Université Paris Cité

Projet : implémentation d'une BD graphe dans Neo4j

Données historiques de la blockchain Bitcoin

Léonard Namolaru

4 janvier 2023

La structure globale du fichier est basée sur un modèle de la galerie des modèles de Google Docs : Fiche de lecture par Reading Rainbow

Informations générales

Les informations d'identification du document

Date du document :	04/01/23
Version du document :	1.00

Les auteurs du document

Léonard Namolaru

Sommaire

Sommaire 2

1. Choix et import du jeu de données 3

2. Requêtes en Cypher 12

3. Plans d'exécution 21

4. Analytique de graphe 25

5. Intégration de Neo4j dans une application 30

1. Choix et import du jeu de données

1.1. Choix du jeu de données

Le jeu de données choisi est « **Bitcoin Blockchain Historical Data** » : données historiques complètes de la Blockchain Bitcoin.

Nous faisons souvent la confusion entre bitcoins et blockchain. Le bitcoin est une application particulière de la blockchain, qui repose sur une valeur monétaire : c'est une crypto-monnaie.[1]

La technologie Blockchain, mise en œuvre pour la première fois en 2009 en tant que composant central de Bitcoin, est un grand livre public distribué enregistrant les transactions. Son utilisation permet une communication peer-to-peer sécurisée en reliant des blocs contenant des pointeurs de hachage à un bloc précédent, un horodatage et des données de transaction. Bitcoin est une monnaie numérique décentralisée (crypto-monnaie) qui exploite la Blockchain pour stocker les transactions de manière distribuée.[2]



https://www.kaggle.com/datasets/bigquery/bitcoin-blockchain

[1] Les blockchains : de la théorie à la pratique, de l'idée à l'implémentation, 2e édition - C 2019
Chouli Billal - Goujon Frédéric - Leporcher Yves-Michel

[2] https://www.kaggle.com/datasets/bigquery/bitcoin-blockchain [Partie Contexte]

1.2. Création de fichiers csv

Les principaux fournisseurs de solutions Cloud proposent des services de Big Data. Google, par exemple, propose **BigQuery** : un service de type Big Data pour manipuler plusieurs téraoctets de données. **BigQuery** peut être testée gratuitement mais nécessite toutefois la création d'un compte Gmail puis la création d'un projet Big Data sur le Cloud Google.[3] Nous avons utilisé ce service après avoir réalisé qu'en raison de la grande quantité de données, le jeu de données que nous avons choisi sur le site **kaggle.com** n'est en fait accessible que via **BigQuery** :

Il est possible de voir sur la page Web du jeu de données que nous avons choisi qu'il n'y a aucun moyen de télécharger les données sous forme de fichiers csv ou dans tout autre format. C'est à ce moment-là que nous nous sommes rendu compte que dans notre cas, nous devions créer nous-mêmes les fichiers csv en utilisant les données sur **BigQuery**. Cela peut sembler anodin, mais en pratique, pour ceux qui ne connaissent pas la procédure (comme nous), c'était à nous d'apprendre et effectuer toutes les étapes, paramétrages et écriture du code Python qui nous ont permis d'obtenir les données. En fait, la compréhension même de la façon dont nous devons nous comporter pour écrire le code Python nous a obligés à faire beaucoup de recherches sur Internet, car il n'y avait pas de source unique qui rassemblait toutes les informations en un seul endroit.

La première étape : pour les réglages et les paramétrages au niveau de Google, nous avons suivi la procédure détaillée ici (étapes 1-6) :
https://cloud.google.com/bigquery/docs/quickstarts/quickstart-client-libraries

Pour la 6ème étape, la commande nécessaire sous Unix (WSL[4] dans le cas de cette exemple) est du type

```
$export GOOGLE_APPLICATION_CREDENTIALS=
"/mnt/c/Users/lenny/Downloads/erudite-scholar- ....-.....json"
```

[3] Bases de données NoSQL et big data : concevoir des bases de données pour le big data
Lacomme Philippe - Aridhi Sabeur - Phan Raksmey [DL 2014, cop. 2014]

[4] Windows Subsystem for Linux

Deuxièmement étape :

```
$ pip install google-cloud-storage
$ pip install google-cloud-bigquery
```

Nous pouvons maintenant exécuter le code suivant qui nous imprime les noms des tables qui existent dans notre jeu de données sur **BigQuery** :

```
from google.cloud import bigquery

project = 'bigquery-public-data'
dataset = 'crypto_bitcoin'

client = bigquery.Client()

print([ds.table_id for ds in client.list_tables(f'{project}.{dataset}')])
```

Si comme nous vous recevez maintenant le message d'erreur suivant,

Access Denied: Project erudite-scholar-174023: User does not have bigquery.jobs.create permission in project

Les étapes suivantes doivent être effectuées :

https://console.cloud.google.com/iam-admin/iam ⇒ Ajouter des comptes principaux ⇒ le “compte de service” qu’on a créée ⇒ Accorder l'accès ⇒ Attribuer des rôles ⇒ Barre de recherche : « Administrateur BigQuery »

Dans ce cas, il s'agit de donner un accès très large pour effectuer une grande variété d'actions, on aurait pu se contenter d'un accès beaucoup plus restreint bien sûr. Nous exécutons à nouveau le code et obtenons le résultat suivant :

```
lenny@DESKTOP-DMJ749K:/mnt/c/Users/lenny/OneDrive/שולחן העבודה$ python3 bdd.py
['blocks', 'inputs', 'outputs', 'transactions']
```

Le test nous montre que nous avons accès à l'API **BigQuery** et nous pouvons maintenant écrire du code qui nous permettra de télécharger ces tables depuis **BigQuery**. En raison de la grande quantité de données, et afin de ne pas ralentir drastiquement les performances de Neo4j, nous avons décidé de nous

concentrer sur les transactions Bitcoin du mois d'octobre 2022. Même les données pour ce mois sont très volumineuses et nous nous sommes donc limités à 10 000 transactions. Sur la base de l'identifiant unique de ces transactions (le hash de chaque transaction) nous avons effectué des requêtes supplémentaires afin d'obtenir les blocs, les entrées (« inputs ») et sorties (« outputs ») les concernant. Une autre raison de la décision de nous limiter à cette quantité de données est qu'après une certaine quantité de téléchargement de données, Google demande un paiement.

```
# -*- coding: utf-8 -*-
# Un article qui nous a ete tres utile dans le cadre de la redaction de ce code :
https://www.kaggle.com/code/nocibambi/getting-started-with-bitcoin-data
# (tout en apportant des modifications et des ajustements selon les besoins de notre projet)

from google.cloud import bigquery # pip install google-cloud-bigquery
import pandas as pd
from time import time

def estimate_gigabytes_scanned(query : str, big_query_client : bigquery.client.Client) -> float :
    """
    Une fonction utile pour estimer la taille d'une requête.
    Sources:
    - https://www.kaggle.com/sohier/beyond-queries-exploring-the-bigquery-api/
    - https://www.kaggle.com/code/nocibambi/getting-started-with-bitcoin-data
    - Apache 2.0 open source license.
    """
    # Nous initions un objet `QueryJobConfig`
    # Description de l'API :
https://googleapis.dev/python/bigquery/latest/generated/google.cloud.bigquery.job.QueryJobConfig.html
    my_job_config = bigquery.job.QueryJobConfig()

    # Nous activons le "dry run" en définissant l'attribut "dry_run" de l'objet "QueryJobConfig".
    # Cela signifie que nous n'exécutons pas réellement la requête, mais estimons son coût de fonctionnement.
    my_job_config.dry_run = True # True si cette requête doit être un essai pour estimer les coûts.

    # We activate the job_config by passing the `QueryJobConfig` to the client's `query` method.
    my_job = big_query_client.query(query, job_config=my_job_config)

    # Les résultats se présentent sous forme d'octets que nous convertissons en gigaoctets pour une meilleure lisibilité
    BYTES_PER_GB = 2**30
    estimate = my_job.total_bytes_processed / BYTES_PER_GB

    return estimate # Cette requête traitera {estimate} Go (Go = GB)

def query_execution(query : str, big_query_client : bigquery.client.Client, csv_file_name : str) -> None :
    """
    Une fonction qui recoit une requete et cree un fichier csv avec les donnees demandees.
    """
    estimate = estimate_gigabytes_scanned(query, big_query_client)
    print(f"Cette requete traitera {estimate} GBs.")

    bytes_in_gigabytes = 2**30
    safe_config = bigquery.QueryJobConfig( maximum_bytes_billed= int(estimate + 10) * bytes_in_gigabytes )

    query_job = client.query(query, job_config=safe_config)
    start = time()

    result = query_job.result()
    data_frame = result.to_dataframe()
```

```
    duration = (time() - start) / 60
    print(f"Temps ecoule : {duration:.2f} minutes")

    print( data_frame.head() )
    data_frame.to_csv(csv_file_name, index=False)


if __name__ == '__main__' :
    # Description de l'API :
https://cloud.google.com/python/docs/reference/bigquery/latest/google.cloud.bigquery.client.Client
    client = bigquery.Client()

    common_part_for_all_queries = 'FROM `bigquery-public-data.crypto_bitcoin.transactions` '\
                                  'WHERE '\
                                  'EXTRACT(YEAR FROM block_timestamp) = 2022 AND '\
                                  'EXTRACT(MONTH FROM block_timestamp) = 10 '\
                                  'ORDER BY `hash` '\
                                  'LIMIT 10000'


    query_1 = 'SELECT `hash`,size,virtual_size,version,lock_time,block_hash,'\
            'block_number,block_timestamp,block_timestamp_month,input_count,output_count,'\
            'input_value,output_value,is_coinbase,fee ' + common_part_for_all_queries

    query_execution(query_1, client, 'transactions_202210.csv')


    query_2 = 'SELECT * '\
            'FROM `bigquery-public-data.crypto_bitcoin.blocks` '\
            'WHERE '\
            '`hash` IN (SELECT block_hash ' + common_part_for_all_queries + ') '

    query_execution(query_2, client, 'blocks_202210.csv')

    query_3 = 'SELECT * '\
            'FROM `bigquery-public-data.crypto_bitcoin.inputs` '\
            'WHERE '\
            'transaction_hash IN (SELECT `hash` ' + common_part_for_all_queries + ') '

    query_execution(query_3, client, 'inputs_202210.csv')

    query_4 = 'SELECT * '\
            'FROM `bigquery-public-data.crypto_bitcoin.outputs` '\
            'WHERE '\
            'transaction_hash IN (SELECT `hash` ' + common_part_for_all_queries + ') '

    query_execution(query_4, client, 'outputs_202210.csv')
```

