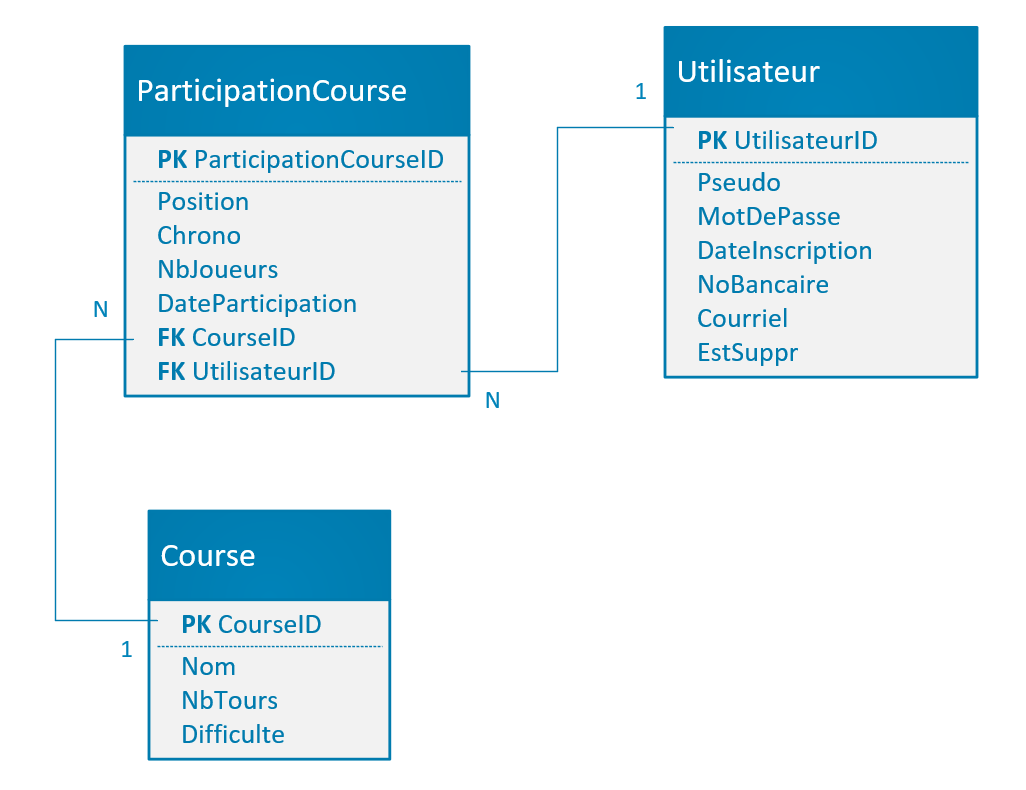
**R20\_Labo**



**Mise en situation :**

* L’application Web **SussyKart** propose un petit jeu de course codé en JavaScript.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

* Vous pouvez essayer le Tutoriel et Jouer des parties qui seront ajoutées à la BD.
* Le site veut mettre à la disposition des joueurs des statistiques sophistiquées (« Scoreboards ») sur le jeu et les participations

Il y a beaucoup de participations.

**Voici un sommaire des choses que vous ferez dans ce laboratoire :**

1. Dans un premier temps, vous devrez **faire une vue SQL** pour obtenir les participations des courses.
2. Vous utiliserez cette vue SQL pour afficher les statistiques demandées dans la vue Razor ci-dessousUne image contenant texte, logiciel, Police, nombre

   Description générée automatiquement
3. Vous devrez ajouter quelques index pour optimiser la performance des requêtes.
4. Finalement, on modifiera la méthode de récupération des données afin de n’exécuter la requête qu’une fois qu’elle est complètement préparée (**.AsQueryable()).**

**Consignes générales**

**1 - Migrations versionnées avec Evolve**

Tous les changements que vous ferez à la BD devront faire partie de migrations standardisées exécutables avec Evolve. Notez qu’il est **interdit** de modifier les migrations **1.0** à **1.3**.

Rien ne vous empêche de « supprimer » ou modifier des objets SQL créés dans ces migrations, mais il faudra le faire à l’aide de nouvelles instructions SQL dans les migrations suivantes !

