



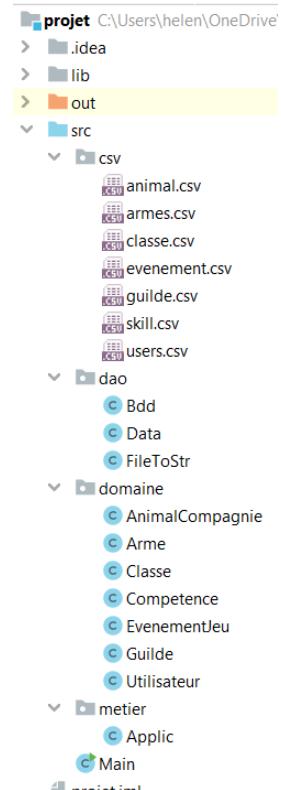
*Groupe: Ardi Ramushi & Hélène Moore*

*Année : 2022 - 2023 - 62-31 Gestion avancée des données*

<b>Sujet: Jeu vidéo DataLand</b>	<b>3</b>
Objectif	3
Choix de la BDD et justification	3
Représentation de la BDD	3
Pourquoi Neo4j ?	3
Pourquoi pas MongoDB ?	3
Générations des données	4
Les données à insérer	4
Les relations	5
Quelques précisions	6
Lancer le projet	6
Définitions des requêtes	7
1. Participation aux évènements	7
2. Parrainage	8
3. Statistique	8
4. Equilibrage	8
Sources	10

## Avant-propos

- Le rendu est sous forme de zip
- La structure du projet doit ressembler à cela pour fonctionner:
  - Dans le dossier *src*:
    - Le dossier *csv* contient toutes les données qui vont être chargées via l'Application
    - Le dossier *dao* contient la partie permettant de lire les csv et les traiter
    - Le dossier *domaine* contient toutes les classes utilisées
    - Le dossier *metier* contient l'application permettant notamment de faire les requêtes dans Neo4j
    - Le tout se lance depuis la classe Main
  - Dans le dossier *lib* se trouvent les drivers



## Sujet: Jeu vidéo DataLand

### Objectif

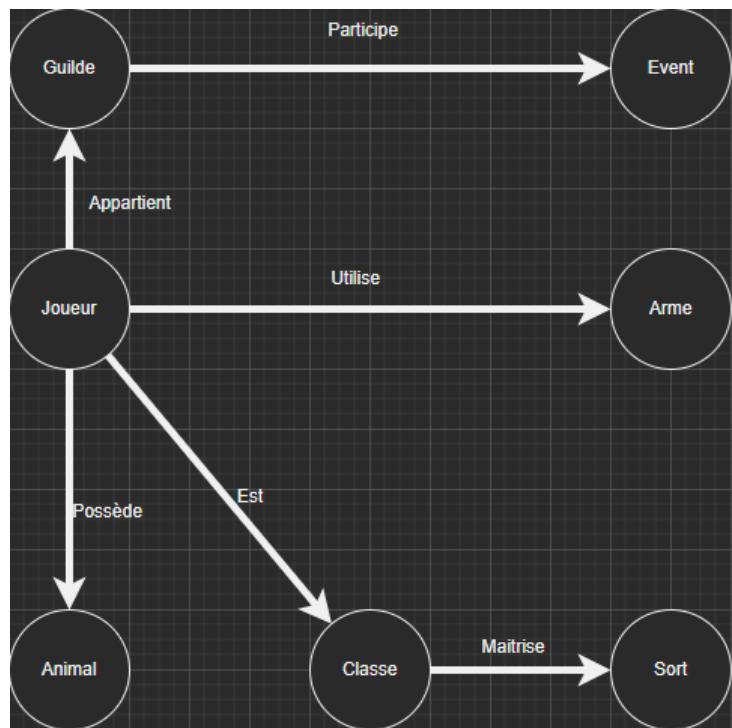
Une entreprise de jeux vidéo nous mandate pour son prochain projet DataLand. Elle souhaite stocker les informations des joueurs et de leurs personnages.



