CentraleSupélec

Infrastructures Modernes

TP1 - Bases de Données Relationnelles et Graphes

Élève:
Daniel Stulberg Huf

Encadrants:
Francesca BUGIOTTI
Thierry RAPATOUT
Luc VO VAN

9 février 2024

Table des matières

1 Base de données relationnelle - Azure SQL Database

Exercice 0 : Décrivez les tables et les attributs.

Table	Column	Type	Null	Description
tArtist	idArtist	nvarchar	False	Primary key
tArtist	primaryName	nvarchar	False	Artist name
tArtist	birthYear	smallint	True	Artist birth year
tFilm	idFilm	nvarchar	False	Primary key
tFilm	primaryTitle	nvarchar	False	Movie name
tFilm	startYear	smallint	True	Release year
tFilm	runtimeMinutes	smallint	True	Movie duration in minutes
tFilmGenre	idFilm	nvarchar	False	Foreign key to tFilm
tFilmGenre	idGenre	nvarchar	True	Foreign key to tGenre
tGenre	idGenre	nvarchar	False	Primary key
tGenre	genre	nvarchar	True	Genre
tJob	idArtist	nvarchar	False	Foreign key to tArtist
tJob	category	nvarchar	True	Role in the movie
tJob	idFilm	nvarchar	True	Foreign key to tFilm

Exercice 1 ($\frac{1}{4}$ pt) : Visualisez l'année de naissance de l'artiste Nicolas Cage.

```sql
SELECT birthYear
FROM [dbo].[tNames]
WHERE primaryName = 'Nicolas Cage'
```

Query

birthYear
1964

Result

This SQL query retrieves the birth year of the artist named Nicolas Cage from the table '[dbo].[tNames]' with a simple select operation and a filter.

Exercice 2 ($\frac{1}{4}$ pt) : Comptez le nombre d'artistes présents dans la base de donnee.

```
SELECT COUNT(*) AS NumberOfArtists
FROM [dbo].[tArtist];
```

Query

NumberOfArtists
5227

Result

This SQL query counts the total number of artists in the table '[dbo].[tArtist]' by simply counting the total number of rows of such table.

Exercice 3 ($\frac{1}{4}$ pt) : Trouvez les noms des artistes nés en 1960, affichez ensuite leur nombre.

```
SELECT primaryName
FROM [dbo].[tArtist]
WHERE birthYear = 1960;
```

Query 1

primaryName
Antonio Banderas
John Leguizamo
Oliver Platt
...

Result 1

This SQL query retrieves the names of artists who were born in the year 1960 from the table '[dbo].[tArtist]' with a simple select operation and a filter.

```
SELECT COUNT(*) AS NumberOfArtistsBornIn1960
FROM [dbo].[tArtist]
WHERE birthYear = 1960;
```

Query 2

NumberOfArtistsBornIn1960
20

Result 2

This SQL query counts the number of artists born in the year 1960 in the table '[dbo].[tArtist]' by simply counting the total number of rows of such table.

Exercice 4 (1 pt) : Trouvez l'année de naissance la plus représentée parmi les acteurs (sauf 0 !), et combien d'acteurs sont nés cette année là.

```sql
SELECT TOP 1 birthYear, COUNT(*) AS NumberOfActors
FROM [dbo].[tArtist]
WHERE birthYear <> 0
GROUP BY birthYear
ORDER BY COUNT(*) DESC;
```

Query

birthYear	NumberOfActors
1979	40

Result

This SQL query finds the most common birth year (excluding the year 0) among artists in the table '[dbo].[tArtist]' and shows how many artists were born in that year. The query groups the artists from the table by birth year, counts how many artists are in each group, then sorts these groups so the one with the most artists is first, and finally, it selects the top group.

Exercice 5 ($\frac{1}{2}$ pt) : Trouvez les artistes ayant joué dans plus d'un film

```sql
SELECT a.idArtist, a.primaryName, COUNT(j.idFilm) AS NumberOfFilms
FROM [dbo].[tArtist] AS a
JOIN [dbo].[tJob] AS j ON a.idArtist = j.idArtist
WHERE j.category = 'acted in'
GROUP BY a.idArtist, a.primaryName
HAVING COUNT(j.idFilm) > 1
ORDER BY NumberOfFilms DESC;
```

Query

idArtist	primaryName	NumberOfFilms
nm5401689	Miles Jonn-Dalton	7
nm0562210	Antonio Mayans	6
nm12302814	Ellen Wing	6
...	...	...

Result

This SQL query lists artists who have acted in more than one film, along with the number of films they acted in. The query joins the tables '[dbo][tArtist]' and '[dbo][tJob]' based on the artist's ID, filters to include only those jobs where the artist's role was 'acted in', groups the result by artist ID and name, counts the number of films for each artist, includes only those artists who have acted in more than one film, and finally, orders the results by the number of films in descending order.

Exercice 6 ($\frac{1}{2}$ pt) : Trouvez les artistes ayant eu plusieurs responsabilités au cours de leur carrière (acteur, directeur, producteur...).

```sql
SELECT a.idArtist, a.primaryName, COUNT(DISTINCT j.category) AS
   NumberOfDistinctCategories
FROM [dbo].[tArtist] AS a
JOIN [dbo].[tJob] AS j ON a.idArtist = j.idArtist
GROUP BY a.idArtist, a.primaryName
HAVING COUNT(DISTINCT j.category) > 1;
```

Query

idArtist	primaryName	NumberOfDistinctRoles
nm0083851	Sam Bisbee	2
nm0164703	Christopher Clarke	2
nm0168002	Glen Coburn	2
...	...	...

Result

This SQL query lists artists who have had more than one distinct type of job in their career, showing each artist's ID, name, and the number of different job types they've had. The query joins the tables '[dbo][tArtist]' and '[dbo][tJob]' on the artist's ID, groups the result by artist ID and name, counts the distinct types of job categories for each artist, and finally, includes only those artists who have more than one distinct type of job.