Assurez-vous que si on ouvre votre projet avec une BD vide, toutes les migrations fonctionnent à l’aide de la commande Evolve.

**Dans ce laboratoire, on vous demandera de commencer la numérotation de vos migrations à 2.0.**

**2 - Models et DbContext générés par scaffolding**

Tous vos **Models** et votre **DbContext** devront être générés grâce à la commande **dotnet ef scaffold** que nous avons abordée. Vous ne pouvez pas modifier les **Models** et le **DbContext** qui ont été générés par cette commande.

**3 - Statistiques (Scoreboards)**

* La page « **Toutes les participations** » doit absolument être fonctionnelle.

## Début du laboratoire :

## Étape1 : Créer la migration V2\_0\_\_VW\_toutesLesParticipations

* **Cette migration devra créer une vue SQL qui gère les jointures pour l’affichage de ces statistiques,** ce qui simplifiera le code dans le contrôleur.
* **L’extrait ci-dessous montre les champs que doivent avoir votre vue.**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* Faire toutes ces jointures dans la base de données plutôt que dans des requêtes Link améliorera la performance.
* Une fois que la migration est complétée exécutez-là à l’aide de la commande Evolve.
* Exécutez la commande dotnet ef scaffold avec -- force pour mettre à jour les modèles.

## Étape2 : Ajustez le code de l’action ToutesParticipations()

* Il faudra aller chercher les **30 premières participations de notre vue sql.**
* Modifiez la signature de l’action pour qu’elle soit **async**.
* Modifiez **FiltreParticipationVM.cs** **pour y ajouter une liste qui contient les données à afficher.**
* Retournez les 30 participations à la vue rasor **ToutesParticipations.cshtml**

## Étape3 : Ajustez la vue correspondante à cette action.

* Complétez la section dans le bas de la vue **toutesParticpations.cshtml** pour afficher les données qui ont été envoyées à cette vue.

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

## Étape 4 : Appliquer les filtres proposés dans la page de statistiques

Vous devrez compléter le code de l’action **ToutesParticipationsFiltre.**

* Vous devrez obtenir **la liste des participations** à partir de la vues SQL.
* Vous devrez faire des **requêtes LINQ pour filtrer les données**. Vous aurez à utiliser
  + .Where
  + .OrderBy
  + .OrderByDescending
* Un système de filtre est déjà partiellement mis en place dans l’action.
* Il y a beaucoup de participations dans la BD, mais vous devrez afficher **30 participations à la fois dans la page**. À l’aide de **.Skip** et **.Take**, affichez les participations, **page par page**.
  + Pour vous aider à y arriver, pensez en termes de noPage.
    - À la page no 1, vous ne voulez pas sauter 30 enregistrements.
    - À la page no 2, vous voulez en sauter 30.
    - À la page no 3, vous voulez en sauter 60.
    - Etc.

**Section optimisation**

Pour s’initier à l’optimisation, nous allons faire deux modifications dans notre projet.

## Étape 5 : Index

* **Créez une migration nommée V2\_1\_\_ Index**
* Ajoutez trois index **non clustered** de votre choix dans une migration. Pour chaque index, justifiez, avec un court commentaire (directement au sein de la migration), pourquoi vous avez choisi la / les colonne(s).
* Exécutez la migration avec Evolve

## Étape 6 : Utilisation de AsQueryable()

Modification dans le projet Web :

* Quand on fait **.ToListAsync()** sur un **DbSet**, Entity Framework doit charger toutes les données dans la mémoire de l’application Web. (Ce qui est très mauvais)
  + Seulement pour l’action **ToutesParticipationsFiltre**, retardez le plus possible le chargement des données dans l’application.
  + Pour y arriver utilisez le type **IQueryable<T>** puisque le filtrage se fait avec plusieurs instructions et **.****AsQueryable()** au lieu de **ToListAsync().**
  + Après avoir appliqué tous les filtres pertinents, utilisez **.ToListAsync()** pour n’aller chercher que les données qu’on veut vraiment.
  + Grâce à ce mode opératoire, Entity Framework chargera seulement les quelques données vraiment nécessaires dans l’application et la mémoire jonglera moins avec des données qui ne nous servent à rien.