ADO.Net Entity Framework

Nouhaila Bensalah

Chercheuse en IA/NLP

nouhaila.bensalah@etu.fstm.ac.ma

Outline I

Outline I

ORM: Mapping Objet-Relationnel

ORM

- Programme informatique jouant le rôle du traducteur entre le modèle relationnel et le modèle objet.
- Permet d'interroger et manipuler les données à partir d'une base de données à l'aide d'un paradigme orienté objet.
- Deux composants dans les ORM :
  - Entités : Instanciation d'une classe (étudiant, enseignant, cours, projet, etc).
  - Gestionnaire d'entités: à utiliser pour persister les entités dans la base de données.

Entity Framework

- Entity Framework est un framework ORM open source pour les applications .NET prises en charge par Microsoft permettant de créer une couche d'accès aux données liées à une base de données relationnelle.
- permettant aux développeurs de manipuler des données à l'aide d'objets de classes C# sans se concentrer sur les tables et colonnes d'une base de données (où ces données sont stockées).
- permettant aux développeurs de créer et maintenir des applications orientées données avec moins de code, par rapport aux applications traditionnelles, grâce à LinQ To Entities.

Outline I

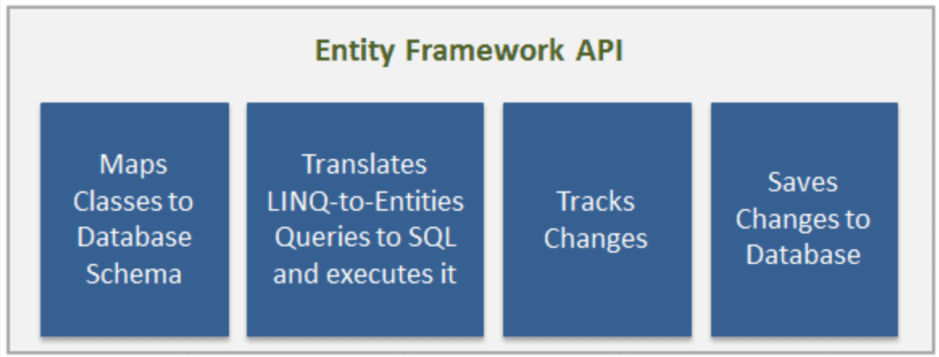
Le fonctionnement de Entity Framework

Dans le cas d'Entity Framework

- Entité (Une classe qui correspond à une table de base de données) = POCO (Plain Old CLR Object) + [classe de configuration ou décorateurs]
- Le gestionnaire d'entités : Linq to Entities

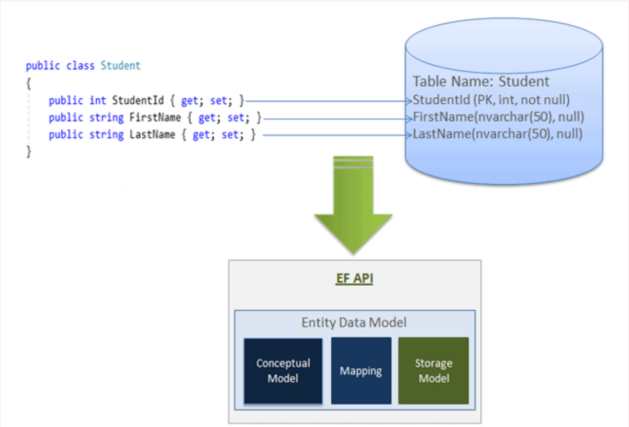
Fonctionnement de Entity Framework

L'API Entity Framework a la possibilité de:



EDM: Entity Data Model

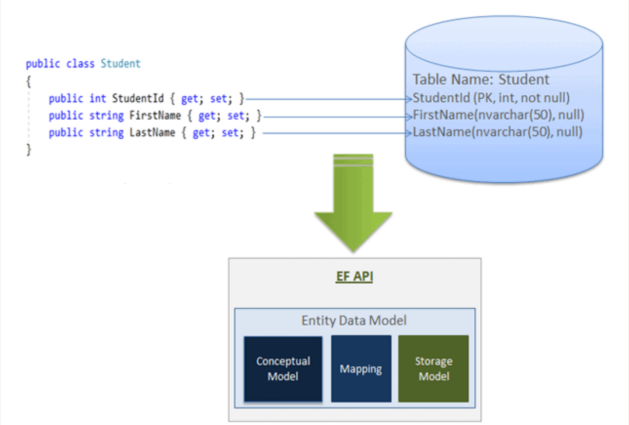
EDM est une représentation en mémoire de l'ensemble des métadonnées.



- EDM consiste en trois parties: Conceptuelle, Mapping et Stockage.
  - Conceptuelle : contient les classes du modèle et ses relations.
  - Stockage : contient le modèle physique de la base : tables, vues, procédures stockées, les relations et les clés.
  - Mapping : définit les mécanismes de passage du modèle conceptuel au stockage.

Maps Classes to Database scheme

EDM est une représentation en mémoire de l'ensemble des métadonnées.



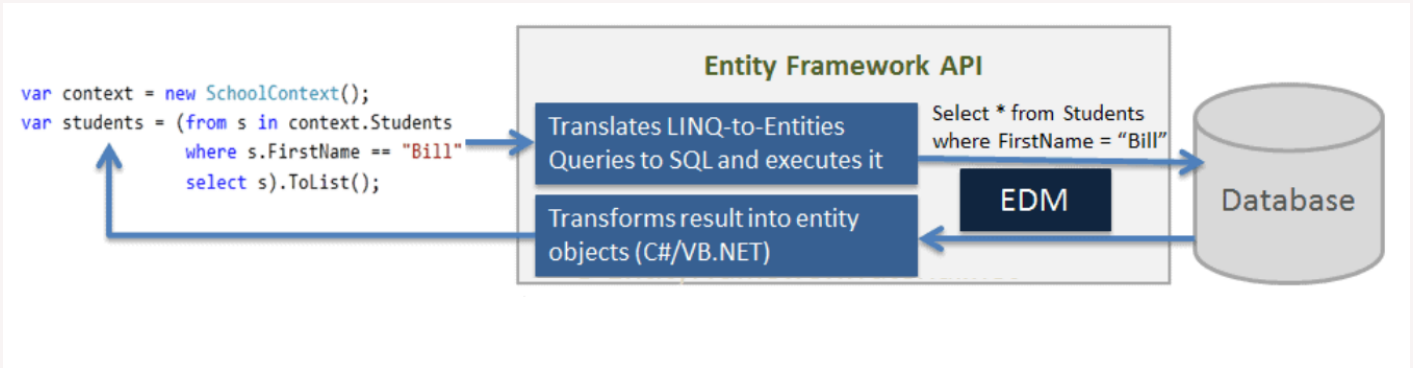
A l'aide de EDM, EF peut:

- Effectuer des opérations **CRUD** (Create, Read, Update and Delete).
- Créer des requêtes **SQL** à partir de requêtes **LINQ**, créer des commandes **INSERT**, **UPDATE** et **DELETE** et transformer le résultat de la base de données en objets d'entité.