L'exécution du code nous donne le résultat suivant :

```
lenny@DESKTOP-DMJ749K:/mnt/c/Users/lenny/OneDrive/Documents/Master 2/Premiere_periode/Bases de donnees specialisees/Neo4
j/bitcoin-blockchain-neo4j$ python3 get_bitcoin_data.py
Cette requete traitera 184.5652655735612 GBs.
Temps ecoule : 0.12 minutes
                                          hash  size  ...  is_coinbase             fee
0  0000006e4c9a8f4d58c5799842d512b53c8fc1dd6430f5...   226  ...        False   1000.000000000
1  00000408aea4b6c821aa95a82bbae5ae5eb8d2ca0a3d70...   419  ...        False  52173.000000000
2  0000055dad28676f6ec4ba19a17a505ef8f3f9fa8b6199...   224  ...        False    958.000000000
3  000006888d81b159f200eccdd5802a82789aa2d6dc3aae...   191  ...        False   6992.000000000
4  0000074b7dfd6b82ce9f5f77d13132cf4cfc220dd1738c...   225  ...        False   2713.000000000

[5 rows x 15 columns]
Cette requete traitera 102.37152341939509 GBs.
Temps ecoule : 0.89 minutes
                                          hash  ...  transaction_count
0  000000000000000000005c697b752fa51389838128ea15b...  ...                983
1  000000000000000000006401d01a38d79c33130bd35471e...  ...               2262
2  000000000000000000031acff4d3e4eb166339680bc8a9...  ...               3086
3  000000000000000000005876cde1f464ecec1696b791ce3...  ...               2327
4  000000000000000000079b874244c6781bf35879bed357...  ...               2243

[5 rows x 13 columns]
Cette requete traitera 1090.249997222796 GBs.
Temps ecoule : 0.83 minutes
                                transaction_hash  ...              value
0  0035535af19120dad1355e4984007f30343ec88807812c...  ...    1392234.000000000
1  002e8b71ce206974cfc780511981bae40541d9726a9173...  ...     212518.000000000
2  0019889c9b25636947f92c9cf183cf820ec9faccca0762...  ...  434740183.000000000
3  0006823b4288557e3f1ec7804680144ae3cb20b10e0b9d...  ...       9687.000000000
4  0006823b4288557e3f1ec7804680144ae3cb20b10e0b9d...  ...      28026.000000000

[5 rows x 14 columns]
Cette requete traitera 535.4311451828107 GBs.
Temps ecoule : 0.53 minutes
                                transaction_hash  ...             value
0  000c179293bc0fbc7c0f48e88561a9512dfb99b9fd66aa...  ...   300000.000000000
1  000c179293bc0fbc7c0f48e88561a9512dfb99b9fd66aa...  ...     9252.000000000
2  000008e090a4d0c51ac2cc851a836098da301e3aee0efa...  ...   107868.000000000
```

1.3. Import des fichiers csv

Au vu de la grande quantité de données (4 fichiers CSV avec beaucoup de données à importer dans chacun d'eux) nous avons décidé d'effectuer l'import en plusieurs étapes. Pour chaque requête d'importation, nous avons ajouté une note sur le temps qu'il nous a fallu pour importer les données.

Import des blocs

```
// Added 7214 labels, created 7214 nodes, set 43284 properties, created 3607 relationships, completed
after 1006 ms.
LOAD CSV WITH HEADERS FROM "file:/blocks_202210.csv" AS blocks_csv
CREATE (block:Block{ hash: blocks_csv.hash,
                          size:toInteger(blocks_csv.size),
                          stripped_size:toInteger(blocks_csv.stripped_size),
                          weight:toInteger(blocks_csv.weight),
                          number:toInteger(blocks_csv.number),
                          version:toInteger(blocks_csv.version),
                          markle_root: blocks_csv.markle_root,
                          timestamp: blocks_csv.timestamp,
                          timestamp_month: date(blocks_csv.timestamp_month),
                          nonce: blocks_csv.nonce,
                          bits: blocks_csv.bits,
                          transaction_count:toInteger(blocks_csv.transaction_count)
                        })-[:coinbase]->(coinbase:Coinbase{value: blocks_csv.coinbase_param})
```

Import des transactions

```
// Added 10000 labels, created 10000 nodes, set 149993 properties, created 10000 relationships, completed
after 20832 ms.
LOAD CSV WITH HEADERS FROM "file:/transactions_202210.csv" as transactions_csv
MERGE (block:Block {hash:transactions_csv.block_hash})
CREATE (transaction:Transaction {  hash: transactions_csv.hash,
                                   size: toInteger(transactions_csv.size),
                                   virtual_size: toInteger(transactions_csv.virtual_size),
                                   version: toInteger(transactions_csv.version),
                                   lock_time: toInteger(transactions_csv.lock_time),
                                   block_hash: transactions_csv.block_hash,
                                   block_number: toInteger(transactions_csv.block_number),
                                   block_timestamp: transactions_csv.block_timestamp,
                                   block_timestamp_month: date(transactions_csv.block_timestamp_month),
                                   input_count: toInteger(transactions_csv.input_count),
                                   output_count: toInteger(transactions_csv.output_count),
                                   input_value: toFloat(transactions_csv.input_value),
                                   output_value: toFloat(transactions_csv.output_value),
                                   is_coinbase: toBoolean(transactions_csv.is_coinbase),
                                   fee: toFloat(transactions_csv.fee)
                                 }
                               )
CREATE (transaction)-[:inc]->(block)
```

Import des inputs

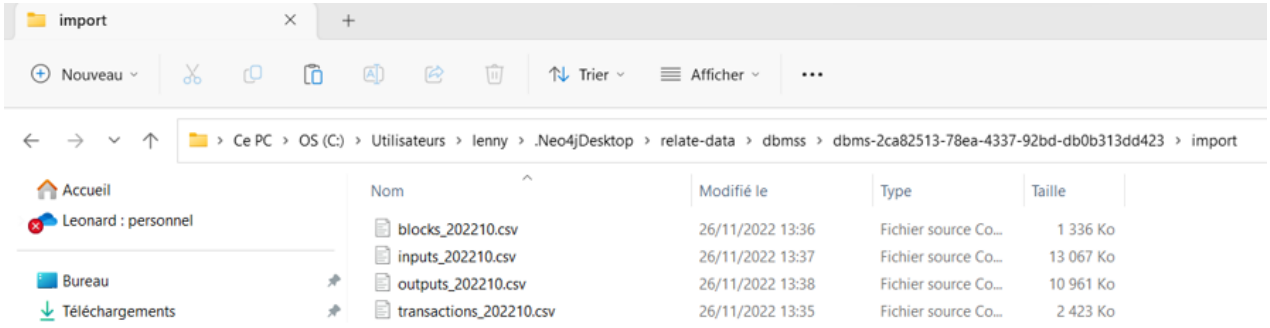
```
// Added 29589 labels, created 29589 nodes, set 178318 properties, created 29589 relationships, completed
after 401933 ms.
LOAD CSV WITH HEADERS FROM "file:/inputs_202210.csv" as inputs_csv
MERGE (transaction:Transaction {hash: inputs_csv.transaction_hash})
CREATE (input :Input {index: toInteger(inputs_csv.index),
                          spent_transaction_hash: inputs_csv.spent_transaction_hash,
                           sent_output_index: toInteger(inputs_csv.sent_output_index),
                           required_signatures: toInteger(inputs_csv.required_signatures),
                           type: inputs_csv.type,
                           value: toFloat(inputs_csv.value)
                         }
                       )
CREATE (input)-[:in {script_asm: inputs_csv.script_asm, script_hex: inputs_csv.script_hex, sequence:
toInteger(inputs_csv.sequence)}]->(transaction)
```

Import des outputs

```
// Added 64062 labels, created 64062 nodes, set 192186 properties, created 64062
relationships, completed after 354477 ms.
LOAD CSV WITH HEADERS FROM "file:/outputs_202210.csv" as outputs_csv
MERGE (transaction:Transaction {hash: outputs_csv.transaction_hash})
CREATE (output :Output {index: toInteger(outputs_csv.index),
                           required_signatures: toInteger(outputs_csv.required_signatures),
```

```
                        type: outputs_csv.type,
                        value: toFloat(outputs_csv.value),
                        script_asm: outputs_csv.script_asm,
                        script_hex: outputs_csv.script_hex
                       }
                      )
CREATE (transaction)-[:out]->(output)
// Les outputs apparaissent sous le format ['yyyxx'], c'est-à-dire sous un format de tableau,
// bien qu'au moins dans notre jeu de données il n'y ait qu'une seule valeur dans le tableau,
// nous utilisons donc la fonction split(original, splitDelimiter) pour extraire l'adresse
elle-même du tableau
CREATE (address :Address {address: split(outputs_csv.addresses, "'")[1]})
CREATE (output)-[:locked]->(address)
```

Pour que les données soient importées, les données doivent se trouver dans le dossier suivant :



