Étude de données NoSQL

TP Réalisé par : Damien CARRIER — Clément SAVINAUD

Pour le 6 mars 2022

Lien Github

Lancement du projet :

```
# Depuis la racine du projet :
cd server/
npm i && npm start

cd ../client
docker-compose up -d
npm i && npm start
```

Contexte

L'objectif de ce TP est de modéliser, implémenter et tester en volumétrie un service d'analyse de comportement d'achat d'utilisateurs regroupés dans un réseau social.

Cette implémentation et ces tests seront effectués *avec un SGBDR traditionnel **et** une base NoSQL* afin de comparer les avantages, inconvénients et performances de chaque solution.

Les tests devront pouvoir être effectués par un utilisateur sans intervention dans le code donc il faut également développer un logiciel (Web ou client lourd au choix) permettant de lancer des requêtes sur les 2 bases avec mesure/affichage des temps de réponse.

Cahier des charges

Les utilisateurs sont regroupés au sein d'un réseau social leur permettant d'avoir des cercles de followers. Le lien de « follows » devra être orienté. En termes de volumétrie pour cette phase de test, on peut envisager de créer 1M utilisateurs. Chaque utilisateur pourrait avoir environ 0 – 20 followers directs.

Attention : sur plusieurs niveaux, un utilisateur peut être son propre follower ! Il faut prendre en compte ce point pour éviter, lors des recherches, de doublonner les résultats. Concernant les achats, la base pourrait contenir 10 000 références de produits. Pour les achats, chaque utilisateur pourrait avoir commandé entre 0 et 5 produits parmi ces références.

Projet

Vous pourrez retrouver ce projet en suivant le lien suivant :

```
https://github.com/MirakuSan/imt-social-network-analysis
```

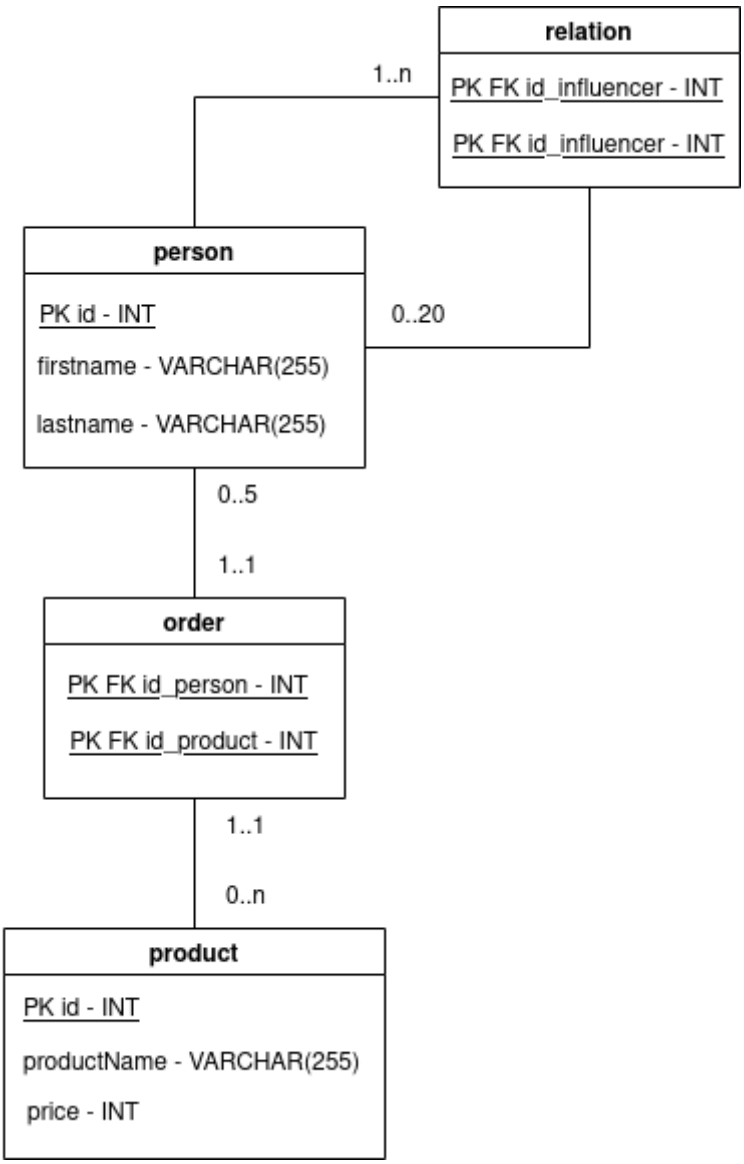
Pour ce projet, nous avons réalisé une application web à l'aide de **ReactJS** pour le front et de **ExpressJS** pour le back.

