Rapport

Base de données orientée graphe : Univers Nintendo

Romain Dreuilhet, Inel Ghazli, Avesta Molaei, Michaël Lebon

Atelier 4.2: Big Data et exploration de données

Juin 2019 *ESIEE PARIS*

Plan

1. Introduction
2. Modèle de la base de donnée
3. Requêtes Cypher & Exploitation
4. Fin

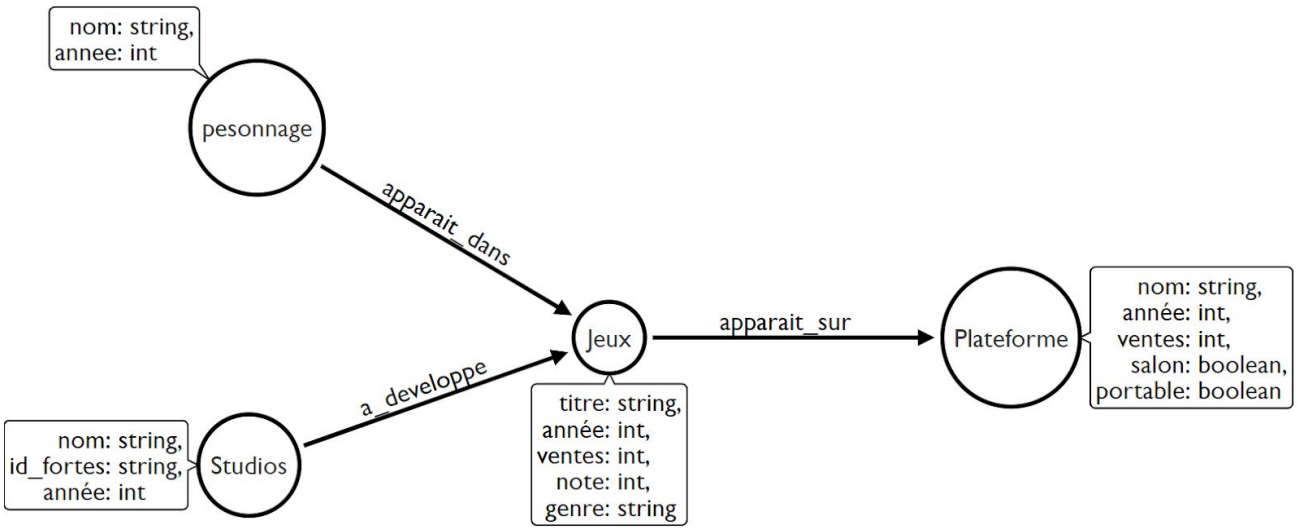
1.Introduction

Étant tous passionnés de jeux vidéos, ~~et, admettons-le, ayant tous passé un peu trop de temps à jouer au lieu de travailler un peu plus cette année,~~ notre choix de sujet pour la construction d'une base de donnée fut assez évident.

Nous avons choisi de créer une base de données sur l'univers du constructeur de console de jeux vidéo et développeur Nintendo comprenant ses licenses (et les personnages des licences), son studio, ses partenaires, et ses logiciels.

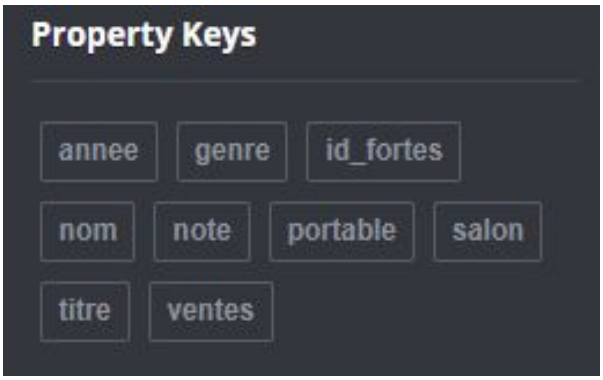
Ainsi, à l'aide de sources comme Wikipédia, MetaCritic, Vimm's Lair, et d'autres, nous nous sommes mis à la recherches des éléments permettant de réaliser notre base de donnée.

2.Modèle de la base de données



- Nœuds:

Dans la base de donnée, il y a quatre types de noeuds, dont le nom est assez parlant, pas besoin d'expliciter à ce niveau la:



Ces nœuds admettent pour propriétés:

Les Jeux ont un titre (string), une année de sortie (int), un nombre de ventes en millions (int), une note donnée par metacritic sur 100 (int), et un genre (string).