### Choix de la BDD et justification

#### *Représentation de la BDD*

Notre base de données sera représentée de la manière suivante. On voit ainsi le besoin et l'utilité des nœuds et les relations entre ceux-ci. Surtout que nous sommes intéressés de voir les liens entre les joueurs par exemple et leurs possessions ou leur guilde.



#### *Pourquoi Neo4j ?*

Nous avons choisi *Neo4j* pour les raisons suivantes:

**Lien entre les nœuds:** en effet, nous avons besoin de garder un lien entre les différents nœuds/entités tel que le lien entre la guilde et les joueurs qui en font partie ou la compétence que maîtrise sa classe.

- **Temps de traitement rapide lors de la recherche entre noeud:** Pour un jeu vidéo en ligne, nous pensons qu'il serait mieux que la base de données répondent rapidement car les interactions entre les joueurs, dans les guildes ou dans les événements sont nombreux
- **Pas/peu de doublons et pas d'imbrication dans les nœuds:** le nœud a ses propres propriétés et est ensuite relié à un autre nœud. Ainsi, il n'y a pas d'imbrication ou de répétitions dans chaque nœud.

#### *Pourquoi pas MongoDB ?*

Nous n'avons pas choisi MongoDB car, comme expliqué auparavant, les liens entre les joueurs et les autres noeuds étaient importants dans ce que nous cherchions. De plus, nous pouvons stocker des données diversifiées.

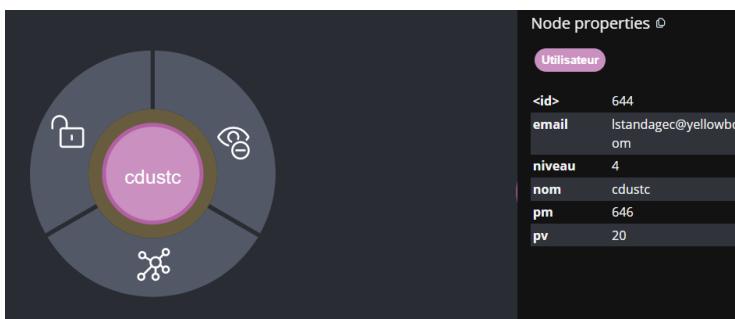
## Générations des données

### *Les données à insérer*

Voici les propriétés des différents noeuds.

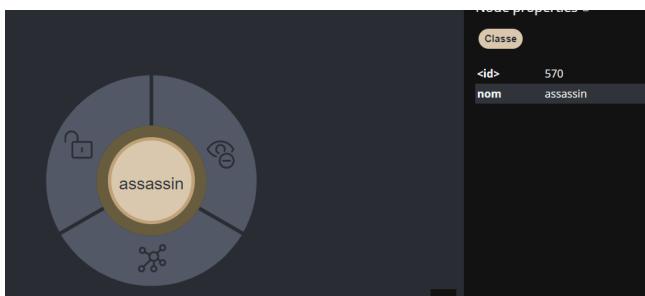
- les **joueurs** (**Utilisateur**)

- Pseudo (String)
- Email (String)
- Niveau (int)
- PV (point de vie) (int)
- PM (point de magie) (int)



- leur classe (**Classe**)

- Nom (String)

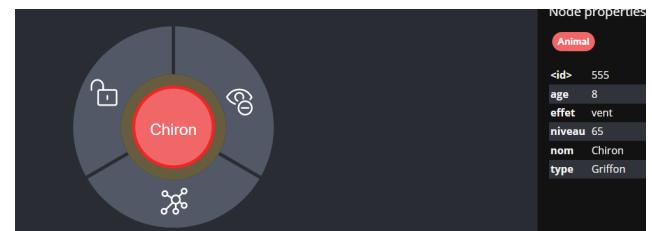


- leur compétence (**Competence**)

- Nom (String)
- Niveau du sort (int)
- Puissance (int)
- Coût en magie (int)
- Temps de recharge