Exercice 7 ($\frac{3}{4}$ pt) : Trouver le nom du ou des film(s) ayant le plus d'acteurs (i.e. uniquement acted in).

```sql
SELECT f.primaryTitle, COUNT(*) AS NumberOfActors
FROM tFilm AS f
JOIN tJob AS j ON f.idFilm = j.idFilm
WHERE j.category = 'acted in'
GROUP BY f.primaryTitle
ORDER BY COUNT(*) DESC;
```

Query

primaryTitle	NumberOfActors
Carole & Grey	10
Full Clip for a Snitch	10
Noches de Sol	10
...	...

Result

This SQL query lists films along with the number of actors in each, ordering first by the greatest number of actors who participated in it. The query joins the tables '[dbo][tFilm]' and '[dbo][tJob]' based on the film ID, filters to include only those jobs where the category is 'acted in', groups the result by the film title, counts the number of actors for each film, and finally, orders the films by the number of actors in descending order.

Exercice 8 (1 pt) : Montrez les artistes ayant eu plusieurs responsabilités dans un même film (ex : à la fois acteur et directeur, ou toute autre combinaison) et les titres de ces films.

```sql
SELECT a.primaryName, f.primaryTitle
FROM tArtist AS a
JOIN tJob AS j ON a.idArtist = j.idArtist
JOIN tFilm AS f ON j.idFilm = f.idFilm
GROUP BY a.idArtist, a.primaryName, f.idFilm, f.primaryTitle
HAVING COUNT(DISTINCT j.category) > 1;
```

Query

primaryName	primaryTitle
Jeffrey Schneider	The Accident
Lenni Uitto	Permafrost
Jeff Profitt	Street Crimes Unit
...	...

Result

This SQL query lists the names of artists who had more than one type of job in the same film, along with the titles of those films. The query joins the tables '[dbo][tArtist]', '[dbo][tJob]', and '[dbo][tFilm]' on artist ID and film ID, groups the results by both artist and film, counts the distinct job categories for each artist in each film, and finally, includes only the pairs of artist-film where the artist had more than one type of job in the same film.

2 Base de données graphe - Neo4j

Modifiez la fin du fichier /.bashrc les variables d'environnement nécessaires à la connexion à votre base Neo4.

The fragment of the bashrc file containing the necessary environment variables is attached to this delivery.

3 Export des données vers un modèle graphe

Dans le dossier tp, complétez le programme export-neo4j.py aux endroits notés A COMPLETER. N'hésitez pas à déboguer en ajoutant des print, créer des programmes de test etc. Utilisez les fonctions create_nodes et create_relationships de py2neo.

The completed export-neo4j.py script is attached to this delivery.

(optionnel, 2 points bonus) Décrivez de quelle manière l'environnement de développement a été préconfiguré pour vous :
- **Environnement Python**
- **Informations de connexion aux bases de données**

The provided script is designed to migrate data from a SQL database to a Neo4j graph database. It demonstrates how to preconfigure a development environment for such a task, which includes setting up the Python environment and connecting to both SQL and Neo4j databases.

Python Environment
This script uses py2neo, which is a client toolkit for working with Neo4j from Python applications. It provides functionalities to interact with the graph database, such as creating nodes and relationships, executing Cypher queries, and handling transactions. For connecting to the SQL database, the script relies on pyodbc, a Python open database connectivity interface that allows connecting to any database using the ODBC drivers installed on the system. Finally, the script relies on environment variables to securely manage the database connection details, which avoids hardcoding sensitive data (like usernames and passwords) directly in the script.

Database Connection
A connection string is constructed with the details provided by the environment variables to connect to the SQL database and perform queries to fetch data that will be migrated to the Neo4j graph database. The Graph object from py2neo is instantiated with the environment variables, enabling the script to interact with the Neo4j database, in order to create nodes, relationships, and run Cypher commands for data manipulation and schema updates.

Workflow
The script starts by deleting existing nodes and relationships in the Neo4j database to prepare for fresh data import. The script then fetches data from the tables tFilm, tArtist, and tJob in the SQL database, and creates the corresponding Film and Artist

nodes in Neo4j, along with their properties. The relationships (ACTED_IN, DIRECTED, etc) between Artist and Film nodes are created based on the roles defined in the tJob table. In addition, the indexes on idFilm for Film nodes and idArtist for Artist nodes are created to optimize query performance in Neo4j.

Data Migration

The script processes data in batches (that are defined by BATCH_SIZE) to efficiently handle large datasets. For each entity type (Film, Artist), the script fetches a batch of records from the SQL database, converts each record into a Node object with appropriate labels and properties, and uses 'create_nodes' to bulk-import these nodes into Neo4j. For relationships, the script constructs tuples representing the relationships from the tJob table and uses 'create_relationships' to bulk-import these relationships into Neo4j, therefore linking artists and films.

4 Requêtes graphe (Cypher)

Exercice 1 ($\frac{1}{4}$ pt) : Ajoutez une personne ayant votre prénom et votre nom dans le graphe. Verifiez qui le noeud a bien éte crée.

```
CREATE (n: Artist { primaryName: "Daniel Stulberg Huf", birthYear:
 ↪  2002 })
RETURN n
```

Query


```
{
  "identity": 6473,
  "labels": [
    "Artist"
  ],
  "properties": {
    "birthYear": 2002,
    "primaryName": "Daniel Stulberg Huf"
  },
  "elementId": "6473"
}
```


Result

This query creates a new node in the graph database with the label Artist and sets the properties primaryName with the value "Daniel Stulberg Huf" and birthYear with the value 2002. Then, it returns the newly created node.

