DEX450 Exercice Appliqué

Cet exercice représente des exemples applicatifs sur les modules de la formation DEX450. L'exercice sera réalisé sur un environnement DEV propre, séparé de l'environnement de la formation, qui peut être créé sur : https://developer.salesforce.com/signup

L'exercice représente la gestion d'une application pour une entreprise de bâtiments et construction.

Jour 1 : Model de Donnés, Application Lightning et SOQL

1- Initialisation du modèle de données

Pour mieux gérer les travaux de constructions, les clients et les produits de l'entreprise, les objets suivants seront créés/utilisés :

- **Ingénieur** : cet objet représente les ingénieurs ou les sociétés d'ingénierie associés à l'entreprise ou travaillant pour la construction des bâtiments de l'entreprise
  - Cet objet sera représenté par l'objet standard **Compte** (**Account**)
  - 2 types de Comptes existent : Personnel ou Entreprise

  Un ingénieur/Société d'ingénieurs sera représenté par les informations suivantes :

  - Un ingénieur peut être de deux types différents : Personnel / Entreprise
  - Nom : Texte
  - Prénom : Texte – visible uniquement pour le type personnel
  - Nombre d'employés : Nombre – visible uniquement pour le type entreprise
  - Nombre d'années d'expériences : Nombre – visible uniquement pour le type personnel
  - Date de démarrage du Contrat : Date

- **Village** : cet objet représente un village contenant plusieurs Bâtiments
  - Etiquette : Village
  - Etiquette plurielle : Villages
  - Nom de l'objet : Village
  - Nom de l'enregistrement : Numéro Automatique

  Un Village est représenté par les informations suivantes :

- Nom : Numéro automatique
- Adresse : Texte

- **Bâtiment** : cet objet représente la liste des bâtiments construits ou gérés par l'entreprise
  - Etiquette : Bâtiment
  - Etiquette plurielle : Bâtiments
  - Nom de l'objet : Batiment
  - Nom de l'enregistrement : Nom du Bâtiment
  - Type de données : Texte

  Un Bâtiment comprend les informations suivantes :

  - Nom : Texte
  - Nombre d'étages : Numéro
  - Date de début des travaux : Date
  - Date de fin des travaux : Date
  - Durée des travaux : Champs formule calculé automatiquement à partir des dates de début et de fin des travaux (nombre total de jours de travaux)
  - Prix de l'appartement : Devise
  - Nombre d'appartements : Nombre
  - Ingénieur de Réalisation : Relation de Recherche vers l'objet **Ingénieur**
  - Village Associé : Relation vers l'objet **Village**
    Le nombre de bâtiments pour chaque Village doit être afficher dans le champ 'Nombre de Bâtiments' sur chaque enregistrement de Village.

- **Ouvriers** : cet objet représente les ouvriers qui ont participé à la construction d'un Bâtiment (l'objet standard **Contact** sera utilisé pour représenter les ouvriers)

  Un Ouvrier sera représenté par les informations suivantes :

  - Nom : Texte
  - Prénom : Texte
  - Date de démarrage du Contrat : Date
  - Date de fin du Contrat : Date

  *Un **Ouvrier** peut participer à la construction de plusieurs **Bâtiments**, ainsi, un **Bâtiment** peut avoir plusieurs **Ouvriers**.*

2- Créer l'application Projets

A partir du 'Gestionnaire des applications', créer une nouvelle application Lightning avec les informations suivantes :

- Nom de l'application : *Projets*
- Nom Développeur : *Projets*

- Ajouter un logo pour votre application

L'application doit permettre aux utilisateurs l'accès aux objets créés

3- Créer un Classe APEX

A partir du console développeur, créer une classe APEX avec le nom : **AP01_Ouvrier**.

Cette classe doit contenir la méthode : **envoyerEmail** qui permet d'envoyer un Email au propriétaire de l'ouvrier au moment où la Date de Fin du Contrat pour l'ouvrier change (durant la création ou modification d'un ouvrier).

La méthode **envoyerEmail** doit être **static** et prend les paramètres suivants :

- Enregistrement ouvrier

La méthode doit utiliser SOQL pour récupérer l'adresse mail du propriétaire de l'exécutant (l'utilisateur propriétaire) et doit implémenter **`Messaging.SingleEmailMessage[].`**

Tester votre méthode à partir de l'exécution Anonymous du Console Développeur et examiner les résultats.