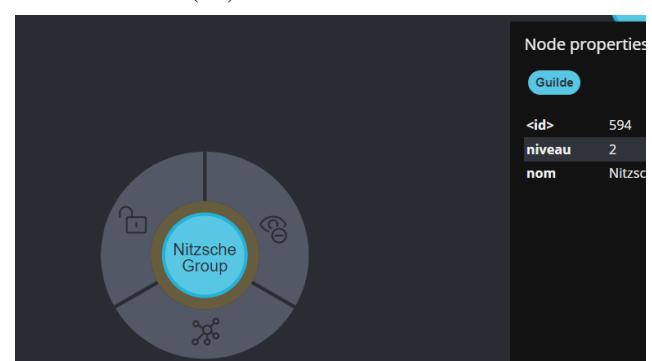
- leur animal de compagnie (**Animal**)

- Nom (String)
- Niveau (int)
- Âge (int)
- Effet (String)



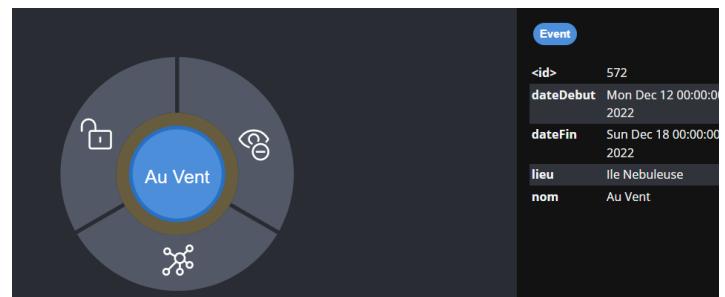
- leur guilde (**Guilde**)

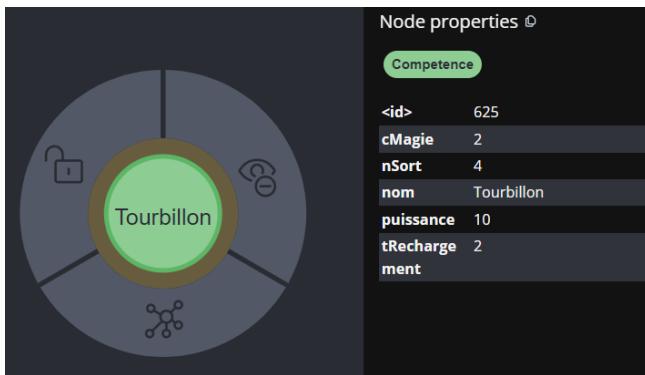
- Nom (String)
- Niveau (int)



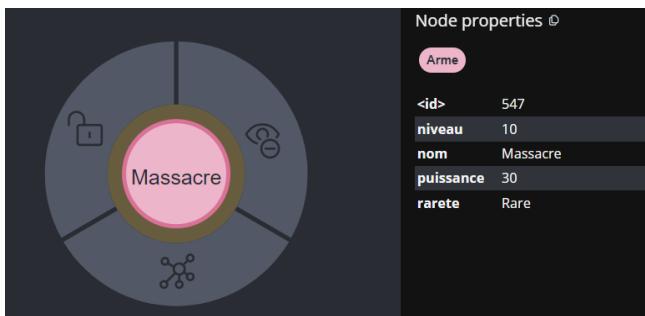
- l'évènements organisé (**Event**)

- Nom (String)
- Date de début (Date)
- Date de fin (Date)
- Lieu (String)





- leur(s) arme(s) (Armes)
  - Nom (String)
  - Niveau (int)
  - Puissance (int)
  - Rareté (String)



