Documents Structurés et NoSQL - INFOA - P1G2
Mamisoa RANDRIANARIMANANA - Xianxiang ZHANG

TP Neo4J & SQLite

Introduction

Dans le cadre du cours Documents structurés et NoSQL, le TP vise à explorer Cypher, langage conçu pour les bases de données orientées graphe, et le confronter à SQL, basé sur l'algèbre relationnelle. Cette comparaison se fera à travers l'utilisation de deux systèmes de gestion de bases de données distincts : Neo4J, avec Cypher, et SQLite, une base de données relationnelle. Neo4J, orienté graphe, utilise Cypher, langage basé sur le filtrage par motif, idéal pour interroger les données dans un contexte graphique. Notre exploration se déroulera dans une petite base de données simulant un réseau social d'étudiants (Highschoolers) avec leurs liens d'amitié et de "likes", disponibles en fichiers CSV et un script SQL (social-data.sql). L'objectif principal est la comparaison des expressions de requête entre les deux environnements (SQL vs. Cypher) pour répondre à diverses questions relatives à la base de données sociale.

1. Find the ID's and the names of all students who are friends with someone named Jordan. Order the answer by student ID's.
Answer: **********
SQL:

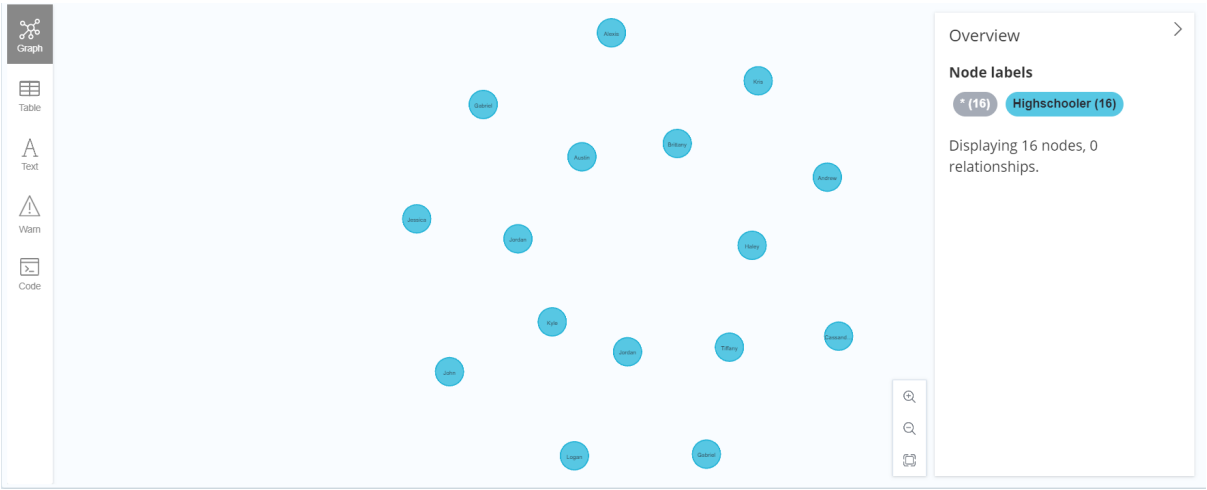
```sql
SELECT name, ID
FROM ( SELECT ID2
FROM (SELECT Jord.id,Jord.name
FROM (SELECT * FROM Highschooler WHERE name='Jordan') as
Jord) as Jord_Ami
INNER JOIN Friend ON Jord_Ami.ID = Friend.ID1
) as AMI_ID
INNER JOIN Highschooler ON AMI_ID.ID2 = Highschooler.ID
ORDER BY ID;
```

name	ID
Tiffany	1381
Logan	1661
Gabriel	1689
Andrew	1782
Kyle	1934

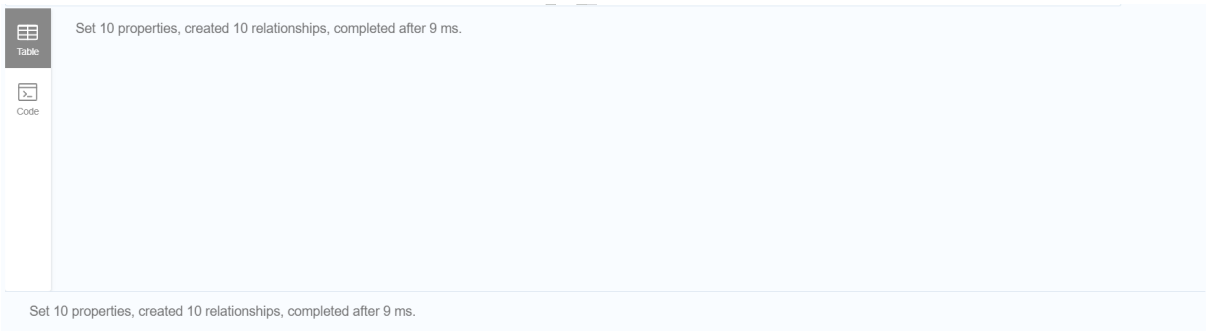
Les réponses sont: Tiffany, Gabriel , Logan, Andrew, Kyle