1.4. Contraintes d'intégrité

```
// DROP CONSTRAINT constraint_name [IF EXISTS]

 CREATE CONSTRAINT unique_block_hash
FOR (block :Block)
REQUIRE block.hash IS UNIQUE

CREATE CONSTRAINT unique_transaction_hash
FOR (transaction :Transaction)
REQUIRE transaction.hash IS UNIQUE
```

```
CREATE CONSTRAINT exist_block_hash
FOR (block :Block)
REQUIRE (block.hash) IS NOT NULL



CREATE CONSTRAINT exist_transaction_hash
FOR (transaction :Transaction)
REQUIRE (transaction.hash) IS NOT NULL

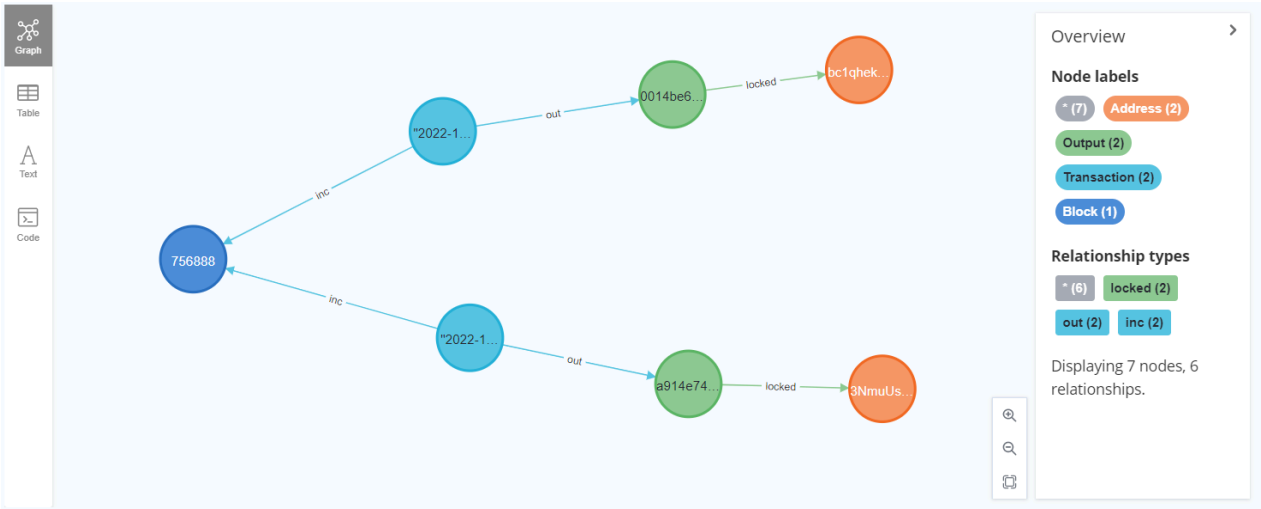
CREATE CONSTRAINT exist_address
FOR (addr :Address)
REQUIRE (addr.address) IS NOT NULL
```

2. Requêtes en Cypher

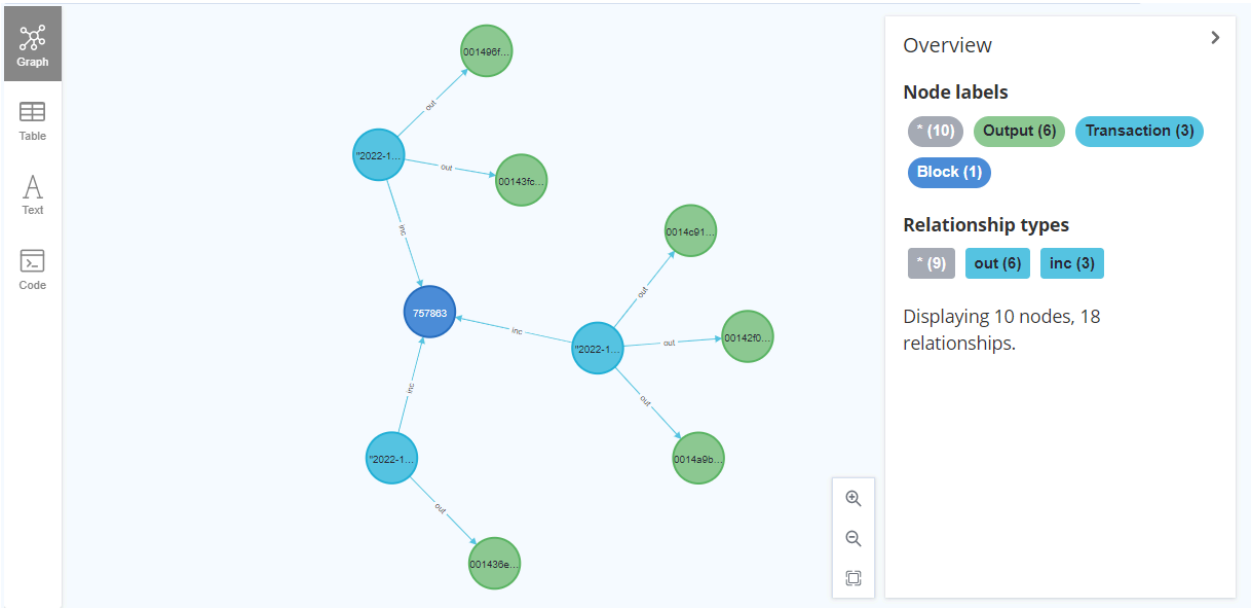
Trouver des chemins entre des transactions et des adresses est probablement l'une des choses les plus intéressantes que nous puissions faire avec les données historiques de la Blockchain Bitcoin à l'aide de Neo4j.

Par exemple :

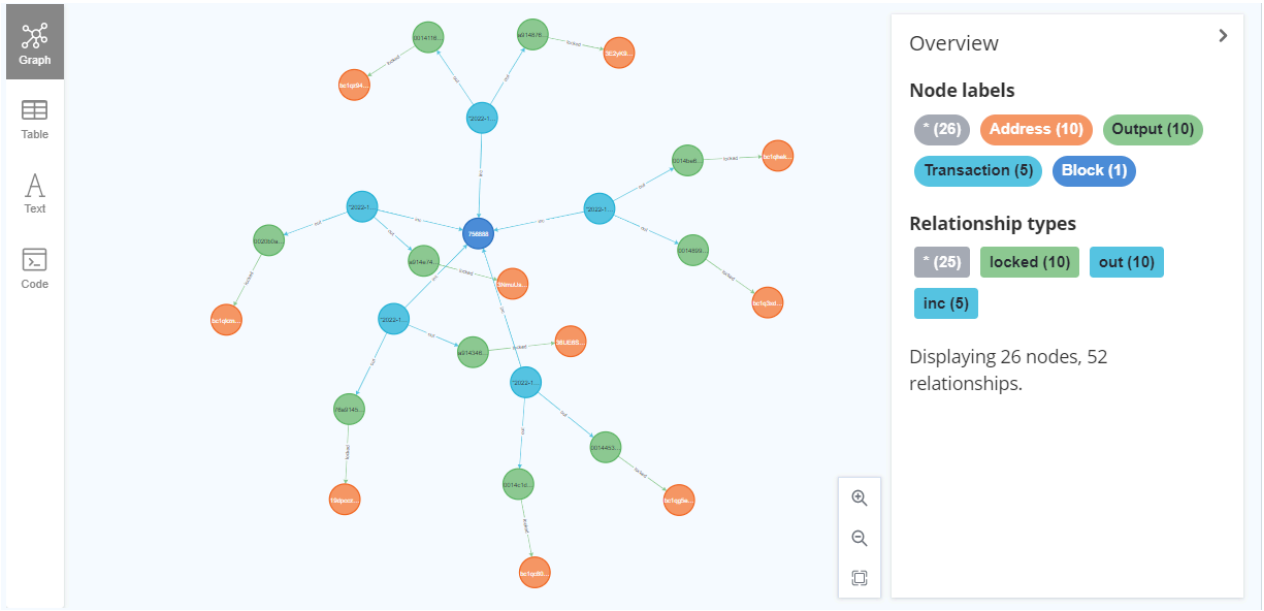
```
MATCH
n=(start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'}
)-[*]-(end:Address{address:'3NmuUsUNAbnuGoSTR47sJ3YLD1q7tc7YxC'})
RETURN n
```




```
MATCH p= (start_output:Output)-[*]-(end_output:Output)
RETURN p
LIMIT 5
```




```
MATCH
n=(start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'})-[*]-(end:
Address)
RETURN n
```



Une requête avec OPTIONAL MATCH

La clause **OPTIONAL MATCH** est utilisée pour rechercher le modèle qui y est décrit, tout en utilisant des valeurs nulles pour les parties manquantes du modèle.[5]

```
MATCH (transaction:Transaction)
OPTIONAL MATCH (transaction{output_count: 2})-[:out]->(output:Output)
RETURN transaction.hash, output.value
```

Cette requête renvoie les valeurs des outputs des transactions qui ont 2 outputs (Si il y a plus ou moins que 2 outputs pour une transaction, on obtient la valeur **null** :

Table | Text | Code

	transaction.hash	output.value
1	"0000006e4c9a8f4d58c5799842d512b53c8fc1dd6430f53357a9db4badcbedab"	2495500.0
2	"0000006e4c9a8f4d58c5799842d512b53c8fc1dd6430f53357a9db4badcbedab"	7813379.0
3	"00000408aea4b6c821aa95a82bbae5ae5eb8d2ca0a3d70936885991ab53666f9"	*null*
4	"0000055dad28676f6ec4ba19a17a505ef8f3f9fa8b619983f09e0256842e16e6"	192159490.0
5	"0000055dad28676f6ec4ba19a17a505ef8f3f9fa8b619983f09e0256842e16e6"	90189300.0
6	"000006888d81b159f200eccdd5802a82789aa2d6dc3aaeea304397ab89399f12"	*null*
7		

Started streaming 16434 records after 8 ms and completed after 10 ms, displaying first 1000 rows.

