

SPRINT 3

Table des matières

Exposé des objectifs du sprint 3 :	3
Mission 1 – Importation des objets stockés sur des fichiers CSV :	3
A propos du format CSV :	3
Description de la structure :	3
Pourquoi le format CSV ? :	4
Pour la culture – l’open data :	4
Exemple de cas d’usage :	4
Etape préalable : Récupération du projet pour le Sprint 3 :	6
Exemple de mise en œuvre - Les objets de la classe Qualite de KotlinAventure :	6
Mise en forme du fichier CSV :	6
Aménagement du code existant :	10
Travail et préparation des tâches de la mission 1 :	11
Répartition des tâches :	11
Quelques précisions complémentaires sur le travail à réaliser (notamment pour les missions 1.1 et 1.2) :	11
Intermission 1 :	12
Mission 2 – Mise en place de la base de données :	13
Travail d’avant mission – Installation & configuration de MariaDB :	13
Configuration du serveur & client :	13
Installation du client mysql-client ou mariadb-client :	13
Vérification de la connexion d’IntelliJ et de notre application à la base de données :	14
Préparation de la base de données :	14
Création des tables dans la base de données :	15
Travail et réparation des tâches de la mission 2.A-Construction de la base de données :	16
Répartition des tâches :	16
Visualiser votre base de données avec IntelliJ :	16
Préambule à la mission 2B -Explication de la configuration pour connecter l’application à la BD :	18
Installation du client :	18
L’API JDBC :	19
TRAVAIL A FAIRE SUR LA CLASSE BDD :	20
Pour en finir sur la classe BDD :	21
Mission 2B – Création des DAO :	22
Analyse de la classe QualiteDAO :	22
Analyse de la méthode findById() :	24
Retour sur la coordination de nos DAO avec la logique de l’application :	27

Travail à faire :	27
Vérification du résultat :	28
Travail à réaliser pour la mission 2B :	28
Mission 3-Rechercher & implémenter une solution anti-redondance :	29
Analyse du problème :	29
Travail à réaliser :	29
Mission 4 – Exportation du contenu des tables vers des CSV :	29
Exposé du besoin :	29
Travail à réaliser :	29
Annexe	30
Pourquoi et quand procéder :	30
Sauvegarde de votre base de données :	30
Sauvegarde manuelle :	30
Script de sauvegarde :	31
Restauration de votre base de données :	31

Exposé des objectifs du sprint 3 :

Dans ce sprint nous allons mettre en œuvre différents de moyen de sauvegarder des données :

- **Mission 1** : Externalisation sur des fichiers **CSV** des objets de l'univers **KotlinAdventure** dans un objectif d'import des objets durant une session de jeu.
- **Mission 2** : Mise en place d'une base de données nous permettant de stocker de façon structurée les objets issus de l'import des fichiers **CSV** (créer les **DAO**).
- **Mission 3** : Implémenter une stratégie permettant d'éviter l'enregistrement multiples des objets importés.
- **Mission 4** : Mise en place d'une solution d'exportation des données de la base de données.

Mission 1 – Importation des objets stockés sur des fichiers CSV :

Nous désirons rendre extensible notre jeu en important les objets de l'univers **KotlinAdventure** avec un ensemble de fichiers où chaque fichier correspond à une classe d'objets.

En pratique, le code de notre application contient les classes, mais les instances de ces classes sont réalisées à partir des informations stockées dans un fichier (au lieu d'avoir ces informations nécessaires aux instances des classes dans le fichier **main.kt**).

Nous allons retrouver sur chaque ligne du fichier l'ensemble des données nécessaires aux instances de la classe concernée.

Nous avons opté pour le format **CSV** pour réaliser cette fonctionnalité.

A propos du format CSV :

Le format **CSV** pour **Comma Separated Values (Valeurs Séparées par des Virgules)**. C'est un format de fichier tabulaire au format texte, il est donc lisible et éditable avec n'importe quel éditeur de texte.

Description de la structure :

- Chaque ligne représente un enregistrement, ce qui peut correspondre à une ligne d'une table d'une base de données ou d'une ligne d'une feuille de calcul.
- Chaque valeur est séparée des autres par un symbole dédié que l'on nomme tout simplement le séparateur. Historiquement c'est la virgule (,) qui fait standard, mais d'autres séparateurs peuvent être choisis comme notamment le point-virgule (;).
- Chaque attribut, à une position identique dans chaque ligne, ce qui permet de retrouver la notion de colonne.

Exemple de structure d'un fichier CSV décrivant des personnes :

Nom, Prénom, Âge
Dupont, Jean, 45
Durand, Marie, 34

Eventuellement la 1^{ère} ligne peut être une ligne dédiée à la description du contenu des lignes/colonnes (comme sur l'exemple présenté ci-dessus).

Pourquoi le format CSV ? :

Le format **CSV** offre des avantages à condition que la structure des données à conserver ne soit pas trop complexe.

Avantages :

- **Simplicité** : La structure est explicite surtout lorsqu'elle est accompagnée d'une 1^{ère} ligne désignant les différents champs/attributs. Le format texte rend la lecture/modification à partir d'un simple éditeur de texte.
- **Compatibilité** : De nombreux logiciels supportent le format **CSV** aussi bien pour l'import que l'export de données. Des options permettent d'adapter le format (choix du standard d'encodage du texte ; présence éventuelle d'une ligne de description des champs, choix du séparateur, etc.).
- **Légèreté** : Le format ne contient pas de superflus avec sa structuration minimaliste.

Pour la culture – l'open data :

L'open data est un mouvement initié depuis une dizaine d'années et qui consiste à l'accès à tous de données en masse. Ces données anonymes sont généralement mises à disposition par des institutions d'états.

Les objectifs de cette démarche sont de favoriser l'innovation en permettant à des chercheurs de réaliser des études, à des entreprises de créer de la valeur ajoutée à partir de ces données.

L'un des principaux défis est de faciliter l'échange des données, les principaux standards utilisés sont généralement :

- Le format **CSV** (*Comma-Separated Values*)
- Le format **XML** (*eXtensible Markup Language*)
- Le format **JSON** (*JavaScript Object Notation*)

Remarque : Nous aurons l'occasion d'utiliser le format **JSON** dans le cadre d'exploitation d'API.

En France les données sont collectées et mis à disposition par l'agence : **data.gouv.fr** dont le site est disponible à l'URL : <https://www.data.gouv.fr/fr/>

Exemple de cas d'usage :

Je désire accéder la base officielle des codes postaux en France, le site **data.gouv.fr** assure la collecte de la base officielle auprès de **La Poste**. La base est accessible sur l'URL : <https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/>

The screenshot shows the 'data.gouv.fr' website interface. At the top, there's a navigation bar with links like 'Se connecter', 'S'enregistrer', and 'Donnez votre avis'. Below this is a search bar with the text 'Recherche'. The main navigation menu includes 'Données', 'Réutilisations', 'Organisations', 'Commencer sur data.gouv.fr', 'Actualités', and 'Nous contacter'. The breadcrumb trail reads 'Accueil > jeux de données > Base officielle des codes postaux'. The dataset title is 'Base officielle des codes postaux'. The description states: 'La base officielle des codes postaux est un jeu de données qui fournit la correspondance entre les codes postaux et les codes INSEE des communes, de France (métropole et DOM), des TOM, ainsi que de MONACO. Ce jeu de données comprend :'. A list of data points follows: 'Le nom de la commune', 'Le code INSEE de la commune', 'La ligne 5 de l'adresse postale : précision du lieu-dit associé ou du nom de la commune déléguée associée', 'Le code postal de la commune', and 'Le libellé d'acheminement'. On the right, it identifies the producer as 'La Poste', notes the data source as 'Ce jeu de données provient d'un portail externe', and provides a link to 'Voir la source originale'. It also shows the 'Dernière mise à jour' as '5 octobre 2023' and the 'Licence' as 'Licence Ouverte / Open licence version 2.0'. At the bottom right, there's a 'Qualité des métadonnées' section with a green progress bar and a warning 'Couverture temporelle non renseignée'.

Vous aurez alors le choix :

- D'exploiter l'**API** mis à disposition avec sa documentation :

API publique du jeu de données : Base officielle des codes postaux

Cette documentation interactive à destination des développeurs permet de consommer les ressources du jeu de données "Base officielle des codes postaux".

Pour protéger l'infrastructure de publication de données, les appels sont limités par quelques règles simples :

- Un utilisateur anonyme ne peut pas effectuer plus de 100 requêtes par interval de 5 secondes. Sa vitesse de téléchargement totale sera limitée à 2 MB/s pour les contenus statiques (fichiers de données, pièces jointes, etc.) et à 500 kB/s pour les autres appels.
- Un utilisateur authentifié (session ou clé d'API) ne peut pas effectuer plus de 100 requêtes par interval de 1 seconde. Sa vitesse de téléchargement totale sera limitée à 4 MB/s pour les contenus statiques (fichiers de données, pièces jointes, etc.) et à 1 MB/s pour les autres appels.

Métadonnées

Données

Select an entry on the left to see detailed information...

