stockées à ajouter au modèle.

Assistant EDM

Choisir vos paramètres et objets de base de données

Quels objets de base de données voulez-vous inclure dans votre modèle ?

- ☒ Tables
- ☐ Vues
- ☐ Procédures et fonctions stockées
 - ☒ dbo
 - ☒ CréerCopies
 - ☐ fn_diagramobjects
 - ☐ selectACTORpage
 - ☐ sp_alterdiagram
 - ☐ sp_creatediagram
 - ☐ sp_dropdiagram
 - ☐ sp_helpdiagramdefinition

☐ Mettre au pluriel ou au singulier les noms d'objets générés

☒ Inclure les colonnes clés étrangères dans le modèle

☐ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

Espace de noms du modèle :

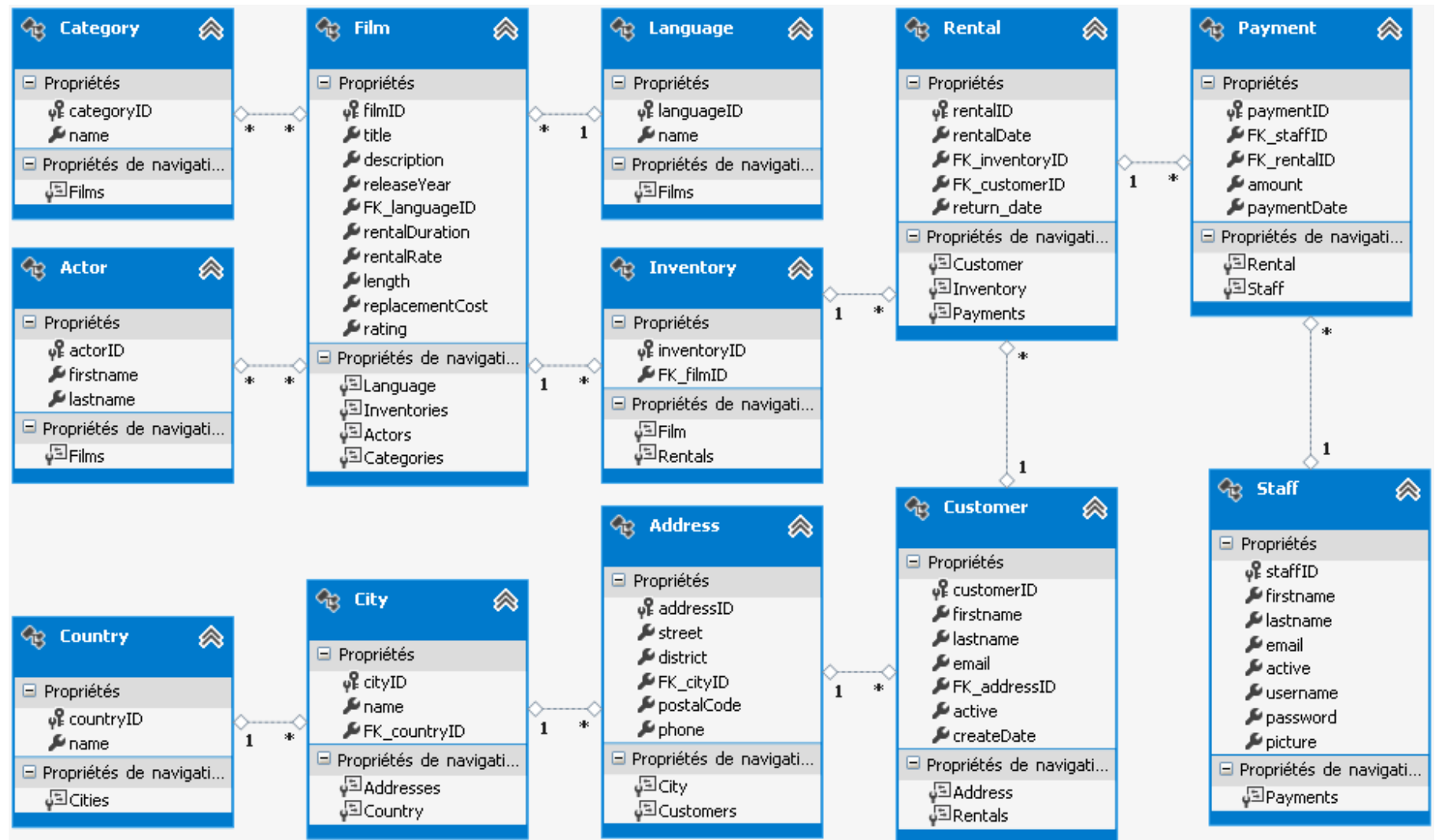
SakilaModel

< Précédent Suivant > Terminer Annuler

Annotations :