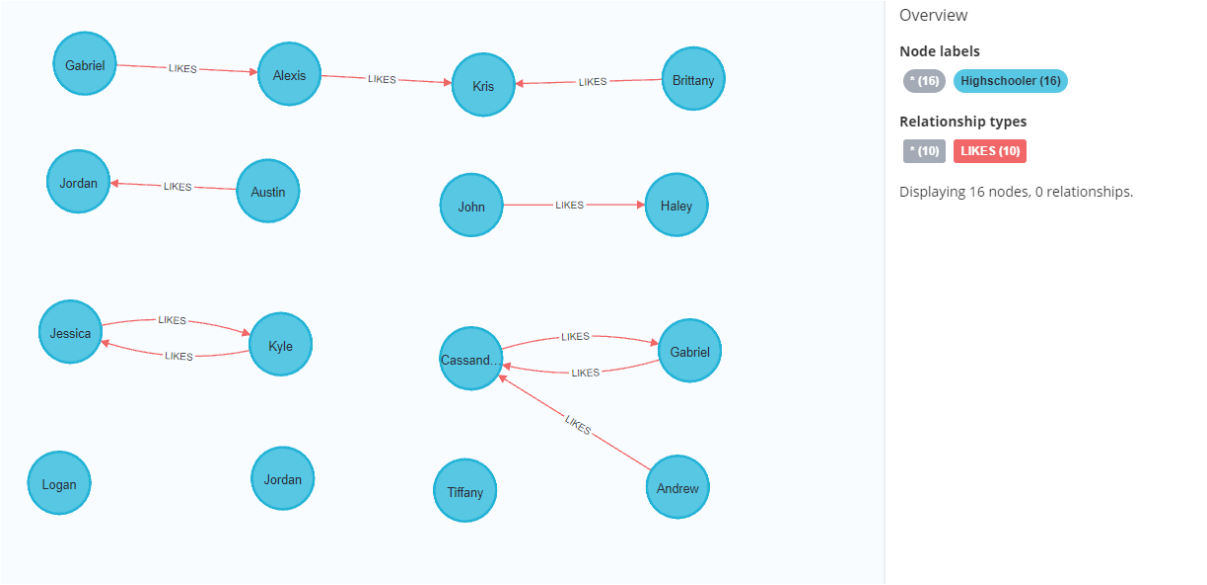
Neo4j:
-Préparer la base de données graphique:

```
LOAD CSV WITH HEADERS
FROM "flle:///Highschooler.csv" AS row
CREATE (n:Highschooler)
SET n=row,
n.ID = toInteger (row.ID),
n.grade = toInteger (row.grade),
n.minor = CASE row.minor WHEN "NULL" THEN null ELSE row.minor END
RETURN n
```

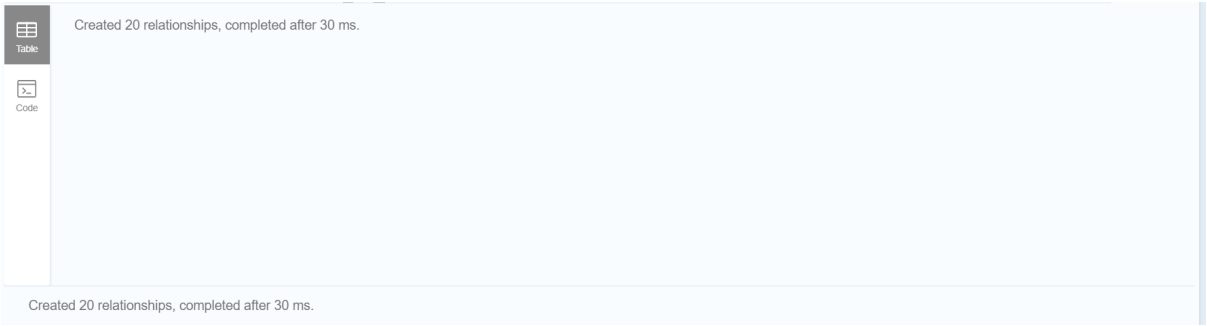



```
LOAD CSV WITH HEADERS FROM "file:///Likes.csv" AS row
MATCH (h1:Highschooler {ID: toInteger (row.ID1) })
MATCH (h2:Highschooler {ID: toInteger (row. ID2)})
MERGE (h1)-[:LIKES {since:date(row.since)} ]-> (h2)
```