- De télécharger au format **CSV** le contenu de la base des codes postaux :

Base officielle des codes postaux

39 193 lignes

Rechercher

Nom de la commune

L ABERGEMENT CLEMENCIAT	
L ABERGEMENT DE VAREY	
AMBERIEU EN BUGÉY	
AMBERIEUX EN DOMBES	
AMBLEON	
AMBRONAY	
AMBUTRIX	01500
ANDERT ET CONDON	01300
ANGLEFORT	01350
APREMONT	01100

Ces téléchargements tiennent compte du tri et de la recherche

format CSV

Les formats suivants sont limités aux 10 000 premières lignes

format XLSX

format ODS

format GeoJSON

Une fois le fichier téléchargé, on peut explorer son contenu en l'ouvrant par exemple avec **Visual Studio Code** comme ci-dessous :

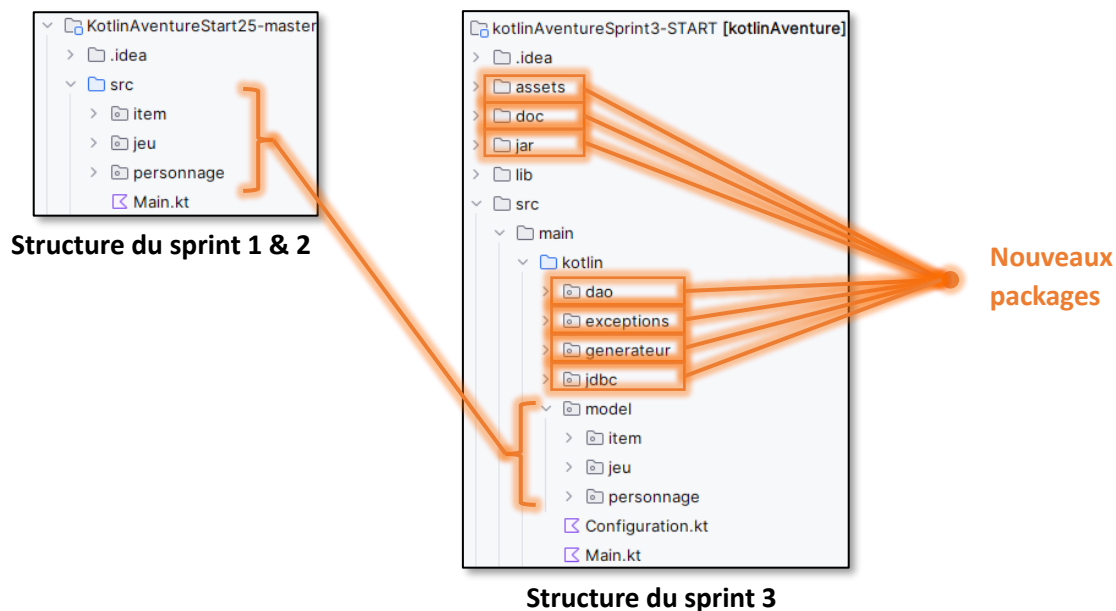
```
laposte-hexasmal.csv - Visua...
Le mode restreint est destiné à la navigation de cod...
Gérer En savoir plus
laposte-hexasmal.csv
D: > HOME > ljules > Desktop > laposte-hexasmal.csv
1 "Nom_de_la_commune","Code_postal"
2 "L ABERGEMENT CLEMENCIAT","01400"
3 "L ABERGEMENT DE VAREY","01640"
4 "AMBERIEU EN BUGÉY","01500"
5 "AMBERIEUX EN DOMBES","01330"
6 "AMBLEON","01300"
7 "AMBRONAY","01500"
8 "AMBUTRIX","01500"
9 "ANDERT ET CONDON","01300"
10 "ANGLEFORT","01350"
```

Etape préalable : Récupération du projet pour le Sprint 3 :

Afin de partir sur de bonnes bases, nous vous proposons de partir sur un projet qui prend la suite du **sprint 2** mais avec une structuration plus poussée des packages.

Adresse du dépôt de départ sur **GitHub** : <https://github.com/ljules/kotlinAventure-Sprint3-Start25>

Comparaison de la structure du projet entre le **sprint 2** et **sprint 3** :



Nous vous expliquerons et exploiterons ces nouveaux packages dans les pages suivantes.

Exemple de mise en œuvre- Les objets de la classe **Qualite** de KotlinAventure :

Avant de vous répartir le travail nécessaire pour importer les principaux objets de notre application, nous vous proposons de vous expliquer un exemple d'implémentation pour les objets de la classe **Qualite**.

Mise en forme du fichier CSV :

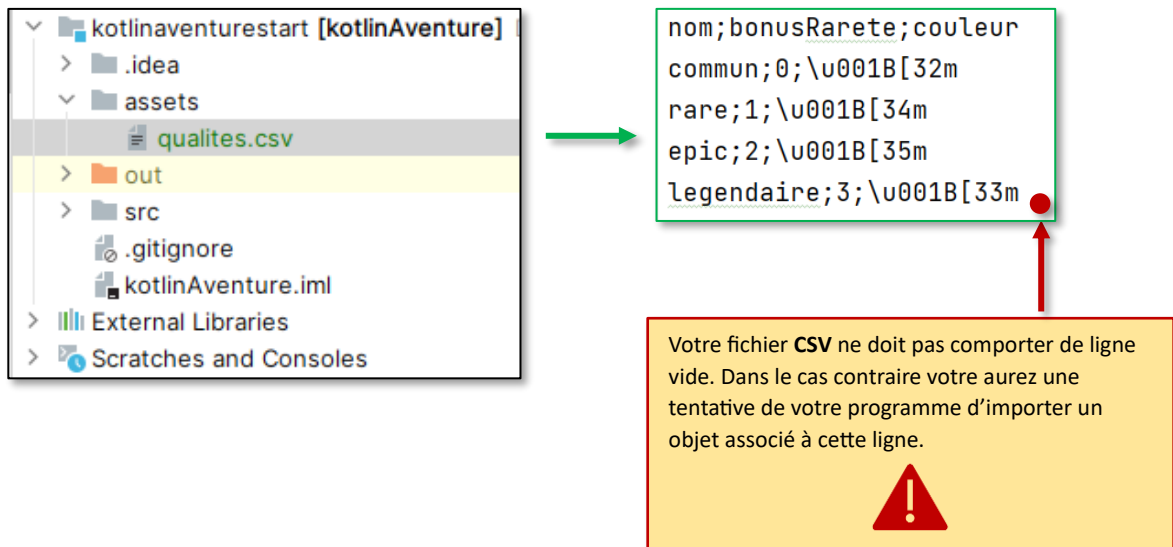
Avant tout, nous devons réaliser un **CSV** de nos objets **Qualite**. La classe nous donne les informations utiles pour réaliser cette tâche. En effet chaque ligne du **CSV** doit fournir les données pour valoriser nos objets importés :

```
class Qualite
    (val nom: String,
     val bonusRarete: Int,
     val couleur: String)
```

Nous avons donc 3 attributs :

- Le **nom**
- Le **bonusRarete**
- La **couleur**

Nous vous proposons de stocker nos **CSV** dans un dossier **assets** à la racine du dossier de notre projet **kotlinAventure**. Le fichier **qualites.csv** contient les objets de la classe **Qualite**.



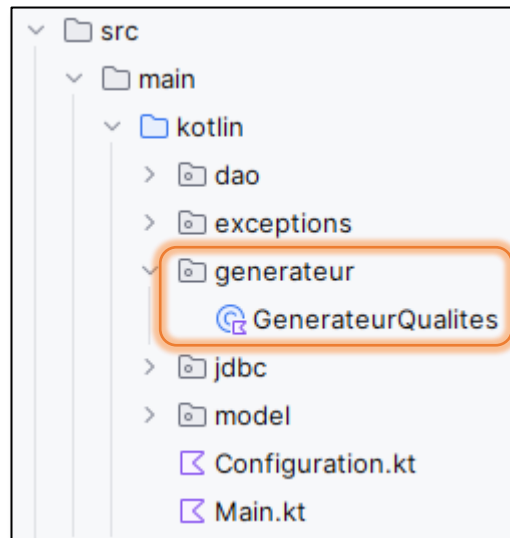
Pour assurer la génération d'objets à partir d'un fichier **CSV** correspondant, nous allons créer une classe dédiée et plus particulièrement une méthode permettant :

- De lire le fichier **CSV**
- De générer une instance à partir des données collectées dans le **CSV**.

La méthode de génération que nous désignerons **generer()** et qui va retourner l'ensemble des objets dans une collection de type **Map**.

Le **Map** retourné contient un ensemble de paires **clés/valeurs** où les clés sont des **Strings** ayant pour valeurs les **noms** des objets et pour valeurs les références des objets.

Nous avons créé plusieurs packages pour structurer notre projet, notamment le package **generateur** dans lequel nous placerons tous nos générateurs sous forme de classes : **GenerateurQualites** ; **GenerateurTypeArmures** ; **GenerateurArmures**, **GenerateurTypeArmes** ; **GenerateurArmes** ; **GenerateurPotions** et **GenerateurBombes** :



Nous vous donnons pour exemple l'implémentation de la classe **GenerateurQualites** :

```
class GenerateurQualites(val cheminFichier: String) {  
    new *  
    fun generer(): MutableMap<String, Qualite> {  
        val mapObjets = mutableMapOf<String, Qualite>()  
  
        // Lecture du fichier CSV, le contenu du fichier est stocké dans une liste mutable :  
        val cheminCSV = Paths.get(this.cheminFichier)  
        val listeObjCSV = Files.readAllLines(cheminCSV)  
  
        // Instance des objets :  
        for (i in 1 ≤ .. ≤ listeObjCSV.lastIndex) {  
            val ligneObjet = listeObjCSV[i].split( ...delimiters: ";")  
            val cle = ligneObjet[0]  
            val objet = Qualite(ligneObjet[0], ligneObjet[1].toInt(), ligneObjet[2])  
            mapObjets[cle] = objet  
        }  
        return mapObjets  
    }  
}
```


Description de la classe **GenerateurQualites** et lien avec le fichier CSV associé :

```
class GenerateurQualites(val cheminFichier: String) {
    @ L. JULES
    fun generer(): MutableMap<String, Qualite> {
        val mapObjets = mutableMapOf<String, Qualite>()

        // Lecture du fichier CSV, le contenu du fichier est stocké dans une liste mutable :
        val cheminCSV = Paths.get(this.cheminFichier)
        val listeObjCSV = Files.readAllLines(cheminCSV)

        // Instance des objets :
        for (i in 1 ≤ .. ≤ listeObjCSV.lastIndex) {
            val ligneObjet = listeObjCSV[i].split( ...delimiters: ";")
            val cle = ligneObjet[0]
            val objet = Qualite(nom = ligneObjet[0], bonusRarete = ligneObjet[1].toInt(), couleur = ligneObjet[2])
            mapObjets[cle] = objet
        }
        return mapObjets
    }
}
```

nom	bonusRarete	couleur
commun	0	\u001B[32m
rare	1	\u001B[34m
epic	2	\u001B[35m
legendaire	3	\u001B[33m

```
{commun=model.item.Qualite@6eceb130,
rare=model.item.Qualite@10a035a0,
epic=model.item.Qualite@67b467e9,
legendaire=model.item.Qualite@47db50c5}
```