Exercice 2 ($\frac{1}{4}$ pt) : Ajoutez un film nommé L'histoire de mon 20 au cours Infrastructure de donnees

```
CREATE (n: Film { primaryTitle: "L'histoire de mon 20 au cours
 ↪  Infrastructure de donnees", startYear: 2024 })
RETURN n
```

Query


```
{
  "identity": 6474,
  "labels": [
    "Film"
  ],
  "properties": {
    "primaryTitle": "L'histoire de mon 20 au cours Infrastructure de
     ↪  donnees",
```

```
    "startYear": 2024
  },
  "elementId": "6474"
}
```

Result

This query creates a new node in the graph database with the label Film and sets the properties primaryTitle with the value "L'histoire de mon 20 au cours Infrastructure de donnees" and startYear with the value 2024. Then, it returns the newly created node.

Exercice 3 ($\frac{1}{2}$ pt) : Ajoutez la relation ACTED_IN qui modélise votre participation à ce film en tant qu'acteur/actrice

```
MATCH (a:Artist), (b:Film)
WHERE a.primaryName = "Daniel Stulberg Huf" AND b.primaryTitle =
 ↪  "L'histoire de mon 20 au cours Infrastructure de donnees"
CREATE (a)-[r:ACTED_IN]->(b)
RETURN type(r)
```

Query

```
type(r)
"ACTED_IN"
```

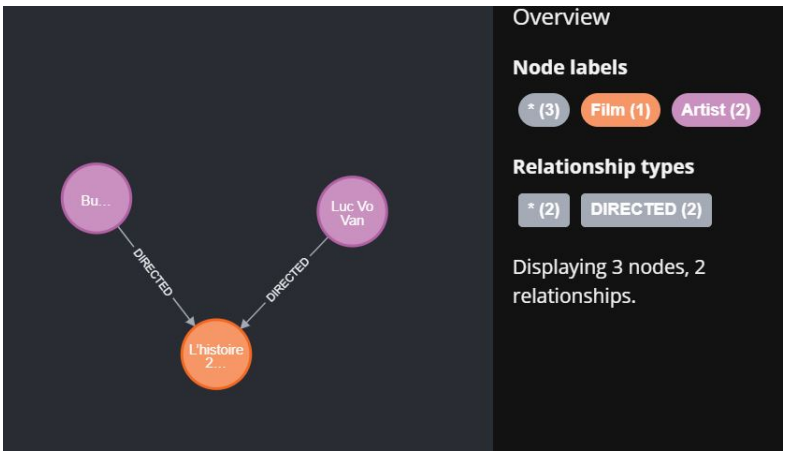
Result

This query frist finds two existing nodes in the graph, one with the label Artist and a primaryName property of "Daniel Stulberg Huf", and another with the label Film and a primaryTitle property of "L'histoire de mon 20 au cours Infrastructure de donnees". It then creates a relationship of type ACTED_IN from the Artist node to the Film node. Finally, it returns the type of the created relationship, which in this case will be "ACTED_IN".

Exercice 4 ($\frac{1}{2}$ pt) : Ajoutez deux de vos professeurs/enseignants comme réalisateurs/réalisatrices de ce film.

```
MATCH (a: Film)
WHERE a.primaryTitle = "L'histoire de mon 20 au cours Infrastructure de donnees"
CREATE (b: Artist { primaryName: "Luc Vo Van" })-[:DIRECTED]->(a)
CREATE (c: Artist { primaryName : "Francesca Bugiotti" })-[:DIRECTED]->(a)
RETURN a, b, c
```

Query



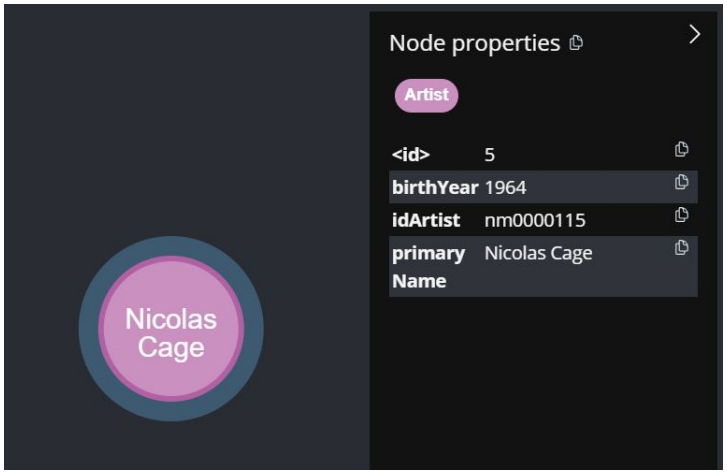
Result

This query adds two new artists as directors to an existing film, creating both the artist nodes and their directing relationships to the film in a single operation. It first searches for a node labeled as Film with the primaryTitle property equal to "L'histoire de mon 20 au cours Infrastructure de donnees", then it creates a new Artist node with the primaryName property set to "Luc Vo Van" and establishes a DIRECTED relationship from this new artist node to the previously matched film node. The same relation is established for the new Artist node with the primaryName property set to "Francesca Bugiotti". Finally, the query returns the film node 'a' and the two newly created artist nodes 'b' and 'c'.

Exercice 5 ($\frac{1}{2}$ pt) : Affichez le noeud représentant l'acteur nommé Nicolas Cage, et visualisez son année de naissance.

```
MATCH (a: Artist)
WHERE a.primaryName = "Nicolas Cage"
RETURN a
```