Jour 2 : DML, Triggers et Classes Test

4- DML

- A partir du Console Développeur, créer une classe **AP01_Batiment**. Dans la classe créer la méthode **attacherIngenieur** qui permet de créer/rattacher un Bâtiment à un Ingénieur dans le cas où ce Bâtiment n'est pas rattaché à un Ingénieur.

  Cette méthode doit être **static**, prend en paramètres l'ID d'un enregistrement de Bâtiment et doit utiliser SOQL pour retrouver l'enregistrement du Bâtiment et les DML pour créer un Ingénieur de type personnel et mettre à jour l'enregistrement de Bâtiment pour le rattacher à l'Ingénieur créé.

  La création doit utiliser la commande Standalone *insert* et la mise à jour doit utiliser la méthode système *Database.update.* Pensez à gérer les Exceptions qui peuvent arriver en utilisant les try/catch.

  Tester votre méthode à partir de l'exécution Anonymous du Console Développeur et examiner les résultats dans Salesforce.

5- Apex Trigger

- Créer un nouveau Trigger '**OuvrierTrigger**' sur l'objet **Ouvrier** qui se déclenche aux événements **update** et **insert**.
  Pour automatiser l'envoie de mail aux propriétaires des exécutants, ce trigger qui doit appeler la méthode **envoyerEmail** créée dans l'étape 4 et modifiée dans l'étape 5. Cette méthode doit être modifiée pour prendre en compte une liste d'ouvrier provenant du trigger.
  Attentions aux SOQL dans les boucles ***for.*** Le code doit être optimisé en utilisant des collections (List et Map) pour éviter de mettre des SOQL dans les boucles for.

- Pour automatiser le rattachement d'un **Ingénieur** à un nouveau bâtiment, le trigger '**BatimentTrigger'** doit appeler la méthode **attacherIngenieur** de la classe **AP01_Batiment** créée dans l'étape 6. La logique de la méthode doit être modifiée pour prendre en compte une liste de Bâtiments au lieu d'un seul bâtiment.
  Attentions aux SOQL dans les boucles ***for.*** Le code doit être optimisé en utilisant des collections (List et Map) pour éviter de mettre des SOQL dans les boucles for.

6- Classes Test

- Assurer une couverture de code **> 90%** pour chaque classe créée dans les étapes précédentes ainsi que les triggers.

Les données de Tests doivent être créées dans une nouvelle classe '**Test_Utils**'.
Pensez à bien utiliser les Assertion pour vérifier les résultats.

7- Trigger et APEX design

- Assurer que les bonnes pratiques sont utilisées dans les classes et les triggers créés pour éviter les limites (heap size, DML, SOQL…) dans le cas d'utilisation des batch d'enregistrements.

Jour 3 : Contrôleurs Visualforce et SOSL

8- Créer une page Visualforce avec le design du Lightning

- L'entreprise souhaite suivre les informations de ventes de ses bâtiments en affichant les bâtiments vendus et les clients qui ont acheté des bâtiments.
    - Créer un nouvel Objet '**Client'** : cet objet représente les clients qui ont acheté des appartements dans les bâtiments.

      Un client sera représenté par les informations suivantes :

        - Nom : Texte
        - Prénom : Texte
        - Numéro d'appartement Acheté : Nombre
        - Bâtiment : Relation vers l'objet Bâtiment
        - Date d'achat : Date
        - Somme payée : Devise
        - Statut : Liste de sélections (Actif/Inactif)

    - Pour suivre les informations de ventes, créer une page Visualforce avec un design Lightning '**VFP01_InformationsVentes**', déclenchée à partir d'un onglet '**Informations de Ventes**', qui permet d'afficher tous les appartements vendus, avec les informations de l'acheteur (Nom, Prénom), le numéro de l'appartement, la date d'achat, le bâtiment concerné et la somme payée pour chaque appartement

    - L'entreprise souhaite avoir une somme totale de tous les appartements vendus

    - L'utilisateur pourra cliquer sur le nom du client pour être rediriger vers la page d'enregistrement du client dans un nouvel onglet

    - Créer la classe Test pour le contrôleur associé à la page Visualforce et assurer une couverture de code > **90%**