Clés : nom

Valeurs : objets Qualite

Une fois la classe **GenerateurQualites** implémentée, nous pouvons l'exploiter dans le fichier **main()** pour générer un map d'objets **qualites**.

```
//DEMO MISSION 1
val generateurQualites = GenerateurQualites( cheminFichier: "assets/qualites.csv")
val qualites = generateurQualites.generer()
```

Aménagement du code existant :

Précédemment (sprint 1 et 2) notre programme contenait 4 objets de type/classe **Qualite** :

```
// Instances de la classe Qualite :  
val qualiteCommun = Qualite( nom: "commun", bonusRarete: 0, couleur: "\u001B[32m")  
val qualiteRare = Qualite( nom: "rare", bonusRarete: 1, couleur = "\u001B[34m")  
val qualiteEpic = Qualite( nom: "epic", bonusRarete: 2, couleur: "\u001B[35m")  
val qualiteLegendaire = Qualite( nom: "legendaire", bonusRarete: 3, couleur: "\u001B[33m")
```

Dans la version du **sprint 3** les instances précédentes ont été mises en commentaire, car nous allons reporter les informations relatives à ces objets (*nom*, *bonusRarete* et *couleur*) dans un fichier de type **CSV**.

```
//instanciation des qualités des objets  
//val qualiteCommun = Qualite(nom="commun", bonusRarete = 0, couleur = "\u001B[32m")  
//val qualiteRare = Qualite(nom="rare", bonusRarete = 1, couleur = "\u001B[34m")  
//val qualiteEpic = Qualite(nom = "epic", bonusRarete = 2, couleur = "\u001B[35m")  
//val qualiteLegendaire = Qualite(nom = "legendaire", bonusRarete = 3, couleur = "\u001B[33m")
```

Mais il faut également remplacer l'ensemble des instances précédentes par leur équivalent présent dans le map **qualites**. Nous avons donc apporté les modifications suivantes :

Modifications dans le fichier **main** :

```
85 val sortInvocatinArme = Sort( nom: "Sort d'invocation d'arme magique") { caster, cible ->  
86     run {  
87         val tirageDes = TirageDes( nbDe: 1, maxDe: 20)  
88         var resultat = tirageDes.lance()  
89         val qualite = when {  
90             resultat < 10 -> qualites["rare"]  
91             resultat < 15 -> qualites["epic"]  
92             resultat <= 20 -> qualites["legendaire"]  
93             else -> qualites["commun"]  
94         }  
95     }
```

```
103 val sortInvocatinArmure = Sort( nom: "Sort d'invocation d'armure magique") { caster, cible ->  
104     run {  
105         val tirageDes = TirageDes( nbDe: 1, maxDe: 20)  
106         var resultat = tirageDes.lance()  
107         val qualite = when {  
108             resultat < 10 -> qualites["rare"]  
109             resultat < 15 -> qualites["epic"]  
110             resultat <= 20 -> qualites["legendaire"]  
111             else -> qualites["commun"]  
112         }  
113     }
```

```
122 fun main() {  
123     //instanciation des armes des monstres  
124     val epee = Arme( nom: "Épée Courte", description: "Une épée courte tranchante", typeEpeeCourte, qualites["commun"]!!)  
125     val lance = Arme( nom: "Lance", description: "Une lance pointue", typeLance, qualites["rare"]!!)  
126     val dague = Arme( nom: "Dague", description: "Une dague extrêmement pointue", typeDague, qualites["epic"]!!)  
127     val marteau = Arme( nom: "Marteau", description: "un marteau legendaire pourfendeur de troll", typeMarteau, qualites["legendaire"]!!)
```

Modifications dans le fichier **jeu.kt** :

```
148 val epee = Arme( nom: "Épée Longue", description: "Une épée longue tranchante", typeEpeeLongue, qualites["commun"]!!)  
149 val potionDeSoin = Potion( nom: "Grande Potion de Soin", description: "Restaure les points de vie", soin: 30)
```

```
130 perso = Guerrier(leNom, pv, pvMax, scoreAttaque, scoreDefense, scoreEndurance, scoreVitesse)  
131 val maDague = Arme( nom: "Dague", description: "Une dague pointue", typeDague, qualites["commun"]!!)
```

Travail et préparation des tâches de la mission 1 :

En s'inspirant des éléments exposés précédemment pour l'implémentation de la solution de persistance des objets de classe **qualites**, chaque membre de l'équipe a pour charge l'implémentation de la persistance pour une catégorie d'objets. Pour chaque objet, vous devrez reconduire la démarche suivante :

1. La réalisation d'un ou plusieurs **CSV** conforme(s) afin de permettre l'importation des objets de sa/ses classe(s) associée(s).
2. Implémentation de la/les classe(s) **Generateur** et de sa méthode **generer()** pour chaque **CSV**.
3. Apporter les modifications nécessaires au reste de votre code (notamment le fichier **main.kt**) pour que votre implémentation soit effective (instanciation et valorisation des objets avec les données du/des **CSV**).

Répartition des tâches :

Etudiant	Mission	Tâches associées
Etudiant 1	Mission 1.1	Gestion des objets des classes : TypeArme + Arme
Etudiant 2	Mission 1.2	Gestion des objets des classes : TypeArmure + Armure
Etudiant 3	Mission 1.3	Gestion des objets des classes : Potion + Bombe

Quelques précisions complémentaires sur le travail à réaliser (notamment pour les missions 1.1 et 1.2) :

Les constructeurs des classes **Arme** et **Armure** réclament en arguments 2 objets :

- 1 objet de la classe **Qualite**
- 1 objet de la classe **TypeArme** pour **Arme** et de la classe **TypeArmure** pour **Armure**

```
class Arme(  
    nom: String,  
    description: String,  
    val type: TypeArme,  
    val qualite: Qualite  
) : Item(nom, description)
```

```
class Armure(  
    nom:String,  
    description:String,  
    val typeArmure: TypeArmure,  
    val qualite: Qualite  
): Item(nom,description) {
```

Il y a donc une dépendance de la classe **Arme** et de ses objets vis-à-vis à la classe **Qualite** et **TypeArme** et leurs objets.

De même pour la classe **Armure** qui a une relation de dépendance vis-à-vis de la classe **Qualite** et **Armure**.

Conséquence : Il est fortement conseillé d'implémenter les générateurs des objets **TypeArme** et **Qualite** avant d'implémenter le générateur de la classe **Arme**. De même pour le générateur des objets **Armure** qui sera implémentée après les générateurs des objets **TypeArmure** et **Qualite**.

Les générateurs produisent des objets, ces objets seront inscrits dans des **maps** (tableaux associatifs) qui auront pour clé la propriété/champs **nom** (inscrit dans le **CSV** et qui sera la valeur de l'attribut nom de l'objet instancié), la valeur associée est l'objet lui-même (ou plus exactement sa référence).

Intermission 1 :

A partir du travail réalisé dans les différentes mission **1.1** ; **1.2** et **1.3** implémenter une solution permettant l'importation de personnages **Monstres** afin d'enrichir l'univers de votre jeu.

Remarques :

- Attention cette intermission nécessite que toutes les missions 1.1 ; 1.2 et 1.3 soient réalisées. Dans le cas contraire vous ne pourrez pas la mener à sa fin.
- Vous utiliserez les 2 adversaires/personnages déjà implémentés dans le **main.kt** pour créer votre fichier **montres.csv**.
- Optionnellement vous pourrez créer une classe spécifique **Monstre** qui hérite de la classe **Personnage**.

Mission 2 – Mise en place de la base de données :

Travail d'avant mission – Installation & configuration de MariaDB :

CE TRAVAIL PREPARATOIRE EST A REALISER PAR CHAQUE MEMBRE DE L'EQUIPE SUR SON PC DE TRAVAIL

Configuration du serveur & client :

Pour pouvoir travailler avec une base de données nous allons avoir besoin des éléments suivants :

- **Côté serveur** : Installer une base de données, nous le ferons avec un serveur mis à votre disposition sur lequel est installé la solution **MariaDB**.
- **Côté client** : Notre application pour accéder à la base de données aura besoin d'un **connecteur/driver** approprié.

Remarque :

Dans le cadre de notre projet nous allons installer le client et le serveur sur la même machine. Mais rien ne nous empêchera par la suite d'installer et d'exécuter notre serveur sur une autre machine de notre réseau local.

Installation du client mysql-client ou mariadb-client :

Votre poste dispose peut-être déjà d'un client **MySQL/MariaDB**, nous allons vérifier cela :

- Ouvrir un terminal avec la combinaison de touches **[CTRL] + [ALT] + [T]**
- Saisir la commande : **mysql -V**
 - Résultat si absence du client **mysql** :

```
ljules@Vvm-ubu22:~$ mysql -v
La commande « mysql » n'a pas été trouvée, mais peut être installée avec :
sudo apt install mysql-client-core-8.0      # version 8.0.39-0ubuntu0.22.04.1, or
sudo apt install mariadb-client-core-10.6   # version 1:10.6.18-0ubuntu0.22.04.1
ljules@Vvm-ubu22:~$
```

- Résultat si présence du client **mysql** :

```
ljules@Vvm-ubu22:~$ mysql -V
mysql Ver 8.0.39-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
ljules@Vvm-ubu22:~$
```

Si le client n'est pas installé, procéder alors à l'installation avec le gestionnaire **APT** :

- **sudo apt update**
- **sudo apt install mysql-client**

```
ljules@Vvm-ubu22:~$ sudo apt update
[sudo] Mot de passe de ljules :
Atteint :1 http://fr.archive.ubuntu.com/ubuntu jammy InRelease
Atteint :2 http://fr.archive.ubuntu.com/ubuntu jammy-updates InRelease
Atteint :3 http://fr.archive.ubuntu.com/ubuntu jammy-backports InRelease
Atteint :4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
390 paquets peuvent être mis à jour. Exécutez « apt list --upgradable » pour les voir.
ljules@Vvm-ubu22:~$ sudo apt install mysql-client
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
```

Vérification de la connexion d'IntelliJ et de notre application à la base de données :

CE TRAVAIL DOIT ETRE REALISE SUR LE POSTE DU RESPONSABLE « GIT » & sessions « CODE WITH ME »

Préparation de la base de données :

Se connecter à votre base de données avec le client **mysql** en utilisant l'un des comptes du groupe. Pour rappel votre compte et mot de passe est identique à celui que vous utilisez pour le compte Wifi ou pour l'accès aux postes Windows du lycée.

