Activité n°3 – *Gestion des réservations*

Nous allons créer une application C# permettant de gérer la réservation des véhicules du parc automobile de la société GSB. Cette application devra utiliser une technologie d'accès aux données pour interagir avec une base MySQL.

I. ORM - Entity Framework

L'ORM (object relationnal mapping ou mapping objet-relationnel) est un outil permettant la génération d'une couche orientée objet d'accès aux données à partir d'une base relationnelle. Pour cela, l'ORM définit des correspondances entre la base de données et des objets. Il existe plusieurs outils d'ORM pour le framework .NET qui permettent de générer des objets C# (Entity Framework, LINQ, NHibernate, ...).

Dans cette activité, nous allons utiliser l'ORM Entity Framework proposé par Microsoft. Entity Framework permet :

- de générer un modèle à partir d'une base de données existante,
- de gérer tous les accès à la base de données (lecture, ajout, suppression, ...)
- de modéliser les données et de générer la base correspondante.

Nous allons pouvoir travailler avec des objets en sous-traitant à l'ORM la persistance dans la base de données.

Entity Framework propose 3 approches différentes pour lesquelles vous pouvez consulter les tutoriaux suivants :

- DataBase First : http://www.entityframeworktutorial.net/EntityFramework5/create-dbcontext-in-entity-framework5.aspx
- Code First : http://www.entityframeworktutorial.net/code-first/entity-framework-code-first.aspx
- Model First : http://www.entityframeworktutorial.net/model-first-with-entity-framework.aspx

Microsoft propose une présentation d'Entity Framework au lien suivant : https://msdn.microsoft.com/fr-fr/library/gg696172(v=vs.103).aspx.

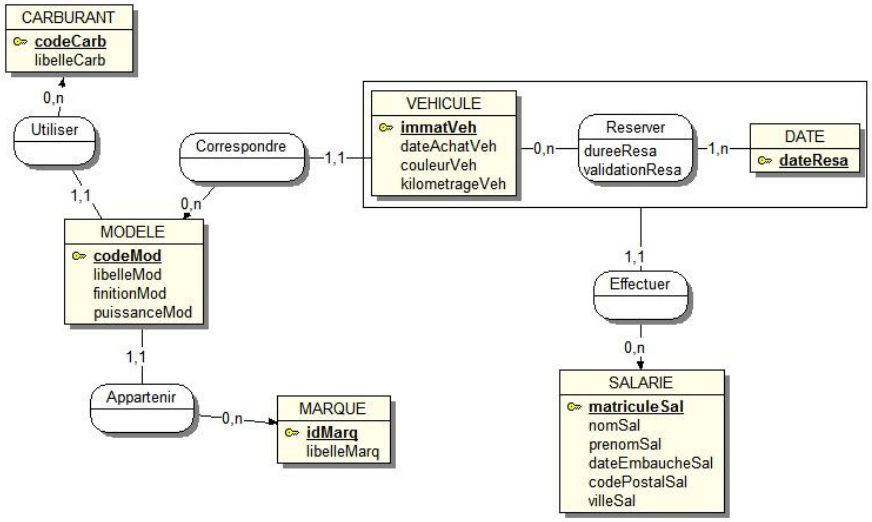
II. Application de gestion du parc automobile

Les véhicules disponibles sont réservables par journée complète et chaque réservation effectuée par un salarié ne peut dépasser deux jours. Une réservation effectuée réellement sera validée dans la base. Un crédit véhicule sera attribué à chaque salarié (par exemple, un développeur a droit à 3 jours d'emprunt).
L'application devra permettre :

- De réserver un véhicule à une date donnée pour un ou deux jours.
- De visualiser les disponibilités d'un véhicule.
- De visualiser l'ensemble des réservations actives ainsi que les réservations par véhicule ou par salarié.

1. La base de données

La base de données sera implémentée sous MySQL à partir du modèle conceptuel des données ci-dessous :



Dans un premier temps, nous ne travaillerons qu'avec les entités VEHICULE, DATE, SALARIE et les associations Reserver et Effectuer.

Travail à réaliser :

- Ecrire le modèle de relations correspondant.
- Créer la base de données ParcAuto sous MySQL à partir de ce modèle.

2. Architecture

L'application se décompose en 3 couches :

- la base de données,
- la couche de mapping utilisant le Framework Entity (modèle objet relié aux tables),
- la couche présentation (application WinForm).

