UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EM GESTÃO E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Lucas Victório Paiola: 792194
Luiza Gandolfi Barioto: 793247

Fase Final
Aplicação de um Sistema de Recomendação em Streaming de Músicas
Grupo 7

Processamento Massivo de Dados

Profª. Drª. Sahudy Montenegro González

Sorocaba
2023

Índice

1. Objetivos

O objetivo principal desta atividade é desenvolver uma aplicação utilizando tecnologias específicas para criar um sistema de recomendação de músicas em um ambiente de streaming. A aplicação deverá realizar o processo de extração, transformação e carregamento (ETL) dos dados utilizando PySpark, uma ferramenta de processamento distribuído de grandes volumes de dados. Após o processamento, esses dados serão armazenados e gerenciados em um banco de dados orientado a grafos, no caso, Neo4j, o que permitirá uma modelagem das relações entre usuários, músicas, artistas e álbuns.

No que diz respeito ao processamento de dados, PySpark será utilizado para realizar a extração de dados brutos de arquivos CSV que contém músicas, álbuns e outros dados relacionados e após a extração, os dados passarão por um processo de transformação, onde serão limpos, enriquecidos e agregados, preparando-os para análises futuras. Em seguida, esses dados transformados serão carregados e armazenados no banco de dados Neo4j, mantendo a integridade das relações e da estrutura para consultas futuras.

O Neo4j será utilizado para a persistência dos dados, permitindo a modelagem de grafos que representem usuários, músicas, artistas, gêneros e playlists, capturando as interações e preferências dos usuários. Além disso, serão implementadas consultas utilizando a linguagem Cypher, própria do Neo4j, para recuperar informações e realizar o sistema de recomendação.

O sistema de recomendação de músicas, parte central desta aplicação, permitirá que o usuário forneça algumas músicas de sua preferência. Com base nessas entradas, o sistema utilizará o Neo4j para explorar a rede de conexões entre as músicas, artistas e outros usuários, identificando músicas similares com base em critérios como popularidade, gênero e comportamento de outros ouvintes. O resultado será uma lista de recomendações personalizadas e relevantes para o usuário, baseada nas músicas fornecidas como ponto de partida.

Espera-se que esta atividade resulte no desenvolvimento de uma aplicação funcional que demonstre a integração eficaz de PySpark e Neo4j, capaz de processar e armazenar dados de forma eficiente.

2. Tecnologias que serão utilizadas

A tecnologia referente ao banco de dados será o Neo4j, uma vez que, sendo um banco de dados orientado a grafos, ele é ideal para armazenar e gerir relacionamentos entre os dados, como por exemplo entre álbuns de mesmo gênero, artistas parecidos e músicas relacionadas. Inclusive as consultas no Neo4j, realizadas em Cypher, são otimizadas para navegar entre esses relacionamentos através dos dados, sendo assim realizadas de maneira muito mais eficiente do que em bancos de dados relacionais tradicionais.

Isso acontece porque o Neo4j utiliza um modelo de dados que representa as entidades como nós e as relações entre eles como arestas. Tanto nós quanto arestas podem ter suas próprias propriedades (normalmente no formato de chave-valor) e diferentemente dos banco de dados relacionais em que as relações entre entidades são modeladas por chaves estrangeiras e junções, no Neo4j, as relações são armazenadas diretamente como parte dos dados, ao serem armazenadas fisicamente ao lado dos nós conectados. Dessa forma, quando o banco de dados precisa acessar um certo nó junto de suas relações, ele não precisa realizar uma busca em outra tabela e realizar junções. Além disso, o Neo4j pode criar e manter índices tanto para nós quanto para arestas, permitindo que a busca inicial em conjuntos de dados massivos seja realizada mais rapidamente.

Já em relação ao Cypher, ele é projetado para trabalhar com grafos uma vez que permite a descrição de consultas de forma declarativa, o que indica padrões de nós e arestas que queremos buscar. A partir do momento que as relações são armazenadas nativamente, o Cypher pode otimizar a execução das consultas para navegar de forma eficiente entre os nós conectados.

A tecnologia para o processamento desses dados será o Apache Spark pois ele se destaca por permitir o processamento dos dados em larga escala. Com sua arquitetura distribuída, PySpark pode lidar com uma grande quantidade de dados em tempo real ou quase real, processando-os em clusters de servidores. Isso facilita a execução de tarefas como a extração, transformação e carga (ETL), essenciais para preparar os dados antes de serem armazenados ou analisados.