L'adresse du serveur **MariaDB** est : **172.16.0.210**

Voici la commande à saisir pour vous connecter :

- **mysql -u monLogin -p -h 172.16.0.210**

```
user@DEB-SRV-125:~$ mysql -u ntran -p -h 172.16.0.210
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 33
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> █
```

Une fois connecté, vous allez devoir créer une base de données pour votre équipe.

Créer la base de données de notre application en exécutant la requête **SQL** :

- **CREATE DATABASE db_kotlinAventure_monLogin;**

Remarque : Vous pouvez remplacer **monLogin** par votre login ou par le nom de votre groupe si vous en avez un.

Puis créons la table **qualite** en exécutant la requête **SQL** :

```
CREATE TABLE Qualite
(
    id            INT AUTO_INCREMENT,
    nom           VARCHAR(255),
    bonusRarete  INT,
    couleur      VARCHAR(50),
    PRIMARY KEY (id)
);
```

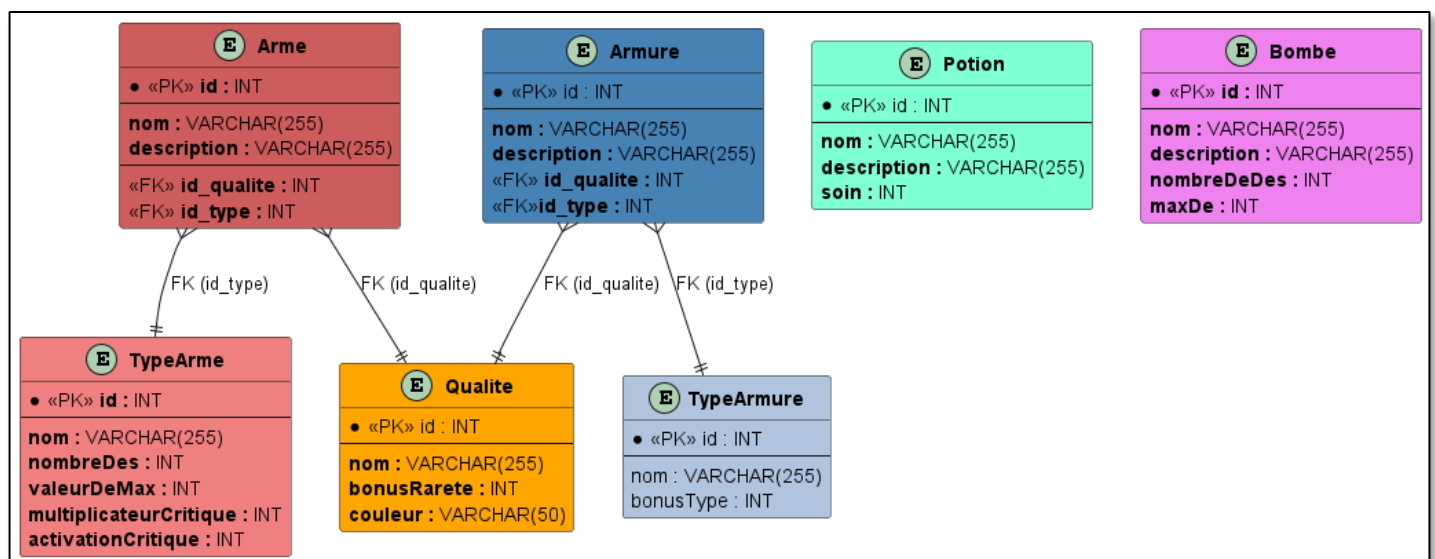
Création des tables dans la base de données :

Notre base de données peut être représentée par le schéma relationnel suivant :

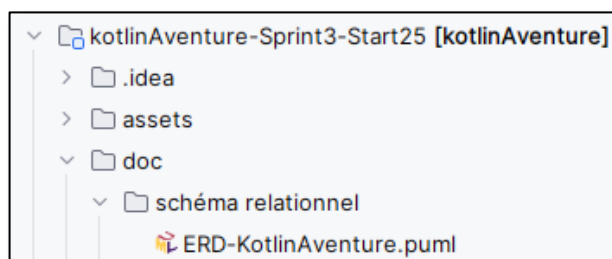
Schéma relationnel sous forme textuel :

- **Qualite**(#id: INT, nom: VARCHAR(255), bonusRarete: INT, couleur: VARCHAR(50))
- **TypeArme**(#id: INT, nom: VARCHAR(255), nombreDes: INT, valeurDeMax: INT, multiplicateurCritique: INT, activationCritique: INT)
- **TypeArmure**(#id: INT, nom: VARCHAR(255), bonusType: INT)
- **Arme**(#id: INT, nom: VARCHAR(255), description: VARCHAR(255), id_qualite => Qualite, id_type => Type)
- **Armure**(#id: INT, nom: VARCHAR(255), description: VARCHAR(255), id_qualite => Qualite, id_type => Type)
- **Potion**(#id: INT, nom: VARCHAR(255), description: VARCHAR(255), soin: INT)
- **Bombe**(#id: INT, nom: VARCHAR(255), description: VARCHAR(255), nombreDeDes: INT, maxDe: INT)

Schéma relationnel sous forme de diagramme Entité-Relation :



Remarque : Vous pouvez accéder au diagramme E-R qui contenu dans le dossier « **doc/schéma relationnel** »



Travail et réparation des tâches de la mission 2.A-Construction de la base de données :

Nous disposons maintenant de l'infrastructure nécessaire pour construire notre base de données selon le schéma relationnel donné ci-dessus.

Nous vous demandons de procéder à l'implémentation du schéma relationnel selon la répartition proposée ci-après.

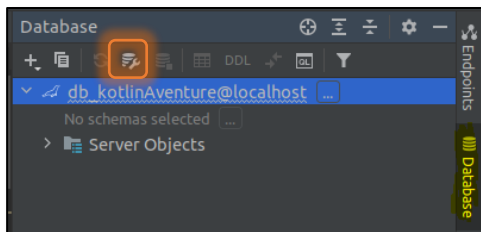
Répartition des tâches :

Etudiant	Mission	Tâches associées
Etudiant 1	Mission 2A.1	Création des tables : TypeArme et Arme
Etudiant 2	Mission 2A.2	Création des tables : TypeArmure et Armure
Etudiant 3	Mission 2A.3	Création des tables : Potion et Bombe

Visualiser votre base de données avec IntelliJ :

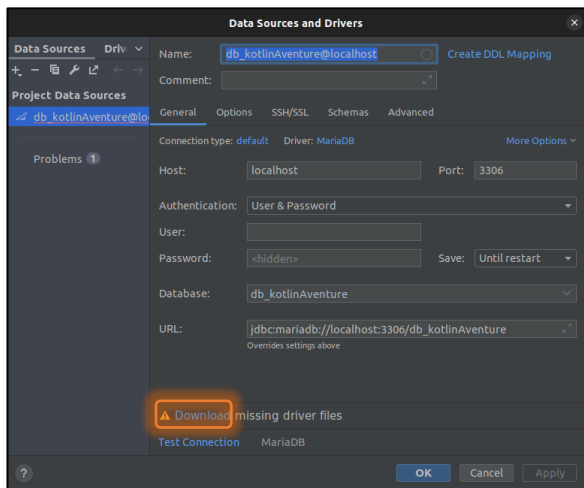
Ouvrir **IntelliJ** et le projet à son état de **Sprint III**.

Développer l'onglet **Database** (à droite de la fenêtre d'**IntelliJ**) :



Cliquer sur l'onglet le bouton de configuration :

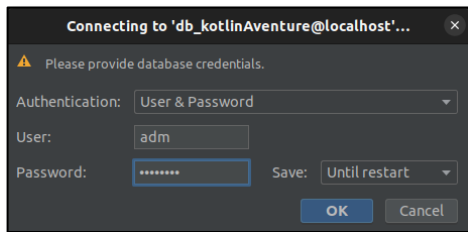
Généralement il manque un driver, téléchargez-le en cliquant le bouton **Download**.