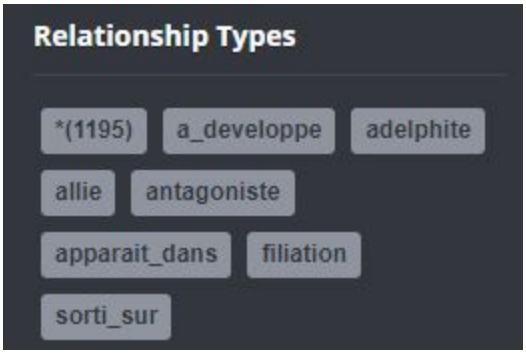
Les Studios ont un nom (string), une année de fondation (int), et des identitées fortes (string).

Les Personnages ont un nom (string), et une année de création (int).

Les Plateformes ont un nom (string), une année de sortie (int), un nombre de ventes en millions (int), et deux booléens: l'un est vrai et l'autre est faux si la console est de salon, et inversement si elle est portable.

- Type des relations

Dans le graphe, il y a 7 types de relations:



`a_developpe`: Lien entre un Jeu et un Studio: montre quels jeux ont été développés par quels studios.

`adelphite`: Lien entre deux Personnages: montre une relation frère/soeur

`allie`: Lien entre deux Personnages: montre une relation amicale

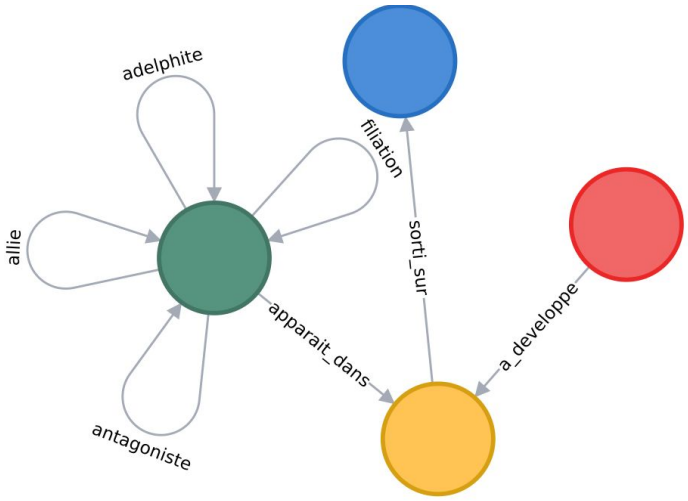
`antagoniste`: Lien entre deux Personnages: montre une relation de conflit

`filiation`: Lien entre deux Personnages: montre une relation père/mère-fils/fille

`apparait_dans`: Lien entre un jeu et un Personnage: montre quel personnage apparait dans quel jeu

`sorti_sur`: Lien entre un jeu et une plateforme: montre quels jeux sortent sur quelles plateformes

3.Requêtes Cypher & Exploitation de la BDD



I./ Personnages qui vendent le plus de copies:

Comment voir quelles sont les Personnages générant le plus de profit? Si l'on a accès à ces données, on pourrait focaliser le développement sur les licences de ces personnages afin d'optimiser la production, et générer le plus de profit. Eh bien, pour ce faire, on réalise les 3 requêtes suivantes.

```
1 MATCH(j:Jeux)<-[:apparait_dans]-(p:personnage)
2 WHERE  j.ventes IS NOT  null
3 RETURN j.ventes AS `nombre de copies vendues`, p.nom AS Personnages ORDER BY j.ventes DESC
```

Nombre de copies vendues en millions	Personnage
650	"Pichu"
650	"Lucario"
650	"Rondoudou"
650	"Herbizarre"
650	"Carapuce"
650	"Dracaufeu"
650	"Pikachu"
650	"Mewtwo"
90	"Luigi"
90	"Princesse Peach"
90	"Bowser"
90	"Mario"
90	"Toad"
82.87	"Mii"
45.8	"Mewtwo"
45.8	"Dracaufeu"
45.8	"Pichu"
45.8	"Carapuce"
45.8	"Rondoudou"
45.8	"Herbizarre"
45.8	"Pikachu"
44.57	"Wario"
44.57	"Donkey Kong"
44.57	"Bowser Jr"
44.57	"Armée de Bowser"
44.57	"Mario"
44.57	"Waluigi"