Une requête utilisant des fonctions de prédicat du type all(), any(), exists(), none(), single()

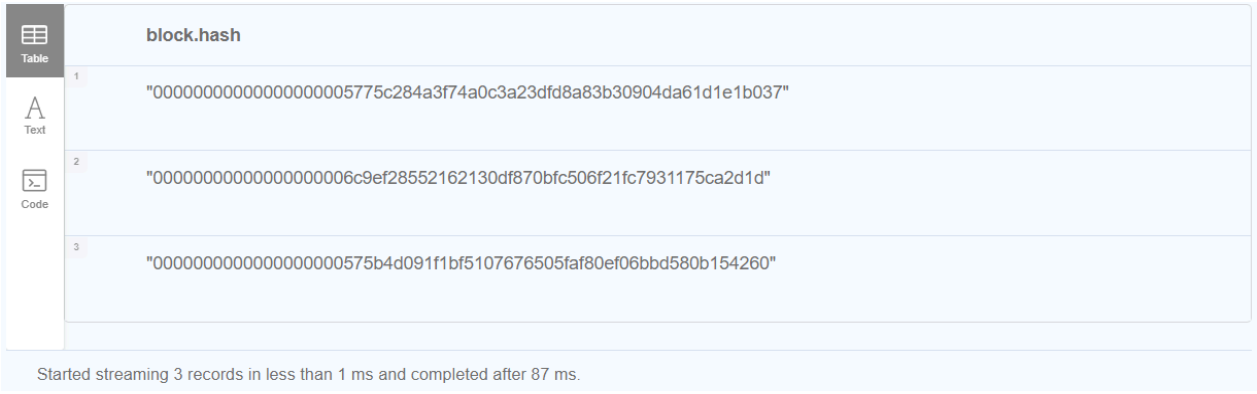
- **all()** : si le prédicat vaut pour tous les éléments d'une liste
- **any()** : si le prédicat vaut pour au moins un élément d'une liste

[5] Cypher Manual : OPTIONAL MATCH : https://neo4j.com/docs/cypher-manual/current/clauses/optional-match/

- **exists()** : s'il y a un match pour le pattern ou si la propriété vaut pour le nœud ou la relation
- **none()** : si le prédicat ne vaut pour aucun élément de la liste
- **single()** : si le prédicat vaut pour exactement un élément de la liste

Source : Slides cours 2 et 3, slide 109

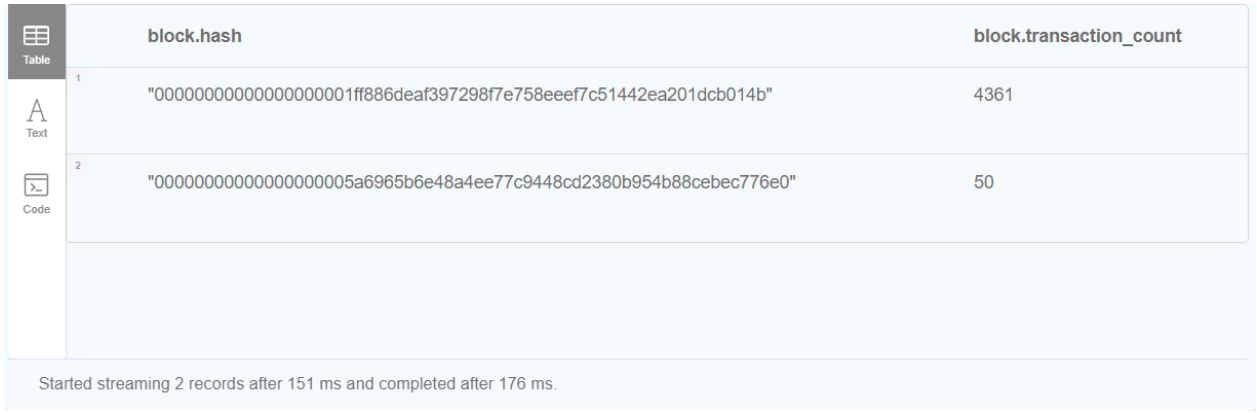
```
MATCH (block:Block)
WHERE NOT EXISTS ((block)<-[:inc]-({is_coinbase: false}))
RETURN block.hash
```



Un filtre post UNION avec CALL

Faut faire attention aux valeurs **null**s car une valeur **transaction_count** non renseignée est classée à la fin par **ORDER BY** (considérée nulle).

```
CALL {
    MATCH (block: Block) WHERE block.transaction_count IS NOT NULL
    RETURN block ORDER BY block.transaction_count ASC LIMIT 1
        UNION
    MATCH (block: Block) WHERE block.transaction_count IS NOT NULL
    RETURN block ORDER BY block.transaction_count DESC LIMIT 1
}
RETURN block.hash, block.transaction_count ORDER BY block.hash
```



	block.hash	block.transaction_count
1	"00000000000000000001ff886deaf397298f7e758eeef7c51442ea201dcb014b"	4361
2	"00000000000000000005a6965b6e48a4ee77c9448cd2380b954b88cebec776e0"	50

Started streaming 2 records after 151 ms and completed after 176 ms.

Une requête utilisant COLLECT et UNWIND

collect(expression)

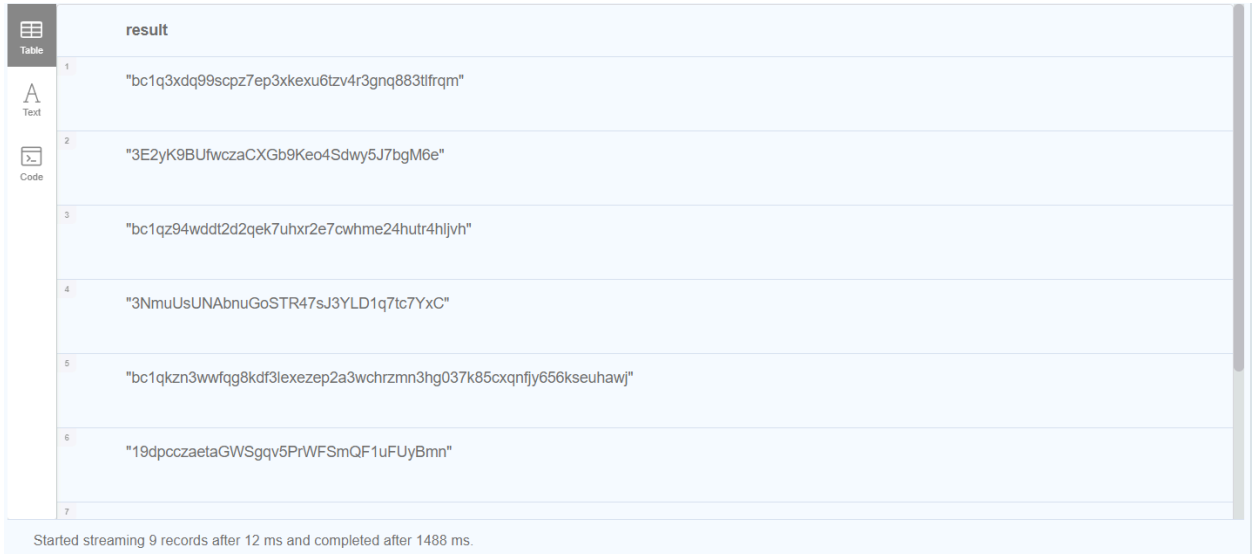
La fonction **collect()** renvoie une seule liste agrégée contenant les valeurs renvoyées par une **expression**. **collect()** renvoie une liste contenant des éléments hétérogènes ; les types des éléments sont déterminés par les valeurs renvoyées par **expression**. **expression** est une expression renvoyant un ensemble de valeurs.[6]

UNWIND développe une liste en une séquence de lignes : la clause **UNWIND** permet de retransformer n'importe quelle liste en lignes individuelles. Ces listes peuvent être des paramètres qui ont été transmis, des résultats précédemment **collect**és ou d'autres expressions de liste. Utilisation courante de la clause **UNWIND** : créer des listes distinctes ; Créer des données à partir des listes de paramètres fournies à la requête. La clause **UNWIND** nous oblige à spécifier un nouveau nom pour les valeurs internes.[7]

```
MATCH (start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'})-[*]-(end:Address)
WITH collect(end.address) AS addr
UNWIND addr AS result
RETURN result
```

[6] Cypher Manual Functions Aggregating functions :
https://neo4j.com/docs/cypher-manual/current/functions/aggregating/#functions-collect

[7] Cypher Manual UNWIND
https://neo4j.com/docs/cypher-manual/5/clauses/unwind/



Une requête manipulant des listes avec reduce()

reduce() renvoie la valeur résultant de l'application d'une expression sur chaque élément successif d'une liste en conjonction avec le résultat du calcul jusqu'à présent. Cette fonction parcourra chaque élément **e** dans la liste donnée, exécutera l'expression sur **e** — en tenant compte du résultat partiel actuel — et stockera le nouveau résultat partiel dans l'accumulateur. Cette fonction est analogue à la méthode **fold** ou **reduce** des langages fonctionnels tels que Lisp et Scala.[8]

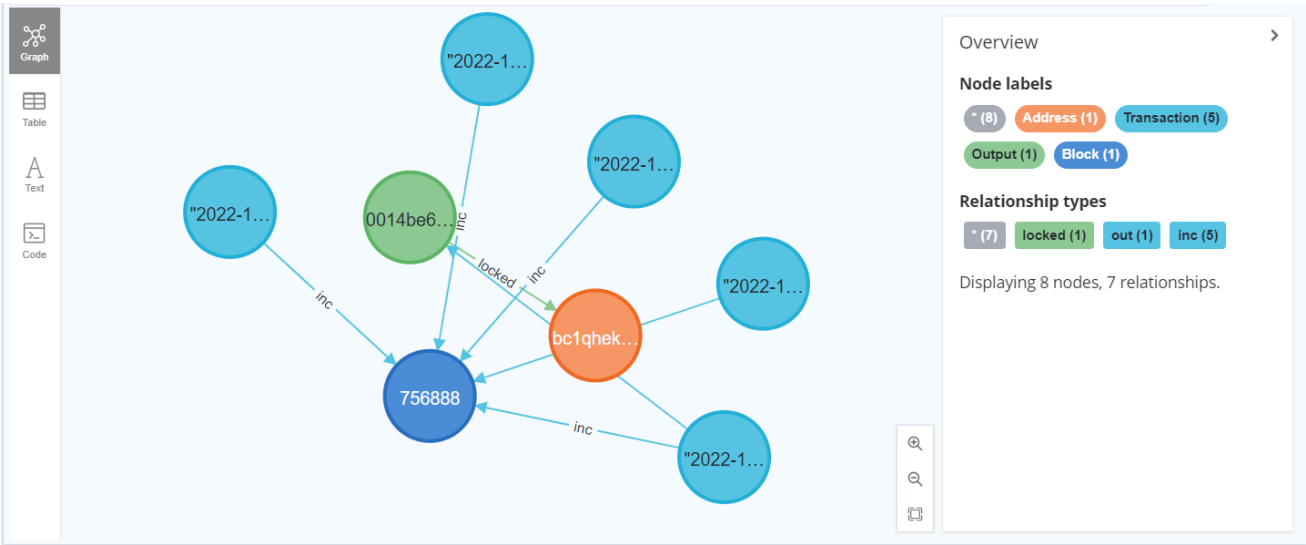
Syntaxe:

```
reduce(accumulator = initial, variable IN list | expression)
```