Renseigner dans les champs idoines :

- L'adresse IP du serveur **MariaDB** dans le champ « Host »
- Votre login dans le champ « User »
- Votre mot de passe dans le champ « Password »
- Le nom de votre base de données dans le champ « Database »

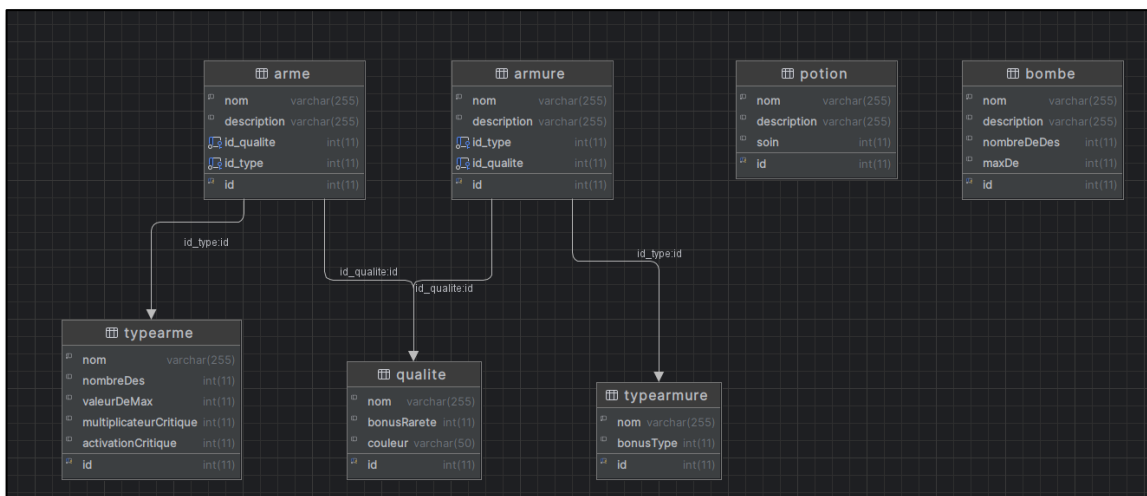
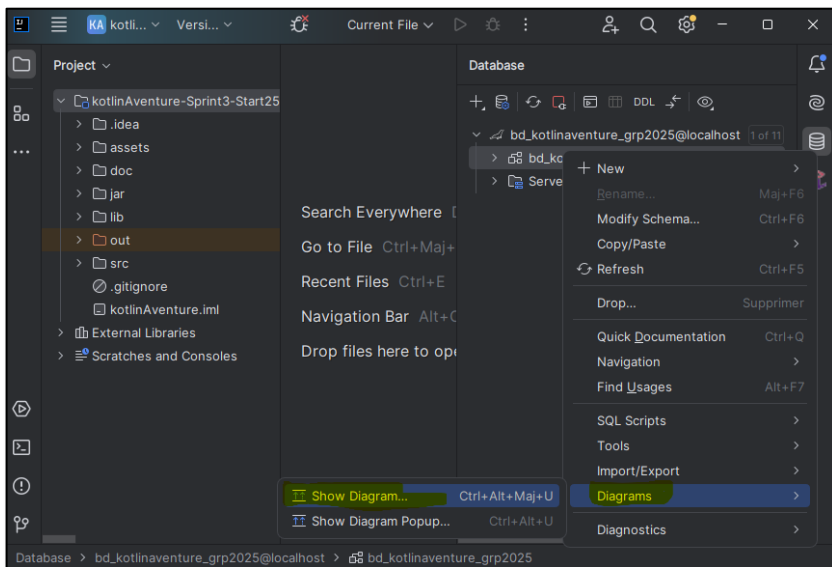
Ensuite cliquer sur **test connection** :



Si la connexion est opérationnelle vous aurez le message suivant :



Vous pouvez ensuite demander la génération du diagramme E-R en cliquant droit sur la connexion de votre base de données :



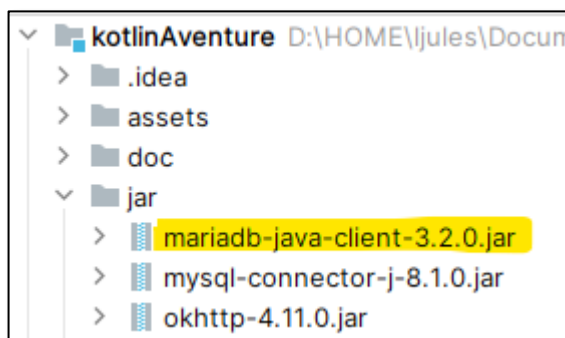
Préambule à la mission 2B-Explication de la configuration pour connecter l'application à la BD :

A ce stade votre base de données est opérationnelle pour recevoir les données d'instances/objets des classes métiers de notre application. Il faut maintenant préparer votre application à se connecter sur votre base de données en utilisant un **connecteur/driver**. Ce **connecteur/driver** ne doit pas être confondu avec la connexion que nous avons réalisé précédemment qui est en fait l'utilisation d'une application client intégré et proposé par **IntelliJ**. Notre application constitue elle-même un client pour notre base de données.

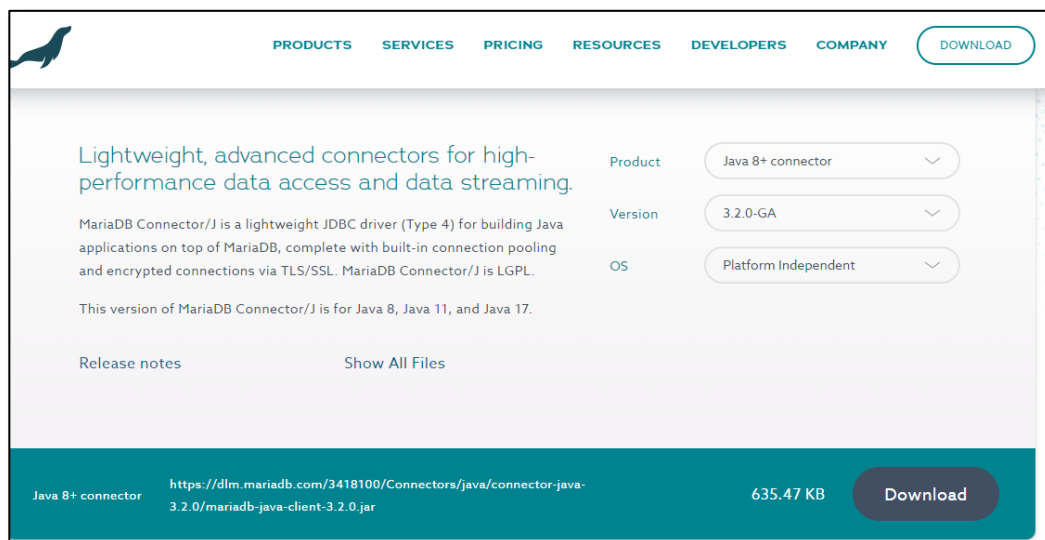
Installation du client :

Le terme client désigne l'application qui ira se connecter sur une base de données. Notre jeu **KotlinAventure** constitue donc un client. Cependant il ne peut dans l'état actuel se connecter à une base de données car il lui manque l'adjonction d'un **connecteur/driver**. Utilisant une base de données **MariaDB**, le **connecteur/driver** fourni par **MariaDB** est le plus adapté, de plus il offre l'avantage d'être également compatible avec le **SGBD MySQL** et de proposer le proposer au format **JAR** compatible à la fois pour les **OS Linux** et **Windows**.

Nous vous avons facilité le travail en intégrant le driver **MariaDB** dans l'arborescence du projet :



Pour information, les connecteurs/drivers officiels de **MariaDB** sont disponibles sur la page suivante : <https://mariadb.com/downloads/connectors/connectors-data-access/java8-connector>



Vous pourrez voir qu'il existe différentes versions de ce driver : **C** ; **Python** ; **Java** ; **NodeJS** ; etc.

Toujours pour information, vous pourrez trouver la documentation du driver/connecteur **Java** ici :

<https://mariadb.com/kb/en/about-mariadb-connector-j/>

Date	Release	Status	Min. Java Compat.	Release Notes	Changelog
25 Aug 2023	3.2.0	Stable (GA)	Java 8	Release Notes	Changelog
1 May 2023	3.1.4	Stable (GA)	Java 8	Release Notes	Changelog
25 Aug 2023	3.0.11	Stable (GA)	Java 8	Release Notes	Changelog
25 Aug 2023	2.7.10	Stable (GA)	Java 8	Release Notes	Changelog

Afin de bien initier la suite du travail, nous avons réalisé l'implémentation du code permettant d'assurer la connexion à la base de données.

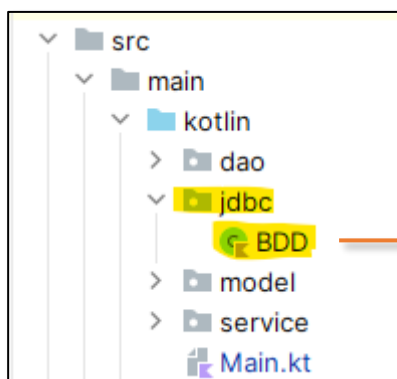
L'API JDBC :

Le **driver/connecteur** nous permet de nous connecter à la base de données mais nous ne pouvons pas dialoguer directement avec le **driver/connecteur**. **Java** nous offre une **API** dédiée à cet usage qui s'appelle **JDBC** (Java DataBase Connectivity). En pratique **JDBC** est une bibliothèque qui repose sur 4 classes principales qui vont nous offrir les méthodes pour interagir avec la base de données :

- **DriverManager** : Classe prenant en charge et la configuration du driver/connecteur ;
- **Connection** : Elle prend en charge la connexion et l'authentification à la base de données ;
- **Statement** : Contient la requête **SQL** et la transmet à la base de données ;
- **ResultSet** : Permet le parcours de l'information renvoyée par la base de données.

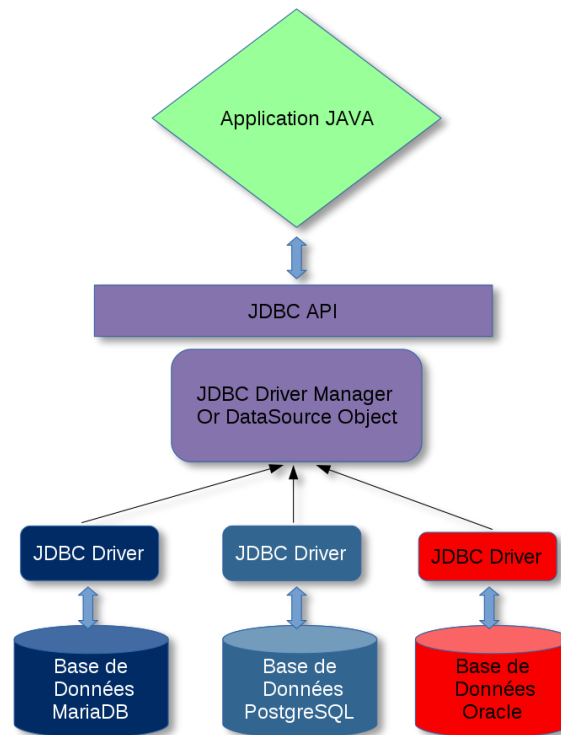
Les classes de la bibliothèque **JDBC** sont accessibles depuis le package : **java.sql.***

L'implémentation est réalisée dans une classe **BDD** qui a été elle-même placée dans un package **jdbc** :



```
class BDD {  
  
    var url: String = "jdbc:mysql://localhost:3306/db_kotlinAventure",  
    var user: String = "root",  
    var password: String = "",  
  
    var connectionBDD: Connection? = null  
  
    /**  
     * Initialise une instance de la classe BDD et établit une connexion à la base de données.  
     */  
    @ Moulin Timothée  
    init {  
        try {  
            // Établir une connexion à la base de données lors de la création de l'objet BDD  
            this.connectionBDD = getConnection()  
        } catch (erreur: SQLException) {  
            // Gérer les exceptions liées à la connexion  
            println("Erreur lors de la connexion à la base de données : ${erreur.message}")  
        }  
    }  
}
```

Le schéma suivant fourni par **Wikipédia** permet de mieux appréhender le flux de communication entre notre application **Kotlin/Java** vers et depuis une base de données via la pile **JDBC/Driver** :



TRAVAIL A FAIRE SUR LA CLASSE BDD :

Avant d'aller plus loin, nous allons **vérifier les informations de connexion inscrites** en dur dans le code de notre classe **BDD** afin d'avoir un connecteur opérationnel pour notre base de données fraîchement construite à l'étape précédente.