- Fonctionne seulement pour les tables dont le nom est en anglais.
- Procédures stockées autres que CRUD

- Le modèle résultant devrait ressembler à :



1.4. Explorer le modèle Entity Framework.

- Comme vous pouvez le constater, le modèle Entity Framework ressemble beaucoup au modèle de données de la base de données, mais avec quelques différences.

1.4.1. Cardinalités

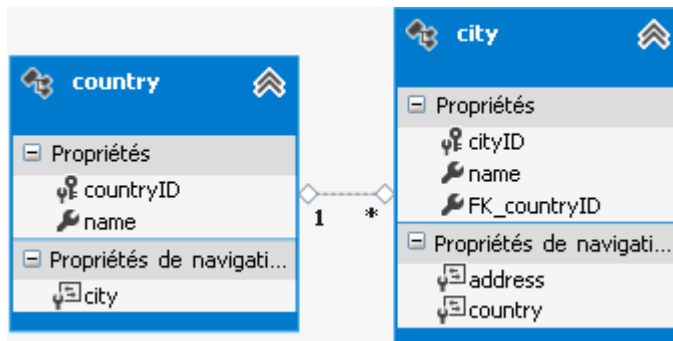
- Une des différences est l'apparence des cardinalités entre les entités.

1.4.1.1. Les relations un à un


- Les relations 1 à 1 sont représentées par le symbole "1" et "0..1"

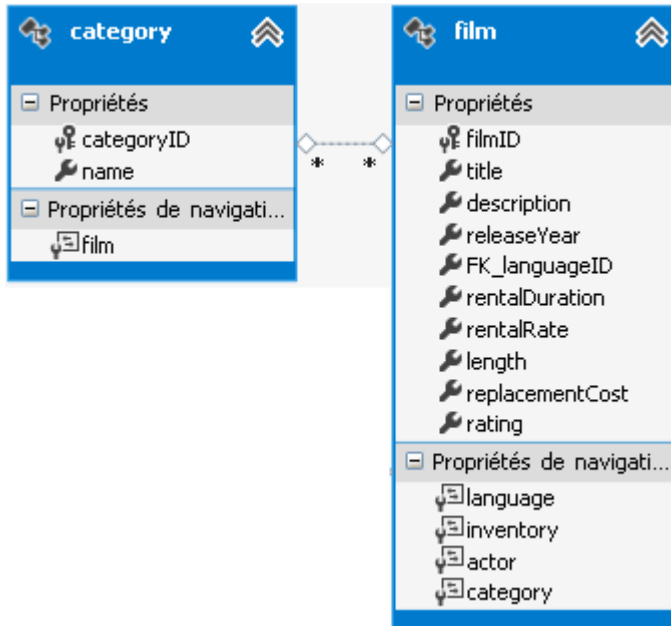
1.4.1.2. Les relations un à plusieurs (1 *)

- Les relations 1 à plusieurs sont représentées par le symbole "1" et "*"
- Par exemple, dans le cas des entités COUNTRY et CITY, un pays peut être associée (ou pas) à une ou plusieurs villes. Une ville **doit** être associée à un pays.