Query



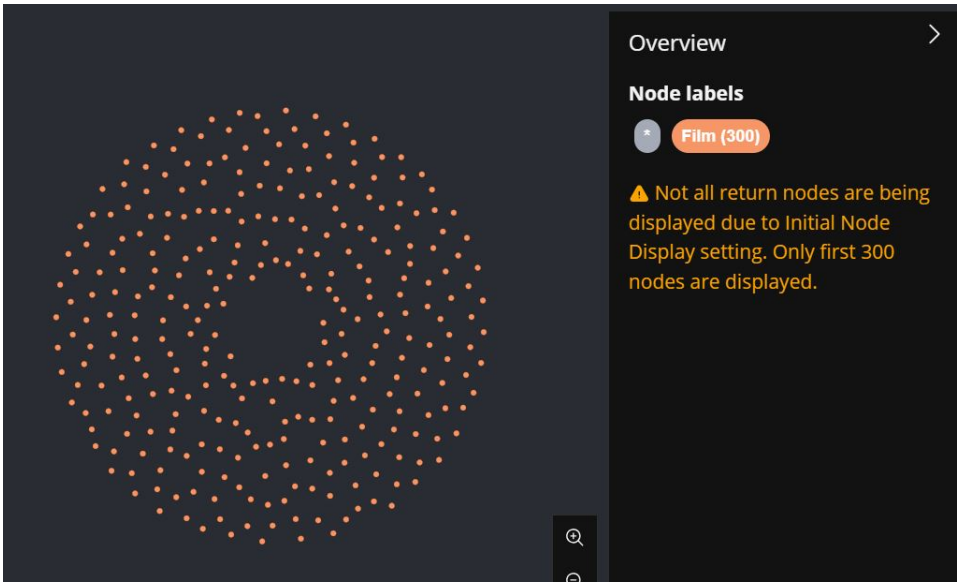
Result

This query searches for a node with the label Artist that has a primaryName property equal to "Nicolas Cage" and returns that node. Essentially, it finds and returns the node representing the artist named Nicolas Cage from the graph database.

Exercice 6 ($\frac{1}{2}$ pt) : Visualisez l'ensemble des films.

```
MATCH (a: Film)
RETURN a
```

Query



Result

This query searches for all nodes in the graph database that have the label Film and returns those nodes, therefore retrieving every node representing a film.

Exercice 7 ($\frac{1}{2}$ pt) : Trouvez les noms des artistes nés en 1960, affichez ensuite leur nombre.

Query 1 :

```
MATCH (a: Artist)
WHERE a.birthYear = 1960
RETURN a
```

Query 1


```
{
  "identity": 0,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm0000104",
    "birthYear": 1960,
    "primaryName": "Antonio Banderas"
  },
  "elementId": "0"
}
{
  "identity": 23,
  "labels": [
    "Artist"
```

```
  ],
  "properties": {
    "idArtist": "nm0000491",
    "birthYear": 1960,
    "primaryName": "John Leguizamo"
  },
  "elementId": "23"
}
{
  "identity": 55,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm0001624",
    "birthYear": 1960,
    "primaryName": "Oliver Platt"
  },
  "elementId": "55"
}
...
```


Result 1

This query searches for all nodes in the graph database that have the label Artist and a birthYear property equal to 1960, and then returns those nodes, therefore returning all artist nodes representing artists born in the year 1960.

Query 2 :

```
MATCH (a: Artist)
WHERE a.birthYear = 1960
RETURN COUNT(a)
```

Query 2

```
COUNT(a)
20
```

Result 2

This query searches for all nodes in the graph database that have the label Artist with a birthYear property equal to 1960. Instead of returning the nodes themselves, it returns the count of these nodes, which represents the number of artists born in the year 1960.

Exercice 8 (1 pt) : Trouver l'ensemble des acteurs (sans entrées doublons) qui ont joué dans plus d'un film.

```
MATCH (n: Artist)
WHERE SIZE([(n)-[:ACTED_IN]->(f) | f]) > 1
RETURN n
```

Query


```
{
  "identity": 6,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm0000128",
    "birthYear": 1964,
    "primaryName": "Russell Crowe"
  },
  "elementId": "6"
}
{
  "identity": 12,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm0000199",
    "birthYear": 1940,
    "primaryName": "Al Pacino"
  },
  "elementId": "12"
}
{
  "identity": 27,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm0000616",
    "birthYear": 1956,
    "primaryName": "Eric Roberts"
  },
  "elementId": "27"
}
...
```

Result

This query searches for artist nodes (:Artist) who have acted in more than one film. It uses a pattern comprehension inside the SIZE function to count the distinct films (f) each artist (n) has acted in. If an artist has acted in more than one film, the artist node is returned.

Exercice 9 (1 pt) : Trouvez les artistes ayant eu plusieurs responsabilités au cours de leur carrière (acteur, directeur, producteur...).

```
MATCH (n: Artist)-[r]->(:Film)
WITH COUNT(DISTINCT type(r)) as nb_roles, n
WHERE nb_roles > 1
RETURN n
```

Query


```
{
  "identity": 1960,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm7700848",
    "birthYear": 0,
    "primaryName": "Maja Prelog"
  },
  "elementId": "1960"
}
{
  "identity": 717,
  "labels": [
    "Artist"
  ],
  "properties": {
    "idArtist": "nm6366616",
    "birthYear": 0,
    "primaryName": "Jeffrey Tipton"
  },
  "elementId": "717"
}
{
  "identity": 1116,
  "labels": [
    "Artist"
  ],
  "properties": {
```

```
    "idArtist": "nm6815189",
    "birthYear": 0,
    "primaryName": "Martin W. Payne"
  },
  "elementId": "1116"
}
...
```


Result

This query finds artists (:Artist nodes) who have more than one type of relationship (r) with any film (:Film nodes), indicating they have had multiple roles in relation to films. The query first matches artists that have any kind of relationship to films, then by using the WITH clause, it counts the distinct types of relationships (type(r)) each artist has with films, and aliases this count as nb_roles. After that, the query filters these results to include only those artists where nb_roles is greater than 1. Finally, it returns the artist nodes that meet this criterion.

Exercice 10 (1 pt) : Montrez les artistes ayant eu plusieurs responsabilités dans un même film (ex : à la fois acteur et directeur, ou toute autre combinaison) et les titres de ces films.

```
MATCH (a:Artist)-[r]->(f:Film)
WITH a, f, COLLECT(DISTINCT type(r)) AS roles
WHERE SIZE(roles) > 1
RETURN a.primaryName AS Artist, f.primaryTitle AS Film, roles
```

Query

Artist	Film	roles
Maja Prelog	Cent'anni	["DIRECTED", "ACTED_IN"]
Jeffrey Tipton	Father Fauci : The Covid Story	["DIRECTED", "ACTED_IN"]
Ben Petrie	The Heirloom	["DIRECTED", "ACTED_IN"]
...	...	...