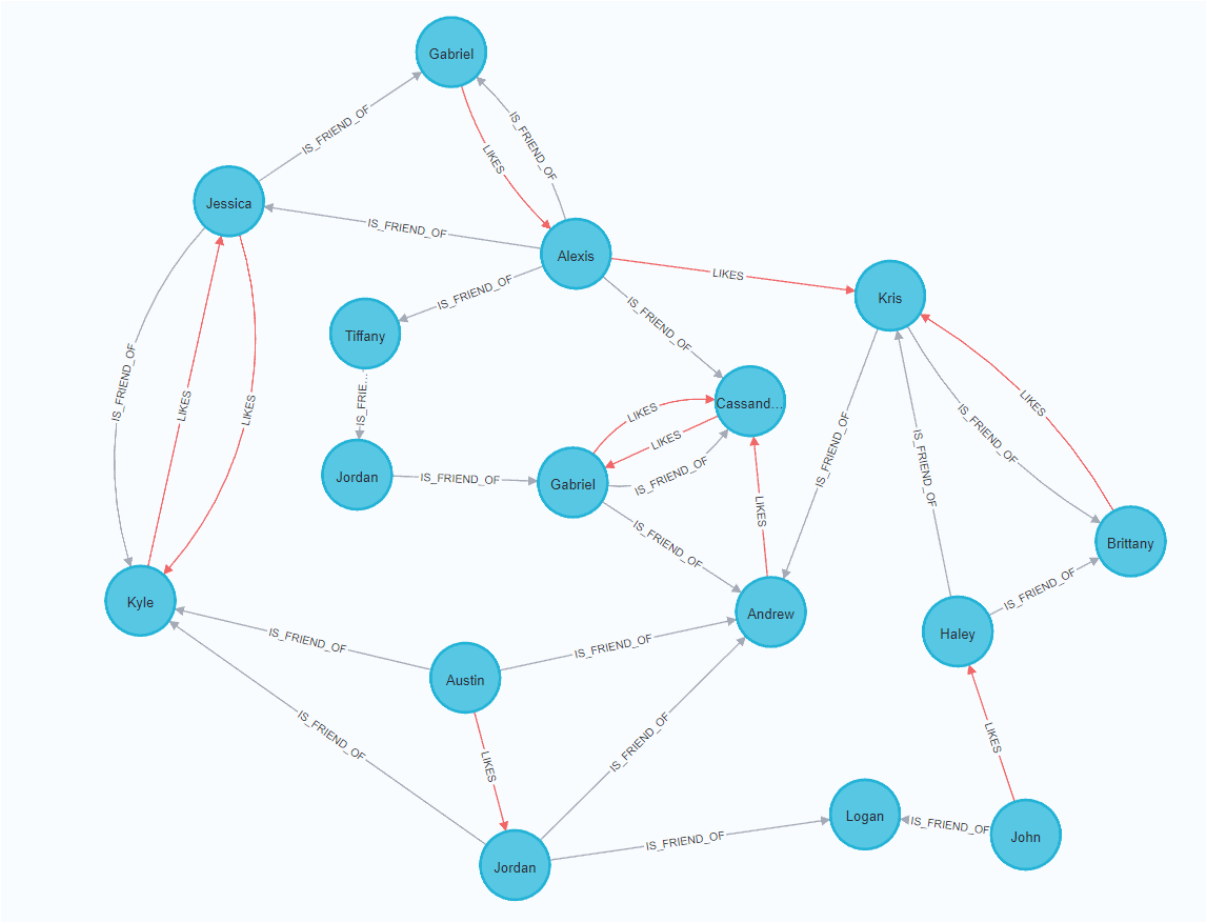




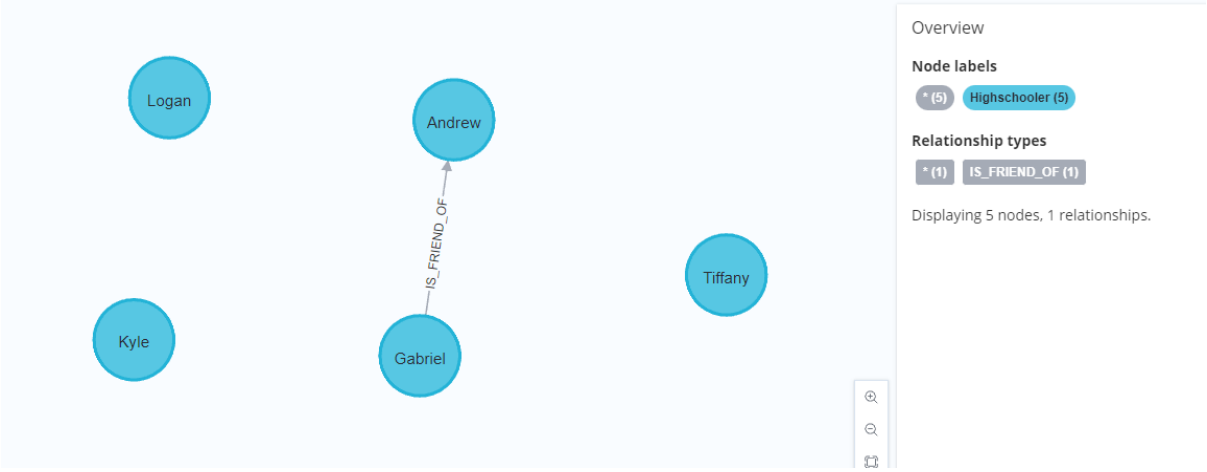

```
LOAD CSV WITH HEADERS FROM "file:///Friend.csv" AS row
MATCH (f1:Highschooler {ID: toInteger (row.ID1)})
MATCH (f2:Highschooler {ID: toInteger (row.ID2)})
WHERE toInteger(row. ID1) < toInteger(row.ID2)
MERGE (f1)-[:IS_FRIEND_OF]-> (f2)
```




```
match(n) return n
```



Utiliser MATCH pour faire correspondre les résultats:
MATCH (n:Highschooler {name:"Jordan"})-[r:IS_FRIEND_OF]-(m) RETURN m



Les réponses sont: Tiffany, Gabriel , Logan, Andrew, Kyle. Ceci est cohérent avec la réponse en SQL.

2. Rank majors by the average grade of students. Cut ranking below 10.
SQL:

```
SELECT major, AVG(grade) AS Moyenne
FROM Highschooler
GROUP BY major
```

```
HAVING AVG(grade) >= 10
ORDER BY Moyenne DESC;
```

major	Moyenne
Maths	12.0
Chemistry	11.0
Physics	10.6666666666667
Art	10.5
Biology	10.3333333333333
Music	10.0

Neo4j:

```
MATCH (n:Highschooler)
WITH n.major AS Major, AVG(n.grade) AS AverageGrade
WHERE AverageGrade >= 10
RETURN Major, AverageGrade
ORDER BY AverageGrade DESC
```

	Major	AverageGrade
1	"Maths"	12.0
2	"Chemistry"	11.0
3	"Physics"	10.666666666666666
4	"Art"	10.5
5	"Biology"	10.333333333333334
6	"Music"	10.0

Q3. For each student A who likes a student B where the two are not friends, find if they have a friend C in common (who can introduce them!). For all such trios, return the name and grade of A, B, and C.