Pour ce faire, il faut ouvrir le code de la classe **BDD** et vérifier ci-dessous :

```
class BDD(  
  
    var url: String = "jdbc:mysql://localhost:3306/db_kotlinAventure",  
    var user: String = "root",  
    var password: String = "",  
) {  
    var connectionBDD: Connection? = null
```


Mettre à jour si nécessaire :

- Remplacer « **localhost** » par l'IP du serveur **MariaDB** dans la chaîne de l'attribut **url**
- Remplacer « **db_kotlinAventure** » par le nom de votre base de données dans la chaîne de l'attribut **url**
- La valeur de l'attribut : **user**
- La valeur de l'attribut : **password**

A propos de l'attribut URL :

Cette **URL** sera interprétée par l'**API JDBC**. Elle utilise le formalisme d'une **URL** pour récupérer les informations nécessaires à la connexion sur notre base de données :

```
var url: String = "jdbc:mysql://localhost:3306/db_kotlinAventure",
```



1. Protocole utilisé par **JDBC**, ici celui pour dialoguer avec une base **MariaDB/MySQL**
2. Adresse (**IP** ou nom **DNS**) de la machine faisant tourner le **SGBD** (**MariaDB** dans notre cas).
3. Numéro de port, par défaut le port est **3306**, mais il peut être modifié dans les fichiers de configuration du serveur **MariaDB**
4. Le nom de la base de données de notre application.

Pour en finir sur la classe BDD :

Pour en finir, nous vous indiquons où trouver l'instanciation de la classe BDD pour établir la connexion à la BDD. Cela se passe dans la fonction **main()** du fichier **main.kt** :

```
19 val coBDD = BDD()
20 //instanciation d'un objet QualiteDAO
21 //val qualiteRepository = QualiteDAO(coBDD)
```

Mission 2B – Création des DAO :

A l'image de la **mission 1** où vous avez dû implémenter les classes **GenerateurObjetImportCSV** pour valoriser des objets d'une classe à partir de valeurs enregistrées dans un fichier **CSV**. Vous allez maintenant réaliser une démarche similaire pour ici valoriser des objets à partir des données de la base de données. Evidemment ici, à chaque classe correspond une table dont chaque ligne/enregistrement contient les valeurs associées au différents objets (1 ligne pour 1 objet).

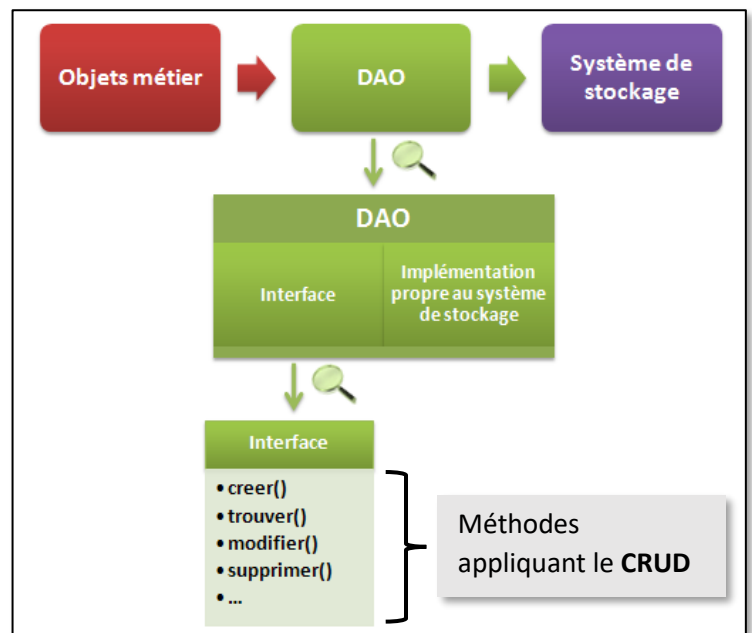
Nous allons procéder également avec la même approche que pour la **mission 1** pour l'organisation de votre travail, c'est-à-dire vous allez vous inspirer de l'implémentation de la classe **QualiteDAO** pour réaliser les **DAO** pour les objets dont vous avez la responsabilité.

Analyse de la classe **QualiteDAO** :

Les **DAO** pour **Data Acces Object** sont des objets qui nous permettent de faire le lien entre la logique (code métier) et le **SGBD**. L'idée étant de centraliser ces échanges dans un « lieu » identifié et dédié. En pratique le **DAO** va instancier les objets de notre application (modèle) à partir de ses classes et des données de la base de données.

Les méthodes d'une classe **DAO** doivent généralement au moins réaliser les 4 opérations **CRUD** :

- **Create** : Créer une nouvelle entrée dans la base de données.
- **Read** : Lire une entrée de la base de données.
- **Update** : Mettre à jour une donnée de la base de données.
- **Delete** : Supprimer une entrée de la base de données.



Chaque méthode va utiliser un mécanisme dit de requête préparée qui se décompose en 3 parties :

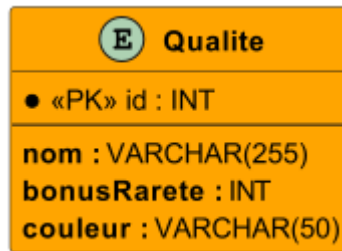
1. Elaboration du template de la requête préparée afin d'obtenir à l'étape 2 une requête **SQL**.
2. Compilation de la requête préparée pour qu'elle puisse être exécutée plus rapidement pour l'étape 3
3. Exécution de la requête compilée à l'étape.

Les requêtes préparées (on parle aussi de *requêtes paramétrables*) nous offrent plusieurs intérêts :

- Un premier niveau de protection en cas d'une tentative d'attaques malveillantes par ce que nous appelons « **attaque par injection SQL** », nous vous invitons à lire les éléments suivants sur ce sujet :
 - https://fr.wikipedia.org/wiki/Injection_SQL
 - <https://www.oracle.com/fr/security/injection-sql-attaque/>
- La phase de compilation permet d'avoir des requêtes qui s'exécutent plus rapidement lorsque la requête est exécutée plusieurs fois, même si les paramètres changent d'une requête à l'autre.

Analyse de la 1^{ère} méthode : `findAll()`

Cette méthode a pour objectif de récupérer les valeurs pour nos objets de la classe **Qualite**, nous allons pour ce faire devoir récupérer tout le contenu de la table **Qualite** :



Nous aurons donc besoin de préparer la requête **SQL** :

- **SELECT FROM * Qualite**

```
fun findAll(): MutableMap<String, Qualite> {
    0 val result = mutableMapOf<String, Qualite>()

    1 val sql = "SELECT * FROM Qualite"
    2 val requetePreparer = this.bdd.connectionBDD!!.prepareStatement(sql)
    3 val resultatRequete = this.bdd.executePreparedStatement(requetePreparer)
    if (resultatRequete != null) {
        4 while (resultatRequete.next()) {
            val id = resultatRequete.getInt( columnLabel: "id")
            val nom = resultatRequete.getString( columnLabel: "nom")
            val bonus = resultatRequete.getInt( columnLabel: "bonusRarete")
            val couleur = resultatRequete.getString( columnLabel: "couleur")
            5 result.set(nom.lowercase(), Qualite(id, nom, bonus, couleur))
        }
    }
    6 requetePreparer.close()
    7 return result
}
```

0. Déclaration et initialisation à vide du **dictionnaire (mutableMap)** qui sera retournée par la méthode.
1. Elaboration du template de la requête préparée (qui ne contient pas de paramètre dans ce cas).
2. Compilation de la requête préparée.
3. Exécution de la requête préparée et récupération du résultat de la requête dans un **itérable**.
4. Boucle d'itération (sur l'itérable issu de la requête pour valoriser les objets).
5. Ajout de l'objet instancié à la volée dans le dictionnaire
6. Fermeture de la requête préparée.
7. Retour du dictionnaire/map contenant les objets valorisés.

Analyse de la méthode `findByNom()` :

Cette méthode permet de retrouver un ou plusieurs objets de classe **Qualite** à partir de leur nom. Elle est intéressante car elle nous permet de voir comment nous pouvons paramétrer la requête paramétrée avec un unique paramètre qui correspond au nom de l'objet recherché. Le paramètre est représenté au moyen du symbole `?`

Nous allons ici commenter que les éléments nouveaux par rapport à la méthode vue précédemment.