Pour les bases de données, nous avons fait le choix d'utiliser **MariaDB** pour la BDD SQL et **Neo4J** pour la BDD NoSQL.

Entités

MariaDB

Modèle Conceptuel de Données :



Person	
Attribute	Type
ID	Integer (Auto Incrément)
firstName	string
lastName	string

Product	
Attribute	Type
ID	Integer (Auto Incrément)
productName	string

Relation	
Attribute	Type
id_influencer	string
id_follower	string

Orders	
Attribute	Type
id_person	Integer
id_product	Integer

NoSQL

Il n'y a pas vraiment d'entité pour le NoSQL mais voilà à quoi pourrait ressembler ces entités :

Person	
Attriute	Type
firstName	string
lastName	string

Product	
Attriute	Type
productName	string

Requêtes

Pour faire nos recherches nous réaliserons 3 requêtes de recherche que nous transformerons en SQL et en Neo4J qui sont :

1. Obtenir la liste et le nombre des produits commandés par les cercles de followers d'un individu (niveau 1, ..., niveau n) → cette requête permet d'observer le rôle d'influenceur d'un individu au sein du réseau social pour le déclenchement d'achats.
2. Même requête, mais avec spécification d'un produit particulier → cette requête permet d'observer le rôle d'influenceur d'un individu suite à un « post » mentionnant un article spécifique.
3. Pour une référence de produit donné, obtenir le nombre de personnes l'ayant commandé dans un cercle de followers « ***orienté*** » de niveau n (à effectuer sur plusieurs niveaux : 0, 1, 2 ...) → permet de rechercher les produits « viraux », c'est-à-dire ceux qui se vendent le plus au

sein de groupes de followers par opposition aux achats isolés pour lesquels le groupe social n'a pas d'impact

Réalisations

Insertions :

Dans les deux cas de type de BDD nous utilisons une méthode assez similaire pour la création des données.

- Insertion de **X** Personnes.
- Génération d'un tableau de 0 à 20 relations pour ces **X** personnes
- Insertion de ces relations créées
    - Pour MariaDB nous utilisons des batchs de 100 000 relations pour l'insertion afin de ne pas faire sauter le container.
- Insertion de **Y** produits
- Génération d'un tableau de 0 à 5 commandes pour ces **X** personnes
- Insertions des commandes créées
    - Idem nous utiliserons pour MariaDB des batchs de 100 000 commandes pour l'insertion.

MariaDB

- Insertions de personnes :

```
const insertPersons = async (arrayPerson) => {
    const batchSize = 100000;
    let insertIndex = 0;
    let duration = 0;

    while (insertIndex < arrayPerson.length) {
        // create batch
        let request = 'INSERT INTO person (firstname, lastname) VALUES';
        const nbPersonToInsert = arrayPerson.length - insertIndex;
        const maxVal = nbPersonToInsert <= batchSize ? nbPersonToInsert :
batchSize;

        for (let i = 0; i < maxVal - 1; i++) {
            let person = arrayPerson[insertIndex];

            request += '("' + person.firstName + '", "' + person.lastName +
'"),';

            insertIndex++;
        }

        request += `("` + arrayPerson[insertIndex].firstName + `", "` +
arrayPerson[insertIndex].lastName + `");`;
        insertIndex++;

        const result = await executeQuery(request);

        if (result.status === '500') {
            return result;
```

```
        }

        duration += result.time;
    }

    return {
        'status': '200',
        'time': duration,
        'data': 'L\'insertion de ' + arrayPerson.length + ' a été réalisée'
    };
}
```