SQL:

```
SELECT DISTINCT A.name AS A_name, A.grade AS A_grade,
                B.name AS B_name, B.grade AS B_grade,
```

```
               C.name AS C_name, C.grade AS C_grade
FROM Highschooler A
JOIN Likes ON A.ID = Likes.ID1
JOIN Highschooler B ON B.ID = Likes.ID2
JOIN Friend AC ON A.ID = AC.ID1
JOIN Friend BC ON B.ID = BC.ID1
JOIN Highschooler C ON (AC.ID2 = C.ID AND BC.ID2 = C.ID)
LEFT JOIN Friend AB ON A.ID = AB.ID1 AND B.ID = AB.ID2
WHERE A.ID <> B.ID
  AND A.ID <> C.ID
  AND B.ID <> C.ID
  AND AB.ID1 IS NULL;
```

A_name	A_grade	B_name	B_grade	C_name	C_grade
Andrew	10	Cassandra	9	Gabriel	8
Austin	11	Jordan	12	Andrew	10
Austin	11	Jordan	12	Kyle	12

Neo4j:

```
MATCH (a:Highschooler)-[:LIKES]->(b:Highschooler)
WHERE NOT (a)-[:IS_FRIEND_OF]-(b)
MATCH (a)-[:IS_FRIEND_OF]-(c:Highschooler)-[:IS_FRIEND_OF]-(b)
RETURN a.name, a.grade, b.name, b.grade, c.name, c.grade
```

	a.name	a.grade	b.name	b.grade	c.name	c.grade
1	"Andrew"	10	"Cassandra"	9	"Gabriel"	8
2	"Austin"	11	"Jordan"	12	"Kyle"	12
3	"Austin"	11	"Jordan"	12	"Andrew"	10

Q4. Find the name and major of all students who are liked by more than one other student.

SQL:

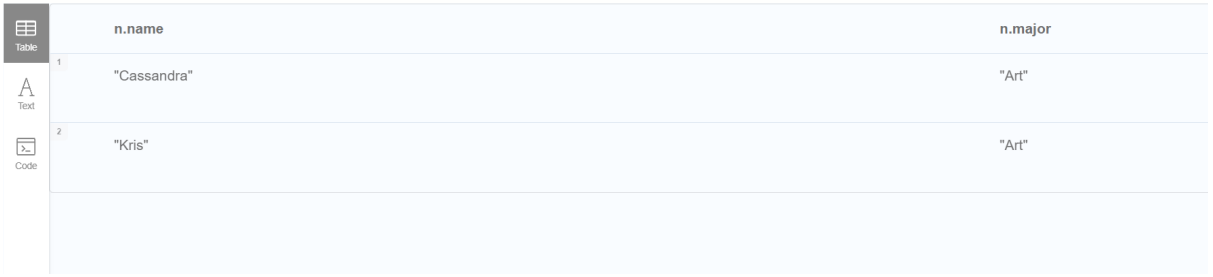
```
SELECT name, major
FROM (
SELECT ID2
FROM Likes L
GROUP BY L.ID2
HAVING count(*)>1
) AS RES_ID
```

```
JOIN Highschooler H ON H.ID = RES_ID.ID2;
```

name	major
Cassandra	Art
Kris	Art

Neo4j:

```
MATCH (n:Highschooler)<-[:LIKES]-(m:Highschooler)
WITH n, COUNT(m) AS likesCount
WHERE likesCount > 1
RETURN n.name, n.major
```



Q5. For each student in CS minor, count the number of friends of friends having grade 12. Do not count in any of the first circle friends.

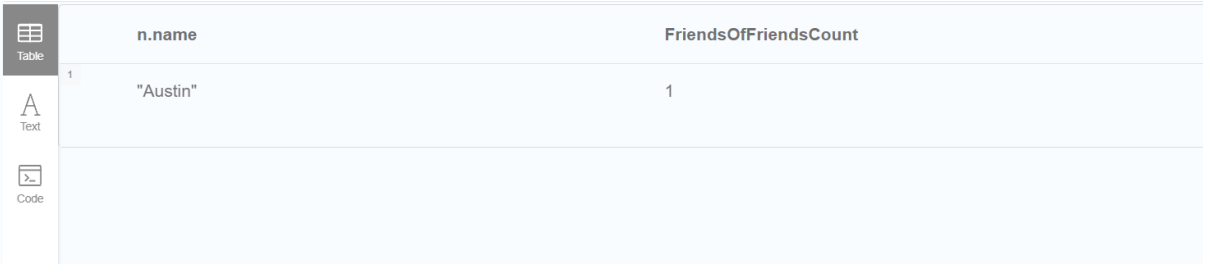
SQL:

```
SELECT h1.*, count(distinct h2.ID) as nb_twelve_graded_foaf
FROM Highschooler h1
         JOIN Friend f1 ON (h1.ID = f1.ID1 AND h1.minor = 'CS')
         JOIN Friend f2 ON f1.ID2 = f2.ID1 AND (h1.ID, f2.ID2) NOT IN Friend
         JOIN Highschooler h2 ON f2.ID2 = h2.ID
WHERE h2.grade = 12
GROUP BY h2.name;
```

ID	name	grade	major	minor	nb_twelve_graded_foaf
1316	Austin	11	Art	CS	1

Neo4j:

```
MATCH (n:Highschooler {minor: 'CS'})-[:IS_FRIEND_OF*2]-(m:Highschooler {grade: 12})
WHERE NOT (n)-[:IS_FRIEND_OF]->(m)
RETURN n.name, COUNT(DISTINCT m) AS FriendsOfFriendsCount
```