### *Les relations*

Nom de la relation	Son utilité
[ :PARTICIPE]	Relie une guilde à un évènement Une guilde peut participer à un à plusieurs évènements et un évènement peut avoir plusieurs guilde.
[ :UTILISE]	Relie le joueur à son arme. En fonction de sa classe, il aura l'arme correspondante Un joueur utilise une arme tandis qu'une arme peut avoir plusieurs joueurs
[ :APPARTIENT]	Relie le joueur à sa guilde (s'il est dans une guilde) Le joueur appartient à une guilde et la guilde a 0 à plusieurs joueurs.
[ :POSSEDE]	Relie le joueur à son animal (s'il en a un). En fonction de sa classe, il aura l'animal correspondant Le joueur possède 0 à un animal de compagnie et un animal est possédé par un joueur
[ :EST]	Relie le joueur à sa classe (= il est de la classe "Mage") Un joueur est d'une classe et une classe a plusieurs joueurs
[ :MAITRISE]	Relie la classe à son sort La classe maîtrise deux sorts et le sort est maîtrisé par qu'une seule classe

## Quelques précisions

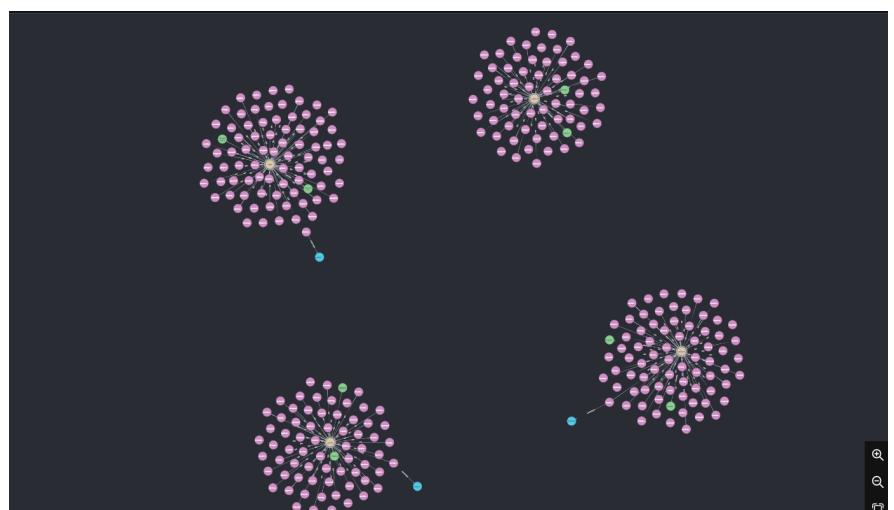
- Les évènements
  - Terre Brûlée - **du** lundi 5 **au** dimanche 11 décembre 2022 dès 20h - La Steppe Malveillante
  - Au Vent - **du** lundi 12 **au** dimanche 18 décembre 2022 dès 20h - Ile Nébuleuse
  - Landes de Pierre - **du** jeudi 1er décembre **au** 31 décembre dès 00h - La Statue d'Immortels
- Leur compétence
  - Les Guerriers : Lancer de bouclier, Tourbillon d'épée
  - Les Mages : Bouclier Aqueux, Boule de Feu
  - Les Soigneurs : Régénération, Réanimation
  - Les Assassins : Assassinat, Empoisonnement
- Leur(s) arme(s)
  - Épée & Bouclier (Guerrier)
  - Bâton (Soigneur)
  - Sceptre (Mage)
  - Double dague (Assassin)
- Leur animal de compagnie
  - Dragon (Guerrier)
  - Griffon (Soigneur)
  - Cerbère (Mage)
  - Basilic (Assassin)