Translates LINQ-to-Entities Queries to SQL

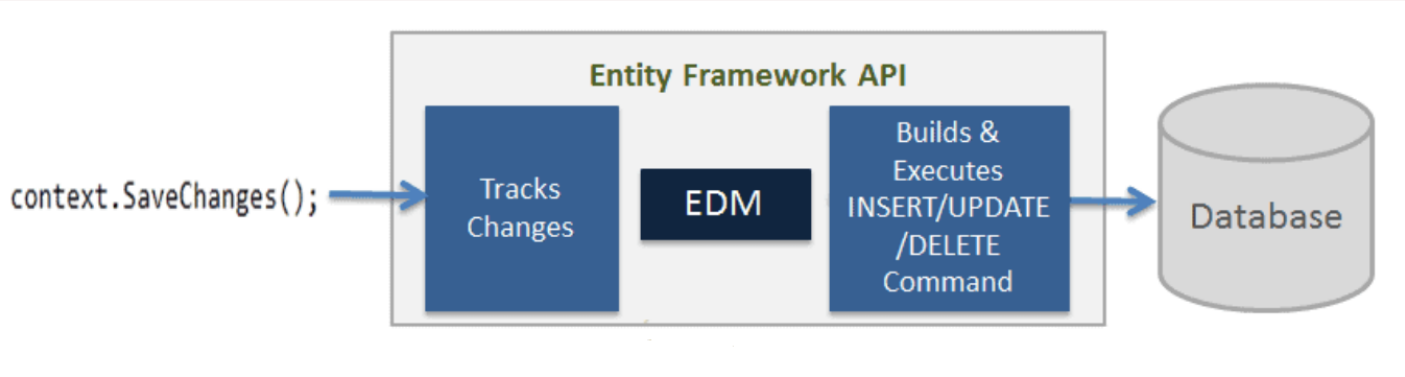
A l'aide d'EDM, l'API EF:

- Traduit les requêtes LINQ-to-Entities en requêtes SQL pour les bases de données relationnelles.
  - LinQ to Entity : Un langage de requête pour un modèle orienté objet. Il retourne les entités définies dans le modèle conceptuel.
- Reconvertit les résultats (requêtes SQL) en objets d'entité.



Tracks and saves changes to database

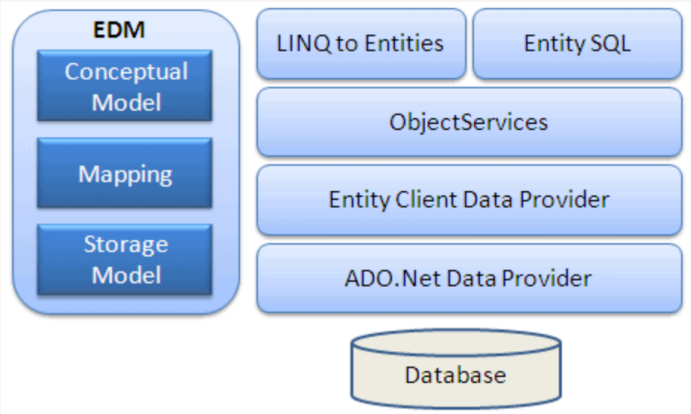
- Lorsque la méthode **SaveChanges()** est appelée, l'API EF déduit les commandes **INSERT**, **UPDATE** et **DELETE** en fonction de l'état des entités.
- Le **ChangeTrack** suit les changements survenus sur les entités.



Outline I

L'architecture de Entity Framework

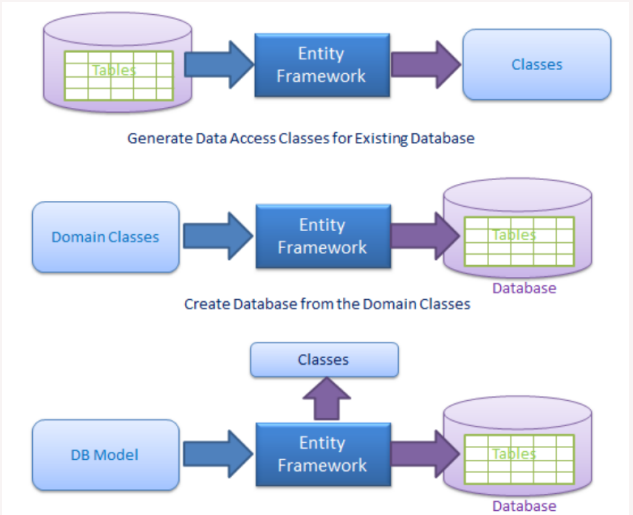
- EDM and LinQ to Entities
- Object Service : est le point d'entrée pour l'accès aux données d'une base de données. Représente le processus de conversion des données retournées par entity client data provider à une entité de structure objet.
- Entity Client Data Provider : la tâche principale est de transformer une expression de LinQ to Entity à une requête SQL.
- ADO.Net Data Provider : communiquer avec la base en utilisant le standard ADO.Net.



Outline I

Troix approches

1. Database First : On crée la base de données (où on a une base de données qui existe déjà) et Entity Framework génère nos entités à partir de cette base de données.
2. Code First : On crée les entités puis et Entity Framework génère la base de données.
3. Model First : On crée notre modèle (de classe) et Entity Framework génère la base de données et les entités correspondantes.



Outline I

Database First approach

Soit la base de données (BD_ACHAT) définie par le modèle relationnel suivant : **CLIENT** (numcli, nomcli, ville, categorie, compte)

1. Chaque ligne décrit un client ; les colonnes décrivent successivement le numéro du client (numcli), son nom (nomcli), sa ville (ville), sa catégorie (categorie) et l'état de son compte (compte). L'identifiant primaire est constitué de numcli .
2. Les enregistrements:

NOMCLI	VILLE	CATEGORIE	COMPTE
AMINE	RABAT	A	11250.00
DRISS	CASABLANCA	A	12300
IMANE	MARRAKECH	D	20
YASSINE	TANGER	C	100
ZINEB	CASABLANCA	B	1000
MAROUANE	KENITRA	D	15
HOUDA	RABAT	A	15000
ALI	TANGER	NULL	0
IMANE	RABAT	B	1500
DRISS	CASABLANCA	NULL	0