Q6. Find the friendship degree (degree of separation) of each pair of students who both have the same name. For all such pair, return the name of students, their ID's and their friendship degree. Include each pair only once.

```
WITH cte_friend(origin, start, goal, current_friend, degree, way)
AS (
/* état initial (de la récursivité) de la table cte table:
ensemble des induvidus ayant le meme nom que quelqu'un dans
le graphe */
SELECT H1.ID, H1.ID, H2.ID, F.ID2, 1, cast(H1.ID as varchar)
|| '/' || cast(F.ID2 as varchar)
FROM Highschooler H1
INNER JOIN Highschooler H2
on H1.name = H2.name
INNER JOIN Friend F
on H1.ID = F.ID1
WHERE H1.ID != H2.ID
AND H1.ID < H2.ID
union all
SELECT cte_friend.origin ,F1.ID1, cte_friend.goal, F1.ID2,
degree + 1, cte_friend.way || cast(F1.ID2 as varchar) || '/'
FROM cte_friend
INNER JOIN Friend F1
on cte_friend.current_friend = F1.ID1
/** Avoid to have loop like 1->2->3->1->2 ...*/
WHERE cte_friend.way NOT LIKE '%' || cast( F1.ID2 as varchar)
|| '%'
)
SELECT origin, H1.name, goal, H2.name, min(degree) as degree
FROM cte_friend
INNER JOIN Highschooler H1
on cte_friend.origin = H1.ID
INNER JOIN Highschooler H2
on cte_friend.goal = H2.ID
WHERE current_friend = goal
GROUP BY goal, current_friend;
```

origin	name	goal	name	degree
1304	Jordan	1510	Jordan	3
1689	Gabriel	1911	Gabriel	3

Neo4j:

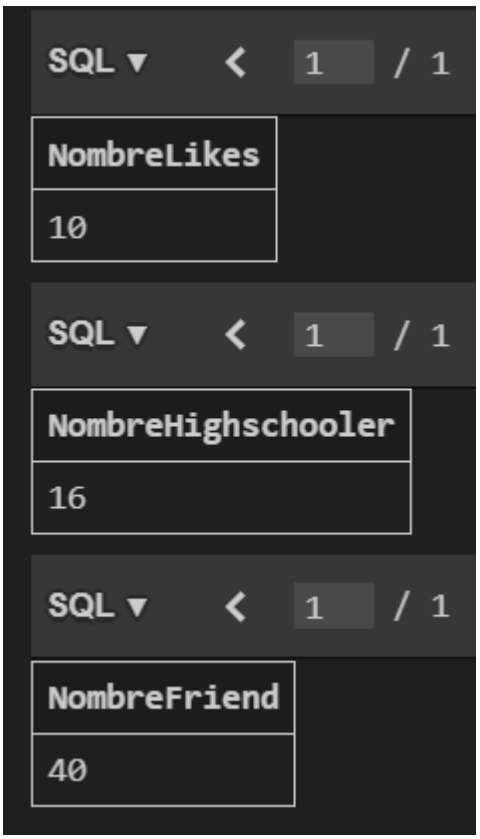
```
MATCH (n1:Highschooler), (n2:Highschooler)
WHERE n1.name = n2.name AND n1.ID < n2.ID
MATCH path = shortestPath((n1)-[:IS_FRIEND_OF*]-(n2))
RETURN n1.name, n1.ID AS ID1, n2.ID AS ID2, LENGTH(path) AS Degree
```

	n1.name	ID1	ID2	Degree
1	"Jordan"	1304	1510	3
2	"Gabriel"	1689	1911	3

II/ MODIFICATION QUERIES

Au départ on a 10 Likes, 16 Highschooler et 40 Friend qu'on peut compter avec la commande:

```
Select COUNT(*) as NombreLikes from Likes;
Select COUNT(*) as NombreHighschooler from  Highschooler;
Select COUNT(*) as NombreFriend from Friend;
```



Q7. It's time for the seniors to graduate. Remove all 12th graders from Highschooler.

```
DELETE
FROM Highschooler
WHERE grade = 12;
SELECT *
FROM Highschooler;
```

La requete a supprimé 4 lignes dans la table Highschooler.