Result

This query identifies artists who have had multiple distinct roles in the same film, then returns the names of these artists, the titles of the films, and the distinct roles they held. The query first finds patterns where an artist (a) has any type of relationship (r) with a film (f), then for each artist-film pair, it collects the distinct types of relationships into a list named roles. The query filters these pairs to only include those where the artist has more than one distinct role in the same film (SIZE(roles) $>$ 1), and finally, returns the primary name of the artist, the primary title of the film, and the list of distinct roles that the artist had in that film.

Exercice 11 (2 pt) : Trouver le nom du ou des film(s) ayant le plus d'acteurs.

```
MATCH (n: Artist)-[:ACTED_IN]->(t: Film)
WITH COUNT(DISTINCT n) as NumberOfActors, t
ORDER BY NumberOfActors DESC
LIMIT 3
RETURN t.primaryTitle, NumberOfActors
```

Query

primaryTitle	NumberOfActors
Full Clip for a Snitch	10
Noches de Sol	10
Carole & Grey	10
...	...

Result

This query finds films with the most distinct actors, limiting the results to the top 3 films with the highest number of actors for simplicity reasons, and returns the titles of these films along with the count of distinct actors in each. The query first matches patterns where an artist (n with the label Artist) acted in (:ACTED_IN relationship) a film (t with the label Film), then for each film t, it counts the number of distinct actors n that have acted in it, grouping by the film. It orders the films by the count of distinct actors in descending order (DESC) and limits the results to the top 3 films. Finally, the query returns the primary title of each of these films and the count of distinct actors that acted in them as NumberOfActors.

5 Requêtes graphe (Gremlin)

Exercice 1 ($\frac{1}{4}$ pt) : Ajoutez une personne ayant votre prénom et votre nom dans le graphe. Verifiez qui le noeud a bien éte crée.

```
g.addV("Artist")
  .property("primaryName", "Daniel Stulberg Huf")
  .property("birthYear", 2002)
  .property("pk", "pk_dh")
```

Query


```
[
    {
      "id": "cbb9c809-10f8-4718-a250-013826b84150",
      "label": "Artist",
      "type": "vertex",
      "properties": {
        "primaryName": [
          {
            "id": "79e2e5ce-6a16-44fa-9d55-dc4f05f4eb0d",
            "value": "Daniel Stulberg Huf"
          }
        ],
        "birthYear": [
          {
            "id": "404b2e2b-b8a4-4dfe-966a-d4203684652d",
            "value": 2002
          }
        ],
        "pk": [
          {
            "id": "cbb9c809-10f8-4718-a250-013826b84150|pk",
            "value": "pk_dh"
          }
        ]
      }
    }
]
```


Result

This query creates a new vertex in the graph database labeled "Artist" and assigns it three properties : ‘primaryName‘ set to "Daniel Stulberg Huf" ; ‘birthYear‘ set to 2002 ; and ‘pk‘ set to "pk_dh," which serves as a primary key for this vertex. The query initiates the addition of a vertex with g.addV("Artist"), followed by chaining .property() steps to assign each of the properties.

Exercice 2 ($\frac{1}{4}$ pt) : Ajoutez un film nommé L'histoire de mon 20 au cours Infrastructure de donnees

```
g.addV("Film")
  .property("primaryTitle", "  L  histoire   de mon 20 au cours
   Infrastructure de donnees")
  .property("startYear", 2024)
  .property("pk", "pk_film")
```

Query


```
[
    {
      "id": "69e4c697-3191-4957-962e-62abdf9661ec",
      "label": "Film",
      "type": "vertex",
      "properties": {
        "primaryTitle": [
          {
            "id": "67594b6f-a3cb-4029-9df6-b0f75f61efe8",
            "value": "  L  histoire   de mon 20 au cours Infrastructure
   de donnees"
          }
        ],
        "startYear": [
          {
            "id": "74496039-3ca2-41fc-b045-a520d92bcd8a",
            "value": 2024
          }
        ],
        "pk": [
          {
            "id": "69e4c697-3191-4957-962e-62abdf9661ec|pk",
            "value": "pk_film"
          }
        ]
      }
    }
]
```


Result

This query creates a new vertex in the graph database labeled "Film" and assigns it three properties : ‘primaryTitle‘ set to "L'histoire de mon 20 au cours Infrastructure de donnees" ; ‘startYear‘ set to 2024 ; and ‘pk‘ set to "pk_film," which serves as a primary key for this vertex. The query initiates the addition of a vertex with g.addV("Film"), followed by chaining .property() steps to assign each of the properties.

Exercice 3 ($\frac{1}{4}$ pt) : Ajoutez la relation ACTED_IN qui modélise votre participation à ce film en tant qu'acteur/actrice

```
g.V().has("Artist", "primaryName", "Daniel Stulberg Huf").as("artist")
  .V().has("Film", "primaryTitle", " L histoire de mon 20 au cours Infrastructure de donnees").as("film")
  .addE("ACTED_IN")
  .from("artist")
  .to("film")
```

Query


```
[
  {
      "id": "7eada684-32c0-4dfd-b428-a5815389ef1d",
      "label": "ACTED_IN",
      "type": "edge",
      "inVLabel": "Film",
      "outVLabel": "Artist",
      "inV": "69e4c697-3191-4957-962e-62abdf9661ec",
      "outV": "cbb9c809-10f8-4718-a250-013826b84150"
  }
]
```


Result

This query creates a relationship by linking the artist named "Daniel Stulberg Huf" to the film titled "L'histoire de mon 20 au cours Infrastructure de donnees" with an edge labeled "ACTED_IN". It begins by searching for a vertex labeled "Artist" with the primaryName property matching "Daniel Stulberg Huf" and aliases this vertex as "artist." It simultaneously searches for another vertex labeled "Film" with the primaryTitle property of "L'histoire de mon 20 au cours Infrastructure de donnees" and aliases it as "film." Then, the query creates an "ACTED_IN" edge from the artist vertex to the film vertex.