Table: Table CLIENT

Database First approach

- Installer MS SQL (Microsoft Structured Query Language) et SSMS (SQL Server Management Studio). See This tutorial.
- Créer une base de données (BD_ACHAT) sous SSMS
  - create database BD_ACHAT; use BD_ACHAT;
  - create table CLIENT
    (NUMCLI int not null IDENTITY(1, 1),
    NOMCLI varchar(25) not null,
    VILLE varchar(25) not null,
    CATEGORIE char(1),
    COMPTE decimal(9,2),
    CONSTRAINT pk_cl_ncli primary key (NUMCLI));
  - insert into CLIENT (NOMCLI, VILLE , CATEGORIE, COMPTE) values
    ('AMINE' ,'RABAT' ,'A',11250.00),
    ('DRISS' ,'CASABLANCA' ,'A',12300),
    ('IMANE' ,'MARRAKECH' ,'D',20),
    ('YASSINE' ,'TANGER' ,'C',100),
    ('ZINEB' ,'CASABLANCA' ,'B',1000);

Database First approach

Étapes à suivre

- Créer un nouveau projet sous Visual Studio **Fichier** → **Nouveau** → **Projet**.
- Cliquer sur **Installé** et choisir *C#*.
- Étendre la rubrique **Web** et sélectionner **Application web ASP.NET (.NET Framework)**.
- Remplir le champs **Nom** par: **App_ACHAT**.
- Sélectionner un modèle **MVC**.
- Valider et attendre la fin de création du projet.

Database First approach

EF designer à partir de la base de données

- Faire un clic droit sur **Models** et aller dans **Ajouter** $\rightarrow$ **Nouvel élément** .
- Sélectionner **ADO.NET Entity Data Model** et cliquer sur **Ajouter**.
- Saisir un nom (**Model1**) et cliquer sur **Ajouter**.
- Les quatres options disponibles:
  - EF Designer from database pour the database-first approach
  - Empty EF Designer model pour the model-first approach
  - Empty Code First model et Code First from database pour Code-First approach.
- Sélectionner **EF designer à partir de la base de données** pour le contenu du modèle et cliquer sur **Ajouter**.

Database First approach

Connecter le serveur sql avec visual studio

- Faire un clic sur **Nouvelle connexion** et saisir le nom du serveur (Faire un clic sur Object explorer dans SSMS).
- Sélectionner le nom de la base de données **BD_ACHAT**.

Choix des objets(tables)

- Sélectionner les objets(les tables) que vous voulez inclure dans votre modèle (**CLIENT**).
- Cocher **Mettre au pluriel ou au singulier les noms des objets générés** et cliquer sur **Terminer**.

Database First approach

Model1.Context.cs

```
public partial class BD_ACHATEntities : DbContext
    {
      public BD_ACHATEntities(): base("name=BD_ACHATEntities")
      {
      }
      protected override void OnModelCreating(DbModelBuilder modelBuilder)
      {
        throw new UnintentionalCodeFirstException();
      }
      public virtual DbSet<CLIENT> CLIENTs { get; set; }
    }
```

Explication

- Une classe héritant de la classe DbContext est appelée classe de contexte (context class) dans le framework d'entité.
- La classe BD_ACHATEntities est une classe C# héritant de la classe System.Data.Entity.DbContext.

DbContext

DbContext est la classe principale responsable de l'interaction avec la base de données. Il est responsable de plusieurs activités :

- Querying: : convertit les requêtes LINQ-to-Entities en requêtes SQL et les envoie à la base de données.
- Change Tracking : assure le suivi des modifications apportées aux entités après une requête à partir de la base de données.
- Persisting Data : effectue les opérations d'insertion, de mise à jour et de suppression dans la base de données, en fonction des états de l'entité.
- Object Materialization : convertit les données brutes de la base de données en objets d'entité.

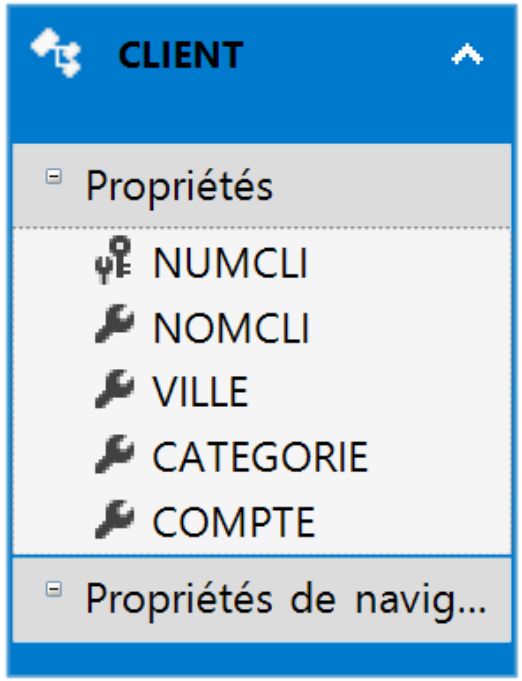
DbSet

- La classe de contexte (BD_ACHATEntities) inclut l'ensemble d'entités de type DbSet<TEntity>.
- DbSet est considérée comme une propriété de la classe de contexte et liée à une table de la base de données.
- DBSet contient un ensemble de méthodes permettant d'effectuer les opérations CRUD nécessaires.

- DbContext correspond à votre base de données (ou à une collection de tables et de vues dans votre base de données).
- DbSet correspond à une table ou une vue dans votre base de données.

Database First approach

Model1.edmx



Database First approach

CLIENT.cs

```
public partial class CLIENT
{
      public int NUMCLI { get; set; }
      public string NOMCLI { get; set; }
      public string VILLE { get; set; }
      public string CATEGORIE { get; set; }
      public Nullable<decimal> COMPTE { get; set; }
}
```

Explication

- Classe client:
  - NUMCLI
  - NOMCLI
  - VILLE
  - CATEGORIE
  - COMPTE
- Les attributs de la classe CLIENT sont les mêmes que ceux de la table CLIENT.

Création du contrôleur

- Faire un clic droit sur **Controllers** et aller dans **Ajouter** $\rightarrow$ **Controleur**.
- Sélectionner **MVC 5 Controller avec vues, utilisant Entity Framework** et cliquer sur **Ajouter**.
- Sélectionner **CLIENT (App_Achat.Models)** pour *la classe de modèle* et **BD_ACHATEntities (App_Achat.Models)** pour *classe de contexte de données*
- Saisir un nom pour le contrôleur (**CLIENTsController**) et cliquer sur **Ajouter**.