## Lancer le projet

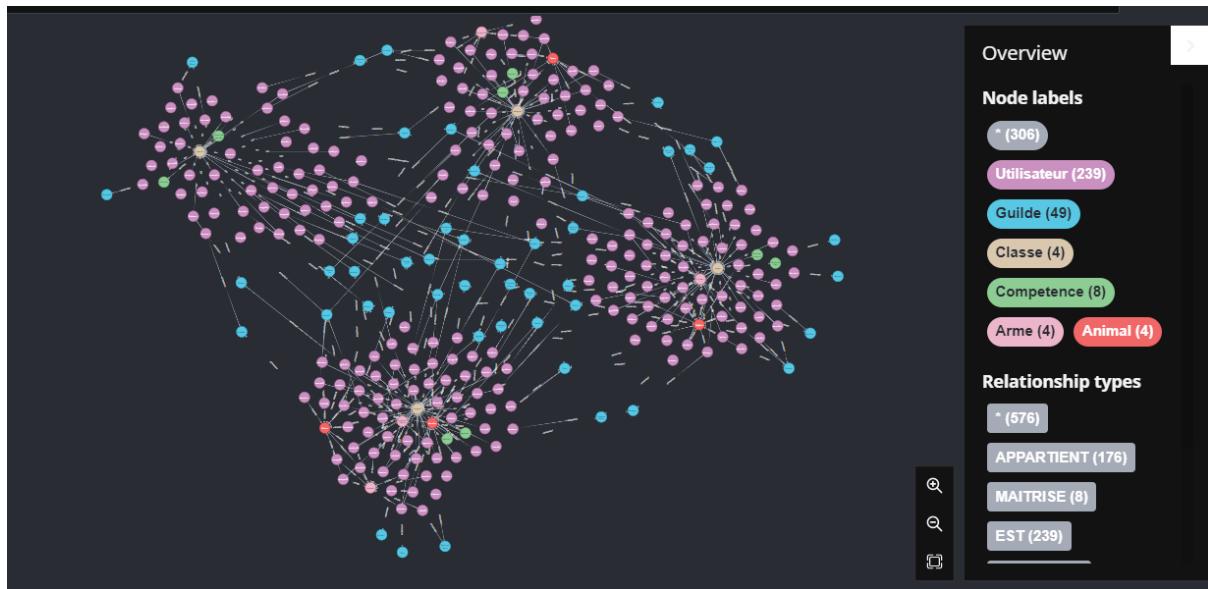
Le code est disponible sous le Projet “projet”. Pour le faire fonctionner, il suffit de lancer la classe *Main*. Celle-ci appelle la classe *Aplic* et les fonctions de la classe *Bdd* permettant de créer l’environnement via la méthode *delete\_all* puis *setup\_env(bdd)*. Ainsi, à chaque lancement du programme, l’environnement se crée et empêche les doublons causés par le fait d’avoir lancé plusieurs fois l’application.

On peut avoir une vue de cette manière et un output annonçant le début et la fin de chaque partie.

```
Main x ...
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" ...
debout link user-guilde
fin link user-guilde
debout link classe-skill
ajout skill guerrier
ajout skill mage
ajout skill soigneur
ajout skill assassin
fin link classe-skill
debout link user-classe
fin link user-classe
debout link user-arme
fin link user-arme
debout link user-animal
fin link user-animal
```



Cette vue permet de regrouper les utilisateurs par classe



### *Vue presque globale*

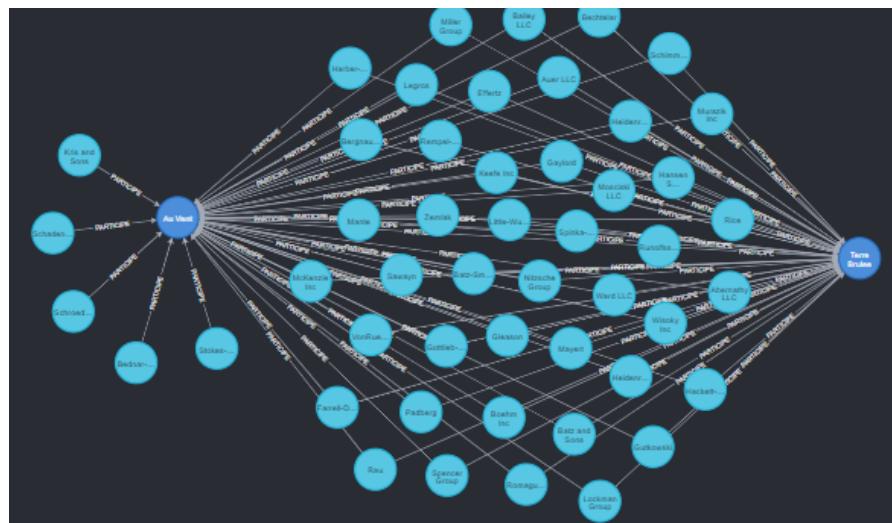
## Définitions des requêtes

## **1. Participation aux évènements**

### *But de la requête:*

- a. Pour participer à l'événement “*Terre Brûlée*”, la guilde doit avoir au moins un joueur de classe Mage possédant la compétence de “Bouclier aqueux”. Si celle-ci l'a, elle est inscrite automatiquement à l'événement.
  - b. Pour participer à l'événement “*Au Vent*”, la guilde doit avoir au moins un joueur possédant soit un Dragon ou Griffon