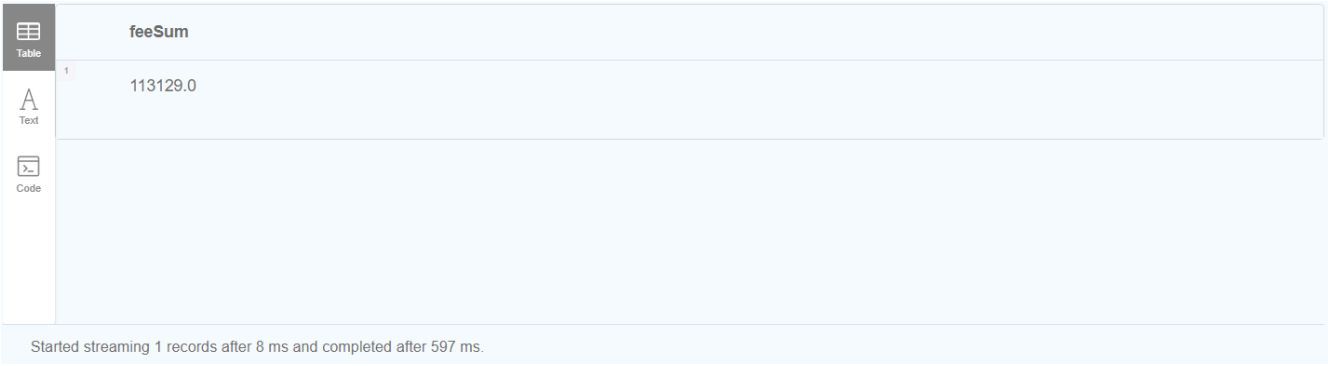
expression : cette expression s'exécutera une fois par valeur dans la liste.

```
MATCH p =
(start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'})-[*]-(transa
ction:Transaction)
RETURN p
```

[8] Cypher Manual - List functions https://neo4j.com/docs/cypher-manual/current/functions/list/#functions-reduce




```
MATCH
(start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'})-[*]-(transa
ction:Transaction)
WITH collect(transaction.fee) AS feeList
RETURN reduce(totalFee = 0, fee IN feeList | totalFee + fee) AS feeSum
```



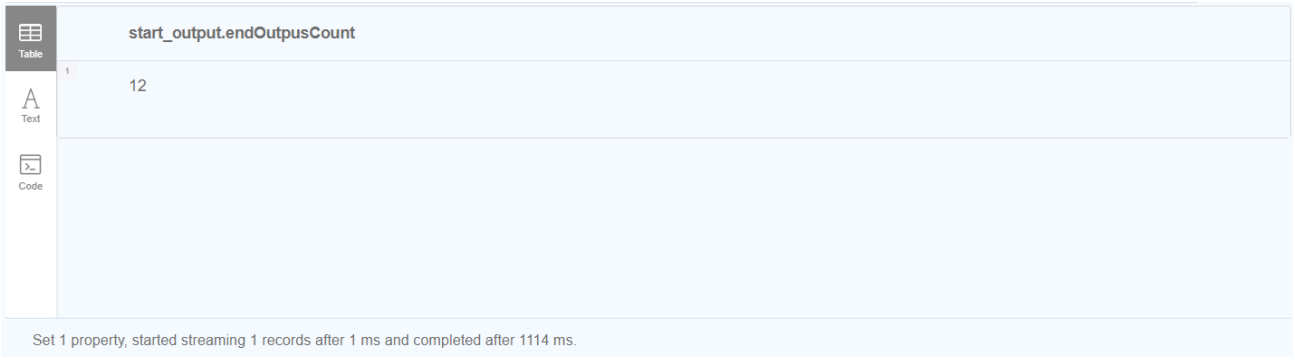
Deux utilisations de WITH différentes **(Par exemple, pour filtrer les résultats d'un agrégat, pour séparer lecture et mise à jour du graphe).**

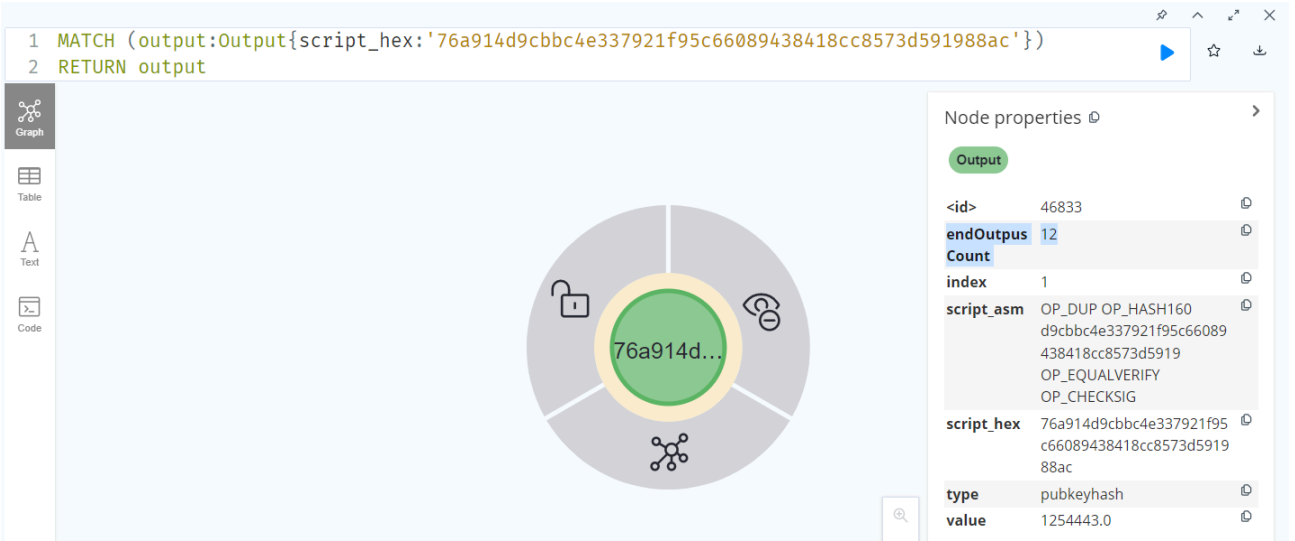
- **WITH** sert à enchaîner les clauses (≃ Unix pipe) : les variables doivent être incluses dans le **WITH** pour être visibles dans la clause suivante.
- Les résultats agrégés doivent passer par un **WITH** pour être filtrés par le **WHERE**

Source : Slides cours 2 et 3, slides 78-79

```
MATCH
(start_output:Output{script_hex:'76a914d9cbbc4e337921f95c66089438418cc8573d591988ac'})-[*]-(end_output:Output) // Lecture
WITH start_output, count(end_output) AS endOutpusCount // Transition
SET start_output.endOutpusCount = endOutpusCount // Mise a jour
RETURN start_output.endOutpusCount // Lecture
```

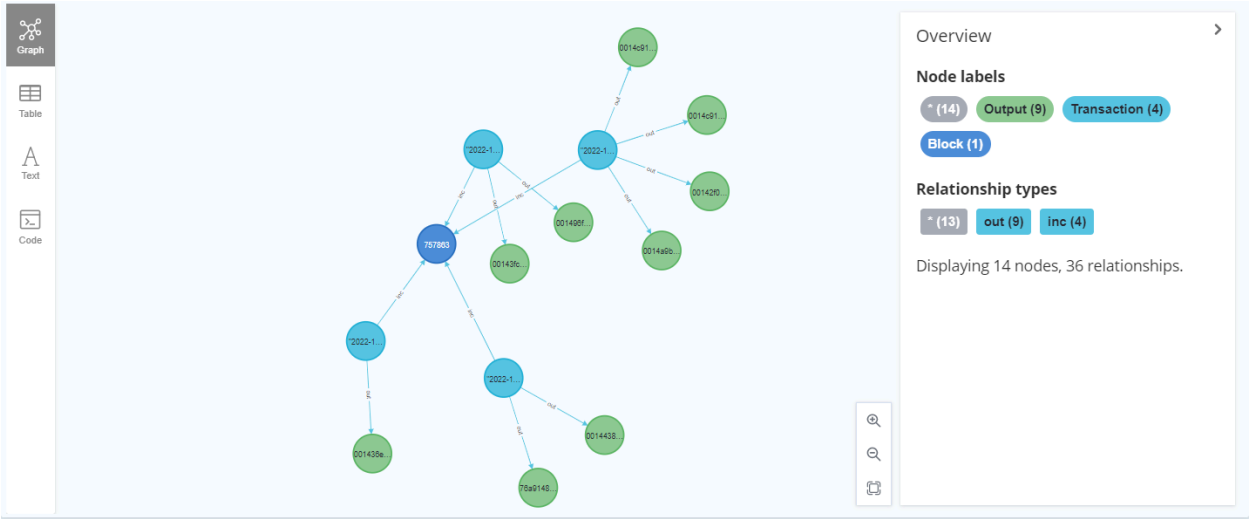
Mise à jour du graphe par ajout des données agrégées ; **WITH** permet ici de passer de la lecture à l'écriture.





Une requête explorant à la fois les données et la topologie du graphe

```
MATCH p=
(start_output:Output)-[*1..5]-(end_output:
Output)
RETURN p
LIMIT 10
```



.