CLIENTsController: les opérations CRUD

Instanciation

```
BD_ACHATEntities db = new BD_ACHATEntities()
// création d'une instance de la classe BD_ACHATEntities
```

Read

```
public ActionResult Index()
    {
      return View(db.CLIENTs.ToList());
    }
//Récupération de la liste des clients ajoutés
```

CLIENTsController: les opérations CRUD

Create

```
// POST
public ActionResult Create([Bind(Include = "NOMCLI,VILLE,CATEGORIE,COMPTE")] CLIENT cLIENT)
    {
      if (ModelState.IsValid)
      {
          db.CLIENTs.Add(cLIENT);
          db.SaveChanges();
          return RedirectToAction("Index");
      }
      return View(cLIENT);
    }
```

// Bind(Include = "NOMCLI,VILLE,CATEGORIE,COMPTE"): les variables dont leurs valeurs vont être récupérées à partir du formulaire
// if (ModelState.IsValid): validation côté serveur
// db.CLIENTs.Add(cLIENT): ajout d'une nouvelle entité
// db.SaveChanges(): Execute la commande INSERT pour l'entité (cLIENT) avec l'état Added (Added state)
// RedirectToAction("Index"): passage de la méthode create à la méthode Index qui récupère la liste des clients ajoutés

CLIENTsController: les opérations CRUD

Update

```
// GET
public ActionResult Edit(int? id)
    {
      if (id == null)
      {
         return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
      }
      CLIENT cLIENT = db.CLIENTs.Find(id);
      if (cLIENT == null)
      {
         return HttpNotFound();
      }
      return View(cLIENT);
    }
```

// CLIENT cLIENT = db.CLIENTs.Find(id): Récupération de l'entité à partir la variable id dont la valeur est récupérée de l'url

CLIENTsController: les opérations CRUD

Update

```
// POST
public ActionResult Edit([Bind(Include = "NOMCLI,VILLE,CATEGORIE,COMPTE")] CLIENT
cLIENT)
    {
      if (ModelState.IsValid)
      {
         db.Entry(cLIENT).State = EntityState.Modified;
         db.SaveChanges();
         return RedirectToAction("Index");
      }
      return View(cLIENT);
    }
```

// db.Entry(cLIENT).State = EntityState.Modified: modification de l'entité cLIENT
// db.SaveChanges(); Execute la commande UPDATE pour l'entité (cLIENT) avec l'état Modified (Modified state)

CLIENTsController: les opérations CRUD

Delete

```
public ActionResult DeleteConfirmed(int id)
    {
      CLIENT cLIENT = db.CLIENTs.Find(id);
      db.CLIENTs.Remove(cLIENT);
      db.SaveChanges();
      return RedirectToAction("Index");
    }
```

// CLIENT cLIENT = db.CLIENTs.Find(id): Récupération de l'entité à partir la variable id dont la valeur est récupérée de l'url
// db.CLIENTs.Remove(cLIENT): suppression de l'entité cLIENT
//db.SaveChanges(): Execute la commande DELETE pour l'entité (cLIENT) avec l'état DELETED (DELETED state)

Outline I

Relations entre les entités dans Entity Framework

- Dans Entity Framework, une entité peut être associée à d'autres entités par le biais d'une association ou d'une relation.
- Chaque relation contient deux extrémités qui décrivent le type d'entité et la multiplicité du type (un, zéro-ou-un ou plusieurs) pour les deux entités de cette relation.

Les propriétés de navigation permettent de parcourir une association entre deux types d'entités. Chaque objet peut avoir une propriété de navigation pour chaque relation à laquelle il participe. Les propriétés de navigation vous permettent de parcourir et de gérer les relations dans les deux directions en retournant

- Un objet de référence (si la multiplicité est un ou zéro-ou-un)
- Une collection (si la multiplicité est plusieurs)

Relations entre les entités dans Entity Framework

Reference Navigation Property vs Collection Navigation Property

- Si une entité inclut une propriété d'un autre type d'entité, elle est appelée propriété de navigation de référence ou **Reference Navigation Property**. Il pointe vers une seule entité et représente la multiplicité d'un (1) dans les relations d'entité.
- Si une entité comprend une propriété de collection générique d'un type d'entité, elle est appelée propriété de navigation de collection **Collection Navigation Property**. Il représente la multiplicité de plusieurs (*).

Database First approach

Soit la base de données (BD_ACHAT) définie par le modèle relationnel suivant :
CLIENT (numcli, nomcli, ville, categorie, compte)
CLIENTAdress (numcli, address1, state, numcli)
COMMANDE (numcom, numcli, datecom)
PRODUIT (numpro, nompro, prix, qstock))
DETAIL (numdetail, numcom, numpro, qcom)

1. Table **CLIENT**: chaque ligne décrit un client; les colonnes décrivent successivement le numéro du client (numcli), son nom (nomcli), sa ville (ville), sa catégorie (categorie) et l'état de son compte (compte). L'identifiant primaire est constitué de numcli .
2. Table **CLIENTAddress**: chaque ligne décrit une adresse d'un client; les colonnes décrivent successivement le numéro du client (numcli), l'adresse du client (address1) ainsi que l'état du client (state). numcli est l'identifiant primaire de la table. numcli est une clé étrangère vers la table CLIENT.
3. Table **COMMANDE**: chaque ligne décrit une commande passée par un client; les colonnes décrivent successivement le numéro de la commande (numcom), le numéro du client qui a passé la commande (numcli) et la date de la commande (datecom). numcom est l'identifiant primaire de la table. numcli est une clé étrangère vers la table CLIENT.
4. Table **PRODUIT**: chaque ligne décrit un produit ; les colonnes décrivent le numéro du produit (numpro), son nom (nompro), son prix unitaire (prix) et la quantité restante en stock (qstock). numpro est l'identifiant primaire.
5. Table **DETAIL**: chaque ligne représente un détail d'une commande ; les colonnes décrivent le numéro du détail d'une commande (numdetail), le numéro de la commande à laquelle le détail appartient (numcom), le numéro du produit commandé (numpro) et la quantité commandée (qcom). L'identifiant primaire est (numdetail). (numcom) et (numpro) sont en outre chacune une clé étrangère respectivement vers les tables **COMMANDE** et **PRODUIT**.

Les enregistrements:

TYPEPRO	NOMPRO	PRIX	QSTOCK
Audio	Casques-micros	750	35
Informatique	Ordinateurs fixes	5400	960
Mobilite	Tablettes	105	830
Informatique	PC portables	9500	134
Audio	Casques-Hi-Fi	1500	82
Informatique	Disques-durs	985	129
Mobilite	Smartphones	220	540
Loisirs	Jeux PS4	3500	10

Table: Table PRODUIT

NUMCOM	NUMPRO	QCOM
1	1	33
2	2	25
2	3	70
3	2	40
4	1	133
4	4	20
5	2	10
5	5	60
5	1	26
6	4	157
7	2	75
7	6	920
7	1	18
7	4	2

Table: Table DETAIL

NUMCLI	DATECOM
3	2015-01-01
9	2015-05-10
3	2015-07-19
1	2015-09-17
3	2015-10-27
6	2015-10-09
2	2015-01-01
8	2015-05-10
5	2015-07-19
2	2015-09-17
6	2015-10-27
10	2015-10-09