Exercice 4 ($\frac{1}{4}$ pt) : Ajoutez deux de vos professeurs/enseignants comme réalisateurs/réalisatrices de ce film.

```
g.V().as("film").has("Film", "primaryTitle", " L histoire de mon 20 au cours Infrastructure de donnees")
  .addV("Artist")
  .as("director1")
  .property("primaryName", "Luc Vo Van")
  .property("pk", "pk_lvv")
  .addE("DIRECTED").from("director1").to("film")
  .addV("Artist")
  .as("director2")
  .property("primaryName", "Francesca Bugiotti")
  .property("pk", "pk_fb")
  .addE("DIRECTED").from("director2").to("film")
```

```
  .select("film", "director1", "director2");
```

Query


```
[
  {
    "film": {
      "id": "69e4c697-3191-4957-962e-62abdf9661ec",
      "label": "Film",
      "type": "vertex",
      "properties": {
        "primaryTitle": [
          {
            "id": "67594b6f-a3cb-4029-9df6-b0f75f61efe8",
            "value": "  L  histoire   de  mon  20  au  cours  Infrastructure
   de  donnees"
          }
        ],
        "startYear": [
          {
            "id": "74496039-3ca2-41fc-b045-a520d92bcd8a",
            "value": 2024
          }
        ],
        "pk": [
          {
            "id": "69e4c697-3191-4957-962e-62abdf9661ec|pk",
            "value": "pk_film"
          }
        ]
      }
    },
    "director1": {
      "id": "3c2fe96a-e056-4307-b428-b683c84af627",
      "label": "Artist",
      "type": "vertex",
      "properties": {
        "primaryName": [
          {
            "id": "bc0d089c-5ccb-4ff0-8752-207fa91a1f89",
            "value": "Luc  Vo  Van"
          }
        ],
        "pk": [
          {
            "id": "3c2fe96a-e056-4307-b428-b683c84af627|pk",
            "value": "pk_lvv"
          }
        ]
      }
    },
    "director2": {
```

```
      "id": "be1087bd-f45b-401a-a6cb-0d5410b74098",
      "label": "Artist",
      "type": "vertex",
      "properties": {
        "primaryName": [
          {
            "id": "51b23b9d-c4e8-4814-81a2-2221fbce69e4",
            "value": "Francesca Bugiotti"
          }
        ],
        "pk": [
          {
            "id": "be1087bd-f45b-401a-a6cb-0d5410b74098|pk",
            "value": "pk_fb"
          }
        ]
      }
    }
  }
]
```


Result

This query begins by identifying a film vertex based on its "primaryTitle" property being "L'histoire de mon 20 au cours Infrastructure de donnees" and aliases it as "film". It then adds two new vertices labeled "Artist", representing directors Luc Vo Van and Francesca Bugiotti, assigning each properties for their names and unique identifiers, and aliases them as "director1" and "director2" respectively. For each director, the query creates a "DIRECTED" edge from the director to the film, signifying their roles as directors in that film. Finally, the query selects and returns the film and the two director vertices.

Exercice 5 ($\frac{1}{4}$ pt) : Affichez le noeud représentant l'acteur nommé Nicolas Cage, et visualisez son année de naissance.

```
g.V().has("Artist", "primaryName", "Nicolas Cage")
  .project("name", "year")
    .by("primaryName")
    .by("birthYear");
```

Query


```
[
  {
    "name": "Nicolas Cage",
    "year": 1960
  }
]
```


Result

This query searches the graph for vertices labeled "Artist" that have a primaryName property equal to "Nicolas Cage." For each vertex that matches this criterion, it constructs a map projection with two keys : "name" and "year." The .by("primaryName") specifies that the value associated with the "name" key in the output map should be the value of the primaryName property of the vertex. Similarly, the .by("birthYear") specifies that the value associated with the "year" key should be the value of the birthYear property of the vertex. This results in the matched vertex detailing Nicolas Cage's name and birth year.

Exercice 6 ($\frac{1}{4}$ pt) : Visualisez l'ensemble des films.

```
g.V().hasLabel("Film")
```

Query


```
[
  {
    "id": "tt15125820",
    "label": "Film",
    "type": "vertex",
    "properties": {
      "primaryTitle": [
        {
          "id": "af2afe0a-bdb9-40fe-9c80-98084edaa392",
          "value": "Bionica"
        }
      ],
      "startYear": [
        {
          "id": "faf1de08-91b0-4b6c-8b0c-34695cfab504",
          "value": 2024
        }
      ],
      "averageRating": [
        {
          "id": "59b18de1-2ad2-484b-8031-4c194579d91c",
          "value": 0
        }
      ],
      "pk": [
        {
          "id": "tt15125820|pk",
          "value": "0"
        }
      ]
    }
  },
  {
    "id": "tt30874320",
    "label": "Film",
    "type": "vertex",
```

```
    "properties": {
      "primaryTitle": [
        {
          "id": "e257af60-15f0-4785-a0a4-1176fe13bbe0",
          "value": "Cent'anni"
        }
      ],
      "startYear": [
        {
          "id": "baef7fec-6b6c-4c20-8e09-ff03700abc63",
          "value": 2024
        }
      ],
      "averageRating": [
        {
          "id": "988a5ea5-11d9-457a-a9b2-90e7649e0b33",
          "value": 0
        }
      ],
      "pk": [
        {
          "id": "tt30874320|pk",
          "value": "0"
        }
      ]
    }
  },
  {
    "id": "tt30877290",
    "label": "Film",
    "type": "vertex",
    "properties": {
      "primaryTitle": [
        {
          "id": "cdb6d0e6-b66f-40c2-bde7-d5e5d41e8f17",
          "value": "Velky Pan"
        }
      ],
      "startYear": [
        {
          "id": "670cd0aa-ddd2-4d83-8909-9f2e7d573b98",
          "value": 2024
        }
      ],
      "averageRating": [
        {
          "id": "076b3338-fa8c-4578-9a45-f62c5887596a",
          "value": 0
        }
      ],
      "pk": [
        {
```

```
            "id": "tt30877290|pk",
            "value": "0"
          }
        ]
      }
  },
  ...
]
```


Result

This query searches through the graph for all vertices that have the label "Film". It starts by looking at all vertices in the graph and then filters these vertices to only include those that are labeled with "Film" using the hasLabel("Film") step.