En guise de résultats, le navigateur Neo4j nous renvoie le tableau ci--contre.
On remarque que les licences générant le plus de profits sont celles avec les personnages des univers Pokémon et Mario, avec (respectivement) 650 millions & 90 millions d'unités vendues

II../ Studio qui vendent le plus de copies:

Comment voir quelles sont les Studios générant le plus de ventes? En obtenant ces informations, de potentiels investisseurs pourraient savoir sur quel “cheval miser”, et optimiser leurs investissements. On peut aussi voir les Studios au succès le plus prominent, et ainsi réaliser des prévisions quant aux achats futurs des consommateurs. Considérons ces quelques requêtes (partielles) de Cypher:

```
MATCH (s8:Studios)-[:a_developpe]->(j8:Jeux)
WHERE j8.ventes IS NOT null AND s8.nom="Studio Partenaire"
return s8.nom as `Studio`, sum(j8.ventes) as `Nombre de
copies vendues en millions`
UNION
MATCH (s:Studios)-[:a_developpe]->(j:Jeux)
WHERE j.ventes IS NOT null AND s.nom="Nintendo
Entertainement Planning & Development"
return s.nom as `Studio`, sum(j.ventes) as `Nombre de copies
vendues en millions`
UNION
MATCH (s6:Studios)-[:a_developpe]->(j6:Jeux)
WHERE j6.ventes IS NOT null AND s6.nom="Gamefreak"
return s6.nom as `Studio`, sum(j6.ventes) as `Nombre de
copies vendues en millions`
```

Ici, on récupère, avec `UNION`, plusieurs noms de studios différents. On trouve ainsi:

Studio	Nombre de copies vendues en millions
"Studio Partenaire"	832.49
"Nintendo Entertainement Planning & Development"	572.5100000000002
"Gamefreak"	110.49999999999999
"HAL Laboratory/SORA Ltd"	64.98
"Square (Enix)"	35.03
"Autre"	34.830000000000005
"Intelligent Systems"	22.78
"Monolith Soft"	14.35
"Retro Studios"	2.84

Le résultat fourni par le navigateur Neo4j est intéressant: le Nintendo EPD n’est pas le premier studio de développement. On remarque en effet que les studios partenaires ont vendu un nombre de copies beaucoup plus important que “Nintendo EPD” dans l’univers Nintendo…

Néanmoins, ceci n’est pas représentatif des revenus générés par ces studios, nous avons comptabilisé comme ventes les applications mobiles qui sont développées par les studios partenaires, étant donné que ces applications sont gratuites et ne génèrent pas autant de revenus que les logiciels payants développés pour consoles traditionnelles. On peut donc bien considérer le studio “Nintendo Entertainement and Planning Development “ comme leader en ce vis-à-vis des jeux concernant l’univers Nintendo.

III.../ Plateformes qui vendent le plus de jeux:

Comment voir quelles sont les Plateformes sur lesquels le plus de jeux ont été vendus? C'est avec ces informations que les développeurs savent comment optimiser leurs ventes: en effet, pourquoi dépenser de l'argent et des ressources à sortir un jeu sur une plateforme "morte" ou le jeu ne se vendra pas? De manière similaire à la requête précédente:

```
MATCH(j8:Jeux)-[:sorti_sur]->(p8:Plateforme)
WHERE j8.ventes IS NOT null AND p8.nom="NES/Famicom"
RETURN p8.nom as `Plateforme`, sum(j8.ventes) as `Nombre de copies vendues en millions`
UNION
MATCH(j9:Jeux)-[:sorti_sur]->(p9:Plateforme)
WHERE j9.ventes IS NOT null AND p9.nom="SNES/Super Famicom"
RETURN p9.nom as `Plateforme`, sum(j9.ventes) as `Nombre de copies vendues en millions`
UNION
MATCH(j10:Jeux)-[:sorti_sur]->(p10:Plateforme)
WHERE j10.ventes IS NOT null AND p10.nom="Nintendo 64"
RETURN p10.nom as `Plateforme`, sum(j10.ventes) as `Nombre de copies vendues en millions`
```

Et on obtient:

Plateforme	Nombre de copies vendues en millions
"Game Boy"	80.22999999999999
"Game Boy Advance"	3.77
"Famille Nintendo DS"	123.96999999999997
"Famille Nintendo 3DS"	144.51
"Appareils Androïd/IOS"	750
"NES/Famicom"	83.94
"SNES/Super Famicom"	42.06
"Nintendo 64"	72.52
"GameCube"	82.78
"Wii"	313.52000000000004
"Wii U"	131.0
"Switch"	120.12
"Autres plateformes"	24.189999999999998

NB: on voit ici que la GBA est la console générant avec 3.77 millions de copies, le moins de ventes. On expose ici une limite de notre BDD: elle n'est pas exhaustive. Rien que les jeux Pokémons de la GBA se sont vendus à 50 millions de copies, et notre BDD n'y fait pas référence.

IV..../ “Les amis de mes amis sont mes amis”

Maintenant, attaquons nous à des requêtes plus complexes. Comme mentionné en page 5, les personnages peuvent avoir une relation d’alliance, c.à.d, que deux personnages soient en situation amicale. Maintenant, trouvons les personnages p1 alliés au premier degré à un Personnage P(Plutôt simple), et les personnages p2 alliés au second degré à p, c.à.d,les personnages p2 alliés à p1, qui sont alliés à P. Comment modéliser cette situation? Eh bien, avec ces quelques lignes de Cypher:

```
MATCH(p1:personnage)-[:allie]->(p2:personnage)<-[:allie]-(p3:personnage)
WHERE  NOT (p1)-[:allie]->(p3)
AND NOT (p1) = (p3)
RETURN  (p1.nom) AS `Personnage`,collect (DISTINCT p2.nom) as `Allié au premier degré`, collect (DISTINCT p3.nom) AS `Allié au deuxième degré`
```

Le navigateur nous renvoie le tableau de listes suivant:

Personnage	Allié au premier degré	Allié au deuxième degré
"Princesse Daisy"	["Mario", "Luigi", "Bebe Mario", "Bebe Peach", "Princesse Peach", "Bebe Luigi"]	["Paper Luigi", "Harmonie", "Diddy Kong", "Donkey Kong", "Paper Mario", "Yoshi", "Toad"]
"Princesse Peach"	["Toad", "Bebe Mario", "Bebe Luigi", "Mario", "Paper Luigi", "Paper Mario", "Harmonie", "Princesse Daisy", "Yoshi", "Luigi"]	["Bebe Peach", "Donkey Kong", "Diddy Kong"]
"Harmonie"	["Princesse Peach", "Mario", "Luigi", "Bebe Peach", "Bebe Luigi", "Bebe Mario"]	["Paper Mario", "Yoshi", "Toad", "Princesse Daisy", "Paper Luigi", "Diddy Kong", "Donkey Kong"]
"Toad"	["Bebe Mario", "Princesse Peach", "Luigi", "Mario", "Bebe Peach", "Bebe Luigi"]	["Donkey Kong", "Diddy Kong", "Princesse Daisy", "Harmonie", "Paper Mario", "Paper Luigi"]

NB: Cette requête montre les limites de notre BDD dans 2 situations:

Exemple 1:

"Roi Dadidou"	["Meta Knight"]	["Kirby"]

On a le “Roi DaDiDou” qui est allié à Meta Knight, et Meta Knight qui est allié à Kirby. Or, dans l’univers Kirby, le Roi DaDiDou est l’ennemi juré de Kirby. Ici, c’est Meta Knight qui pose problème, ayant un peu un rôle de traître: il est vassal de Roi DaDidou, mais il est aussi allié de Kirby. Notre Base de données ne prend pas en compte cette situation.

Exemple 2: Mais où est Bowser?

On remarque que lors de l'exécution, le navigateur Neo4j nous renvoie un tableau, qui omet le fameux antagoniste de Mario. Pour quelle raison? Il se trouve que Bowser ne dispose pas de beaucoup d'alliés… Mise à part part son armée (Armée de Bowser) et son fils (Bowser Jr), il n'a aucun autre lien d'alliance au premier degré. Si on regarde ces 2 derniers personnages cités précédemments (son fils et son armée), eux non plus n'ont pas d'alliés à part Bowser. Son absence est due à la 3ème ligne de la requête:

```
AND NOT (p1) = (p3)
```

Ici, on interdit le lien au premier degré, ce qui explique son absence.