Table: Table COMMANDE

NUMCLI	ADRESSE	ETAT
3	RUE A AV B	Oriental
9	RUE B AV C	Souss-Massa
4	RUE H AV E	Oriental
1	RUE K AV V	Fès-Meknès
6	RUE M AV L	Drâa-Tafilalet
8	RUE B AV V	Casablanca-Settat
5	RUE N AV D	Drâa-Tafilalet
2	RUE A AV F	Souss-Massa
6	RUE A AV C	Fès-Meknès
10	RUE N AV A	Souss-Massa

Table: Table CLIENTAddress

Database First approach

Création de la table CLIENTAddress

- use BD_ACHAT;
- create table CLIENTAddress
  (NUMCLI int not null,
  ADDRESS1 varchar (25),
  STATE varchar(25),
  CONSTRAINT pk_pr primary key (NUMCLI));
- Alter table CLIENTAddress
  Add constraint fk_cm_claddr foreign key (NUMCLI) references
  CLIENT (NUMCLI);
- insert into CLIENTAddress (NUMCLI, ADDRESS1 , STATE) values
  (3 ,'RUE A AV B' ,'Oriental'),
  (9 ,'RUE B AV C' ,'Souss-Massa'),
  (4 ,'RUE H AV E' ,'Oriental');

Création de la table PRODUIT

- create table PRODUIT
  (NUMPRO int not null IDENTITY(1, 1),
  TYPEPRO varchar (25),
  NOMPRO varchar(25),
  PRIX decimal(5) ,
  QSTOCK decimal(6,0) check(qstock between 0 and 1000),
  CONSTRAINT pk_pr_npro primary key (NUMPRO),
  CONSTRAINT ck_pr_prix check(prix>=0));
- insert into PRODUIT (TYPEPRO,NOMPRO,PRIX,QSTOCK) values
  ('Audio', 'Casques-micros', 750, 35),
  ('Informatique','Ordinateurs fixes', 5400,960),
  ('Mobilite','Tablettes',105, 830),
  ('Informatique','PC portables', 9500, 134);

Création de la table COMMANDE

- create table COMMANDE
  (NUMCOM int not null IDENTITY(1, 1),
  NUMCLI int not null,
  DATECOM date,
  CONSTRAINT pk_cm_ncom primary key (NUMCOM));
- Alter table commande
  Add constraint fk_cm_ncli foreign key (NUMCLI) references CLIENT (NUMCLI);
- insert into COMMANDE (NUMCLI,DATECOM) values
  (3,'2015-01-01'),
  (9,'2015-05-10'),
  (3,'2015-07-19'),
  (1,'2015-09-17'),
  (3,'2015-10-27'),
  (6,'2015-10-09');

Création de la table DETAIL

- create table DETAIL
  (NUMDETAIL int not null IDENTITY(1, 1),
  NUMCOM int not null,
  NUMPRO int not null,
  QCOM decimal(4),
  CONSTRAINT pk_dt_cmproo primary key (NUMDETAIL));
- Alter table DETAIL
  Add constraint fk_dt_ncomm foreign key (NUMCOM) references COMMANDE (NUMCOM);
- Alter table DETAIL
  Add constraint fk_dt_nproo foreign key (NUMPRO) references PRODUIT(NUMPRO);
- insert into DETAIL (NUMCOM,NUMPRO,QCOM) values
  (1,1,33),
  (2,2,25),
  (2,3,70);

Database First approach

Étapes à suivre

- Créer un nouveau projet sous Visual Studio **Fichier** → **Nouveau** → **Projet**.
- Cliquer sur **Installé** et choisir *C#*.
- Étendre la rubrique **Web** et sélectionner **Application web ASP.NET (.NET Framework)**.
- Remplir le champs **Nom** par: **App_ACHAT**.
- Sélectionner un modèle **MVC**.
- Valider et attendre la fin de création du projet.

Database First approach

EF designer à partir de la base de données

- Faire un clic droit sur **Models** et aller dans **Ajouter** → **Nouvel élément** .
- Sélectionner **ADO.NET Entity Data Model** et cliquer sur **Ajouter**.
- Saisir un nom (**Model1**) et cliquer sur **Ajouter**.
- Sélectionner **EF designer à partir de la base de données** pour le contenu du modèle et cliquer sur **Ajouter**.

Database First approach

Connecter le serveur sql avec visual studio et choix des objets(tables)

- Faire un clic sur **Nouvelle connexion** et saisir le nom du serveur (Faire un clic sur Object explorer dans SSMS).
- Sélectionner le nom de la base de données **BD_ACHAT**.
- Sélectionner les objets(les tables) que vous voulez inclure dans votre modèle **CLIENT, CLIENTAddress, COMMANDE, PRODUIT, DETAIL**.
- Cocher **Mettre au pluriel ou au singulier les noms des objets générés** et cliquer sur **Terminer**.

Database First approach

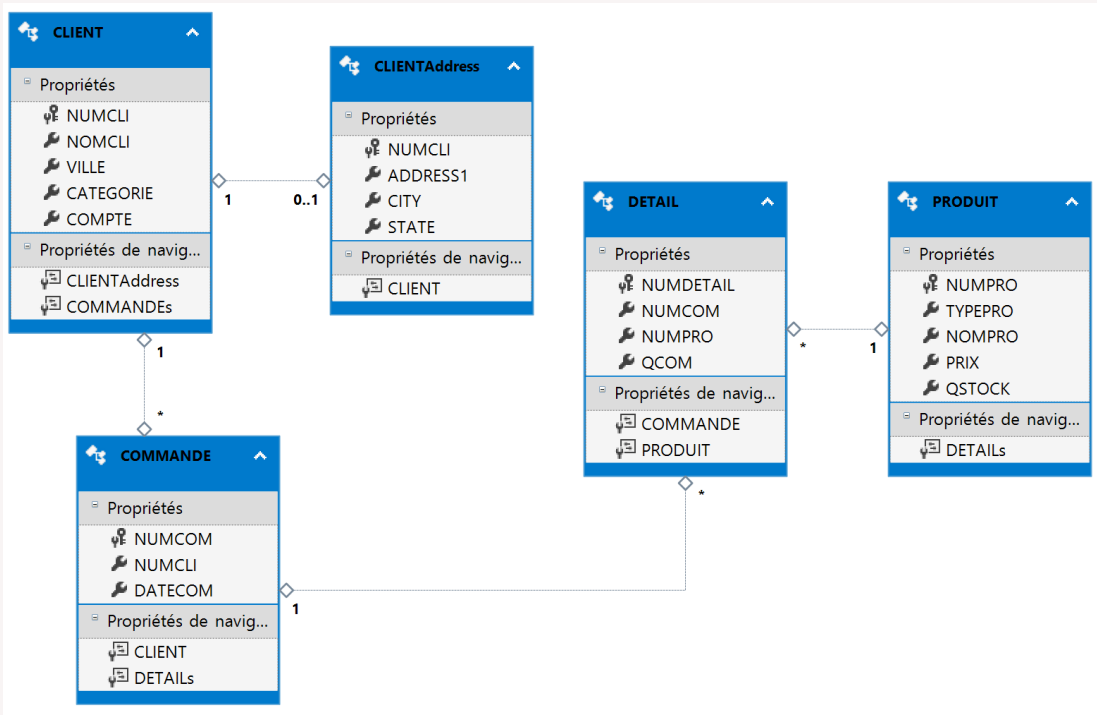
Model1.Context.cs

```
public partial class BD_ACHATEntities : DbContext
    {
        public BD_ACHATEntities(): base("name=BD_ACHATEntities")
        {
        }
        public virtual DbSet<CLIENT> CLIENTs { get; set; }
        public virtual DbSet<CLIENTAddress> CLIENTAddresses { get; set; }
        public virtual DbSet<COMMANDE> COMMANDEs { get; set; }
        public virtual DbSet<DETAIL> DETAILs { get; set; }
        public virtual DbSet<PRODUIT> PRODUITs { get; set; }
    }
```