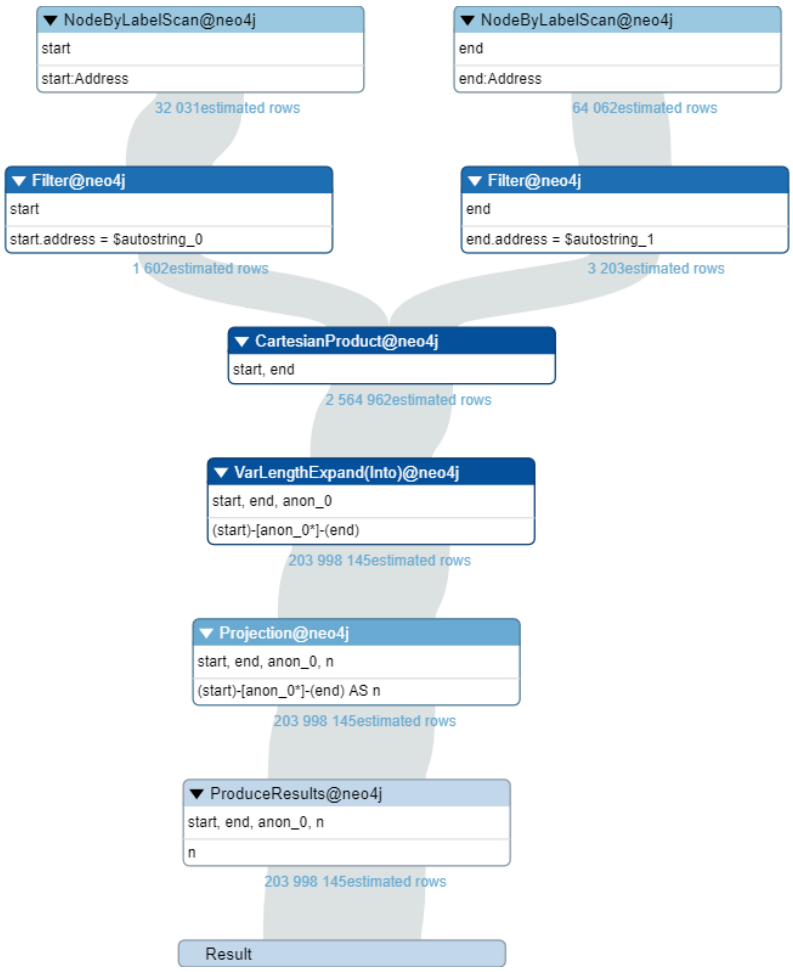
3. Plans d'exécution

- **PROFILE** : Afficher le plan d'exécution sans exécuter la requête.
- **EXPLAIN** : Afficher le plan d'exécution et exécuter la requête.

Source : Slides cours 2 et 3, slide 132

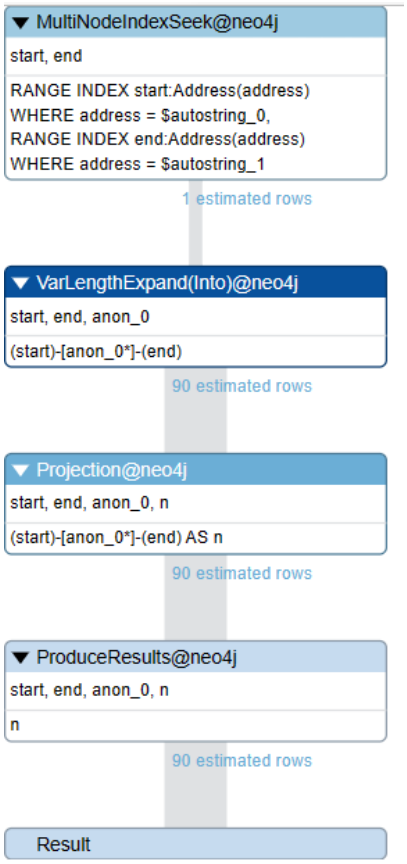
Exemple 1

```
EXPLAIN
MATCH
n=(start:Address{address:'bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6'})-[*]-(end:
Address{address:'3NmuUsUNAbnuGoSTR47sJ3YLD1q7tc7YxC'})
RETURN n
```



Création d'un index :

```
CREATE INDEX index_address for (addr:Address) on (addr.address)
```



Exemple 2

```
EXPLAIN CALL {

    MATCH (block: Block) WHERE block.transaction_count IS NOT NULL

    RETURN block ORDER BY block.transaction_count ASC LIMIT 1

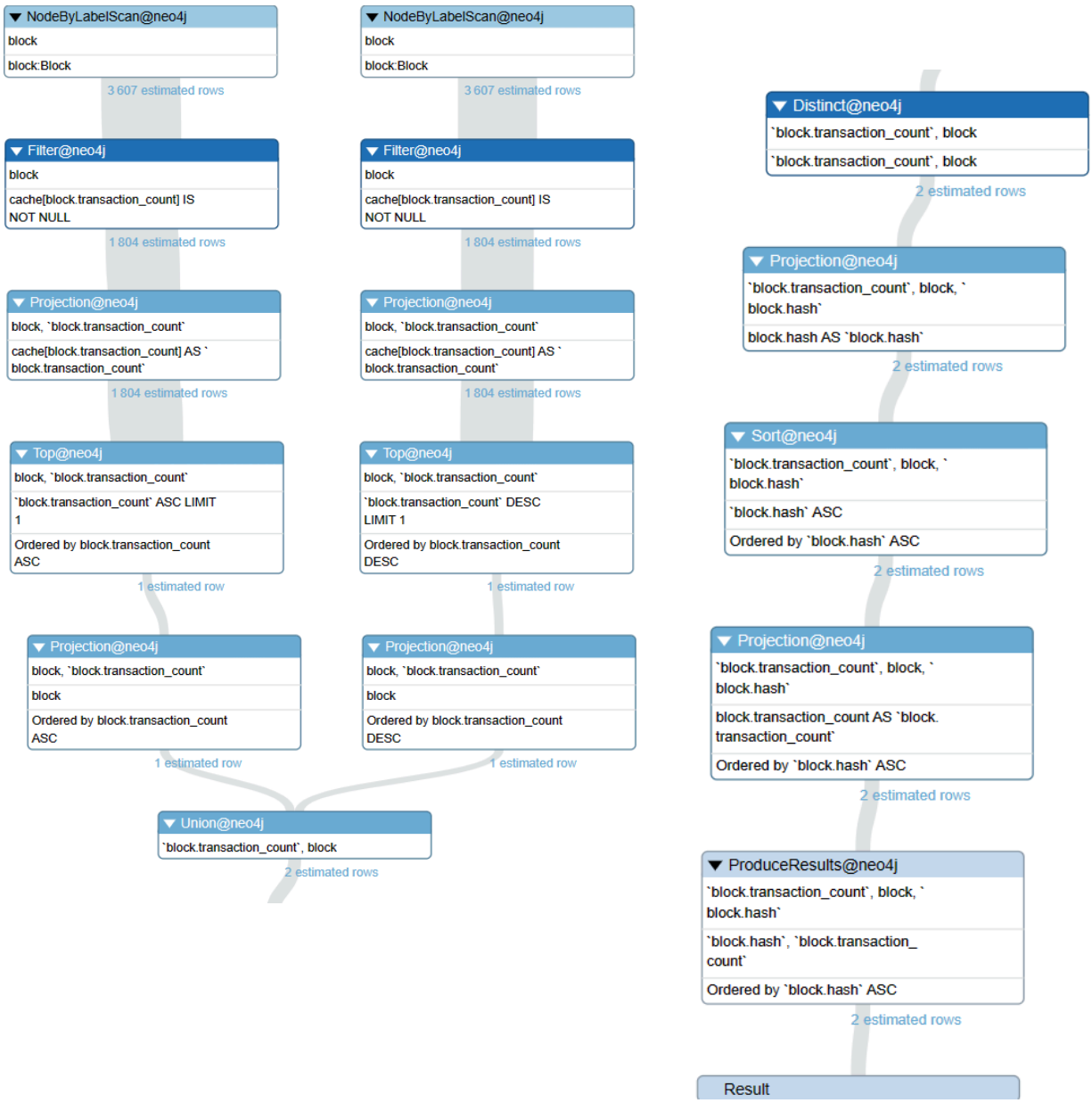
        UNION

    MATCH (block: Block) WHERE block.transaction_count IS NOT NULL

    RETURN block ORDER BY block.transaction_count DESC LIMIT 1

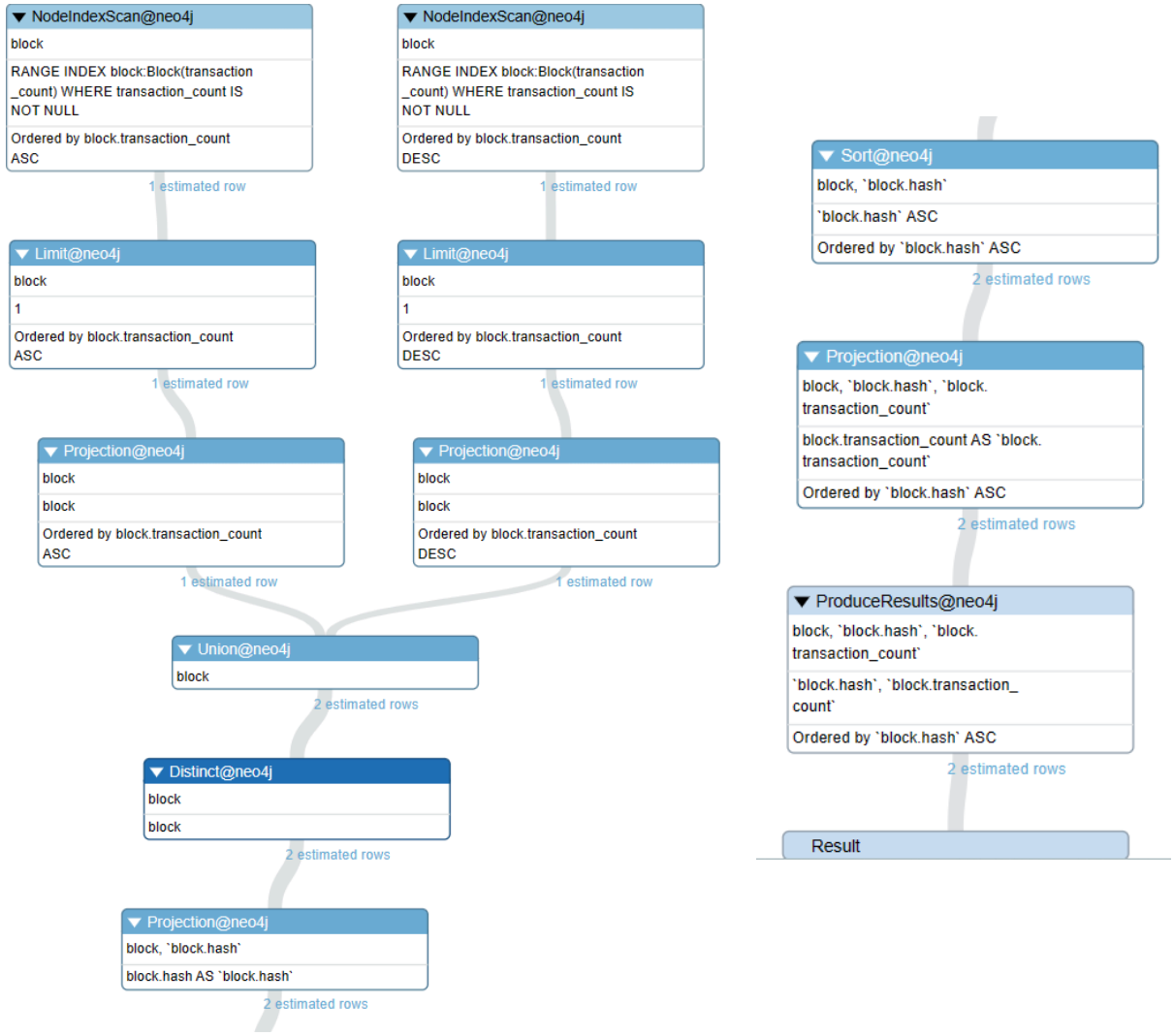
}
```

```
RETURN block.hash, block.transaction_count ORDER BY block.hash
```