V...../ “Les ennemis de mes amis sont mes ennemis”

De manière analogue à la situation précédente, trouvons des relations d’antagonisme de manière indirecte. Ainsi, si un personnage P sait qu’un personnage p3 est l’ennemi de son allié p2, P sait qu’il ne doit pas être dans le camp de p3! (Si l’on omet les relations de traîtres/double camp, comme celles de Meta Knight (Univers Kirby) et Lakitu (Univers Mario). De même que précédemment, on utilise les listes avec collect():

```
MATCH(p1:personnage)-[:allie]->(p2:personnage)<-[:antagoniste]-(p3:personnage)
WHERE  NOT (p1)-[:allie]->(p3)
AND NOT (p1)=(p3)
RETURN (p1.nom) AS `Personnage` ,collect (DISTINCT p2.nom) as `Allié au premier degré du personnage` ,collect
(DISTINCT p3.nom)  as `ennemi au deuxième degré du personnage`
```

Personnage	Allié au premier degré du personnage	ennemi au deuxième degré du personnage
"Mario"	["Toad", "Yoshi", "Bebe Mario", "Harmonie", "Bebe Peach", "Princesse Peach", "Bebe Luigi", "Donkey Kong", "Luigi"]	["Bowser", "Roi Boo", "Maskass", "Bebe Bowser", "Wario", "Armée de Bowser", "Waluigi"]
"Luigi"	["Toad", "Bebe Luigi", "Donkey Kong", "Yoshi", "Bebe Mario", "Mario", "Harmonie", "Bebe Peach", "Princesse Peach"]	["Bowser", "Wario", "Roi Boo", "Bebe Bowser", "Armée de Bowser", "Waluigi", "Maskass"]
"Bowser"	["Maskass", "Armée de Bowser", "Bebe Bowser"]	["Bebe Peach", "Bebe Mario", "Bebe Luigi", "Mario", "Princesse Peach", "Luigi"]
"Princesse Daisy"	["Bebe Luigi", "Bebe Peach", "Luigi", "Princesse Peach", "Mario", "Bebe Mario"]	["Wario", "Donkey Kong", "Roi Boo", "Bebe Bowser", "Bowser", "Armée de Bowser", "Waluigi", "Maskass"]
"Princesse Peach"	["Yoshi", "Toad", "Roi Boo", "Bebe Mario", "Bebe Luigi", "Mario", "Luigi", "Harmonie"]	["Bowser", "Bebe Peach", "Maskass", "Bebe Bowser", "Wario", "Armée de Bowser", "Waluigi", "Donkey Kong"]

Explicitons quelques résultats: on voit que Mario a pour alliés Toad, Yoshi, Bébé Mario, Harmonie, etc (liste 1). Bowser, Roi Boo, Maskass, Bébé Bowser, Wario, etc (liste 2) étant ennemis au second degré de la liste 1, ils sont donc ennemis de Mario.

NB: ici, on omet les liens d’antagonisme au premier degré. En effet:

"Donkey Kong"	["Bebe Mario", "Mario", "Bebe Luigi", "Luigi"]	["Roi Boo", "Maskass", "Bowser", "Bebe Bowser", "Wario", "Armée de Bowser",

Dans la licence Donkey Kong, le primate admet pour ennemi juré King K. Rool, qui n'apparaît pas dans sa liste d’ennemis. Ici, on n’affiche que les ennemis d’amis, pour prévoir une vengeance pour son ami par exemple :-)

VI……/ “Les ennemis de mes ennemis sont mes amis”

Que faire si un personnage P est trop fort? Eh bien, comme il est souvent dit, l’union fait la force: p1 et p2, tous deux ennemis de P, devraient s’allier afin de le battre! Mettons les en relation en Cypher:

```
MATCH(p1:personnage)-[:antagoniste]->(P:personnage)<-[:antagoniste]-(p2:personnage)
WHERE NOT (p1)-[:allie]->(p2)
AND NOT (p1)=(p2)
RETURN (p1.nom) AS `Personnage 1`, collect(DISTINCT P.nom) as `Ennemi Commun`, collect(DISTINCT p2.nom) as `Personnage 2`
```

Résultats (partiels):