- Insertion de relations :

```
const insertRelations = async (arrayRelations) => {
    const batchSize = 100000;
    let insertIndex = 0;
    let duration = 0;

    while (insertIndex < arrayRelations.length) {
        // create batch
        let request = 'INSERT INTO relation (id_influencer, id_follower)
VALUES';
        const nbRelationToInsert = arrayRelations.length - insertIndex;
        const maxVal = nbRelationToInsert < batchSize ? nbRelationToInsert :
batchSize;

        for (let i = 0; i < maxVal - 1; i++) {
            let relation = arrayRelations[insertIndex];

            request += '(' + relation.follower + ', ' + relation.followed +
'),';

            insertIndex++;
        }

        request += `(` + arrayRelations[insertIndex].follower + `, ` +
arrayRelations[insertIndex].followed + `);`;
        insertIndex++;

        const result = await executeQuery(request);

        if (result.status === '500') {
            return result;
        }

        duration += result.time;
    }

    return {
        'status': '200',
        'time': duration,
        'data': 'L\'insertion de ' + arrayRelations.length + ' a été réalisée'
    };
}
```

- Insertion de produits :

```
const insertProducts = async (arrayProduct) => {
    const batchSize = 100000;
    let insertIndex = 0;
    let duration = 0;

    while (insertIndex < arrayProduct.length) {
        let request = 'INSERT INTO product (productName, price) VALUES ';
        const nbProductToInsert = arrayProduct.length - insertIndex;
        const maxVal = nbProductToInsert < batchSize ? nbProductToInsert :
batchSize;

        for (let i = 0; i < maxVal - 1; i++) {
            let product = arrayProduct[insertIndex];
            request += '("' + product.productName + '", "' + product.price +
'"),';
            insertIndex++;
        }

        request += '("' + arrayProduct[insertIndex].productName + '", "' +
arrayProduct[insertIndex].price + '");';
        insertIndex++;

        const result = await executeQuery(request);

        if (result.status === '500') {
            return result;
        }

        duration += result.time;
    }

    return {
        'status': '200',
        'time': duration,
        'data': 'L\'insertion de ' + arrayProduct.length + ' produits a été
réalisée.'
    };
}
```

- Insertion de commandes :

```
const insertPurchase = async (arrayPurchase) => {
    const batchSize = 100000;
    let insertIndex = 0;
    let duration = 0;

    while (insertIndex < arrayPurchase.length) {
        let request = 'INSERT INTO orders (id_person, id_product) VALUES ';
        const nbRelationToInsert = arrayPurchase.length - insertIndex;
        const maxVal = nbRelationToInsert < batchSize ? nbRelationToInsert :
batchSize;

        for (let i = 0; i < maxVal - 1; i++) {
            let relation = arrayPurchase[insertIndex];

            request += '(' + relation.person + ', ' + relation.product + '),';
```

```
            insertIndex++;
        }

        request += `(` + arrayPurchase[insertIndex].person + `, ` +
arrayPurchase[insertIndex].product + `);`;
        insertIndex++;

        const result = await executeQuery(request);

        if (result.status === '500') {
            return result;
        }

        duration += result.time;
    }

    return {
        'status': '200',
        'time': duration,
        'data': 'Les relations de personnes -> produits ont été insérées'
    };
}
```