La requête **SQL** à mettre en œuvre ici est de la forme :

- **SELECT * FROM qualite WHERE nom**

L'attribut **nom** sera le paramètre de notre requête, nous la compléterons avec la syntaxe suivante **nom= ?** afin d'indiquer que cela est effectivement un paramètre.

```
0 fun findByNom(nomRechercher:String): MutableMap<String,Qualite> {  
    val result = mutableMapOf<String,Qualite>()  
  
    1 val sql = "SELECT * FROM Qualite WHERE nom=?"  
    2 val requetePreparer = this.bdd.connectionBDD!!.prepareStatement(sql)  
    3 requetePreparer?.setString( parameterIndex: 1, nomRechercher)  
    4 val resultatRequete = this.bdd.executePreparedStatement(requetePreparer)  
    if (resultatRequete != null) {  
        while (resultatRequete.next()) {  
            val id =resultatRequete.getInt( columnLabel: "id")  
            val nom=resultatRequete.getString( columnLabel: "nom")  
            val bonus= resultatRequete.getInt( columnLabel: "bonusRarete")  
            val couleur= resultatRequete.getString( columnLabel: "couleur")  
            result.set(nom.lowercase(),Qualite(id,nom,bonus,couleur))  
        }  
    }  
    requetePreparer.close()  
    return result  
}
```

0. L'entête de la méthode comporte un paramètre **nomRechercher** qui récupérera l'argument correspondant au nom
1. Template de la requête préparée avec le paramètre `?` en valeur de **nom**
2. Compilation de la requête préparée, du moins de la partie ne variant pas à chaque exécution.
3. Ici on a une ligne supplémentaire qui permet de définir la partie variable de la requête, notre paramètre étant unique, il est associé au 1^{er} indice.
4. Exécution de la requête préparée.

Analyse de la méthode `findById()` :

Cette méthode repose sur le même mécanisme que la requête **finByNom()**, nous vous laissons apprécier son implémentation en autonomie.

Analyse de la méthode save() :

Cette méthode va permettre d'aborder 2 nouveautés par rapport aux méthodes précédentes :

- Mise en œuvre d'une requête d'écriture dans la **BD** (les précédentes étaient des accès en lecture).
- La requête comporte plusieurs paramètres, car un paramètre pour chaque attribut de l'objet.

La requête créera un nouvel enregistrement (une nouvelle mise ligne dans la table **Qualite**) si l'**id** de l'objet est absent de la table. Si l'objet est présent, les attributs de l'objet correspondant à l'**id** seront mis à jour.

Voici l'analyse de la 1^{ère} partie de la structure conditionnelle permettant de préparer une requête d'insertion d'un nouvel enregistrement/ligne dans la table. Nous avons donc une requête de type **Insert Into** qui sera multi-paramétrées pour insérer le **nom**, le **bonusRarete** et la **couleur**, soit une requête ayant pour template :

- **INSERT INTO Qualite (nom, bonusRarete, couleur) VALUES (?; ?; ?)**

Nous allons pouvoir constater que les paramètres sont automatiquement indexés de **1** à **3**.

```
if (uneQualite.id == null) { 0
    val sql =
    1 "Insert Into Qualite (nom,bonusRarete,couleur) values (?,?,?)"
    2 requetePreparer = this.bdd.connectionBDD!!.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)
    requetePreparer?.setString( parameterIndex: 1, uneQualite.nom)
    requetePreparer?.setInt( parameterIndex: 2, uneQualite.bonusRarete)
    requetePreparer?.setString( parameterIndex: 3, uneQualite.couleur)
```

0. Un objet ayant un **id null** est par nature un nouvel objet, il fait donc objet d'une requête d'insertion dans la table.
1. Template de la requête préparée avec 3 paramètres représentés par les 3 ? en argument de **VALUES**
2. A partir de cette ligne nous utilisons les méthodes **setString()** ou **setInt()** pour renseigner les bonnes valeurs grâce à leur indices.

Si l'objet comporte déjà un **id** non **null**, c'est qu'il existe déjà dans la table **Qualite**, nous allons donc ici mettre en œuvre une requête de mise à jour **UPDATE**., le template sera de la forme :

- **UPDATE Qualite SET NOM= ?; bonusRarete= ?, couleur= ? WHERE id= ?**

```
else {
    var sql = ""
    sql =
    "Update Qualite set nom=?,bonusRarete=?,couleur=? where id=?"
    requetePreparer = this.bdd.connectionBDD!!.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)

    requetePreparer?.setString( parameterIndex: 1, uneQualite.nom)
    requetePreparer?.setInt( parameterIndex: 2, uneQualite.bonusRarete)
    requetePreparer?.setString( parameterIndex: 3, uneQualite.couleur)
    requetePreparer?.setInt( parameterIndex: 4, uneQualite.id!!)
```

La préparation de la requête est assez similaire à la préparation de la requête pour rajouter un objet.

Il faut ensuite exécuter la requête, ce qui est intéressant ici, c'est que dans le cas de l'application de l'insertion d'un nouvel enregistrement d'un objet, un nouveau **id** est généré !

```
// Exécutez la requête d'insertion
val nbLigneMaj = requetePreparer?.executeUpdate()
// La méthode executeUpdate() retourne le nombre de lignes modifié par un insert,
// update ou delete sinon elle retourne 0 ou -1
```

Nous allons enfin dans la partie de code à venir récupérer le nouvel **id** pour renvoyer un nouvel objet **Qualite** avec son **id** fraîchement généré (pour rappel, un nouvel objet **Qualite** à un **id null** au début de la méthode).

L'objet est retourné, ce qui permettra de mettre à jour l'objet dans la logique de notre application.

```
// Si l'insertion a réussi
if (nbLigneMaj != null && nbLigneMaj > 0) {
    // Récupérez les clés générées (comme l'ID auto-incrémenté)
    val generatedKeys = requetePreparer.generatedKeys
    if (generatedKeys.next()) {
        val id = generatedKeys.getInt( columnIndex: 1) // Supposons que l'ID est la première col
        uneQualite.id = id // Mettez à jour l'ID de l'objet Qualite avec la valeur générée
        return uneQualite
    }
}
```

Méthodes `saveAll()` et `deleteById()` :

Ces 2 méthodes nous demandent moins de commentaires, car nous avons déjà l'ensemble des traitements de base.

La méthode **saveAll()** nous permet de sauvegarder un lot d'objets placés dans une collection. Elle va itérer sur l'ensemble des objets de la collection (collection ici appelé **lesQualites**) et va à chaque itération appeler la méthode **save()** vu juste précédemment. Cette méthode nous retournera tous les objets fraîchement sauvegardés et comportant leur nouvel **id** dans un dictionnaire afin que nous puissions en retour mettre à jour ces objets dans la logique de l'application.

La méthode **deleteById()** met en œuvre une requête **SQL DELETE** comportant 1 paramètre qui est l'**id** de l'objet à supprimer.

Retour sur la coordination de nos DAO avec la logique de l'application :

Avant de vous laisser travailler sur le développement des **DAO**, nous allons commenter la liaison entre les **DAO** et la logique de l'application.

```
10 //DEMO MISSION 1
11 val generateurQualites = GenerateurQualites( cheminFichier: "assets/qualites.csv")
12 val qualitesFromCSV = generateurQualites.generer()
13
14
15
16 //DEMO MISSION 2 :
17 // TODO Retirer les commentaires des lignes 22 et 25
18 // TODO : A la ligne 13 renom   la variable qualites en qualitesFromCSV
19 //instanciation de la co    la BDD
20 val coBDD = BDD()
21 //instanciation d'un objet QualiteDAO
22 val qualiteRepository = QualiteDAO(coBDD)
23 //
24 //Sauvegarde des Qualites dans la BDD
25 val qualites = qualiteRepository.saveAll(qualitesFromCSV.values)
```

- A la ligne **n  12** nous importons/valorisons nos objets    partir du fichier **CSV** associ   aux objets de la classe **Qualite**.
Pour rappel la variable **qualitesFromCSV** est un dictionnaire (voir **mission 1**).
- A la ligne **n  20** nousinstancions un objet de connexion    la Base de Donn  es appel  e **coBDD**.
- A la ligne **n  22** nousinstancions un objet de la classe **QualiteDAO** avec la connexion **coBDD**.
- A la ligne **n  25** nous proc  dons    la vol  e :
 -    A l'enregistrement de tous les objets import  s du **CSV** dans la **BDD**. (m  thode **saveAll()**)
 -    Le retour de la m  thode **saveAll()** nous permet de cr  er notre **map** d'objets appel  e **qualites**.

Travail    faire :

Ouvrir le fichier **main.kt**. Renommer    la ligne **12** la variable **qualites** en **qualitesFromCSV**.

```
10 //DEMO MISSION 1
11 val generateurQualites = GenerateurQualites( cheminFichier: "assets/qualites.csv")
12 val qualites = generateurQualites.generer()
```

```
10 //DEMO MISSION 1
11 val generateurQualites = GenerateurQualites( cheminFichier: "assets/qualites.csv")
12 val qualitesFromCSV = generateurQualites.generer()
```

Décommenter les lignes permettant d'activer l'enregistrement des objets **Qualite** dans la table dédiée :