Com APIs em Python disponíveis para ambas as ferramentas (como por exemplo, Neo4j Connector for Apache Spark), a integração entre PySpark e Neo4j é simplificada, permitindo um fluxo de dados integrado entre os sistemas.

3. Fontes de Dados

114000 Spotify Songs, disponível no link. É um dataset público na plataforma Kaggle, que possui registros de músicas da plataforma Spotify com os seguintes dados:

- **track_id**: O ID único do Spotify para cada faixa.
- **artists**: Nomes dos artistas que performaram a faixa, separados por ';'.
- **album_name**: O nome do álbum no qual a faixa aparece.
- **track_name**: O título da faixa.
- **popularity**: Um valor entre 0 e 100, indicando a popularidade da faixa com base em reproduções recentes.
- **duration_ms**: A duração da faixa em milissegundos.
- **explicit**: Booleano indicando se a faixa contém conteúdo explícito.
- **danceability**: Descreve quão adequada é a faixa para dançar (0.0 = menos dançável, 1.0 = mais dançável).
- **energy**: Representa a intensidade e atividade de uma faixa (0.0 = baixa energia, 1.0 = alta energia).
- **key**: A tonalidade musical da faixa mapeada usando a notação padrão de Classe de Tom.
- **loudness**: Volume geral da faixa em decibeis (dB).
- **mode**: Indica a modalidade (maior ou menor) da faixa.
- **speechiness**: Detecta a presença de palavras faladas na faixa.
- **acousticness**: Medida de confiança de se a faixa é acústica (0.0 = não acústica, 1.0 = altamente acústica).
- **instrumentalness**: Prediz se uma faixa contém vocais (0.0 = contém vocais, 1.0 = instrumental).
- **liveness**: Detecta a presença de uma audiência na gravação (0.0 = gravação de estúdio, 1.0 = performance ao vivo).
- **valence**: Mede a positividade musical transmitida por uma faixa (0.0 = negativa, 1.0 = positiva).
- **tempo**: Tempo estimado da faixa em batidas por minuto (BPM).
- **time_signature**: Assinatura de tempo estimada da faixa (3 a 7).

4. Consultas a serem respondidas

Consulta 1: Dadas as músicas: ‘21 guns’ - Bailey Jehl, ‘I will wait’ - The Mayries e ‘Say something’ - A Great Big World, me recomende 10 novas músicas relacionadas.

Explicação da Consulta 1: Essa consulta deverá retornar o nome das 10 músicas recomendadas, levando em consideração as características: *‘danceability’*, *‘energy’*, *‘valence’* e *‘popularity’* em comum das músicas: ‘21 guns’ - Bailey Jehl, ‘I will wait’ - The Mayries e ‘Say something’ - A Great Big World. As 10 músicas recomendadas devem possuir características semelhantes também às músicas mencionadas.

Consulta 2: Dado o álbum ‘When the morning comes’, me recomende outros 3 álbuns com características relacionadas.

Explicação da Consulta 2: Essa consulta deverá retornar o nome de 3 álbuns que tenham características relacionadas ao álbum ‘When the morning comes’. Para isso, será considerado a média de dançabilidade e energia das músicas contidas nos álbuns, e então, comparar quais são os 3 álbuns que possuem a média mais próxima do álbum ‘When the morning comes’.

Consulta 3: Entre os artistas Angelina Cruz, Jason Mraz e Andrew Foy, qual deles possui as músicas mais dançantes e populares?