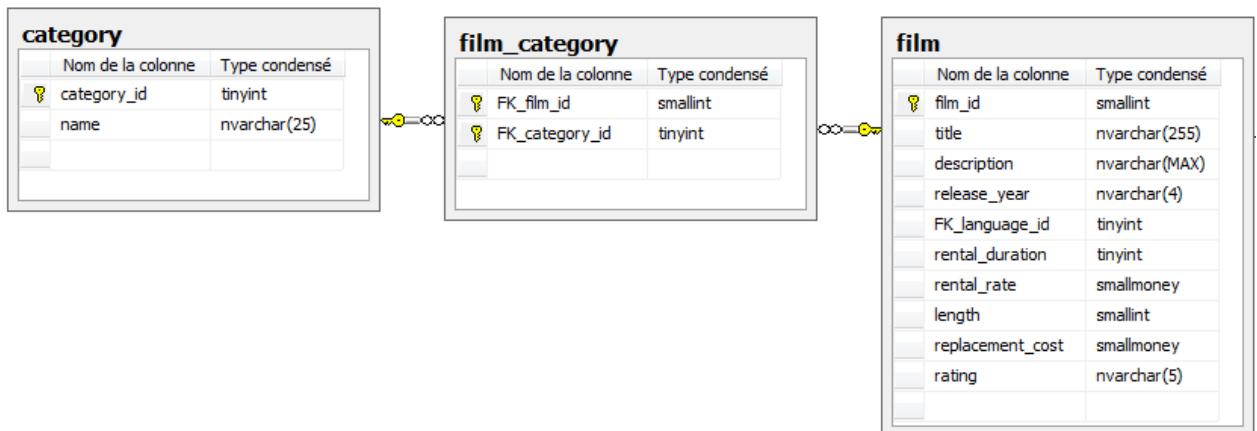


1.4.1.3. Les relations plusieurs à plusieurs ()

- Les relations plusieurs à plusieurs sont représentées par le symbole "*" et "*" 
- Par exemple, dans le cas des entités CATEGORY et FILM, une catégorie peut être associée (ou pas) à un ou plusieurs films et un film peut être associé, ou pas, à une ou plusieurs catégories.

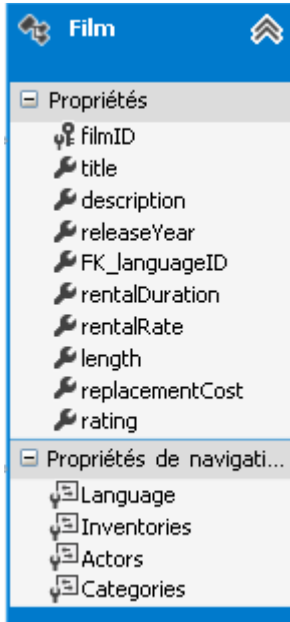


- Vous remarquerez que la table d'intersection **FILM_CATEGORY** n'apparaît pas dans le modèle Entity Framework.



1.4.2. Propriétés de navigation

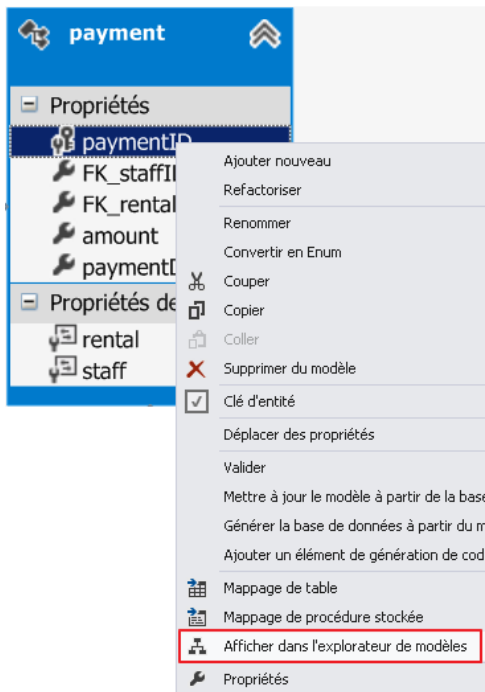
- Vous avez peut-être remarqué la section Propriété de navigation. Cet élément apparaît pour chaque table dans le modèle Entity Framework mais pas dans le modèle de base de données.
- Les propriétés de navigation permettent de consulter les entités reliées.
- Par exemple la table film est reliée aux tables LANGUAGE, INVENTORY, ACTOR et CATEGORY. Il est donc possible de consulter les categories d'un film donné en consultant sa propriété de navigation categories.