Personnage 1	Ennemi Commun	Personnage 2
"Wario"	["Mario", "Luigi", "Princesse Peach", "Bebe Mario", "Bebe Luigi", "Bebe Peach"]	["Armée de Bowser", "Donkey Kong", "Maskass", "Waluigi", "Bowser", "Bebe Bowser", "Roi Boo"]
"Donkey Kong"	["Mario", "Luigi", "Bowser", "Bebe Mario", "Bebe Luigi"]	["Armée de Bowser", "Wario", "Maskass", "Waluigi", "Bowser", "Bebe Bowser", "Roi Boo", "Princesse Peach", "Harmonie", "Toad", "Bebe Peach", "Yoshi"]

Interprétation: Ici, si l’on prend la première ligne par exemple, on pourrait prévenir Wario que si il souhaite enfin battre Mario, Luigi et leur bande, il pourrait s’allier avec un ou des personnages de la liste Personnage 2!

VII……./ Classement des meilleurs Jeux

Désormais, nous ne voulons plus accéder à des données “sérieuses”, pour le travail: On veut la crème de la crème, le top du top: comment trouver le meilleur jeu parmi tous? On rappelle que l’un des attributs des jeux est leur note donnée par Metacritic. Voici le top 15 de ce que notre base a à nous offrir:

```
1 Match(j:Jeux)
2 WHERE j.note IS NOT NULL
3 return DISTINCT j.titre as Nom ,j.note as Note,j.annee as Annee
4 ORDER BY j.note DESC limit 15
```

Nom	Note	Annee
"The Legend Of Zelda: Ocarina Of Time"	99	1998
"The Legend Of Zelda: Breath Of The Wild"	97	2017
"The Legend Of Zelda: The Wind Waker"	97	2003
"Super Metroid"	97	1994
"Super Mario Galaxy"	97	2007
"Mario & Sonic aux Jeux Olympiques"	97	2007
"Super Mario Odyssey"	97	2017
"Metroid Prime"	97	2002
"Super Mario 3D World"	96	2013
"The Legend Of Zelda: A Link to the Past"	95	1991
"The Legend Of Zelda: Majoras Mask"	95	2000
"Metal Gear Solid"	94	1998
"Super Mario 64"	94	1996
"Super Smash Bros. Ultimate"	93	2018
"Super Mario 3D Land"	93	2011

4. Fin

>Conclusion:

Pour conclure, nous pouvons affirmer avoir construit une base de donnée intéressante sur l'univers, et aussi assez conséquente: si nous disposions de quelques semaines de plus, nous aurions pu la rendre très exhaustive. Nous aurions pu la rendre plus complexe (et comme dit juste précédemment) complète, mais il a fallu rentrer dans le cadre scolaire afin de pouvoir finir dans les temps.

Aussi, nous avons pu approfondir nos connaissances de l'univers et de l'histoire de Nintendo, entreprise nous tenant à coeur, autant en tant qu'amateurs de jeux vidéos qu'en tant que futurs ingénieurs.

>Appréciation:

Pour nous tous, le monde de la base de donnée fut nouveau. L'apprentissage "alternatif" (la majorité commençant par un autre type de base que celle de graphe) à changé notre vision des infrastructures de stockage: les informations, la manière de les exploiter, leurs relations; tout est un peu plus clair, et tout à coup plus intéressant. De ce fait, nous sommes reconnaissants envers nos instituteurs, Monsieur Natowicz et Monsieur Cela, de nous avoir proposé cette introduction aux bases de données de graphe, et de nous avoir fait découvrir un autre secteur de l'informatique.

Quand à nos remarques, elles sont minimes. L'atelier est très bien mené, si ce n'est que le temps proposé pour la création de la base de donnée est un peu limité afin de créer quelque chose de conséquent. Nous avons trouvé l'approche par MOOC bien faite, et le Cypher fut très rapidement assimilé. Aussi, la partie plus technique, c.à.d concernant la programmation en java et l'algorithme plus théorique des parcours/création de graphe, fut, à notre goût, plus particulièrement intéressante. Enfin, l'organisation en projet de groupe est évidemment une bonne idée, à conserver.

>Note finale:
Trouvez la totalité de notre code en ligne, à cette URL ou à ce QR code:

https://github.com/avmolaei/db_nintendo