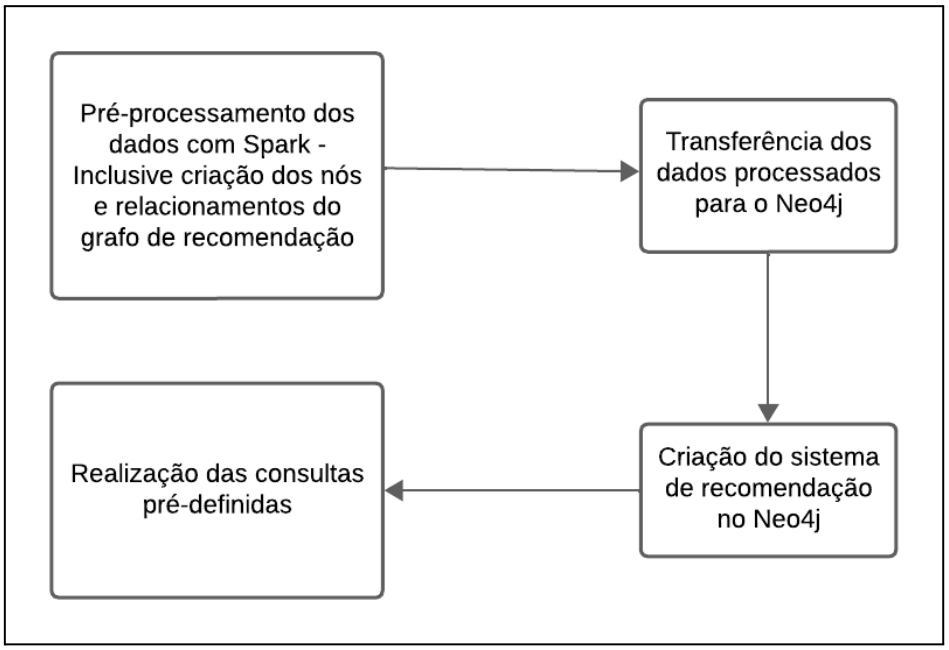
Explicação da Consulta 3: Essa consulta deverá retornar o nome do artista que possui as músicas mais dançantes e populares, de acordo com os parâmetros *‘danceability’* e *‘popularity’*. Para isso, deverá ser feita a média das músicas de cada um dos artistas mencionados, e em seguida, comparar as médias para analisar qual deles possui em sua totalidade as músicas mais populares e dançantes.

Consulta 4: Quais músicas do artista Eddie Vedder são as mais positivas e energéticas?

Explicação da Consulta 4: Essa consulta deverá retornar uma ordenação das músicas mais positivas e enérgicas do artista Eddie Vedder, levando em consideração os campos *‘valence’* e *‘energy’* de suas músicas.

5. Fluxograma

O fluxograma do desenvolvimento da aplicação pode ser visto abaixo e no link.



O processo começa com o carregamento e pré-processamento dos dados no Databricks, utilizando Spark. Nessa fase inicial, os dados são importados e processados, abrangendo um grande volume de informações sobre faixas musicais, artistas, álbuns e suas respectivas características. O pré-processamento envolve a limpeza das colunas, eliminando caracteres especiais, ajustando capitalizações e normalizando valores. A coluna 'explicit', por exemplo, é convertida para valores binários, e o 'track_id' é substituído por um identificador serial. Com os dados limpos e preparados, são criados nós que representam as entidades principais no grafo de recomendação, como Faixa, Artista e Álbum. Esses nós são estruturados para refletir o relacionamento entre as entidades, **onde um artista grava uma faixa, uma faixa pertence a um álbum, e um álbum é feito por um artista**.

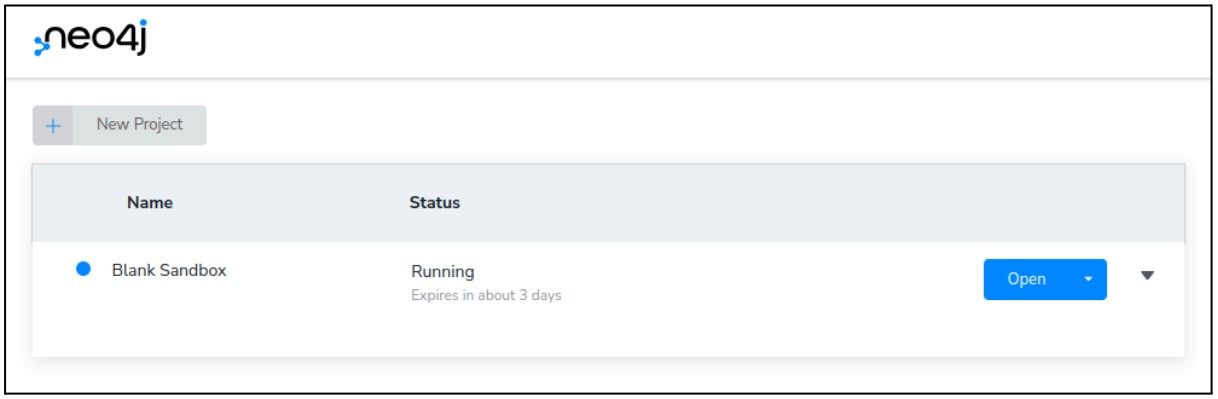
Após o pré-processamento e a criação dos nós e relacionamentos no Spark, os dados são transferidos para o Neo4j automaticamente respeitando a estrutura criada no Databricks. Com a estrutura do grafo em funcionamento no Neo4j, o próximo passo é a criação do sistema de recomendação em si.