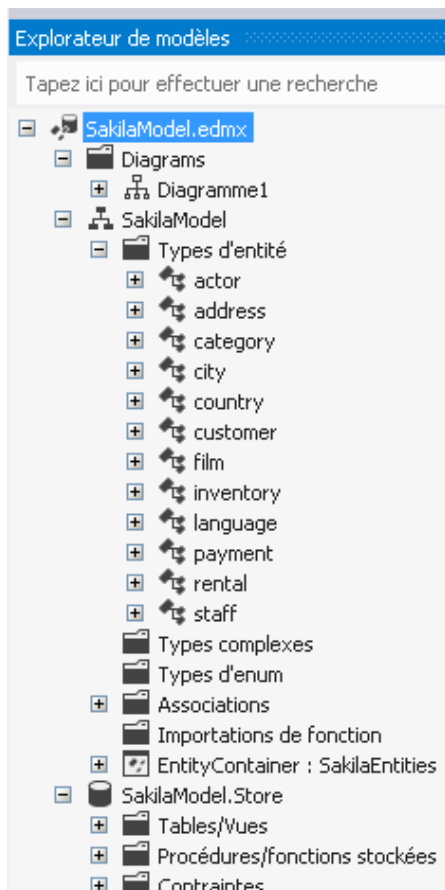
- Par convention, on conjuguera au pluriel les propriétés de navigation s'il peut exister plusieurs occurrences de l'entité reliée.
- Par la même mesure, on conjuguera au singulier les propriétés de navigation s'il ne peut exister qu'une seule occurrence de l'entité reliée

1.4.3. L'explorateur de modèle

- Pour utiliser l'explorateur de modèle, on clique sur l'entité et le menu contextuel permet de sélectionner l'explorateur de modèle.



- L'explorateur de modèle présente un arbre des attributs des entités.






1.4.4. L'onglet propriété

- On peut consulter/modifier les propriétés des champs

Propriétés

SakilaModel.actor.actorID Property




☒ **Général**

Clé d'entité	True
<input checked="" type="checkbox"/> Documentation	
Mode d'accès concurrentiel	None
Nom	actorID
Nullable	False
StoreGeneratedPattern	Identity
Type	Int16
Valeur par défaut	(Aucune)
<input checked="" type="checkbox"/> Génération de code	
Getter	Public
Setter	Public


- Ou encore des entités

Propriétés

SakilaModel.payment EntityType

☒ **Diagramme**

Couleur de remplissage	 0; 122; 204
<input checked="" type="checkbox"/> Général	
<input checked="" type="checkbox"/> Documentation	
Nom	payment
Nom du jeu d'entités	payment
Type de base	(Aucune)
<input checked="" type="checkbox"/> Génération de code	
Abstrait	False
Accès	Public

1.4.5. Mappage de table

- L'option mappage de table, obtenue du même menu contextuel que l'explorateur de modèle, permet de lier les attributs à des champs du support de données.

Détails de mappage - payment

Colonne	Opérateur	Valeur/Propriété
Tables		
Est mappé à payment		
<Ajouter un Condition>		
Mappage de colonnes		
paymentID : smallint	↔	paymentID : Int16
FK_staffID : tinyint	↔	FK_staffID : Byte
FK_rentalID : int	↔	FK_rentalID : Int32
amount : decimal	↔	amount : Decimal
paymentDate : datetime	↔	paymentDate : DateTime

- Il est aussi possible d'utiliser des procédures stockées pour effectuer les opérations de mise-à-jour des entités.

Détails de mappage - payment

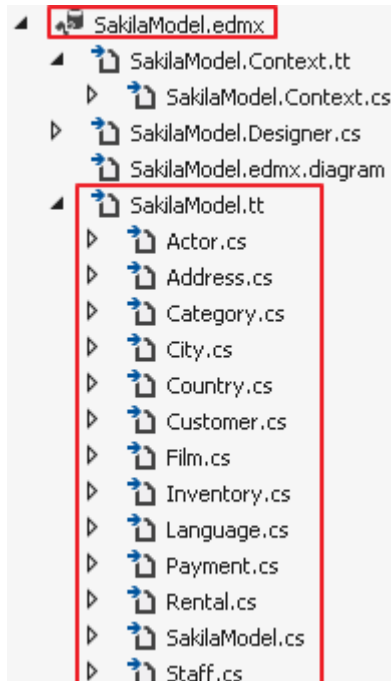
Paramètre/Colonne	Opérateur	Propriété	Utiliser Original Value	Paramètre des lignes affectées
Fonctions				
<Sélectionner un Insert Function>				
<Sélectionner un Update Function>				
<Sélectionner un Delete Function>				

1.5. Le code généré par EF5.

- Il existe plusieurs processus de génération de code qui peuvent être utilisés avec le modèle EF.