Neo4j

- Insertion de personnes ou produit :

```
const insertObject = async function (arrayObject, tableName) {
    const session = driver.session();
    let arrayDatas;

    if (tableName === 'Person') {
        arrayDatas = arrayObject.map((person) => ({
            firstname: person.firstName, lastname: person.lastName
        }));
    } else if (tableName === 'Product') {
        arrayDatas = arrayObject.map((product) => ({
            productName: product.productName, price: product.price
        }));
    }

    let startTimeCreation;

    try {
        await session.writeTransaction((tx) => {
            tx.run(
                'UNWIND $props AS map CREATE (p:' + tableName + ') SET p =
map',
                {
                    props: arrayDatas,
                }
            );

            startTimeCreation = Date.now();
```

```
        });

        const endTimeCreation = Date.now();
        await session.close();
        const timeCreation = endTimeCreation - startTimeCreation;

        let dataString = 'Les ' + arrayObject.length;

        if (tableName === 'Person') {
            dataString += ' personnes ont été correctements ajoutées';
        } else if (tableName === 'Product') {
            dataString += ' produits ont été correctements ajoutés';
        }

        return {
            status: 200,
            data: dataString,
            time: timeCreation / 1000
        };
    } catch (error) {
        console.error(error);

        return {
            status: 409,
            data: error,
            time: null,
        };
    }
};
```

Insertion de relations :

```
const insertRelations = async (tabRelations) => {
    try {
        const session = driver.session();

        let start;
        await session.writeTransaction((tx) => {
            tx.run(
                `UNWIND $listFollow as followTab
         MATCH (p1:Person) WHERE ID(p1) = followTab.influencer
         UNWIND followTab.follower as followerTab
         MATCH (p2:Person) WHERE ID(p2) = followerTab
         CREATE (p1)-[:Relation]->(p2)
         RETURN p1, p2`,
                {
                    listFollow: tabRelations
                });

            start = Date.now();
        });
        const end = Date.now();
        const duration = (end - start) / 1000;
        await session.close();

        return {time_creation_relations: duration};
    } catch (error) {
```

```
        console.error(error);

        return {
            status: 409,
            data: error,
            time: null,
        };
    }
}
```

Insertion de commandes :

```
const insertOrders = async (tabOrders) => {
    try {
        const session = driver.session();

        let start;
        await session.writeTransaction((tx) => {
            tx.run(
                `UNWIND $ordersList as orderTab
               MATCH (p1:Person) WHERE ID(p1) = orderTab.idPerson
               UNWIND orderTab.ordersList as productTab
               MATCH (p2:Product) WHERE ID(p2) = productTab
               CREATE (p1)-[:Order]->(p2)
               RETURN p1,p2`,
                {
                    ordersList: tabOrders
                });
            start = Date.now();
        });

        const end = Date.now();
        const duration = (end - start) / 1000;
        await session.close();

        return {
            status: 200,
            time: duration,
            data: {}
        };
    } catch (error) {
        console.error(error);

        return {
            status: 409,
            data: error,
            time: null,
        };
    }
}
```

Résultats :

- MariaDB :


```
{
    "response": {
        "durationGenerateStructure": {
            "status": 200,
            "data": "done",
            "time": 0.12s
        },
        "durationInsertPerson": {
            "durationInsertPersons": {
                "status": "200",
                "time_insertions_persons": 6.383s,
                "data": "L'insertion de 1 000 000 personnes a été réalisée"
            },
            "durationInsertRelations": {
                "status": "200",
                "time_creation_relations": 516.538s,
                "data": "L'insertion de 9 985 147 relations a été réalisée"
            }
        },
        "durationInsertProduct": {
            "durationInsertProducts": {
                "status": "200",
                "time_insertion_product": 0.353s,
                "data": "L'insertion de 10 000 produits a été réalisée."
            },
            "durationInsertPurchase": {
                "status": "200",
                "time_creation_orders": 59.18899999999999s,
                "data": "Les relations de personnes -> produits ont été
insérées"
            }
        }
    }
}
```


- Neo4J :


```
{
    "resultInsertPersons": {
        "status": 200,
        "data": "Les 1000000 personnes ont été correctements ajoutées",
        "time_insertions_persons": 34.349s
    },
    "resultInsertRelations": {
        "time_creation_relations": 189.436s
    },
    "resultGenerationProduct": {
        "resultInsertProduct": {
            "status": 200,
            "data": "Les 10000 produits ont été correctements ajoutés",
            "time_insertion_product": 1.444s
        },
        "resultInsertRelations": {
```

```
            "status": 200,
            "time_creation_orders": 72.64s,
        }
    }
}
```

Nous pouvons remarquer que globalement le temps d'insertions est plus rapide sur Neo4j pour les ~10 millions de
relations que pour Mysql. Pour la partie sur les produits, les temps sont globalement identiques. Par contre, concernant
l'insertion de 1 million de personnes, Mysql est plus rapide que Neo4j.