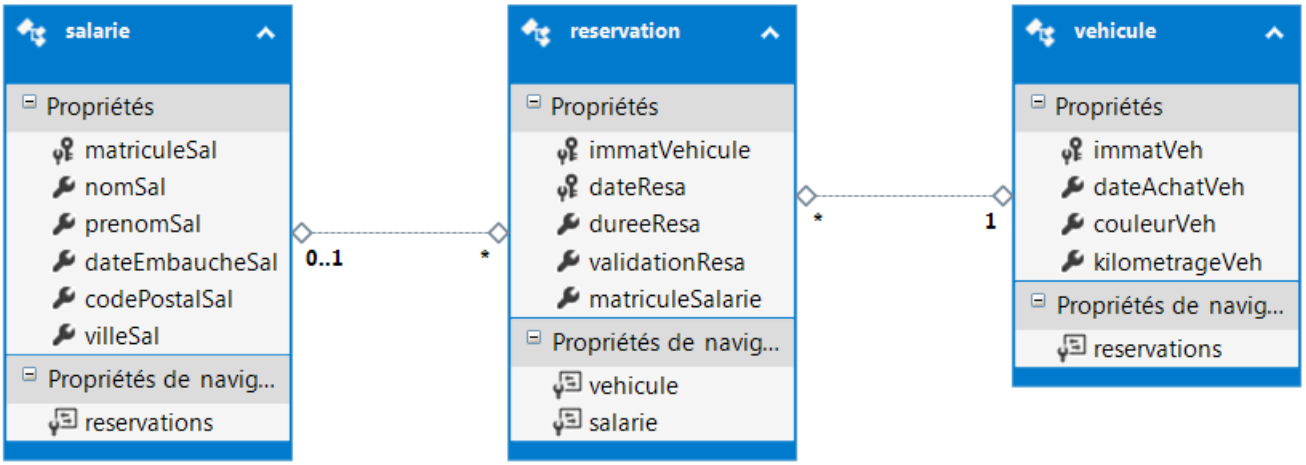
3. Création du projet

Cette activité doit vous permettre de découvrir et manipuler les fonctionnalités de l'accès aux données avec Entity Framework.

Travail à réaliser :

- Créer un projet WinForm nommé GSBParcAuto.
- Ajouter un nouvel élément ADO.NET Entity Data Model au projet, nommé mdlParcAuto.edmx.
- Avec l'aide de l'assistant, demander à créer une nouvelle connexion :
  - sélectionner le driver d'accès à la base MySQL (MySQL Database),
  - configurer la connexion en cochant l'enregistrement du mot de passe,
  - sélectionner la base de données ParcAuto dans la liste déroulante puis tester la connexion,
  - accepter le nom de connexion proposée,
  - sélectionner les tables salarie, vehicule et reservation,
  - cocher les cases Mettre au pluriel... et Inclure les ... ,
  - enfin, avant de terminer, modifier le nom de l'espace de noms du modèle en parcautoEF.

Entity Framework génère un modèle objet et le charge en mémoire :



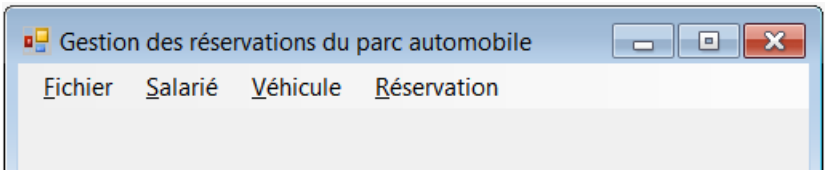
Chaque table devient ainsi une classe. Les propriétés de navigation offriront la possibilité de naviguer de classe en classe. Le singulier ou le pluriel sur ces propriétés signifie par exemple qu'une réservation ne concerne qu'un seul véhicule et qu'un seul salarié ; en revanche, le salarié a plusieurs réservations, et le véhicule aussi.

La classe Reservation contient des champs correspondant aux clés étrangères : immatVehicule et matriculeSalarie. Il est possible de ne pas générer les clés étrangères en décochant la case Inclure les colonnes clés étrangères vue avec l'assistant.

Il est possible de modifier le modèle en supprimant des propriétés de navigation.

Vous pouvez vérifier que les paramètres de connexion apparaissent bien dans le fichier XML App.Config.