1.5.1. Classes POCO

- Celui créé initialement par EF5 utilise le modèle créé dans le designer pour générer des classes POCO (*Plain Old CLR Objects*). Les classes POCO sont des classes simples, sans héritage ou implémentation d'interface particulière.
- Pour chaque entité, une classe représentant une occurrence de l'objet dans la base de données est créée. Le code généré peut être consulté à l'aide de l'explorateur de solution.



Exemple de classes POCO générées

- Par exemple, la classe Country telle que générée par EF

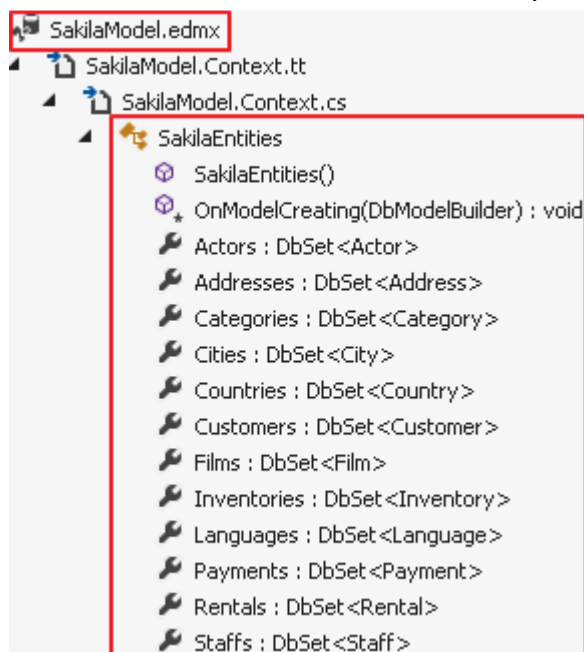
```
public partial class country
{
    public country()
    {
        this.city = new HashSet<city>();
    }

    public short countryID { get; set; }
    public string name { get; set; }

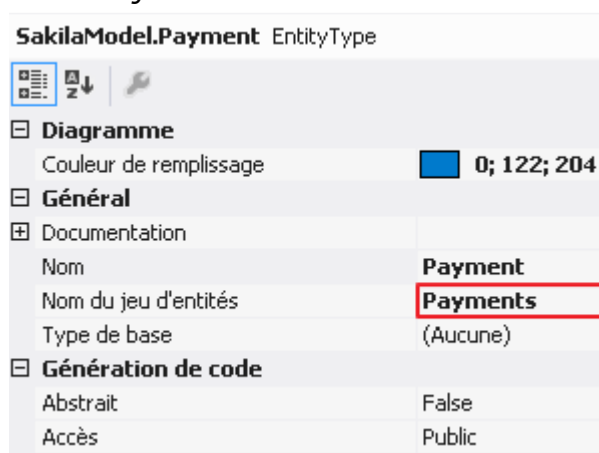
    public virtual ICollection<city> city { get; set; }
}
```

1.5.2. DbContext

- EF5 génère aussi une classe plus complexe, héritant de DbContext, qui représente l'ensemble des entités du modèle (les tables).
- Le DbContext ainsi créé sera utilisé pour interroger le modèle.



- Remarquez que la classe SakilaEntities (qui hérite de DbContext) possède une propriété de type DbSet par entité du modèle.
- Comme ces propriétés représentent plusieurs occurrences des entités du même nom, elles sont généralement conjuguées au pluriel.
- Il est possible de changer le nom de ces propriétés en changeant la propriété *Nom du jeu d'entités* de l'entité concernée.