Création d'un index :

```
CREATE INDEX index_transaction_count for (block:Block) on
(block.transaction_count)
```


NodeIndexScan@neo4j
block
RANGE INDEX block:Block(transaction _count) WHERE transaction_count IS NOT NULL
Ordered by block.transaction_count ASC
1 estimated row
NodeIndexScan@neo4j
block
RANGE INDEX block:Block(transaction _count) WHERE transaction_count IS NOT NULL
Ordered by block.transaction_count DESC
1 estimated row
Sort@neo4j
block, `block.hash`
`block.hash` ASC
Ordered by `block.hash` ASC
2 estimated rows
Limit@neo4j
block
1
Ordered by block.transaction_count ASC
1 estimated row
Limit@neo4j
block
1
Ordered by block.transaction_count DESC
1 estimated row
Projection@neo4j
block, `block.hash`, `block. transaction_count`
block.transaction_count AS `block. transaction_count`
Ordered by `block.hash` ASC
2 estimated rows
Projection@neo4j
block
block
Ordered by block.transaction_count ASC
1 estimated row
Projection@neo4j
block
block
Ordered by block.transaction_count DESC
1 estimated row
ProduceResults@neo4j
block, `block.hash`, `block. transaction_count`
`block.hash`, `block.transaction_ count`
Ordered by `block.hash` ASC
2 estimated rows
Union@neo4j
block
2 estimated rows
Distinct@neo4j
block
block
2 estimated rows
Projection@neo4j
block, `block.hash`
block.hash AS `block.hash`
2 estimated rows
Result

4. Analytique de graphe

PageRank

L'algorithme PageRank mesure l'importance de chaque nœud dans le graphique, en fonction du nombre de relations entrantes et de l'importance des nœuds source correspondants. L'hypothèse sous-jacente est qu'une page est aussi importante que les pages qui y renvoient.[9]

Nous avons utilisé cet algorithme pour classer l'importance des différents blocs en fonction du nombre de transferts enregistrés dans chaque bloc et de l'importance de chaque transfert.

Dans un premier temps nous avons créé un graphe nommé :

```
CALL gds.graph.project(
  'Graph_1',
  ['Block', 'Transaction'],
  'inc'
)
```



[9] Neo4j Graph Data Science - PageRank
https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/

En mode d'exécution **stream**, l'algorithme renvoie le score pour chaque nœud. Cela nous permet d'inspecter les résultats directement ou de les post-traiter dans Cypher sans aucun effet secondaire. Par exemple, nous pouvons ordonner les résultats pour trouver les nœuds avec le score PageRank le plus élevé[10]

```
CALL gds.pageRank.stream('Graph_1')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).hash AS hash, score
ORDER BY score DESC, hash ASC
```

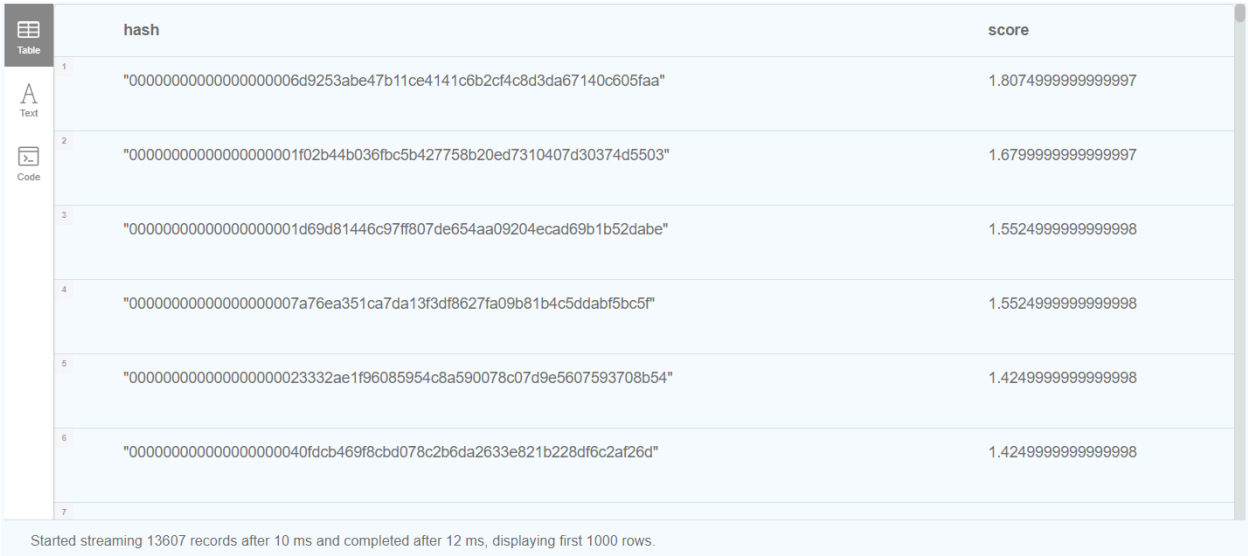


Table | Text | Code

	hash	score
1	"0000000000000000006d9253abe47b11ce4141c6b2cf4c8d3da67140c605faa"	1.8074999999999997
2	"00000000000000000001f02b44b036fbc5b427758b20ed7310407d30374d5503"	1.6799999999999997
3	"0000000000000000001d69d81446c97ff807de654aa09204ecad69b1b52dabe"	1.5524999999999998
4	"0000000000000000007a76ea351ca7da13f3df8627fa09b81b4c5ddabf5bc5f"	1.5524999999999998
5	"00000000000000000023332ae1f96085954c8a590078c07d9e5607593708b54"	1.4249999999999998
6	"0000000000000000040fdcb469f8cbd078c2b6da2633e821b228df6c2af26d"	1.4249999999999998
7		

Started streaming 13607 records after 10 ms and completed after 12 ms, displaying first 1000 rows.

En mode d'exécution **stats** (statistiques), l'algorithme renvoie une seule ligne contenant un résumé du résultat de l'algorithme. Par exemple, les statistiques PageRank renvoient un histogramme de centralité qui peut être utilisé pour surveiller la distribution des valeurs de score PageRank sur tous les nœuds calculés. [11]

```
CALL gds.pageRank.stats('Graph_1', {
  maxIterations: 20,
  dampingFactor: 0.85
})
YIELD centralityDistribution
RETURN centralityDistribution AS stats
```

[10] Neo4j Graph Data Science - PageRank
https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/

[11] Neo4j Graph Data Science - PageRank
https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/



Label Propagation[12]

L'algorithme de propagation d'étiquettes (Label Propagation algorithm, LPA) est un algorithme rapide pour trouver des communautés dans un graphe. Il détecte ces communautés en utilisant uniquement la structure du réseau comme guide et ne nécessite pas de fonction objective prédéfinie ni d'informations préalables sur les communautés. LPA fonctionne en propageant des étiquettes à travers le réseau et en formant des communautés basées sur ce processus de propagation d'étiquettes. L'intuition derrière l'algorithme est qu'une seule étiquette peut rapidement devenir dominante dans un groupe de nœuds densément connectés, mais aura du mal à traverser une région peu connectée.

Les étiquettes seront piégées à l'intérieur d'un groupe de nœuds densément connectés, et les nœuds qui se retrouvent avec la même étiquette lorsque les algorithmes se terminent peuvent être considérés comme faisant partie de la même communauté.

L'algorithme fonctionne comme suit:

- Chaque nœud est initialisé avec une étiquette de communauté unique (un identifiant).
- Ces étiquettes se propagent à travers le réseau.
- A chaque itération de propagation, chaque nœud met à jour son étiquette, à celle à laquelle appartient le nombre maximum de ses voisins.
- LPA atteint la convergence lorsque chaque nœud a l'étiquette majoritaire de ses voisins.
- LPA s'arrête si la convergence ou le nombre maximal d'itérations défini par l'utilisateur est atteint.

[12] Neo4j Graph Data Science - Label Propagation
https://neo4j.com/docs/graph-data-science/2.3-preview/algorithms/label-propagation/

Au fur et à mesure que les étiquettes se propagent, des groupes de nœuds densément connectés parviennent rapidement à un consensus sur une étiquette unique. À la fin de la propagation, il ne restera que quelques étiquettes - la plupart auront disparu.