Exercice 7 ($\frac{1}{2}$ pt) : Trouvez les noms des artistes nés en 1960, affichez ensuite leur nombre.

```
g.V().hasLabel("Artist")
.has("birthYear", 1960)
.values("primaryName");
```

Query 1

```
[
  "Joan Chen",
  "Desmond Eastwood",
  "Alicia Witt",
  ...
]
```

Result 1

This query searches the graph for vertices labeled as "Artist" who were born in the year 1960, and then extracts the primaryName property from each of these vertices. It begins by selecting all vertices and narrows down this selection to those with the label "Artist" using .hasLabel("Artist"). It further filters these artist vertices to only those with a birthYear property equal to 1960 through .has("birthYear", 1960). Finally, for each vertex that matches these criteria, it retrieves the value of the primaryName property using .values("primaryName"), listing the names of all artists born in 1960.

```
g.V().hasLabel("Artist").has("birthYear", 1960).count()
```

Query 2

```
[
  5289
]
```

Result 2

This query counts the number of vertices in the graph that are labeled as "Artist" and have a birthYear property equal to 1960. It starts by selecting all vertices in the graph, then filters these to only include those with the label "Artist" (hasLabel("Artist")). It then narrows down the selection to those artists who were born in 1960 (has("birthYear", 1960)). Finally, it applies the count() method, which counts the number of vertices that meet these criteria, resulting in the total number of artists born in that year.

Exercice 8 (1,5 pt) : Trouver l'ensemble des acteurs (sans entrées doublons) qui ont joué dans plus d'un film.

```
g.V().hasLabel("Artist")
  .where(__.out("acted in").count().is(gt(1)))
  .dedup()
  .values("primaryName")
```

Query

```
[
  "Tim Neff",
  "Lilly Singh",
  "Olwen Fouere",
  ...
}
```

Result

This query finds and lists the unique names of artists who have acted in more than one film. It starts by selecting all vertices labeled "Artist" from the graph. Then, for each artist, it checks if there are more than one outgoing edges labeled "acted in," indicating the artist has acted in multiple films. This is done using a where step that contains a sub-traversal : it looks at outgoing "acted in" edges, counts them, and filters for artists with a count greater than one using .count().is(gt(1)). To ensure each artist's name is listed only once, even if they've acted in many films, the dedup() step removes any duplicates. Finally, it extracts and returns the primaryName property of these filtered artist vertices, giving a list of artists who have acted in more than one film.

Exercice 9 (2 pt) : Trouvez les artistes ayant eu plusieurs responsabilités au cours de leur carrière (acteur, directeur, producteur...).

```
g.V().hasLabel('Artist')
  .where(outE().groupCount().by(label).unfold().
  select(keys).dedup().count().is(gt(1)))
  .values('primaryName')
```

Query

```
[
  "Carley Harrison",
  "Josef Hader",
  "Pavel Nikolajev",
```

```
  ...
]
```

Result

This query identifies and retrieves the names of artists who have more than one type of relationship to other entities in the graph, indicating they've held multiple roles. It starts by selecting vertices labeled "Artist". For each artist, it examines their outgoing relationships, counting the occurrences of each relationship type using groupCount().by(label). This count is then unfolded to work with individual entries, from which it extracts the relationship types (select(keys)), removes duplicates with dedup(), and counts the distinct types of relationships. If an artist has more than one distinct type of outgoing relationship (count().is(gt(1))), it implies they've had multiple roles. Finally, it extracts and returns the primaryName property of these artists, providing a list of artists with multiple roles in their careers.

Exercice 10 (2 pt) : Montrez les artistes ayant eu plusieurs responsabilités dans un même film (ex : à la fois acteur et directeur, ou toute autre combinaison) et les titres de ces films.

```
g.V().hasLabel('Film').has("primaryTitle")
  .as('film')
  .inE().as('roles')
  .outV().hasLabel('Artist').has("primaryName")
  .as('artist')
  .select('artist', 'film')
    .by('primaryName')
    .by('primaryTitle')
  .group().by(select('artist', 'film'))
    .by(__.select('roles').label().dedup().count())
  .unfold()
  .where(select(values).is(gt(1)))
  .select(keys)
```

Query

This query finds artists who have had more than one type of role in the same film, then groups and lists these artists along with the films. It begins by selecting all vertices labeled "Film" that have a "primaryTitle", marking these vertices as "film". Then, it looks for incoming edges to these film vertices, marking them as "roles", which represent the different roles artists have had in these films. Next, it follows those edges to vertices labeled "Artist" that have a "primaryName", marking these vertices as "artist". The query constructs a pair of "artist" and "film" and uses this pair as a key to group the data. For each artist-film pair, it counts the distinct types of roles (using deduped edge labels to ensure uniqueness) associated with the pair. After grouping, it unfolds the results to filter out artist-film pairs for only those with more than one distinct role. Finally, it selects the keys from this filtered set, which include the artist's name and the film title.

However, this query did not work properly as it lasted forever to be completed. That could have happened due to the complexity of several traversal steps, or because of

the volume of data. One possible solution for the problem would be simplifying the query by avoiding the use of traversal-based calculations within the group() step, by performing some of the data processing application-side after fetching more basic data from the database. Other approach would be breaking down the query into multiple simpler queries by firstly retrieving artist-film pairs, and then, in subsequent queries, process these pairs to count distinct roles.