4. Formulaire de menu



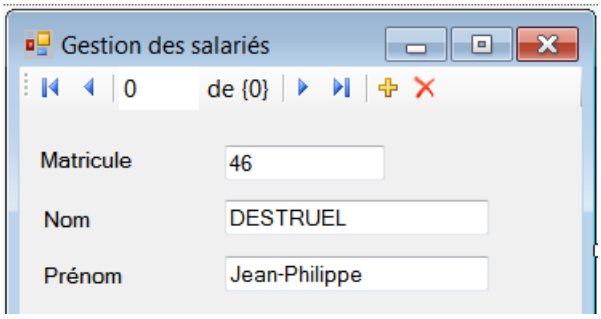
Travail à réaliser :

- Modifier le nom du formulaire en FormMenu puis ajouter un menu.
- Dans le menu, ajouter les sous-menus suivants :
  - Fichier / Quitter
  - Salarié / Gérer
  - Véhicule / Gérer et Véhicule / Liste
  - Réservation / Ajouter et Réservation / Valider
- Afin de créer un objet d'accès aux données dans ce formulaire, déclarer un attribut privé de type parcautoEntities qui est la classe de contexte. Cette classe assure la liaison à la base, gère le mapping avec le modèle relationnel et sauvegarde les données en base.

```
public partial class FormMenu : Form
{
    private parcautosEntities mesDonneesEF;
    public FormMenu()
    {
        InitializeComponent();
        this.mesDonneesEF = new parcautosEntities();
    }
```

Chaque formulaire associé à une option du menu sera construit avec ce contexte en paramètre (argument du constructeur).

5. Formulaire de gestion des salariés



Travail à réaliser :

- Ajouter au projet un nouveau formulaire nommé FormSalarie.
- Réaliser la conception de ce formulaire en ajoutant les différents composants que vous prendrez soin de nommer correctement (txtMatricule, ...). Ne prenez pas en compte la barre de navigation pour le moment.
- Lier l'ouverture de ce formulaire au formulaire FormMenu :

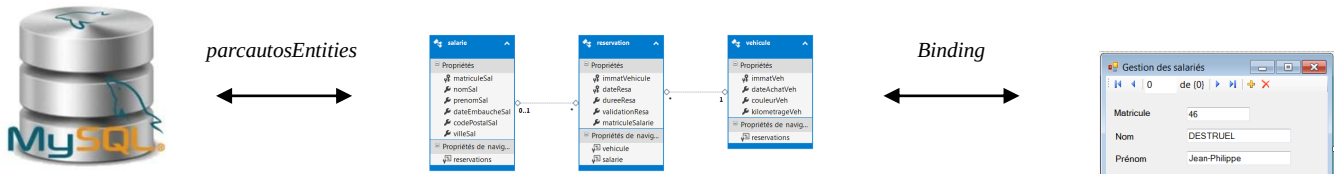
```
public partial class FormSalarie : Form
{
    private parcautosEntities mesDonneesEF;
    public FormSalarie(parcautosEntities mesDonneesEF)
    {
        InitializeComponent();
        this.mesDonneesEF = mesDonneesEF;
    }
```

Dans le code de FormMenu :

```
    private void gérerToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormSalarie f = new FormSalarie(this.mesDonneesEF);
        f.Show();
    }
```

6. Notion de binding

Le binding est un mécanisme qui permet de lier un composant graphique de présentation à une source de données. Cette liaison est bi-directionnelle ; en effet, si l'on modifie la source, l'information sera transmise au composant et si on modifie l'information présente dans le composant, la modification sera répercutée dans la source de données.
Une source de données peut être de nature très variée : une table d'une base, une classe en mémoire, une liste typée.

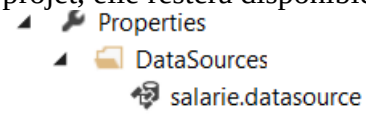


Le binding est mis en œuvre grâce à un composant à installer dans le formulaire.

Travail à réaliser :

- Déposer un composant BindingSource dans le formulaire FormSalarie. Il va apparaitre en bas du formulaire car il ne s'agit pas d'un composant graphique. Renommer le en bindingSourceSalarie et indiquer comme source de données la classe salarie (et non la table salarie). Pour cela, dans les propriétés du composant, cliquer sur DataSource, ajouter une nouvelle source de données, indiquer que la source est un objet et sélectionner la classe salarie.