Explication

- La classe BD_ACHATEntities est une classe C# héritant de la DbContext.
- La classe de contexte (BD_ACHATEntities) inclut les entités de type DbSet: CLIENTs, CLIENTAddresses, COMMANDEs, DETAILs et PRODUITs

Model1.edmx


CLIENT
Propriétés
NUMCLI
NOMCLI
VILLE
CATEGORIE
COMPTE
Propriétés de navig...
CLIENTAddress
COMMANDEs
CLIENTAddress
Propriétés
NUMCLI
ADDRESS1
CITY
STATE
Propriétés de navig...
CLIENT
DETAIL
Propriétés
NUMDETAIL
NUMCOM
NUMPRO
QCOM
Propriétés de navig...
COMMANDE
PRODUIT
PRODUIT
Propriétés
NUMPRO
TYPEPRO
NOMPRO
PRIX
QSTOCK
Propriétés de navig...
DETAILs
COMMANDE
Propriétés
NUMCOM
NUMCLI
DATECOM
Propriétés de navig...
CLIENT
DETAILs
1
0..1
1
*
*
1
*
1

Création du contrôleur CLIENTAddresses

- Faire un clic droit sur **Controllers** et aller dans **Ajouter** → **Controleur**.
- Sélectionner **MVC 5 Controller avec vues, utilisant Entity Framework** et cliquer sur **Ajouter**.
- Sélectionner **CLIENTAddress (App_Achat.Models)** pour *la classe de modèle* et **BD_ACHATEntities (App_Achat.Models)** pour *classe de contexte de données*
- Saisir un nom pour le contrôleur (**CLIENTAddressesController**) et cliquer sur **Ajouter**.

CLIENTAddressesController: les opérations CRUD

Instanciation

```
BD_ACHATEntities db = new BD_ACHATEntities()
// création d'une instance de la classe BD_ACHATEntities
```

Read

```
public ActionResult Index()
    {
      var cLIENTAddresses = db.CLIENTAddresses
        .Include(c => c.CLIENT);
      return View(cLIENTAddresses.ToList());
    }
//Récupérer la liste des adresses ajoutées
//Récupérer la liste des clients (noms) ainsi que leurs adresses.
// Include: Il s'agit de "Eager méthode", utilisée pour récupérer un
objet et les autres objets en relation dans une seule requête.
```

CLIENTAddressesController: les opérations CRUD

Create

```
// POST
public ActionResult Create([Bind(Include = "NUMCLI,ADDRESS1,STATE1")] CLIENTAddress cLIENTAddress)
    {
      if (ModelState.IsValid)
      {
          db.CLIENTAddresses.Add(cLIENTAddress);
          db.SaveChanges();
          return RedirectToAction("Index");
      }
      return View(cLIENT);
    }
```

// ViewBag.NUMCLI = new SelectList(db.CLIENTs, "NUMCLI", "NOMCLI", cLIENTAddress.NUMCLI): afin de récupérer la liste des numéros **NUMCLI** des clients créés. Les noms de ces clients sont affichées dans la la liste déroulante (DropDownList) pour que l'utilisateur attribut une adresse à un client bien précis. (source, value, text, selectedValue)

Update

```
// GET
public ActionResult Edit(int? id)
    {
      if (id == null)
      {
         return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
      }
      CLIENTAddress cLIENTAddress = db.CLIENTAddresses.Find(id);
      if (cLIENTAddress == null)
      {
         return HttpNotFound();
      }
      return View(cLIENTAddress);
    }
```

//CLIENTAddress cLIENTAddress = db.CLIENTAddresses.Find(id): Récupération de l'entité à partir la variable id dont la valeur est récupérée de l'url

Update

```
// POST
public ActionResult Edit([Bind(Include = "NUMCLI,ADDRESS1,STATE1")] CLIENTAddress
cLIENTAddress)
    {
      if (ModelState.IsValid)
      {
         db.Entry(cLIENTAddress).State = EntityState.Modified;
         db.SaveChanges();
         return RedirectToAction("Index");
      }
      return View(cLIENTAddress);
    }
```

// db.Entry(cLIENTAddress).State = EntityState.Modified: modification de l'entité cLIENTAddress
// db.SaveChanges(); Execute la commande UPDATE pour l'entité (cLIENTAddress) avec l'état Modified (Modified state)

CLIENTAddressesController: les opérations CRUD

Delete

```
public ActionResult DeleteConfirmed(int id)
    {
      CLIENTAddress cLIENTAddress = db.CLIENTAddresses.Find(id);
      db.CLIENTAddresses.Remove(cLIENTAddress);
      db.SaveChanges();
      return RedirectToAction("Index");
    }
```

// CLIENT cLIENT = db.CLIENTs.Find(id): Récupération de l'entité à partir la variable id dont la valeur est récupérée de l'url
// db.CLIENTAddresses.Remove(cLIENTAddress): suppression de l'entité cLIENTAddress
//db.SaveChanges(): Execute la commande DELETE pour l'entité (cLIENTAddress) avec l'état DELETED (DELETED state)

Outline I

Les requêtes simples

Dans la fonction Index (PRODUITsController)

Pour afficher la liste des produits

```
var pRODUITs = db.PRODUITs;
return View(pPRODUITs.ToList());
```

Dans la fonction Index (PRODUITsVue)

Pour afficher la liste des produits dont le prix est supérieur à 400 en utilisant les expressions Lambda

```
@{
   var pQuery = Model
      .Where(elt => elt.PRIX > 500);
}
@foreach (var item in pQuery)
{
     ...
}
```

Les requêtes simples

Pour afficher la liste des produits dont le prix est supérieur à 500 en sélectionnant seulement quelques attributs (et les renommer) avec les expressions Lambda

Dans la fonction Index (PRODUITsVue)

```
   <tr>
     <th>
       @Html.DisplayNameFor(model => model.TYPEPRO)
     </ th>
     <th>
       @Html.DisplayNameFor(model => model.PRIX)
     </ th>
     <th></ th>
   </ tr>
@{
     var pQuery = Model.Where(elt => elt.PRIX > 500).Select(elt => new
     { NUMPRO= elt.NUMPRO,
       TYPE = elt.TYPEPRO,
       PRICE = elt.PRIX       }
     );
@foreach (var item in pQuery)
{
     <td>
        @Html.DisplayFor(modelItem => item.TYPE)
     </ td>
     <td>
        @Html.DisplayFor(modelItem => item.PRICE)
     </ td>
}
```

orderby

orderby : permet de trier dans l'ordre croissant ou décroissant.

