**Laboratoire 10**

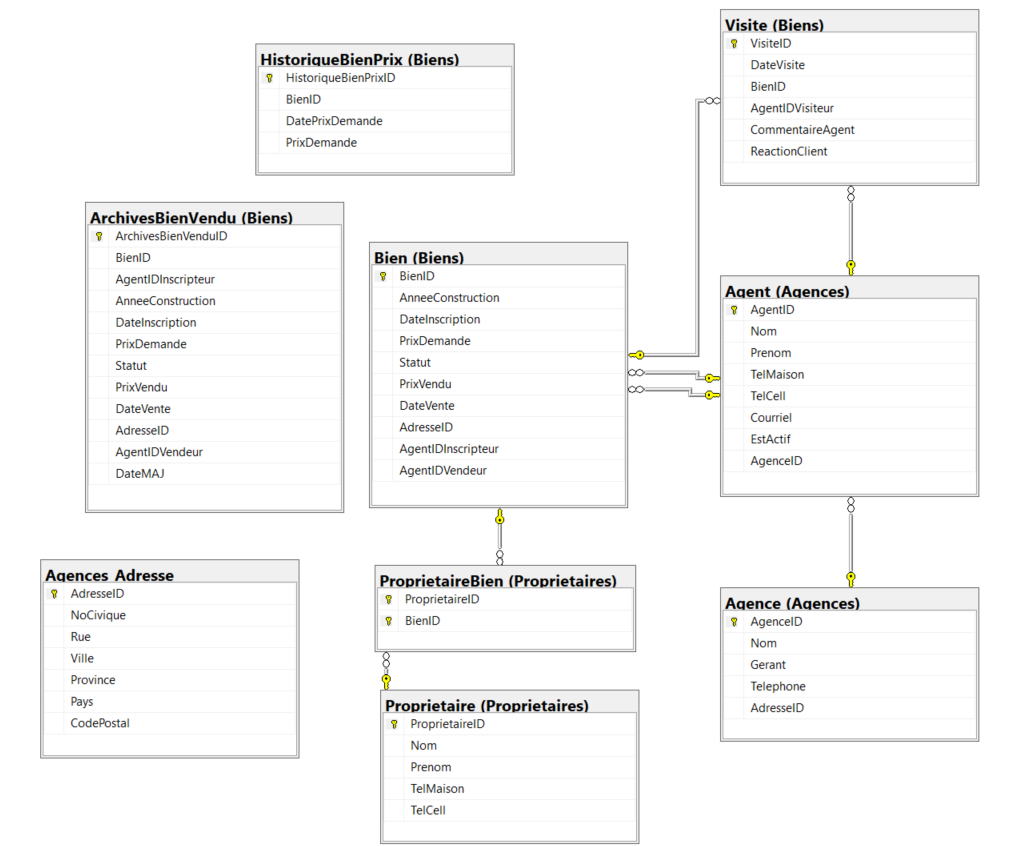
Une fonction et des déclencheurs

**Partie 1 - Un trigger pour deux actions DML 👁**

ATTENTION ! Avant de commencer :

* Exécutez les scripts **420-4D5\_R10\_Labo\_create** et **420-4D5\_R10\_Labo\_insert** dans SSMS.

Dans cette étape, vous aurez à remplir le fichier **420-4D5\_R10\_Labo\_Fonction et Trigger.sql** pour créer un déclencheur qui s’exécutera **après l’insertion OU l’update d’un bien**.



**PARTIE 1 :**

* **Faites un trigger qui se déclenchera quand :**
* **On fera l'insertion d'un bien** (INSERT)
* **OU qu'on mettra à jour** (UPDATE) **le champ** **PrixDemande** **de la table** Biens.Bien **OU le champ** **DateVente** **de la table** Biens.Bien.

Ce trigger s'exécutera **après** l'instruction **INSERT** ou **UPDATE** sur la table **Biens.Bien**.

1. Dans le cas d'un **INSERT**, on voudra insérer dans la table **Biens.HistoriqueBienPrix** les données **BienID**, **DateInscription** et **PrixDemande** comme premier enregistrement lié à ce bien.
2. Dans le cas d'un **UPDATE,** Il faudra vérifier si :
   1. L'update a eu lieu sur le champ **PrixDemande** de la table **Biens.Bien**.
      1. Si tel est le cas, on voudra insérer dans la table **Biens.HistoriqueBienPrix** les données **BienID**, la **date d'aujourd'hui** et le **nouveau PrixDemande**. Cela nous permettra de garder un historique de fluctuation des prix demandés.
   2. L’update a eu lieu sur le champ **DateVente** de la table **Biens.Bien.**
      1. Si tel est le cas, on voudra **insérer les valeurs de cet enregistrement** dans la table **Biens.ArchivesBienVendu** tout en conservant l'**ID** du bien identique à celui de la table **Biens.Bien.**
      2. **Supprimez** ensuite cet enregistrement de la table **Biens.Bien**.
      3. **Supprimez** également les données sur **les visites (Biens.Visite)** sur le bien nouvellement vendu, car nous ne garderons pas les informations sur les visites des biens vendus. De sorte que dans la table **Biens.Bien,** il n'y aura, toujours, que les biens non vendus.
      4. **Insérez** également dans la table **Biens.HistoriqueBienPrix** les données **BienID**, la **DateVente** et le **PrixVendu**. Ce sera la dernière entrée de notre table **Biens.HistoriqueBienPrix** pour ce bien.

**PARTIE 2:**

1. **Faites un trigger qui se déclenchera quand on voudra supprimer un agent de la table dbo.Agent parce qu'il quitte l'AGENCE.**

Ce trigger s'exécutera **à la place** de l'instruction DELETE. Nous ferons ce que l’on appelle un **‘Soft delete’**. On ne supprimera pas vraiment l’agent, on va plutôt le rendre inactif après avoir transférés les biens qui sont sous sa gestion à une autre agent (celui qui est le plus ancien et qui a le moins de biens à vendre présentement).

**CE TRIGGER :**

1. Vérifiera si l’agent est présentement **l'agentIDVendeur** de biens non vendus.
   1. Si c'est le cas, il retirera les biens de l'agent qui quitte l'agence pour les donner à l'agent le plus ancien (celui dont l'AgentID est le plus bas) **ET** qui a le moins de biens à vendre présentement.
2. Mettra le **champ EstActif** de la **table Agences.Agent** à **0** pour l'agent qui quitte l'agence.

**CONSEILS :**

* + - * Faites une fonction pour retourner l’ID de l’agent le plus ancien **ET** ayant le moins de biens à vendre pour le moment, dont l’ID est différent de l’ID de l’agent qui veut quitter l’agence.
* VOTRE TRIGGER utilisera cette fonction.

**TESTS :**

* Faites vos tests en supposant que **l’agentID 1 quitte l’agence**.
* Normalement, si votre trigger est bien fait, vous devriez avoir les biens de **l'agentIDVendeur** 1 transférés à l'agent dont l'ID est 5 et ce dernier devrait se retrouver avec 4 biens à gérer.