Cette source de données est ajoutée au projet, elle restera disponible pour d'autres formulaires.



Le composant de binding connait maintenant sa source de données. La liaison se fait sur la classe mais il n'y a pas encore de chargement des données venant de la classe de contexte de données.

Travail à réaliser :

- Réaliser le chargement des données à l'aide du code suivant :

```
public FormSalarie(parcautosEntities mesDonneesEF)
{
    InitializeComponent();
    this.mesDonneesEF = mesDonneesEF;
    this.bindingSourceSalarie.DataSource = mesDonneesEF.salaries;
}
```

Le composant de binding est lié à la source de données. Nous allons maintenant lier les composants graphiques et le composant de binding.

Travail à réaliser :

- Modifier la propriété DataBinding-Text du textBox correspondant au matricule du salarié en le liant au matriculeSal du bindingSourceSalarie.
- Effectuer les modifications sur les autres composants du formulaire.
- Ajouter au formulaire (si ce n'est pas encore fait) un composant BindingNavigation dédié à la navigation et le nommer bdgNavigateur.
- Paramétrer la propriété BindingSource du navigateur (bindingSourceSalarie).

Nous allons maintenant modifier les boutons de la barre de navigation. Le bouton d'ajout du navigateur pose le problème de l'identifiant du salarié car celui-ci n'est pas auto-incrémenté dans la base de données. Nous utiliserons donc, dans le code du bouton ajouter, une méthode qui recherche le dernier matricule de salarié à l'aide d'une requête LINQ.

Travail à réaliser :

- Se documenter sur les requêtes LINQ et préparer un diaporama de présentation.
- Ecrire la méthode de gestion du matricule :

```
private int getMatriculeSalarie()
{
    var requeteDernier = (from sal in this.mesDonneesEF.salaries
                          orderby sal.matriculeSal descending
                          select sal);
    salarie dernierSalarie = requeteDernier.First();
    int matricule = dernierSalarie.matriculeSal + 1;
    return matricule;
}
```

- Après avoir mis le champ txtMatricule en lecture seule, utiliser cette méthode pour ajouter un nouveau salarie :

```
private void bindingNavigatorAddNewItem_Click(object sender, EventArgs e)
{
    this.textMatricule.Text = this.getMatriculeSalarie().ToString();
    this.bindingSourceSalarie.EndEdit();
    this.mesDonneesEF.SaveChanges();
}
```

L'instruction endEdit() permet de valider dans le modèle de classe la création. Cet appel doit se faire explicitement parfois lorsque nous voulons faire prendre en compte immédiatement une modification. Cet appel est le plus souvent fait automatiquement.

L'appel à SaveChanges() se justifie ici car la méthode getMatriculeSalarie est un requête LINQ qui va chercher dans la base de données un nouveau numéro ; si nous n'enregistrons pas dans la base ce numéro, la requête retournera toujours la même valeur.

Travail à réaliser :

- Pour terminer, ajouter un bouton Enregistrer à la barre de navigation :

```
private void toolStripEnregistrer_Click(object sender, EventArgs e)
{
    this.bindingSourceSalarie.EndEdit();
    this.mesDonneesEF.SaveChanges();
}
```

7. Autres formulaires

En vous inspirant du formulaire précédent, réaliser les différents formulaires de l'application.

Travail à réaliser :

- Formulaire Véhicule / Gérer : similaire au précédent. Pensez à gérer toutes les situations d'utilisation de la barre de navigation.
- Formulaire Véhicule / Liste :
  - utiliser un composant de binding (bdgVehicules)
  - déposer ensuite un DataGridView et le paraméter
  - ajouter un bouton dans le formulaire pour enregistrer les modifications.
- Formulaire Ajout d'une réservation :
  - utiliser deux ComboBox pour les salariés et les véhicules.
  - écrire une méthode qui génère le numéro de la nouvelle réservation à l'aide d'une requête LINQ :

```
private int nouveauNumReservation()
{
    int num;
    ...
    return num;
}
```

  - écrire une méthode qui retourne une nouvelle réservation :

```
private reservation nouvelleReservation()
{
    int num = nouveauNumReservation();
    salarie sal = (salarie)cmbSalarie.SelectedValue;
    ...
    reservation resa = ...;
    ...
    return resa;
}
```

  - écrire le code du bouton enregistrer.
- Formulaire Réservation / Valider : utiliser un DataGridView.