NombreLikes
10

SQL ▾ < 1 / 1 >

NombreHighschooler
12

SQL ▾ < 1 / 1 >

NombreFriend
40

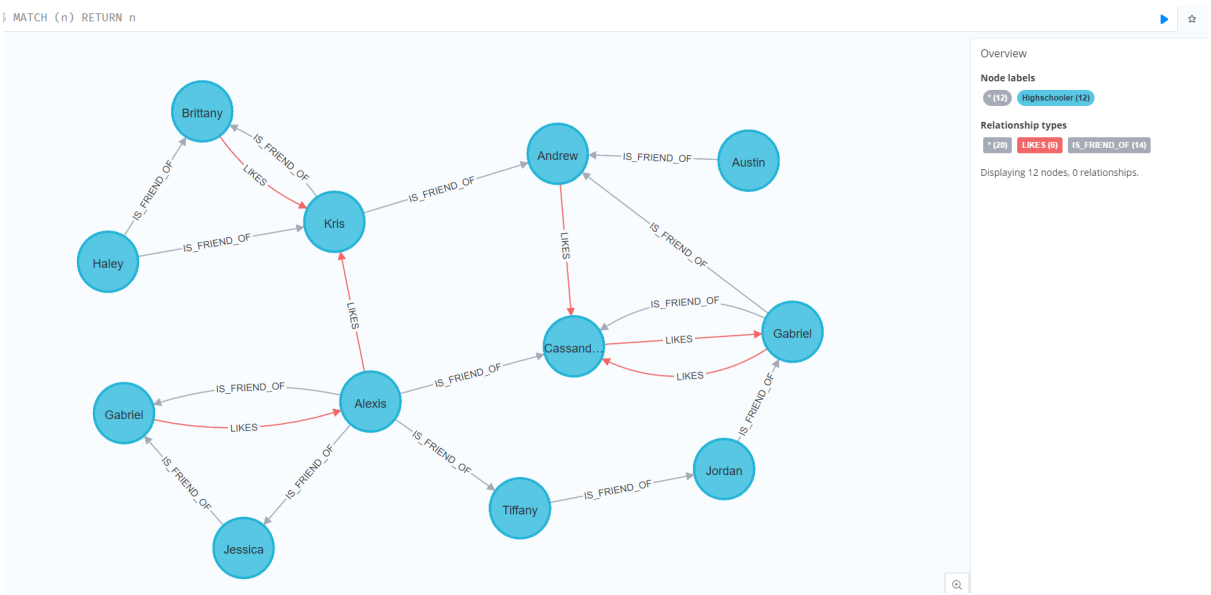
ID	name	grade	major	minor
1510	Jordan	8	CS	Maths
1689	Gabriel	8	CS	NULL
1381	Tiffany	9	Physics	Maths
1709	Cassandra	9	Art	NULL
1101	Haley	10	Biology	Chemistry
1782	Andrew	10	Music	Maths
1468	Kris	10	Art	Music
1641	Brittany	10	Biology	CS
1247	Alexis	11	Biology	NULL
1316	Austin	11	Art	CS
1911	Gabriel	11	Chemistry	Physics
1501	Jessica	11	Physics	NULL

Neo4j:

```
MATCH (n:Highschooler {grade: 12})
```

DETACH DELETE n

```
Deleted 4 nodes, deleted 10 relationships, completed after 12 ms.
```




```
neo4j$ MATCH (n) RETURN n
```

Graph | Table | Text | Code

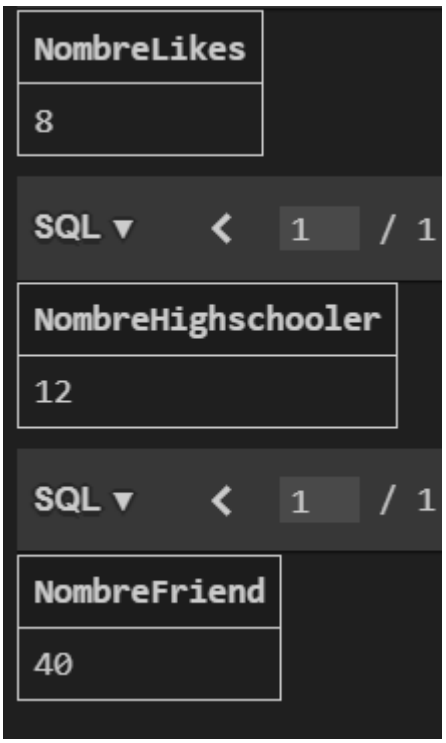
n
(:Highschooler {major: "CS",minor: "Maths",grade: 8,name: "Jordan",ID: 1510})
(:Highschooler {major: "CS",grade: 8,name: "Gabriel",ID: 1689})
(:Highschooler {major: "Physics",minor: "Maths",grade: 9,name: "Tiffany",ID: 1381})
(:Highschooler {major: "Art",grade: 9,name: "Cassandra",ID: 1709})
(:Highschooler {major: "Biology",minor: "Chemistry",grade: 10,name: "Haley",ID: 1101})
(:Highschooler {major: "Music",minor: "Maths",grade: 10,name: "Andrew",ID: 1782})
(:Highschooler {minor: "Music",major: "Art",grade: 10,name: "Kris",ID: 1468})
(:Highschooler {major: "Biology",minor: "CS",grade: 10,name: "Brittany",ID: 1641})
(:Highschooler {major: "Biology",grade: 11,name: "Alexis",ID: 1247})
(:Highschooler {major: "Art",minor: "CS",grade: 11,name: "Austin",ID: 1316})
(:Highschooler {major: "Chemistry",minor: "Physics",grade: 11,name: "Gabriel",ID: 1911})
(:Highschooler {major: "Physics",grade: 11,name: "Jessica",ID: 1501})

Q8. If two students A and B are friends, and A likes B but not vice-versa, remove the Likes tuple.

```sql
DELETE FROM Likes
WHERE (ID1, ID2) IN (
    SELECT L.ID1, L.ID2
    FROM Likes L
    JOIN Friend F ON L.ID1 = F.ID1 AND L.ID2 = F.ID2
    WHERE NOT EXISTS (
        SELECT 1
        FROM Likes L2
        WHERE L2.ID1 = F.ID2 AND L2.ID2 = F.ID1
    )
    UNION
    SELECT L.ID1, L.ID2
    FROM Likes L
    JOIN Friend F ON L.ID1 = F.ID2 AND L.ID2 = F.ID1
    WHERE NOT EXISTS (
        SELECT 1
        FROM Likes L2
        WHERE L2.ID1 = F.ID1 AND L2.ID2 = F.ID2
    )
);
```

Name1	Name2	since
Gabriel	Cassandra	2023-09-02
Cassandra	Gabriel	2023-09-03
Andrew	Cassandra	2023-09-03
Alexis	Kris	2023-09-11
Austin	NULL	2023-09-02
Jessica	NULL	2023-09-09
NULL	Jessica	2023-09-11
NULL	Haley	2023-09-13

On a effectivement 2 lignes qui ont été supprimées.