Dans un premier temps nous avons créé un graphe nommé :

```
CALL gds.graph.project(
    'Graph_2',
    ['Block', 'Transaction', 'Output', 'Address'],
    ['inc', 'out', 'locked']
)
```

En mode d'exécution **stream** :

```
CALL gds.labelPropagation.stream('Graph_2')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).hash AS Hash, Community
ORDER BY Community, Hash
```

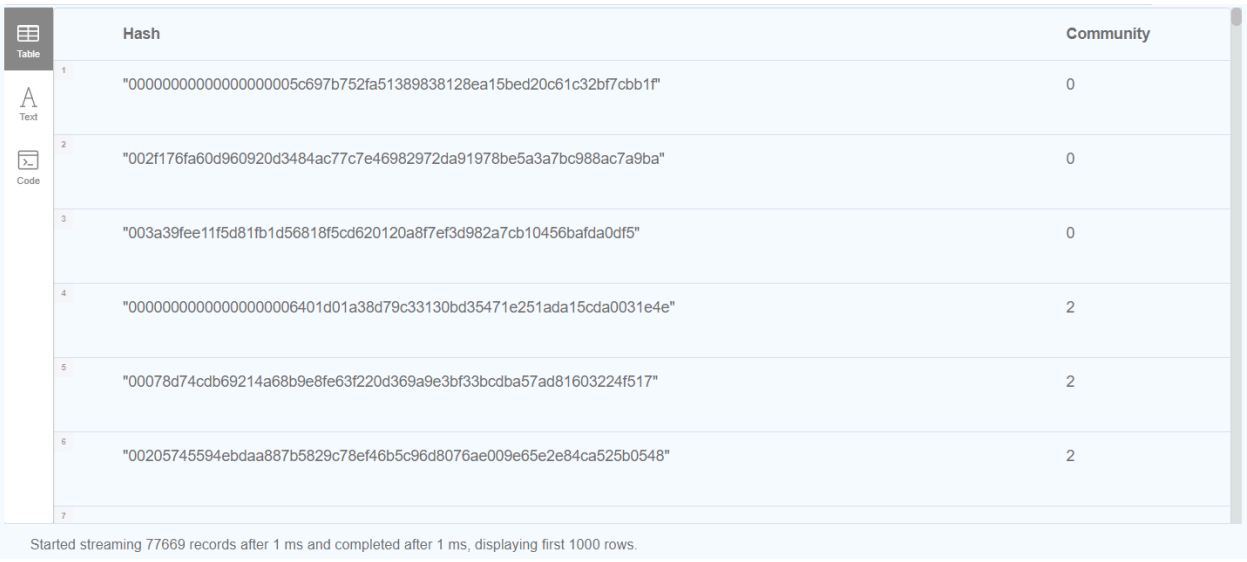


Table | Text | Code

	Hash	Community
1	"0000000000000000005c697b752fa51389838128ea15bed20c61c32bf7cbb1f"	0
2	"002f176fa60d960920d3484ac77c7e46982972da91978be5a3a7bc988ac7a9ba"	0
3	"003a39fee11f5d81fb1d56818f5cd620120a8f7ef3d982a7cb10456bafda0df5"	0
4	"00000000000000000006401d01a38d79c33130bd35471e251ada15cda0031e4e"	2
5	"00078d74cdb69214a68b9e8fe63f220d369a9e3bf33bcdba57ad81603224f517"	2
6	"00205745594ebdaa887b5829c78ef46b5c96d8076ae009e65e2e84ca525b0548"	2
7		

Started streaming 77669 records after 1 ms and completed after 1 ms, displaying first 1000 rows.

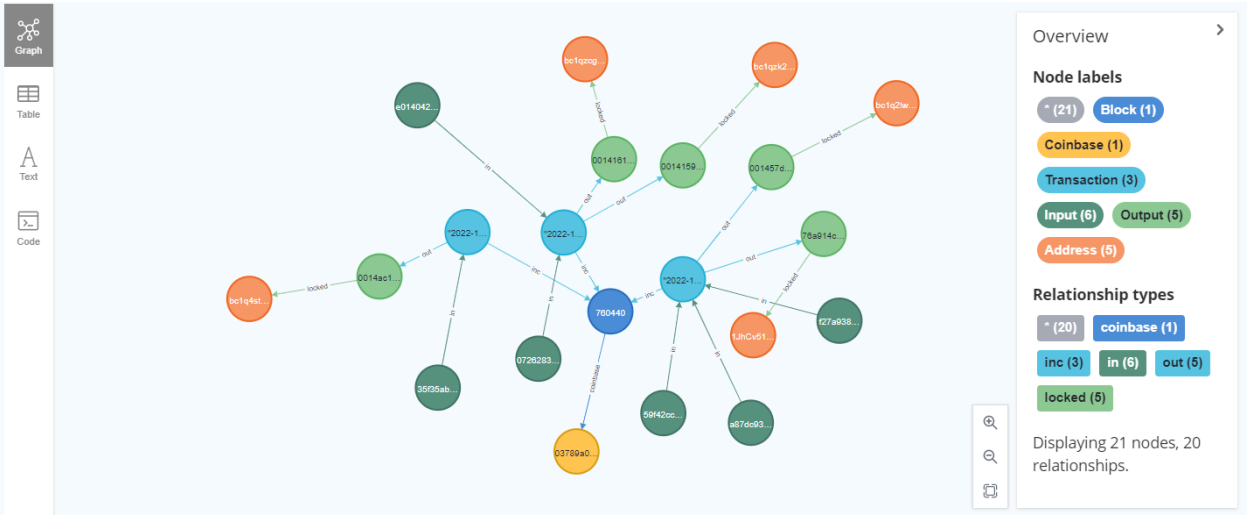
	Hash	Community
995	"0000000000000000004f02e0196a9cc3e005d86cf6a042f5873e437ddd71b9d"	534
996	"00029ba9d358c0d3744bb731be8a6bff17a01b2b3bd71ec8b733fe1527b4dcee"	534
997	"0006157425ed487686702e1dfb655403ec62ba7acbacb553bb60ead96e9bb75b"	534
998	"0000000000000000006e56f608594ca700e14920cfb348fbdd2f68f589315d8"	536
999	"00135af7f6496cb5f6e7b8a7259c33db0da1978bf29c5dfadc73ef567bfc12af"	536
1 000	"0000000000000000003a44c124ea9471869d13617c917be07eaf2e7a5a6a6f6"	538

Started streaming 77669 records after 1 ms and completed after 1 ms, displaying first 1000 rows.

Par exemple, si nous avons ce bloc :

```
// Community : 538
MATCH (n:Block{hash:'0000000000000000003a44c124ea9471869d13617c917be07eaf2e7a5a6a6f6'})
RETURN n
```

Nous pouvons remarquer que les transactions, les sorties, etc. appartenant à ce bloc ont de nombreux facteurs communs entre eux.



Et ceci est en contraste, par exemple, avec ce bloc :

```
// Community : 2
MATCH (n:Block{hash:'00000000000000000006401d01a38d79c33130bd35471e251ada15cda0031e4e'})
RETURN n
```

5. Intégration de Neo4j dans une application

Exemple d'intégration de notre base de données Neo4j dans une application au moyen du Neo4j Python Driver

Pour installer la toute dernière version stable,

```
python -m pip install neo4j
```

```
PS C:\Users\lenny\OneDrive\Documents\Master 2\Premiere_periode\Bases de donnees specialisees\Neo4j\bitcoin-blockchain-neo4j> python -m pip install neo4j
Collecting neo4j
  Downloading neo4j-5.3.0.tar.gz (157 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 157.8/157.8 kB 2.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing metadata (pyproject.toml) ... done
Collecting pytz
  Downloading pytz-2022.7-py2.py3-none-any.whl (499 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 499.4/499.4 kB 2.0 MB/s eta 0:00:00
Building wheels for collected packages: neo4j
  Building wheel for neo4j (pyproject.toml) ... done
  Created wheel for neo4j: filename=neo4j-5.3.0-py3-none-any.whl size=221872 sha256=b3015f50c96b30b063324f62b0590527f0e4bf50de3615c3a3feeed18409f87b
  Stored in directory: c:\users\lenny\appdata\local\pip\cache\wheels\8a\c2\63\120f949d48b7a9577c5b5bcaae8d32d698794be94206d41c3d
Successfully built neo4j
Installing collected packages: pytz, neo4j
Successfully installed neo4j-5.3.0 pytz-2022.7
```

Exemple d'utilisation :

```
from neo4j import GraphDatabase, ManagedTransaction

class DonneesHistoriquesBlockchainBitcoin(object) :

    def __init__(self, uri : str, utilisateur : str, mdp : str):

        self.driver = GraphDatabase.driver(uri, auth=(utilisateur, mdp))

    def close(self):

        self.driver.close()

    def print_end_addresses(self, hash : str):

        def get_end_addresses(managed_transaction : ManagedTransaction, hash : str) -> list :

            addresses = []

            result = managed_transaction.run("MATCH p=(start:Address)-[*]-(end:Address) "
```

```
                                        "WHERE start.address = $hash "
                                        "RETURN end.address AS address ", hash=hash)
            for record in result:
                addresses.append(record['address'])
            return addresses
        with self.driver.session() as session:
            addresses_list = session.execute_read(get_end_addresses, hash)
        for address in addresses_list:
            print(address)
if __name__ == "__main__":
    uri = "bolt://localhost:7687"
    utilisateur = "neo4j"
    mdp = "..."
    donneesHistoriquesBlockchainBitcoin = DonneesHistoriquesBlockchainBitcoin(uri, utilisateur, mdp)
    donneesHistoriquesBlockchainBitcoin.print_end_addresses("bc1qhekuaxsq7ps2w7fzw9qwun2x2yacsd6ev4csf6")
    donneesHistoriquesBlockchainBitcoin.close()
```

```
PS C:\Users\lenny\OneDrive\Documents\Master 2\Premiere_periode\Bases de donnees specialisees\Ne
o4j\bitcoin-blockchain-neo4j> python app.py
bc1q3xdq99scpz7ep3xkexu6tzv4r3gnq883tlfrqm
3E2yK9BUfwczaCXGb9Keo4Sdwy5J7bgM6e
bc1qz94wddt2d2qek7uhxr2e7cwhme24hutr4hljvh
3NmuUsUNAbnuGoSTR47sJ3YLD1q7tc7YxC
bc1qkzn3wwfqg8kdf3lexezep2a3wchrzmn3hg037k85cxqnfjy656kseuhawj
19dpcczaetaGWSgqv5PrWFSmQF1uFUyBmn
36UE6S1VswwS1a7VQf4cyQ3y6gh6c5BLuB
bc1qg5exgc76kf8495763wzkgph7mm2sv6v2t50p0x
bc1qc80ps88xpxckkn096urphcgqmfxe03glzavuv6
```