# Mise en situation : S-EAU-S

Lors de son élection en 2015, le gouvernement Trudeau avait promis de donner accès à l'eau potable à toutes les communautés autochtones du Canada. En 2021, il est devenu clair que cette promesse ne sera pas complétée dans le second mandat du gouvernement non plus. Afin de garder un œil sur la situation, vous décidez de concevoir un outil de suivi des différents projets dans les différentes communautés.

Votre outil, en plus de s'intéresser aux **projets** et aux **communautés**, va aussi suivre les **compagnies** qui exécutent ces projets.

Plus spécifiquement, un **projet** :

- Est mené par une **compagnie** ;
- Est supervisé par une personne chargée du projet ;
- A un budget initial (prévision). C'est le coût estimé du projet avant le début des travaux;
- A un budget final (coût réel). C'est le coût final du projet, une fois les travaux complétés ;
- A lieu dans une **communauté** ;
- A une date d'annonce initiale ;
- A une date de début des travaux ;
- A une date de fin des travaux ;
- A un état d'avancement.

Les états d'avancement possibles sont : « **Planifié** », « **En cours** » et « **Terminé** ».

Une **communauté** a :

- Un nom ;
- Une nation d'appartenance ;
- Une personne qui dirige ;
- Des coordonnées géographiques.

Une communauté peut avoir plus d'un projet.

Une **compagnie** a :

- Une adresse ;
- Un nom ;
- Une ou des compagnies parentes.

Une compagnie peut être à la fois parente ou parentée par une ou plusieurs autres compagnies.

# Quiz 3-4 et Labo 3-4 – Exercice de modélisation

## Objectifs

Ce laboratoire a comme objectifs de :

1. Modéliser une situation à l'aide d'un diagramme entité-association;
2. Transcrire un modèle entité-association en schéma relationnel;
3. Concevoir une base de données relationnels en SQL.

Pour ce laboratoire, vous utiliserez PostgreSQL pour écrire la base de données.

## Tâches

Cet exercice se divise en 3 étapes :

1. Complétez le modèle entité-association du quiz 3-4 sur Moodle pour 1%.
2. Écrivez le schéma relationnel résultant. Cette étape n'est pas remise, donc vous pouvez utiliser le médium de votre choix (papier ou logiciel de votre choix). Faites valider votre schéma par la personne enseignante.
3. Écrivez votre script SQL et remettez-le sur Turnin. Un script qui a été bien transcrit à partir d'un modèle exact se vaudra tous les points (5%) sur Turnin. Plusieurs remises sont possibles jusqu'à l'obtention des points.

# Labo 5

Maintenant que vous avez créé la base de données, on voudrait pouvoir écrire un logiciel en Java qui se connecte à la BD pour y envoyer des requêtes.

## Objectif

Ce laboratoire a comme objectifs de :

1. Écrire et exécuter des PreparedStatement
2. Parcourir et lire des ResultSet
3. Utiliser une connexion JDBC à une base de données PostgreSQL.

## Tâche

Pour ce laboratoire, vous devrez compléter le code du fichier SeauS.java se trouvant dans Labo5.zip. Voici les étapes à compléter :

1. Télécharger Labo5.zip sur Turnin et extraire les fichiers.
2. Complétez le code de SeauS.java (package SeauS).
3. Zippez tout le contenu de SeauS-JDBC et nommez le zip « Labo5.zip ».

4. Remettez Labo5.zip sur Turnin. Des tests d'exécution s'effectueront et vous retourneront une note sur 5.

### Consignes

Respectez les consignes suivantes pour obtenir tous vos points :

1. Complétez tous les endroits avec des commentaires « TODO ». Regardez les exemples déjà dans le fichier.
2. Suivez les indications en commentaires.
3. Fermez vos ResultSet une fois que vous avez terminé avec (n'oubliez pas de les fermer aussi en cas d'exception)!
4. Faites un commit en cas de réussite et un rollback en cas d'échec. Toujours important de le faire, même pour un SELECT, pour libérer les ressources.
5. Ne supprimez pas une donnée si d'autres données dépendent d'elles (ex. un projet réfère une communauté à supprimer ou une compagnie parente ou enfant ou un projet réfère une compagnie à supprimer). Retournez une exception à la place.
6. Lorsque vous lancez une exception, utilisez SeauSException. Indiquez un message d'erreur significatif dans votre exception.