1.6. Opérations CRUD avec DbContext

- Soit le contrôleur Category

```
public class CategoryController : Controller {
    public ActionResult Index() {
        string contenu = "";
        contenu += "<a href='/Category/Read'>Test READ</a><br/>";
        contenu += "<a href='/Category/Create'>Test CREATE</a><br/>";
        contenu += "<a href='/Category/Delete'>Test Delete</a><br/>";
        contenu += "<a href='/Category/Update'>Test Update</a><br/>";
        return this.Content(contenu);
    }
    public ActionResult Read() {
        SakilaEntities se = new SakilaEntities();
        IEnumerable<Category> lesCategories = se.Categories;
        string contenu = "";
        foreach (Category c in lesCategories) {
            contenu += c.categoryID + " " + c.name + "<br/>";
        }
        return this.Content(contenu);
    }
    public ActionResult Create() {
        SakilaEntities se = new SakilaEntities();
        Category c = new Category();
        c.name = "Nouvelle catégorie";
        se.Categories.Add(c);
        se.SaveChanges();
        return this.Content("La catégorie <strong>" + c.name
            + "</strong> a été créée, son id est <strong>"
            + c.categoryID + "</strong>");
    }
    public ActionResult Delete(int? id) {
        if (!id.HasValue)
            return this.Content("spécifiez /category/delete/id");
        SakilaEntities se = new SakilaEntities();
        Category c = se.Categories.Find(id.Value);
        se.Categories.Remove(c);
        se.SaveChanges();
        return this.Content("la catégorie " + id.Value + " a été supprimée");
    }
    public ActionResult Update(int? id) {
        if (!id.HasValue)
            return this.Content("spécifiez /category/update/id");
        SakilaEntities se = new SakilaEntities();
        Category c = se.Categories.Find(id.Value);
        c.name = "test update";
        se.SaveChanges();
        return this.Content("la catégorie " + id.Value + " a été modifiée");
    }
}
```

1.7. DataAnnotations avec EF

- Étant donné que les classes du modèle EF sont générées automatiquement à partir du modèle, il ne sera pas possible de les modifier en ajoutant des DataAnnotations directement dans le code. D'ailleurs, chaque classe qui a été générée par le modèle est précédée d'un avertissement à cet effet.

```
//-----  
// <auto-generated>  
//   Ce code a été généré à partir d'un modèle.  
//  
//   Des modifications manuelles apportées à ce fichier peuvent  
//   conduire à un comportement inattendu de votre application.  
//   Les modifications manuelles apportées à ce fichier sont remplacées  
//   si le code est régénéré.  
// </auto-generated>  
//-----
```

1.7.1. Classes Partielles

- En C#, il est possible de diviser les classes en plusieurs fichiers à l'aide du mot clé partial.
- Les classes partielles du même nom (et du même espace de nom) seront assemblées avant la compilation.
- Soit le fichier A1.cs, contenant une partie de la classe partielle A de l'espace de nom Module01.Models.

```
namespace Module01.Models {  
    public partial class A {  
        public int propriété01 { get; set; }  
    }  
}
```

- Soit aussi le fichier A2.cs, contenant une partie de la classe partielle A de l'espace de nom Module01.Models.

```
namespace Module01.Models {  
    public partial class A {  
        public String propriété02 { get; set; }  
    }  
}
```

- La classe A finale possède deux propriétés, la propriété1 et la propriété2. La fonctionnalité CodeMap et l'explorateur de solution le montre aisément.



1.7.2. Metadata

- Les classes partielles nous donnent une partie de la solution du problème énoncé au début du module 1.6 mais pas toute la solution puisqu'il n'est pas possible de déclarer deux fois les mêmes propriétés. Il ne sera toujours pas possible d'annoter les propriétés même en déclarant une classe partielle.
- Il existe toutefois un DataAnnotation nommé MetadataType qui désigne une classe cible où l'engin de vues Razor pourra consulter les DataAnnotations. Cette annotation spéciale doit être déclarée sur la classe parent.

```
[MetadataType(typeof(CustomerMetaData))]  
public partial class Customer {  
  
    private class CustomerMetaData {  
        public short customerID { get; set; }  
        [MaxLength(20)]  
        public string firstname { get; set; }  
        public string lastname { get; set; }  
  
        [DataType(DataType.EmailAddress)]  
        public string email { get; set; }  
        public short FK_addressID { get; set; }  
        public bool active { get; set; }  
        public System.DateTime createDate { get; set; }  
  
        public virtual Address Address { get; set; }  
        public virtual ICollection<Rental> Rentals { get; set; }  
    }  
}
```