**Comment y arriver:** Pour arriver au résultat ci-dessous, la méthode `participerEvent(Session session, String nomEvenement, String requis1, String requis2)` a été appelée. Il fallait d'abord faire une requête pour trouver les guildes possédant les critères pour chaque compétition. Puis, il fallait vérifier si celle-ci était déjà inscrite à l'évènement (c.à.d s'il existe une relation de type PARTICIPE entre la guilde et l'évènement). Si elle n'était pas inscrite alors on relie la guilde et si elle est déjà inscrite, alors on ne fait rien (la relation existe déjà).



## 2. Parrainage

**But de la requête:** Un joueur de niveau 1, lors de son inscription/création dans le jeu, sera parrainé par un joueur de niveau 60 ayant la même classe que lui. Si on ne trouve pas de parrain, il y a aura un message alertant sur le fait que le joueur est premier dans sa catégorie. Si on trouve un formateur, le joueur ainsi que le futur parrain seront alerté.

**Comment y arriver:** Pour arriver au résultat ci-dessous, nous avons du d'abord générer via la méthode `randomJoueur(Session session)` un joueur aléatoirement qui se nomme `BddSlayer01` et qui vient de s'inscrire dans le jeu. Il est donc de niveau 1 et va avoir une classe de manière aléatoire. Après sa création, il est inséré dans la base de données.

Ensuite, nous cherchons dans la base de données via la méthode `trouverParrain(Session session)` les joueurs possédant la même classe que `BddSlayer01` et un niveau égal ou supérieur à 60.

Si on en trouve, alors on affiche les pseudonymes sinon, il y a un message indiquant qu'aucun parrain n'a été trouvé.

Nous vous avons trouvé des parrains!

- "hgaveltonep"
- "hlamberti5o"
- "cruddle9e"
- "ntivenanbx"
- "plambdond7"
- "mmarfellib"
- "gogdenj0"
- "rhellyerjy"
- "gsheptonmm"
- "idigginnf"
- "eboddisps"
- "pdailer1"
- "ypluesri"

## 3. Statistique

**But de la requête:** Les créateurs du jeu souhaitent connaître le nombre de joueurs par guilde regroupés par classe ainsi que le nombre de joueurs dans cette guilde. Voici l'exemple d'affichage<sup>1</sup>.

**Comment y arriver:** Pour obtenir ce résultat, nous avons deux méthodes.

La première `compterJoueurGuilde(Session session, String nomGuilde)` calcule le nombre de joueurs par classe et a pour paramètre le nom de la guilde. La fonction d'affichage `nombreJoueurParGuilde(Session session)` quant à elle fait une requête pour récolter toutes les guildes présentes dans le jeu.

La guilde Farrell-Daugherty :

Total de joueurs : 12 joueurs. Sur ces joueurs, elle contient:

- Mage : 3
- Guerrier : 4
- Soigneur : 3
- Assassin : 2

## 4. Equilibrage

**But de la requête:** Les créateurs souhaitent faire du tri et savoir quelle classe est la moins utilisée et créer un événement (“Autour des lacs” se produisant dans le lieu “Grottes Macabres” du 19 décembre au 24 décembre) entre guilde ayant des spécialistes de la classe (niveau 50 minimum pour être considéré être un spécialiste).