## Labo6

Dans ce laboratoire, vous allez réusiné votre code du précédent laboratoire pour respecter l'architecture 3-tiers.

### Objectif

Ce laboratoire a comme objectifs de :

1. Écrire un logiciel de connexion à une base de données selon l'architecture 3-tiers.

### Tâches

À partir de votre solution pour le laboratoire 5, vous allez déplacer les fonctions se trouvant dans SeauS.java dans les bons gestionnaires. Voici en détails ce que vous allez faire :

#### *Création des gestionnaires de transactions – Couche traitement*

1. Créez un sous-package nommé « gestion » dans le package SeauS.
2. Dans ce package, créez 3 nouvelles classes : GestionCompagnie, GestionCommunaute et GestionProjet.

3. Déplacez maintenant les fonctions dans les bonnes classes selon leurs concepts. Déplacez les fonctions « listerProjetsCompagnie » dans GestionCompagnie et « listerProjetsCommunaute » dans GestionCommunauté. Déplacez les fonctions gérant les parents dans GestionCompagnie.

### *Création des gestionnaires de tables – Couche données*

Vous avez maintenant vos classes de gestionnaires. Cependant, ces classes ne font toujours pas abstraction du système de base de données utilisé. Il faudrait maintenant réusiner chacune des fonctions afin de déplacer l'accès aux données dans des gestionnaires de tables.

1. Créez un sous-package nommé « tables » dans le package SeauS.
2. Dans ce package, créez 4 nouvelles classes : Compagnies, Communautes, Projets et Parents (AU PLURIEL).
3. Déplacez les PreparedStatement de la classe principale vers chacune de ces classes. Les Prepared Statement devraient être déclarés comme des propriétés de classe privées et initialisées dans le constructeur.
4. Justement, pour chacune des classes, créez un constructeur qui prendra en paramètre un objet Connexion. Chaque gestionnaire de table aura une propriété Connexion qui sera initialisée dans le constructeur. Si ce n'est pas déjà fait, initialisez vos PreparedStatement dans ce constructeur (avec leur SQL).
5. Pour chacun des statement, il y aura une méthode publique dans la classe qui pourra être appelée :
    a. Les statements de type « select » qui retourne un seul résultat auront 2 méthodes : une méthode « existe » et une méthode « getX » (ex. getCompagnie). La méthode « existe » retourne un booléen, et la méthode « get » va retourner un tuple (nous y reviendrons). Ils prendront en paramètre le nom ou l'id, selon (ou la paire d'id pour la table Parent).
    b. Les statement de type « insert » auront une méthode « ajouterX » (ex. ajouterCompagnie). Elles prendront en paramètres un tuple à ajouter (ex. une Compagnie pour Compagnies, nous y reviendrons). Elles retourneront un int représentant le nombre de lignes modifiées dans la base de données.
    c. Les statement de type « update » auront une méthode « editerX » (ex. editerCompagnie). Elles prendront un tuple comprenant les valeurs à modifier et l'identifiant de l'item à modifier (ex. Compagnie c, String

nomActuel pour Compagnies). Elles retourneront un int représentant le nombre de lignes modifiées dans la base de données.

d. Les statement de type « delete » auront une méthode « supprimerX » (ex. supprimerCompagnie). Elles prendront en paramètres l'identifiant de la valeur à supprimer (ex. String nom pour une Compagnie ou une Communauté). Elles retourneront un int représentant le nombre de lignes modifiées dans la base de données.
e. Les statements de type « select » qui retournent plusieurs tuples (ex. « selectProjetsCompagnie ») auront une méthode « getProjetsX » (ou getParents). Elles prendront en paramètre l'identifiant de la compagnie ou de la communauté et retourneront une List<Projet> ou une List<Compagnie>, selon, où Projet et Compagnie sont des tuples (nous y arriverons). Les méthodes getProjetsX devraient se retrouver dans la classe Projets, tandis que getParents devraient se trouver dans la classe Parents.