Pour trier la liste des produits selon le prix du produit

Dans la fonction Index (PRODUITsVue)

```
@{
    var pQuery = Model
      .Where(elt => elt.PRIX > 500)
      .OrderByDescending(elt => elt.PRIX);
}
@foreach (var item in pQuery)
{
    ...
}
```

orderby

Pour trier la liste des produits selon le type du produit

Dans la fonction Index (PRODUITsVue)

```
@{
    var pQuery = Model
      .Where(elt => elt.PRIX > 500)
      .OrderBy(elt => elt.TYPEPRO);
}
@foreach (var item in pQuery)
{
    ...
}
```

orderby

Pour trier trier la liste des produits selon le type et ensuite le prix

Orderby selon plusieurs colonnes avec lambda

```
@{
    var pQuery = Model
      .Where(elt => elt.PRIX > 500)
      .OrderBy(elt => elt.TYPEPRO)
      .ThenByDescending(elt => elt.PRIX);
}
@foreach (var item in pQuery)
{
    ...
}
```

group ... by ... into

group ... by ... into : permet de retourner une séquence composée de clé (correspondant au critère du groupement) et valeur (correspondant à l'objet)

Pour calculer la valeur totale des produits appartenant à un même type

group ... by ... into avec lambda

```
@{
    var pQuery = Model
    .GroupBy(u => u.TYPEPRO, (TYPEPRO, PRIX) =>
    new { TYPE = TYPEPRO, TOTALE = PRIX.Sum(u => u.PRIX) });
}
@foreach (var item in pQuery)
{
    <td>
        @Html.DisplayFor(modelItem => item.TYPE)
    </td>
    <td>
        @Html.DisplayFor(modelItem => item.TOTALE)
    </td>
}
```

join ... on ... equals

join ... on ... equals

join ... on ... equals : permet de faire des jointures entre deux enumérables

Pour afficher le nom et la ville des clients qui commandent le produit de numéro 4 avec les expressions Lambda

Dans la fonction Index (COMMANDEsController)

```
public ActionResult Index()
{
    var cOMMANDEs = db.COMMANDEs.Include(c => c.CLIENT);
    var commande = (from c in cOMMANDEs
       join d in db.DETAILs
       on c.NUMCOM equals d.NUMCOM
       where d.NUMPRO == 4
       select c).ToList();
       return View(commande);
}
```

Outline I

Exercices: : Linq to entities

1. Afficher le numéro, le nom et la ville des clients de catégorie B n'habitant pas à Rabat.
2. Afficher les caractéristiques des produits Informatiques.
3. Afficher les caractéristiques de tous les casques.
4. Donner le numéro, le nom et le compte des clients de Tanger et de Kenitra dont le compte est positif.
5. Quelles catégories de clients trouve-t-on à Casablanca ?
6. Afficher le numéro, le nom et la ville des clients dont le nom précède alphabétiquement la ville où ils résident.
7. Afficher les numéros des clients qui commandent le produit de numéro 4.
8. Afficher les villes des clients qui commandent le produit de numéro 6.
9. Quels sont les Casques qui font l'objet d'une commande ?
10. Dans quelles villes a-t-on commandé en Novembre 2015 ?
11. Afficher les commandes qui spécifient une quantité du produit 1 inférieure à celle que spécifie la commande 4 pour ce même produit.
12. Calculer, pour chaque ville, le nombre de catégories distinctes.
13. Calculer, par jour, le total des montants des commandes.
14. Afficher pour chaque ville, les noms des produits qui y sont commandés tout en triant les villes et les noms des produits par ordre lexicographique.

Outline I

View-Razor

Avant 2012, on avait le choix entre deux moteurs de vue

- ASPX (assez proche de JSP): utilise l'extension .aspx
- Razor: utilise l'extension .cshtml (pour C# + HTML) ou .vbhtml (pour VB + HTML)

Création d'une vue

Pour créer une vue qui correspond à une méthode (Index) créée dans un contrôler (Home)

- Faire un clic droit sur le répertoire *Home* (le nom du controlleur) situé dans *Views*, aller dans **Ajouter** > **Vue**
- Dans ViewName: saisir Index
- Dans Template: choisir **Empty (without model)**
- Décocher toutes les cases
- Cliquer sur **Add**

Création d'une vue

Pour appeler la vue depuis le contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View();
    }
}
```

Explication

- Le compilateur va chercher la vue dans le répertoire *Home* (nom du contrôleur).
- La vue recherchée doit porter le même nom que la méthode appelante.

Création d'une vue

Pour appeler une vue par son nom

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View("ViewName");
    }
}
```

Pour appeler une vue qui n'est pas dans le répertoire associé au contrôleur

```
public class HomeController : Controller
{
     // GET: Home
     public ActionResult Index(int ? id)
     {
          return View("~/Views/Vehicule/Create.cshtml" );
     }
}
```

View- Razor:

- Utilise @ au lieu de <% %>

Un bloc d'instructions

```
@{
    var nom = "Bensalah"; // the compiler determines the type at run time
                                    depending on the value stored in them
    var prenom = "Nouhaila";
    var nomComplet = prenom + " " + nom;
    var date = DateTime.Now.ToShortDateString();
}
```

Une instruction sur une seule ligne

```
<p>Bonjour @nomComplet</p>
```

Pour mettre en commentaire un bloc

```
@*@{
var x = 3;
}*@
```

if/ if ... else

Un bloc if

```
@{
    int nbr = 10;
    if (nbr % 2 == 0)
    {
        <p> @nbr est pair </p>
    }
}
```

Un bloc if ... else

```
@{
    int nbr = 10;
    if (nbr % 2 == 0)
    {
        <p> @nbr est pair </p>
    }
    else
    {
        <p> @nbr est impair </p>
    }
}
```

switch/for

switch ... case

```
<ul>
    @{
        int nbr=10;
        switch (nbr)
        {
            case 0: <li>zéro</li>; break;
            case 1: <li>un</li>;break;
            default: <li>autre</li>;break;
        }
    }
</ul>
```