**Comment y arriver:** Pour obtenir ce résultat, la méthode `classeMoinsUtilisee(Session session)` est appelée. Elle compte en premier le nombre de joueur par classe puis avec `calculePourcentage(float totalClasse, int totalJoueurs)`, elle affiche le pourcentage de joueur par classe. Ensuite, via la méthode `trouverMinimum`, on trouve la classe minimum et on crée l'évènement. En dernier, on relie les guildes à “Autour des lacs”. Comme pour la

---

<sup>1</sup> Le nombre peut varier en fonction du nombre de joueurs générés et de la répartition aléatoire de ceux-ci dans les guildes

première requête, on vérifie en premier si la guilde participe déjà à l'évènement: si elle n'était pas inscrite alors on relie la guilde et si elle est déjà inscrite, alors on ne fait rien.

La méthode `classeMoinsUtilisee` (`Session session`) possède deux utilités: la première affiche dans la console les nombres de joueurs par classe en tout tandis que la deuxième, permet de relier les guildes contenant des spécialistes à l'évènement.

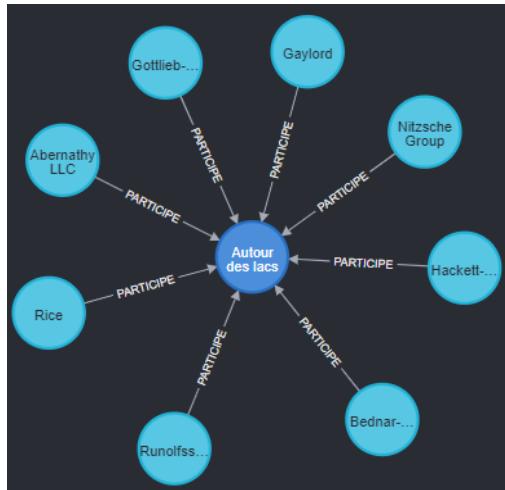
En fonction de la quantité de joueurs insérée à la création de la bdd, voici l'affichage qu'on peut avoir:

*Dans IntelliJ*

-----Évènement des spécialistes-----

- Mage : 252 (soit 25.174826 % des joueurs)
- Guerrier : 247 (soit 24.675325 % des joueurs)
- Soigneur : 249 (soit 24.875124 % des joueurs)
- Assassin : 253 (soit 25.274725 % des joueurs)

*Dans Neo4j*



## Sources

- Draw.io pour le schéma
- Canva pour la page de couverture
- Educative: Interactive Courses for Software Developers. « How to Generate Random Numbers in Java ». Consulté le 11 décembre 2022. <https://www.educative.io/answers/how-to-generate-random-numbers-in-java>.
- www.javatpoint.com. « Java Convert String to Date - Javatpoint ». Consulté le 11 décembre 2022. <https://www.javatpoint.com/java-string-to-date>.
- « Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel ». Consulté le 11 décembre 2022. <https://mockaroo.com/>.
- Neo4j Graph Data Platform. « WHERE - Neo4j Cypher Manual ». Consulté le 11 décembre 2022. <https://neo4j.com/docs/cypher-manual/5/clauses/where/>.
- « Fantasy Name Generators. Noms pour tous vos personnages. ». Consulté le 11 décembre 2022. <https://fr.fantasynamegenerators.com/>.
- « Capabilities of the Neo4j Graph Database with Real-Life Examples ». Consulté le 23 décembre 2022. <https://rubygarage.org/blog/neo4j-database-guide-with-use-cases>.
- Agence Web Kernix. « Exploiter des données hétérogènes avec le graphe de Neo4J ». Consulté le 23 décembre 2022. <https://www.kernix.com/neo4j/>.
- « Qu'est-ce que MongoDB et comment fonctionne-t-elle ? | Pure Storage ». Consulté le 23 décembre 2022. <https://www.purestorage.com/fr/knowledge/what-is-mongodb.html>.
- Rowe, Walker. « Introduction to the Neo4j Graph Database ». BMC Blogs. Consulté le 23 décembre 2022. <https://www.bmc.com/blogs/neo4j-graph-database/>.
- Cours disponible sur Cyberlearn <https://cyberlearn.hes-so.ch/course/view.php?id=22552>