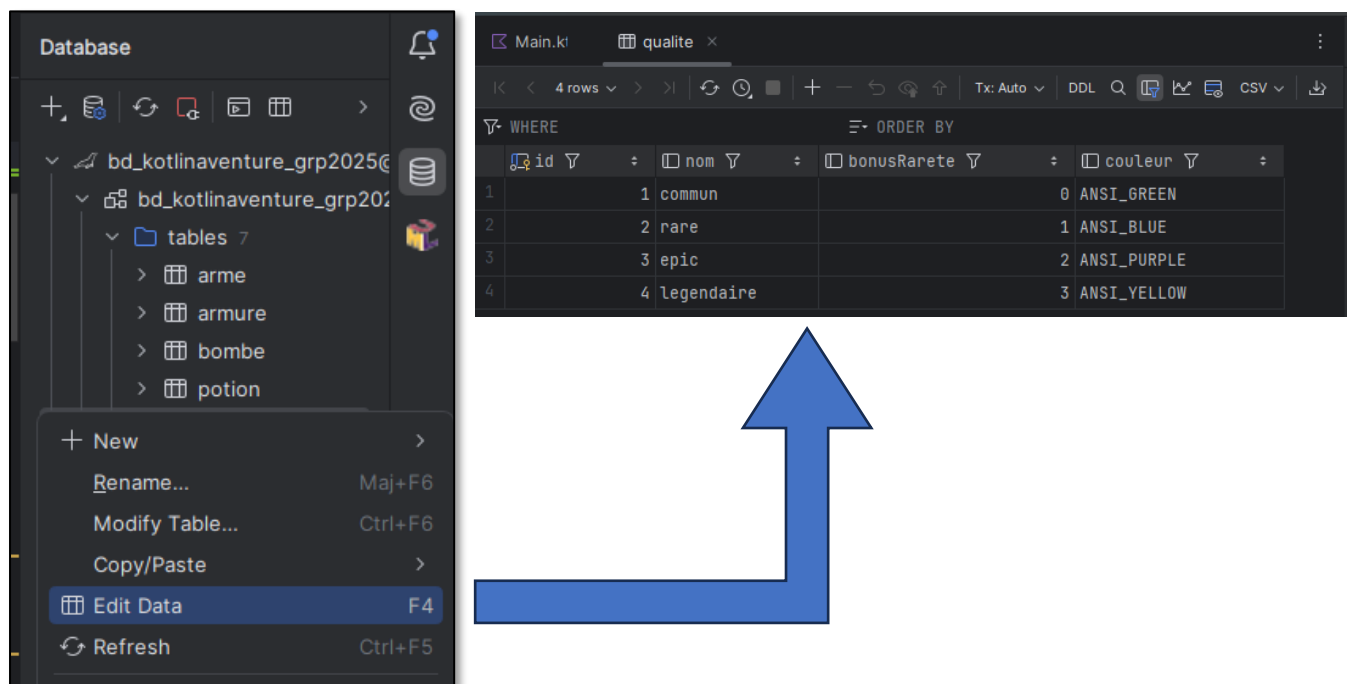
```

16 //DEMO MISSION 2 :
17 // TODO Retirer les commentaires des lignes 22 et 25
18 // TODO : A la ligne 13 renommé la variable qualites en qualitesFromCSV
19 //instanciation de la co à la BDD
20 val coBDD = BDD()
21 //instanciation d'un objet QualiteDAO
22 //val qualiteRepository = QualiteDAO(coBDD)
23 //
24 //Sauvegarde des Qualites dans la BDD
25 //val qualites=qualiteRepository.saveAll(qualitesFromCSV.values)
26

```

Vérification du résultat :

Exécuter votre application sur **IntelliJ**, si votre application compile et s'exécute, vous avez validé en partie vos modifications. Il reste à vérifier que les données du **CSV** ont bien été inscrites dans la table **qualite** de la base de données. Pour ce faire, cliquer droit sur la table **qualite** dans l'outil d'exploration de BD d'**IntelliJ** et choisir **Edit Data**. Vous devriez alors observer que votre table a bien reçu ses nouveaux objets :



The screenshot shows the IntelliJ IDEA interface. On the left, the 'Database' tool window displays a tree structure of the database 'bd_kotlinaventure_grp2025'. The 'qualite' table is selected, and a context menu is open with 'Edit Data' highlighted (F4). On the right, the 'Edit Data' window for the 'qualite' table is shown, displaying 4 rows of data:

id	nom	bonusRarete	couleur
1	commun	0	ANSI_GREEN
2	rare	1	ANSI_BLUE
3	epic	2	ANSI_PURPLE
4	legendaire	3	ANSI_YELLOW

A large blue arrow points from the 'Edit Data' menu item to the 'Edit Data' window, indicating the action taken to view the data.

Travail à réaliser pour la mission 2B :

Etudiant	Mission	Tâches associées
Etudiant 1	Mission 2B.1	Création des DAO : TypeArmeDAO et ArmureDAO
Etudiant 2	Mission 2B.2	Création des DAO : TypeArmureDAO et ArmureDAO
Etudiant 3	Mission 2B.3	Création des DAO : PotionDAO et BombeDAO

Mission 3-Rechercher & implémenter une solution anti-redondance :

Analyse du problème :

Si vos **DAO** et votre **Base de Données** sont opérationnelles, vous aurez peut-être anticipé et nous vous invitons à le constater, qu'actuellement notre système d'importation des objets au lancement du programme souffre d'un gros défaut ! A chaque lancement de l'application **TOUS** les objets des **CSV** sont importés dans la **Base de Données**, même s'ils étaient déjà présents !

Travail à réaliser :

1. Trouver une stratégie commune à tous les membres de l'équipe pour déterminer si un objet est déjà présent dans la base de données.
2. Faire valider votre stratégie par vos professeurs.
3. Mettre en œuvre votre stratégie.

Mission 4 – Exportation du contenu des tables vers des CSV :

Exposé du besoin :

Nous désirons procéder à la sauvegarde de notre **Base de Données**. Pour ce faire, nous désirons réaliser un fichier **CSV** par table.

Travail à réaliser :

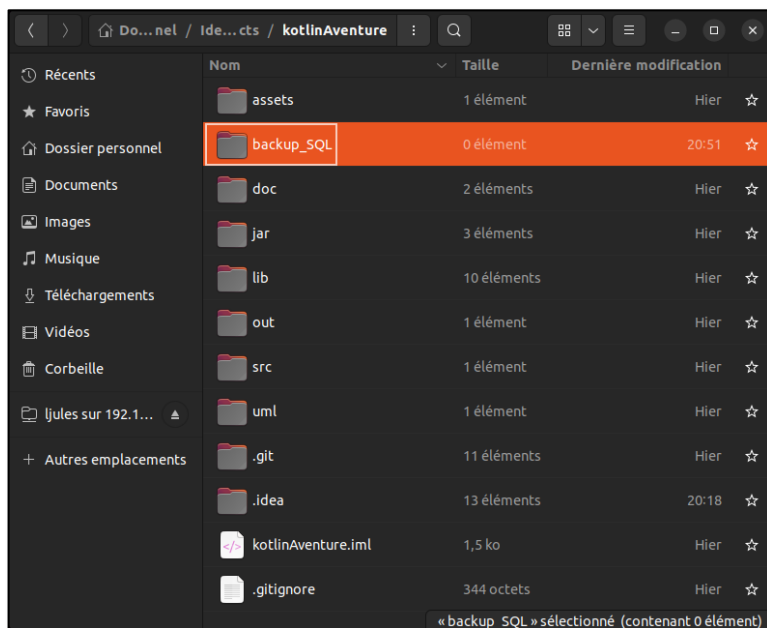
Proposer une solution, la faire valider par vos professeurs et la mettre en œuvre.

Pourquoi et quand procéder :

Votre base de données pour ce sprint du projet est mise en place sur votre machine de travail du lycée. Le problème est que cette machine est partagée par les étudiants du **BTS SIO** et d'autres section du lycée. Vous êtes donc susceptible de perdre votre travail réalisé sur la base de données dans le cas où une personne réalise de mauvaises manipulations volontaires ou involontaires. Ou encore dans le cas d'une restauration complète de votre poste, tout le contenu de ce poste sera effacé.

On vous conseille donc de procéder à une sauvegarde de votre base de données en fin de séance et de placer le fichier **SQL** généré dans le dossier de travail de votre projet afin qu'il bénéficie d'une copie sur votre dépôt distant **GitHub**.

EXEMPLE :

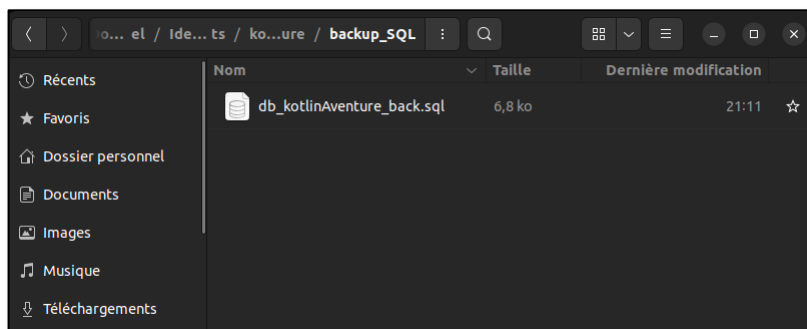


Sauvegarde de votre base de données :

Sauvegarde manuelle :

- Ouvrir un terminal à partir de votre dossier de sauvegarde :
 - **Exemple :** `./KotlinAventure/backup_SQL`
- Saisir la commande suivante :
 - `mysqldump -u adm -p db_kotlinAventure > db_kotlinAventure_back.sql`

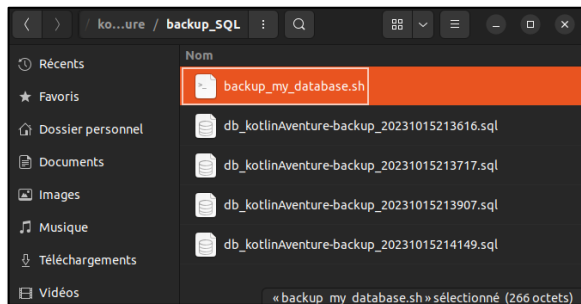
Vous avez maintenant un fichier de sauvegarde dans le dossier prévu à cet effet :



Script de sauvegarde :

La méthode suivante est déjà intéressante mais il faudra chaque fois saisir la commande et choisir un nom pour le fichier.

Afin de se simplifier la vie nous vous proposons d'écrire un script qui réalisera l'opération pour nous tout en rajoutant la date au fichier de sauvegarde :



```
#!/bin/bash

# SCRIPT DE SAUVEGARDE BASE DE DONNEES :
# -----

USER_BD="adm"
MDP="123passe"
DATABASE="db_kotlinAventure"
DATE=$(date +%Y%m%d%H%M%S)
FILE="$DATABASE-backup_$DATE.sql"

mysqldump -u $USER_BD -p$MDP $DATABASE > $FILE
```

Il suffira de rendre « **exécutable** » le script (clic droit Propriétés/Permissions/Autoriser l'exécution du fichier comme programme).

Puis pour l'exécuter de faire clic droit : **Exécuter comme un programme**

Restauration de votre base de données :

Pour restaurer la base de données, cette dernière doit déjà exister :

- `mysqldump -u adm -p db_kotlinAventure < db_kotlinAventure_back.sql`