Recherches

Requête 1

- MariaDB :

```
const getProductsOrderedByFollowersMysql = async (idInfluencer, depth, limit)
=> {
    if (idInfluencer === 0) {
        return {
            status: '500',
            'data': 'Error 500'
        };
    }

    depth++

    let request = `WITH RECURSIVE person_req(id, depth) AS (
            SELECT id, 0 FROM person WHERE id=${idInfluencer}
            UNION ALL
            SELECT p.id, req.depth + 1
            FROM person p, person_req req, relation r
            WHERE p.id=r.id_influencer AND req.id = r.id_follower AND req.depth
< ${depth}
        )
        SELECT pr.id, pr.productName, COUNT(o.id_person) AS nbOrders
        FROM product pr
        JOIN orders o ON pr.id=o.id_product
        WHERE o.id_person IN (
            SELECT id FROM person_req
        )
        GROUP BY pr.id ORDER BY nbOrders DESC LIMIT ` + limit;

    return await executeQuery(request);
}
```

- Neo4j :

```
const getProductsOrderedByFollowersNeo4j = async (influencer, depth, limit) =>
{
    let products = [];
```

```
    try {
        const session = driver.session();
        const query = ` MATCH (:Person {firstname: "${influencer.firstName}",
lastname: "${influencer.lastName}"})<-[:Relation *0..${depth}]-(p:Person)
              WITH DISTINCT p
              MATCH (p)-[:Order]->(n:Product)
              RETURN n.productName, COUNT(*)
              LIMIT ${limit}`;

        const start = Date.now();
        const data = await session.run(query, {});

        for (let i = 0; i < data.records.length; i++) {
            products.push(
                {
                    name: data.records[i].get(0),
                    nbrOrders: data.records[i].get(1).low
                }
            )
        }

        await session.close();
        const duration = Date.now() - start;

        return {
            status: 200,
            time: duration / 1000,
            influencer: influencer.firstName + ' ' + influencer.lastName,
            result: products
        }
    } catch (e) {
        console.error(e);

        return {
            status: 409,
            data: e,
            time: null,
        };
    }
}
```

Résultats

Tous les résultats sont donnés pour une base contenant ~1 million de personnes, ~10 millions de relations, ~10 000
produits et ~2.5 millions de commandes.

- MariaDB — Profondeur 0 (limite 2 résultats) :


```
{
    "status": "200",
    "time": "0.03s",
    "influenceur": 50906,
    "data": [
        {
```

```
            "id": 5696,
            "productName": "Sleek Metal Mouse",
            "nbOrders": 1
        },
        {
            "id": 6659,
            "productName": "Refined Rubber Tuna",
            "nbOrders": 1
        }
    ]
}
```


- Neo4J — Profondeur 0 (limite 2 résultats) :


```
{
    "status": 200,
    "time": "0.621s",
    "influencer": "Kari Rice",
    "data": [
        {
            "name": "Sleek Concrete Computer",
            "nbrOrders": 1
        },
        {
            "name": "Sleek Steel Salad",
            "nbrOrders": 1
        }
    ]
}
```


- MariaDB — Profondeur 1 (limite 2 résultats) :


```
{
    "status": "200",
    "time": "0.03s",
    "influenceur": 657078,
    "data": [
        {
            "id": 630,
            "productName": "Ergonomic Soft Chair",
            "nbOrders": 2
        },
        {
            "id": 5062,
            "productName": "Intelligent Cotton Table",
            "nbOrders": 1
        }
    ]
}
```