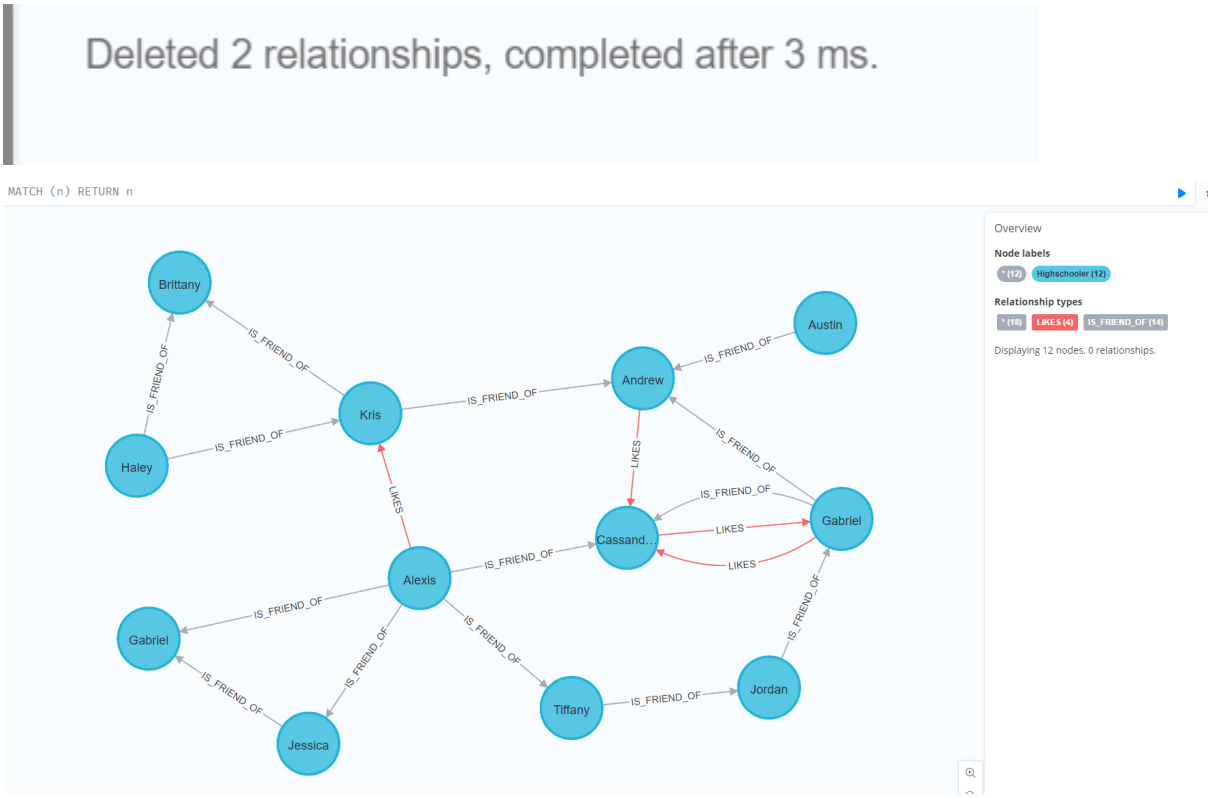
En faisant un SELECT et une Jointure avec la table Highschooler, on peut voir qu'on a 4 relations Likes restant pour les étudiants ayant une correspondance entre Likes et Highschooler comme on a trouvé dans Neo4j:

```
SELECT H1.name AS Name1, H2.name AS Name2, L.since
FROM Likes L
JOIN Highschooler H1 ON L.ID1 = H1.ID
JOIN Highschooler H2 ON L.ID2 = H2.ID;
```

Name1	Name2	since
Gabriel	Cassandra	2023-09-02
Cassandra	Gabriel	2023-09-03
Andrew	Cassandra	2023-09-03
Alexis	Kris	2023-09-11

Neo4j:

```
MATCH (a:Highschooler)-[r:LIKES]->(b:Highschooler)
WHERE NOT (b)-[:LIKES]->(a) AND ((a)-[:IS_FRIEND_OF]->(b) OR
(b)-[:IS_FRIEND_OF]->(a))
DETACH DELETE r
```



Q9. For all cases where A is friends with B, and B is friends with C, add a new friendship for the pair A and C. Do not add duplicate friendships, friendships that already exist, or friendships with oneself.