No Neo4j, o sistema de recomendação é configurado utilizando algoritmos específicos que geram sugestões de músicas personalizadas para os usuários. Para isso, são considerados parâmetros como Popularity, Danceability, Valence e Energy, que ajudam a identificar e sugerir faixas e álbuns de acordo com os gostos e preferências dos usuários. Com o sistema configurado, são realizadas consultas pré-definidas para validar e testar a eficácia do sistema de recomendação, permitindo a criação de playlists personalizadas e a descoberta de novas músicas de forma eficiente e alinhada com as preferências dos usuários.

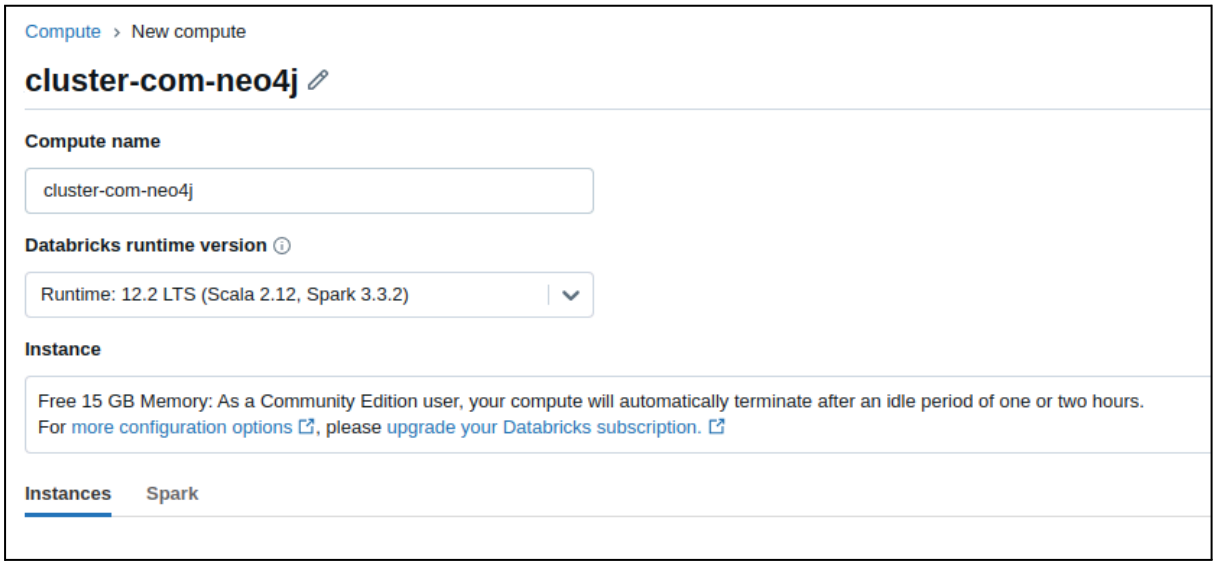
6. Explicação da conexão entre Databricks e Neo4j

A principal fonte para a descoberta de como fazer essa conexão foi este link. Abaixo seguimos os passos com as pequenas alterações que contemplam atualizações de versão do Spark, entre outros.

1. Criar uma nova instância em branco no Neo4j, bem como conectar com os dados baixados em um .txt (as credenciais deste arquivo .txt serão utilizadas também na instância do Databricks).