- Neo4J — Profondeur 1 (limite 2 résultats) :


```
{
    "status": 200,
    "time": "0.63s",
    "influencer": "Merl Watsica",
```

```
    "data": [
        {
            "name": "Fantastic Cotton Bacon",
            "nbrOrders": 1
        },
        {
            "name": "Incredible Steel Shoes",
            "nbrOrders": 1
        }
    ]
}
```


- MariaDB — Profondeur 3 (limite 2 résultats) :


```
{
    "status": "200",
    "time": "1.588s",
    "influenceur": 583774,
    "data": [
        {
            "id": 2995,
            "productName": "Incredible Wooden Shirt",
            "nbOrders": 11
        },
        {
            "id": 3557,
            "productName": "Awesome Concrete Chair",
            "nbOrders": 10
        }
    ]
}
```


- Neo4J — Profondeur 3 (limite 2 résultats) :


```
{
    "status": 200,
    "time": "0.758s",
    "influencer": "Arturo Mayer",
    "data": [
        {
            "name": "Handcrafted Metal Shirt",
            "nbrOrders": 12
        },
        {
            "name": "Incredible Wooden Car",
            "nbrOrders": 11
        }
    ]
}
```


- MariaDB — Profondeur 5 (limite 2 résultats) :


```
{
    "status": "200",
    "time": "77.306s",
    "influenceur": 889170,
```

```
    "data": [
        {
            "id": 9883,
            "productName": "Gorgeous Concrete Soap",
            "nbOrders": 176
        },
        {
            "id": 4098,
            "productName": "Tasty Concrete Chips",
            "nbOrders": 171
        }
    ]
}
```


- Neo4J — Profondeur 5 (limite 2 résultats) :


```
{
    "status": 200,
    "time": "2.149s",
    "influencer": "Francisco Thompson",
    "data": [
        {
            "name": "Small Concrete Keyboard",
            "nbrOrders": 126
        },
        {
            "name": "Ergonomic Wooden Tuna",
            "nbrOrders": 120
        }
    ]
}
```


Requête 2

- MariaDB :

```
const getProductsOrderedByFollowersAndByProductMysql = async (idInfluencer,
idProduct, depth) => {
    if (!idInfluencer) {
        return {
            'status': '500',
            'data': 'Error 500 : influenceur'
        };
    }

    if (!idProduct) {
        return {
            'status': '500',
            'data': 'Error 500 : product'
        };
    }

    depth++

    let request = `WITH RECURSIVE person_req(id, depth) AS (
            SELECT id, 0 FROM person WHERE id=${idInfluencer}
```

```
            UNION ALL
            SELECT p.id, req.depth + 1
            FROM person p, person_req req, relation r
            WHERE p.id=r.id_influencer AND req.id = r.id_follower AND req.depth
< ${depth}
        )
        SELECT pr.id, pr.productName, COUNT(o.id_person) AS nbOrders
        FROM product pr
        JOIN orders o ON pr.id=o.id_product
        WHERE o.id_person IN (
            SELECT id FROM person_req
        ) AND o.id_product=${idProduct}
        GROUP BY pr.id`;

    return await executeQuery(request);
}
```

- Neo4j :

```
const getProductsOrderedByFollowersAndByProductNeo4j = async (influencer,
productName, depth) => {
    try {
        const session = driver.session();
        const query = ` MATCH (:Person{firstname: "${influencer.firstName}",
lastname: "${influencer.lastName}"})<-[:Relation *0..${depth}]-(p:Person)
                WITH DISTINCT p
                MATCH (p)-[:Order]->(n:Product {productName:
"${productName}"})
                RETURN n.productName, COUNT(*)`;

        const start = Date.now();
        const data = await session.run(query, {});
        let count = 0;

        if (data.records[0]) {
            count = data.records[0].get(1);
            count = count.low
        }

        await session.close();
        const duration = Date.now() - start;

        return {
            status: 200,
            time: duration / 1000,
            productName: productName,
            influencer: influencer.firstName + ' ' + influencer.lastName,
            nbOrders: count
        }
    } catch (e) {
        console.error(e);

        return {
            status: 409,
            data: e,
            time: null,
        };
```

```
    }
}
```