Exercice 11 (5 pt BONUS) : Trouver le nom du ou des film(s) ayant le plus d'acteurs.

```
g.V().hasLabel('Film').has("primaryTitle")
  .group()
    .by('primaryTitle')
    .by(__.in("acted in").count())
  .unfold()
  .order().by(select(values), decr)
  .limit(3)
```

Query

```
[
  {
    "L'histoire de mon 20 au cours Infrastructure de donnees": 19
  },
  {
    "Noches de Sol": 10
  },
  {
    "Carole & Grey": 10
  }
]
```

Result

This query identifies the top three films with the highest number of actors and retrieves their titles along with the count of actors for each. It starts by selecting vertices labeled "Film" that also have a "primaryTitle" property to ensure consistency (some films did not have this property correctly written and that was raising an error). The query then groups these films by their titles and counts the number of incoming "acted in" edges for each film, which represents the number of actors associated with that film. After grouping, the query unfolds the results to break down the grouped structure into individual items that can be sorted. It then orders these items in descending order based on the count of actors (using order().by(select(values), decr)), ensuring the films with the most actors are listed first. Finally, the query limits the output to the top three films.

A bashrc.sh

```
export TPBDD_NEO4J_SERVER="bolt://44.214.180.114:7687"
export TPBDD_NEO4J_USER="neo4j"
export TPBDD_NEO4J_PASSWORD="commands-atom-sheets"
```

B export-neo4j.py

```
from py2neo import Graph
from py2neo.bulk import create_nodes, create_relationships
from py2neo.data import Node
import os
import pyodbc

server = os.environ["TPBDD_SERVER"]
database = os.environ["TPBDD_DB"]
username = os.environ["TPBDD_USERNAME"]
password = os.environ["TPBDD_PASSWORD"]
driver= '{ODBC Driver 18 for SQL Server}'

neo4j_server = os.environ["TPBDD_NEO4J_SERVER"]
neo4j_user = os.environ["TPBDD_NEO4J_USER"]
neo4j_password = os.environ["TPBDD_NEO4J_PASSWORD"]

graph = Graph(neo4j_server, auth=(neo4j_user, neo4j_password))

BATCH_SIZE = 10000

print("Deleting existing nodes and relationships...")
graph.run("MATCH ()-[r]->() DELETE r")
graph.run("MATCH (n:Artist) DETACH DELETE n")
graph.run("MATCH (n:Film) DETACH DELETE n")

with pyodbc.connect('DRIVER='+driver+';SERVER=tcp:'+server+';PORT
   =1433;DATABASE='+database+';UID='+username+';PWD='+ password) as
   conn:
    cursor = conn.cursor()

    # Films
    exportedCount = 0
    cursor.execute("SELECT COUNT(1) FROM tFilm")
    totalCount = cursor.fetchval()
    cursor.execute("SELECT idFilm, primaryTitle, startYear FROM tFilm
   ")
    while True:
        importData = []
        rows = cursor.fetchmany(BATCH_SIZE)
        if not rows:
            break

```

```
40          i = 0
41          for row in rows:
42              # Creer un objet Node avec comme label Film et les
    proprietes adequates
43              n = Node("Film", idFilm=row[0], primaryTitle=row[1],
    startYear=row[2])
44              importData.append(n)
45              i += 1
46
47          try:
48              create_nodes(graph.auto(), importData, labels={"Film"})
49              exportedCount += len(rows)
50              print(f"{exportedCount}/{totalCount} film records
    exported to Neo4j")
51          except Exception as error:
52              print(error)
53
54      # Artists
55      # En vous basant sur ce qui a ete fait dans la section precedente
    , exportez les donnees de la table tArtist
56      # A COMPLETER
57      exportedCount = 0
58      cursor.execute("SELECT COUNT(1) FROM tArtist")
59      totalCount = cursor.fetchval()
60      cursor.execute("SELECT idArtist, primaryName, birthYear FROM
    tArtist")
61      while True:
62          importData = []
63          rows = cursor.fetchmany(BATCH_SIZE)
64          if not rows:
65              break
66
67          i = 0
68          for row in rows:
69              # Creer un objet Node avec comme label Film et les
    proprietes adequates
70              # A COMPLETER
71              n = Node("Artist", idArtist=row[0], primaryName=row[1],
    birthYear=row[2])
72              importData.append(n)
73              i += 1
74
75          try:
76              create_nodes(graph.auto(), importData, labels={"Artist"})
77              exportedCount += len(rows)
78              print(f"{exportedCount}/{totalCount} artist records
    exported to Neo4j")
79          except Exception as error:
80              print(error)
81
82      try:
83          print("Indexing Film nodes...")
```

```
        graph.run("CREATE INDEX FOR (n:Film) ON n.idFilm")
        print("Indexing Artist nodes...")
        graph.run("CREATE INDEX FOR (n:Artist) ON n.idArtist")
    except Exception as error:
        print(error)


    # Relationships
    exportedCount = 0
    cursor.execute("SELECT COUNT(1) FROM tJob")
    totalCount = cursor.fetchval()
    cursor.execute(f"SELECT idArtist, category, idFilm FROM tJob")
    while True:
        importData = { "acted in": [], "directed": [], "produced": [], "composed": [] }
        rows = cursor.fetchmany(BATCH_SIZE)
        if not rows:
            break

        for row in rows:
            relTuple=(row[0], {}, row[2])
            importData[row[1]].append(relTuple)

        try:
            for cat in importData:
                # Utilisez la fonction create_relationships de py2neo pour creer les relations entre les noeuds Film et Artist
                # (les tuples necessaires ont deja ete crees ci-dessus dans la boucle for precedente)
                # https://py2neo.org/2021.1/bulk/index.html
                # ATTENTION: remplacez les espaces par des _ pour nommer les types de relation
                create_relationships(graph.auto(), importData[cat], cat.upper().replace(" ", "_"), start_node_key=("Artist", "idArtist"), end_node_key=("Film", "idFilm"))
            exportedCount += len(rows)
            print(f"{exportedCount}/{totalCount} relationships exported to Neo4j")
        except Exception as error:
            print(error)
```