6. Dans tous les cas, les paramètres que prendront ces méthodes devraient être les mêmes que les paramètres de vos statements.
7. Tous les appels à executeQuery ou executeUpdate, ainsi que le parcours des ResultSet (s'il y a lieu) vont se faire maintenant dans ces méthodes. Déplacez la logique correspondante au bon endroit. Appelez maintenant ces méthodes dans vos gestionnaires de transaction.
8. Référez tous les gestionnaires de tables utilisés dans vos gestionnaires de transactions via une propriété. Initialisez ces propriétés dans le constructeur. Ajoutez également une propriété « Connexion » qui sera initialisée avec la Connexion d'un des gestionnaires de tables reçus en paramètre dans le constructeur (ce sera la même Connexion pour tous). Faites un constructeur qui recevra en paramètres les gestionnaires de tables utilisés. Ex. pour GestionCompagnie, vous devriez recevoir en paramètre les objets suivants : Compagnies, Parents et Projets (pour listerProjetsCompagnie).

Vous devriez maintenant avoir réussi à séparer l'accès aux données des logiques des transactions. Attention! Les try, catch, les vérifications (et s'il y a lieu, l'envoi d'exception), les appels à commit et rollback devraient DEMEURER dans vos méthodes de gestionnaires de transactions.

### *Création des classes de tuples de données – Couche données*

La dernière étape consiste à avoir des classes pour représenter les tuples retournés par la base de données. Ces classes simples visent seulement à rendre le code plus

propre pour éviter de traîner les paramètres représentant toutes les colonnes d'un tuple. Elles vont s'apparenter à des struct en C++.

1. Créez un sous-package « tuples » dans le package SeauS.
2. Dans ce package, créez les classes « Compagnie », « Communaute » et « Projet » (AU SINGULIER).
3. Dans ces classes, vous allez créez un champ pour chacune des colonnes de la base de données. Ex. La classe Compagnie aura les champs « idCompagnie », « nomCompagnie » et « adresse ». Utilisez des types appropriés et représentatifs de ce qu'il y a dans la base de données (dans l'exemple précédent, ce serait int, String, String).
4. Pour éviter d'avoir des « getters » pour chacun de ces champs et alourdir la lecture du code, vous pouvez mettre tous ces champs publics.
5. Créez maintenant un constructeur prenant en paramètre une valeur pour tous ces champs. Vous pouvez aussi créez avec tous les champs sauf l'id. Ça peut vous être utile pour certaines fonctions.
6. Référez ces nouvelles classes aux bons endroits dans vos classes précédentes des 2 autres packages.

*Finition*

Il est maintenant temps de terminer le réusinage pour faire fonctionner le tout. Pour cette partie, vous pouvez vous aider du code disponible ici : UdeS-Charges-maym2104/ift287-labo6-seaus: Code de base pour le labo 6 (architecture 3-tiers) avec la mise en situation SeauS.

1. Clonez le repo précédent.
2. Dans votre solution, remplacez votre classe SeauS.java par celle du repo git.
3. Téléchargez également la classe GestionSeauS que vous placerez dans le package SeauS.
4. Créez un sous-package bdd. Placez-y la classe Connexion.
5. Téléchargez et téléversez les classes GestionTransactions dans gestion, GestionTables dans tables et l'interface IConnexion dans bdd. Faites héritez vos gestionnaires de transactions de GestionTransactions, vos gestionnaires de Tables de GestionTables et la classe Connexion de IConnexion. Enlevez la déclaration et l'initialisation de la Connexion dans vos classes et appelez le constructeur super dans vos constructeurs. L'initialisation de la Connexion sera maintenant gérée par les classes abstraites.

En principe, après ces modifications, votre solution devrait fonctionner telle qu'avant, à la différence qu'elle est maintenant modulée selon l'architecture 3-tiers. Vous pouvez testez votre solution.

Pour la remise sur turnin :

1. Zippez votre code dans un fichier appelé Labo6.zip.
2. Remettez-le sur turnin.
3. Attention! Je vais remplacer les classes SeauS.java, GestionSeauS.java, Connexion.java et SeauSException.java par mes solutions (celles disponibles sur git) afin de valider que vous avez bien utilisé l'architecture 3-tiers et non seulement remis la solution du Labo5. Ne changez donc pas ces classes car vos changements ne seront pas pris en compte!

Les instructions de ce laboratoire sont longues, mais vous n'avez pas de nouveau code à écrire. Ce laboratoire devrait donc bien se faire si vous suivez les instructions telles qu'écrites.

En cas de doute, je laisserai aussi sur le repo git des exemples de la solution finale sur laquelle vous pouvez vous baser. Je vous conseille également de consulter l'exemple de la Bibliothèque disponible sur Moodle.

Bon succès!