Boucle for

```
@for(var i = 0; i < 10; i++)
{
<p> @i</p>
}
```

Listes

```
@{
    var list = new List¡int¿();
    list.Add(2);
    list.Add(1);
    list.Add(7);
}
<ul>
    @{
       @foreach (var x in list)
       {
          <li>@x</li>
       }
    }
</ul>
```

Tableaux

```
@{
    var tab = { 2, 1, 7 };
}
<ul>
    @{
        var j = 0;
        while (j < tab.Length)
        {
            <li>@tab[j]</li>
            j++;
        }
    }
</ul>
```

Envoi et récupération d'attribut

ViewData

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        if (id != null)
        {
            ViewData["id"] = id;
            return View();
        }
        else return View("Error");
    }
}
```

Récupérer un attribut dans la vue (Index)

```
Number @ViewData["id"]
```

Envoi et récupération d'attribut

TempData

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        if (id != null)
        {
            TempData["id"] = id;
            return View();
        }
        else return View("Error");
    }
}
```

Récupérer un attribut dans la vue (Index)

```
Number @TempData["id"]
```

Envoi et récupération d'attribut

ViewBag

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        if (id != null)
        {
            ViewBag.id = id;
            return View();
        }
        else return View("Error");
    }
}
```

Récupérer un attribut dans la vue (Index)

```
Number @ViewBag.id
```

Envoi et récupération d'attribut

ViewBag vs ViewData vs TempData

1. ViewBag est un objet, ViewData et TempData sont des dictionnaires ({clé, valeur}).
2. ViewBag est moins rapide que ViewData et TempData.
3. ViewBag, ViewData et TempData permettent de passer les données d'un contrôleur vers une vue.
4. Seul TempData permet de transmettre les données entre contrôleurs.
5. Contrairement à ViewData et TempData, ViewBag ne nécessite pas une conversion de type lors d'une énumération.

Envoi et récupération d'attribut

ViewBag vs ViewData avec les listes

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int? id)
    {
      var list = new List<int>();
      list.Add(2);
      list.Add(1);
      list.Add(7);
      ViewData["list"] = list;
      ViewBag.list = list;
      return View();
    }
}
```

Envoi et récupération d'attribut

ViewBag vs ViewData avec les listes

Récupérer la liste dans la vue (cas d'un ViewBag)

```
<ul>
@foreach (var elt in @ViewBag.list)
{
<li> @elt </li>
}
</ul>
```

Récupérer la liste dans la vue (cas d'un ViewData)

```
<ul>
@foreach (var elt in @ViewData["list"] as List<int>)
{
<li> @elt </li>
}
</ul>
```

Envoi et récupération d'attribut

1. HtmlHelper est une classe qui spécifie une propriété (Age d'un client) à lier avec les HTML controls (Text Input Controls, Checkboxes Controls, Radio Box Controls etc).
2. @Html est un objet de la classe HtmlHelper. (@ symbole est utilisé pour accéder à l'objet côté serveur dans la syntaxe razor).
3. La classe HtmlHelper comprend plusieurs extension methods (TextBox, TextArea, Display etc) et strongly typed methods (TextBoxFor, TextAreaFor, DisplayFor etc)

En utilisant Client model class

```
namespace FirstAspNet.Models
{
    public class Client
    {
        public int ID { get; set; }
        public int Age { get; set; }
        public string Nom { get; set; }
        public Gender ClientGender  get; set;
    }
    public enum Gender
    {
        Male,
        Female
    }
}
```

Envoi et récupération d'attribut

TextBoxFor

@FirstAspNet.Models.Client
@Html.TextBoxFor(m => m.Nom)

Explication

Dans l'exemple ci-dessus, l'expression lambda (m => m.Nom) spécifie la propriété Nom à lier à une zone de texte. Il génère un élément de texte d'entrée avec les attributs id et name.

Html result

<input id="Nom" name="Nom" type="text" value="" />

Envoi et récupération d'attribut

DropdownList

@FirstAspNet.Models.Client
@Html.DropDownListFor(m => m.ClientGender, new SelectList(Enum.GetValues(typeof(Gender))), "Select Gender")

Explication

Dans l'exemple ci-dessus, l'expression lambda (@Html.DropDownListFor(m => m.ClientGender, new SelectList(Enum.GetValues(typeof(Gender))), "Select Gender")) crée une liste déroulante pour la propriété Gender.

Html result

```
<select class="form-control" id="ClientGender" name="ClientGender">
    <option>Select Gender</option>
    <option>Male</option>
    <option>Female</option>
</select>
```

Les plus utilisés

Extension Method	Strongly Typed Method	
Html.ActionLink		Génère un lien vers une méthode du contrôleur avec la balise <a href>
Html.BeginForm		Génère une balise de formulaire <form>
Html.CheckBox	Html.CheckBoxFor	Génère une case à cocher de type "checkbox"
Html.TextBox	Html.TextBoxFor	Génère une zone de texte modifiable sur une seule ligne
Html.TextArea	Html.TextAreaFor	Génère une zone de texte modifiable sur plusieurs lignes
Html.RadioButton	Html.RadioButtonFor	Génère un bouton radio
Html.DropDownList()	Html.DropDownListFor()	crée une liste déroulante pour une propriété.
Html.Password	Html.PasswordFor	Génère une zone de texte permettant de saisir un mot de passe
Html.ListBox	Html.ListBox	Génère une liste déroulante multi-sélectionnable
Html.Hidden	Html.HiddenFor	Génère un champ caché
Html.Display	Html.DisplayFor	Affiche la valeur d'une propriété
Html.Label	Html.LabelFor	Affiche le nom d'une propriété
Html.Editor	Html.EditorFor	Affiche la valeur d'une propriété à l'aide d'un champ de saisie

Envoi et récupération d'attribut

On peut aussi utiliser View pour passer un Modèle

Créer une classe Client dans Models

```
namespace FirstAspNet.Models
{
    public class Client
    {
        public int Age { get; set; }
        public string Nom { get; set; }
        public string Prenom { get; set; }
    }
}
```

Dans l'action Index() du contrôleur

```
Client c = new Client()
{
    Age = 25,
    Nom = "NomA",
    Prenom = "PrenomA"
};
return View(c);
```

Envoi et récupération d'attribut

Dans la vue (Index.cshtml)

```
<ul>
@{
<li> @Model.Age </li>
<li> @Model.Nom </li>
<li> @Model.Prenom </li>
}
</ul>
```

N'oubliez pas d'ajouter dans le contrôleur

```
using FirstAspNet.Models;
```

N'oubliez pas d'ajouter dans la vue

```
@model FirstAspNet.Models.Client
```