Résultats

- MariaDB — Profondeur 0 :

```
{
    "status": "200",
    "time": "0.015s",
    "influenceur": 470780,
    "product": 6789,
    "nbOrders": 1
}
```

- Neo4J — Profondeur 0 :

```
{
    "status": 200,
    "time": "0.764s",
    "influencer": "Jace Schinner",
    "productName": "Intelligent Cotton Soap",
    "nbOrders": 0
}
```

- MariaDB — Profondeur 1 :

```
{
    "status": "200",
    "time": "0.03s",
    "influenceur": 792399,
    "product": 4733,
    "nbOrders": 1
}
```

- Neo4J — Profondeur 1 :

```
{
    "status": 200,
    "time": "0.813s",
    "influencer": "Pascale Gutmann",
    "productName": "Generic Frozen Tuna",
    "nbOrders": 0
}
```

- MariaDB — Profondeur 3 :

```
{
    "status": "200",
    "time": "2.523s",
    "influenceur": 724386,
    "product": 5416,
    "nbOrders": 5
}
```

- Neo4J — Profondeur 3 :


```
{
    "status": 200,
    "time": "0.85s",
    "influencer": "Florine Swaniawski",
    "productName": "Handcrafted Rubber Gloves",
    "nbOrders": 0
}
```


- MariaDB — Profondeur 5 :


```
{
    "status": "200",
    "time": "166.814s",
    "influenceur": 813931,
    "product": 3002,
    "nbOrders": 205
}
```


- Neo4J — Profondeur 5 :


```
{
    "status": 200,
    "time": "4.892s",
    "influencer": "Jeff Kuhlman",
    "productName": "Gorgeous Wooden Chicken",
    "nbOrders": 142
}
```


Récapitulatif

Insertions

	MariDB	Neo4j
Person	6.383s	34.349s
Product	0.353s	1.444s
Orders	59.1889s	72.64s
Relations	516.538s	189.436s

Recherches

- Requête 1 :

	MariDB	Neo4j
Profondeur 0	0.03s	0.621s
Profondeur 1	0.03s	0.63s
Profondeur 3	1.588s	0.758s
Profondeur 5	77.306s	2.149s

- Requête 2 :

	MariDB	Neo4j
Profondeur 0	0.015s	0.764s
Profondeur 1	0.03s	0.813s
Profondeur 3	2.523s	0.85s
Profondeur 5	166.814s	4.892s

Conclusion

A travers ce projet, nous avons pu comparer l'utilisation d'une base de données type SQL et une base NoSQL type graphe. Pour chacune de ces technologies nous avons pu constater plusieurs points forts, mais également des points faibles.

Côté base de données SQL, nous avons pu constater que le temps d'insertion de données dans une table est très performante. En effet, l'insertion de personnes ou de produits est plutôt rapide et reste tout à fait acceptable. Cependant, lorsqu'on parle de création de relations complexes entre les données, on observe bien des problèmes de performances avec des insertions très lentes dans les tables Order et Relation.

La base de données Neo4j qui est une base de type graphe, permet de s'affranchir des limites posées par une base SQL. En effet, malgré un temps d'insertion des données important sur des données sans relations, on a pu observer un énorme gain au niveau de la création de relations entre ces données.

Ce gain de performances avec la base de données NoSQL se retrouve également sur la sélection de données. La récupération de données relationnelles est plus simple et performante avec la base Neo4j grâce à son architecture orientée graphe, tandis que la base SQL est limitée en terme de complexité de requêtes à ce niveau.

Par conséquent, la réalisation de ce projet nous a permis de prendre connaissance d'une nouvelle technologie de stockage de données et avons pu en mesurer l'intérêt.