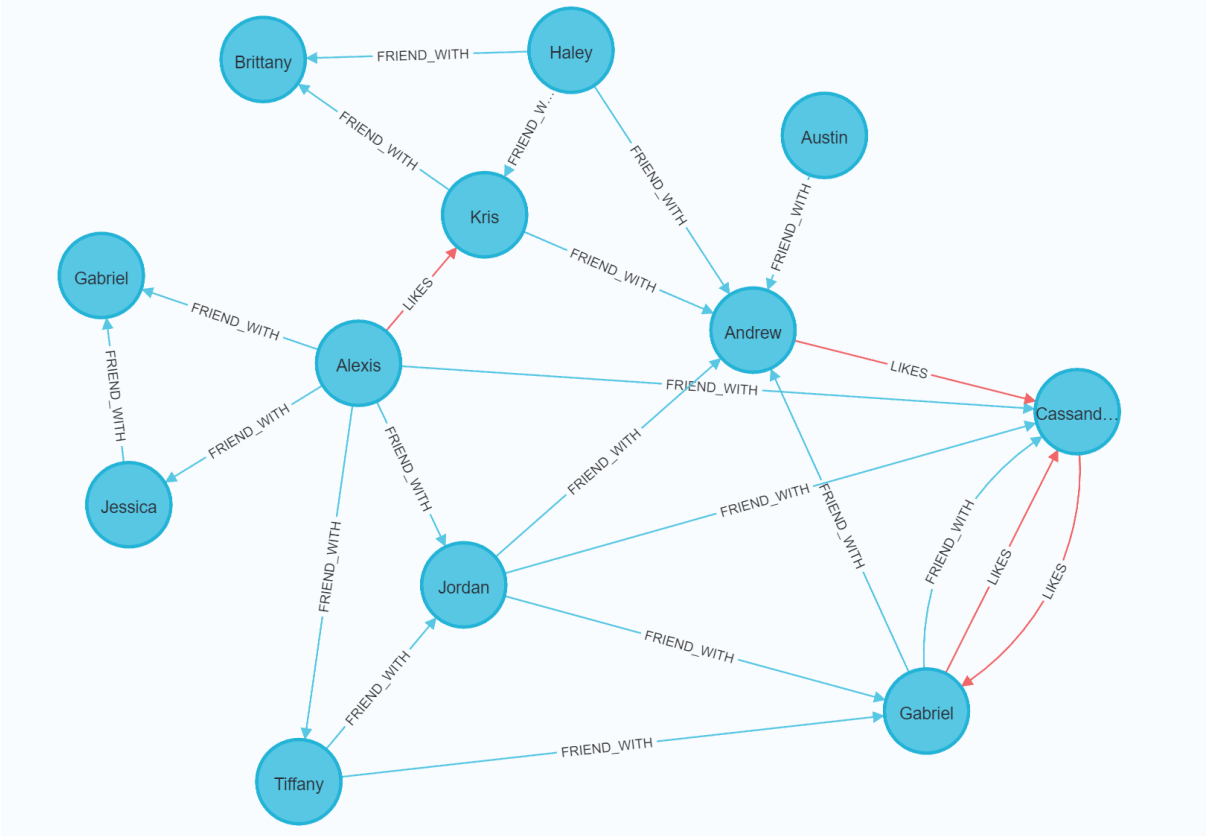
```
INSERT INTO Friend (ID1, ID2)
SELECT DISTINCT F1.ID1, F2.ID2
FROM Friend F1
JOIN Friend F2 ON F1.ID2 = F2.ID1
JOIN Highschooler H1 ON F1.ID1 = H1.ID
JOIN Highschooler H2 ON F1.ID2 = H2.ID
JOIN Highschooler H3 ON F2.ID2 = H3.ID
WHERE F1.ID1 <> F2.ID2
AND NOT EXISTS (
    SELECT 1
    FROM Friend F3
    WHERE (F3.ID1 = F1.ID1 AND F3.ID2 = F2.ID2)
       OR (F3.ID1 = F2.ID2 AND F3.ID2 = F1.ID1)
);
```

Name	Value
Updated Rows	32
Query	INSERT INTO Friend (ID1, ID2)
	SELECT DISTINCT F1.ID1, F2.ID2
	FROM Friend F1
	JOIN Friend F2 ON F1.ID2 = F2.ID1
	JOIN Highschooler H1 ON F1.ID1 = H1.ID
	JOIN Highschooler H2 ON F1.ID2 = H2.ID
	JOIN Highschooler H3 ON F2.ID2 = H3.ID
	WHERE F1.ID1 <> F2.ID2
	AND NOT EXISTS (
	SELECT 1
	FROM Friend F3
	WHERE (F3.ID1 = F1.ID1 AND F3.ID2 = F2.ID2)
	OR (F3.ID1 = F2.ID2 AND F3.ID2 = F1.ID1)
	)

Neo4j:

```
MATCH
(a:Highschooler)-[:FRIEND_WITH]->(b:Highschooler)-[:FRIEND_WITH]->(c:Highschooler)
WHERE NOT (a)-[:FRIEND_WITH]->(c) AND NOT (c)-[:FRIEND_WITH]->(a) AND a <> c
MERGE (a)-[:FRIEND_WITH]->(c)
```

Created 32 relationships, completed after 2 ms.



Conclusion

En conclusion, l'exploration des requêtes sur des bases de données orientées graphe, réalisée à travers les langages SQL et Cypher, révèle des différences significatives en termes de complexité, de concision et d'efficacité. Les requêtes SQL sur un graphe présentent une complexité accrue, nécessitant parfois des CTE récursifs et impliquant un coût de calcul élevé, notamment lors de la réalisation de nombreuses jointures pour traverser le graphe.

À l'inverse, Cypher se démarque par sa concision et son adaptabilité aux bases de données de graphes. Les requêtes en Cypher sont intuitives, facilitant leur rédaction, tandis que la complexité reste notablement inférieure par rapport aux équivalents SQL. De plus, Cypher offre des algorithmes graphiques prêts à l'emploi, comme le plus court chemin, simplifiant considérablement certaines requêtes.

La comparaison entre les requêtes Cypher et SQL met en lumière la lisibilité accrue des premières, notamment sur des requêtes longues ou complexes.

Cypher élimine le besoin de récursivité pour interroger un graphe et propose des fonctions de parcours graphiques natives, telles que shortest Path, qui manquent dans SQL. Cette simplification se traduit également par des requêtes plus courtes et plus compréhensibles.

En outre, l'outil de visualisation de graphe interactif supporté par Neo4J constitue un avantage supplémentaire, offrant une expérience immersive absente des IDE SQL traditionnels.

En résumé, Cypher se révèle être un choix judicieux pour interroger et modifier des bases de données orientées graphe, offrant une simplicité d'utilisation et une efficacité supérieure par rapport au SQL dans ce contexte spécifique. Cependant, il est important de souligner que SQL reste incontestablement le meilleur langage pour les bases de données relationnelles.