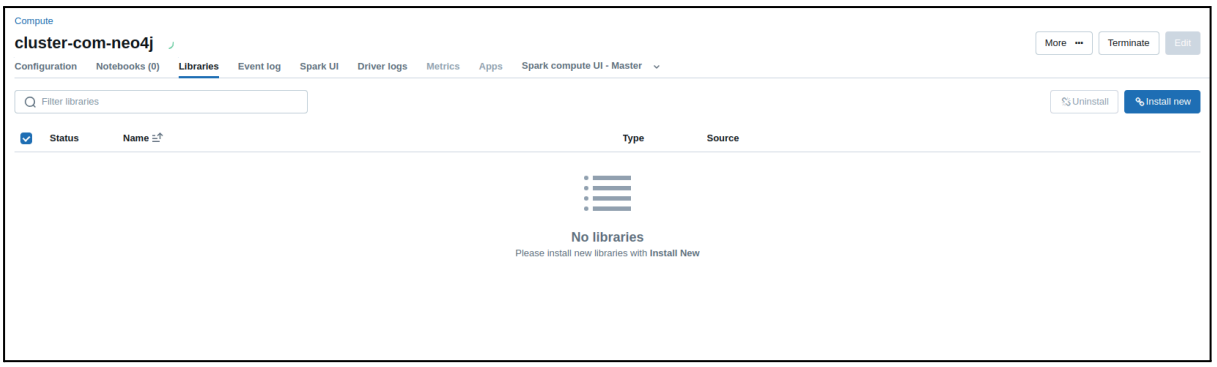


2. Criar um novo cluster no Databricks, porém com algumas modificações:

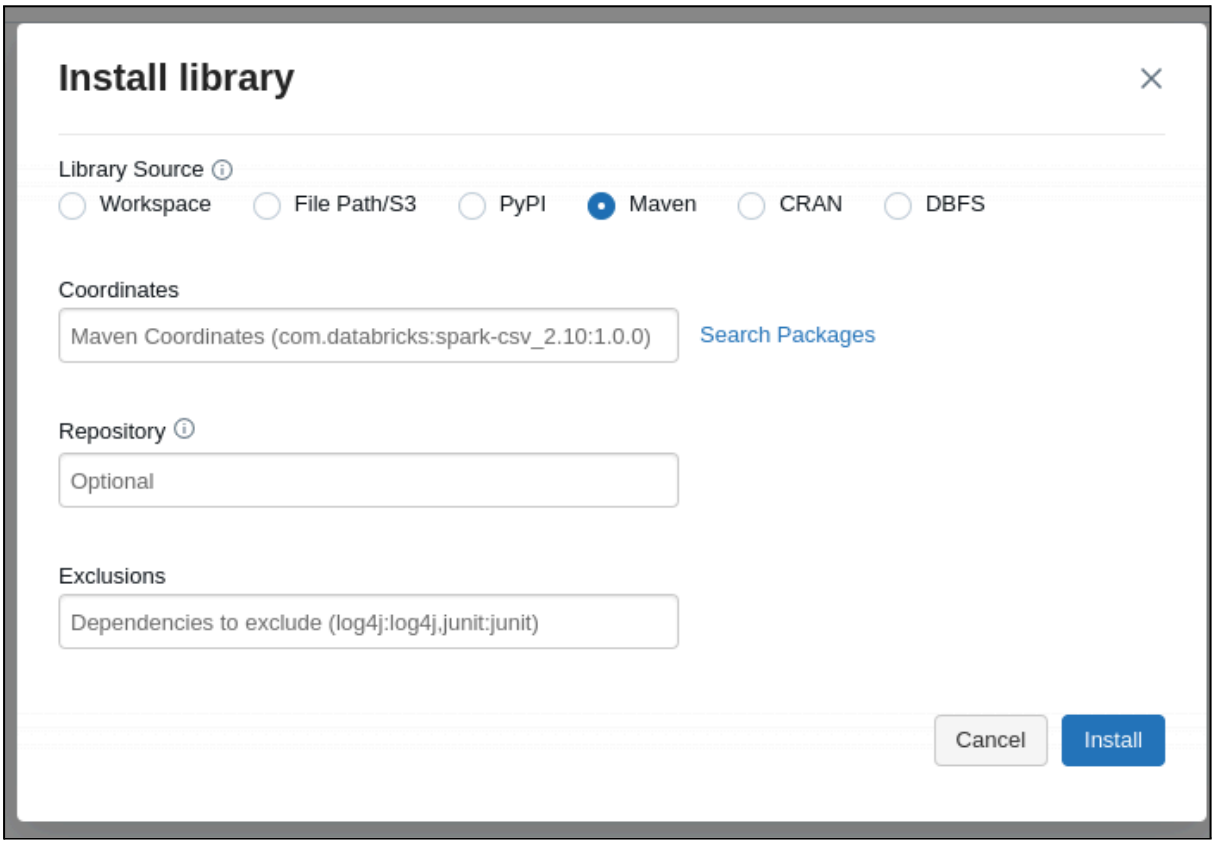


Essas foram as especificações do Spark utilizadas nesse projeto, que inclusive já estão mais adiante do que a utilizada no link de tutorial, porém conforme novas versões são lançadas, pode haver mudança e necessidade de checagem se a versão do Spark se conecta com a biblioteca usada para se conectar ao Neo4j.

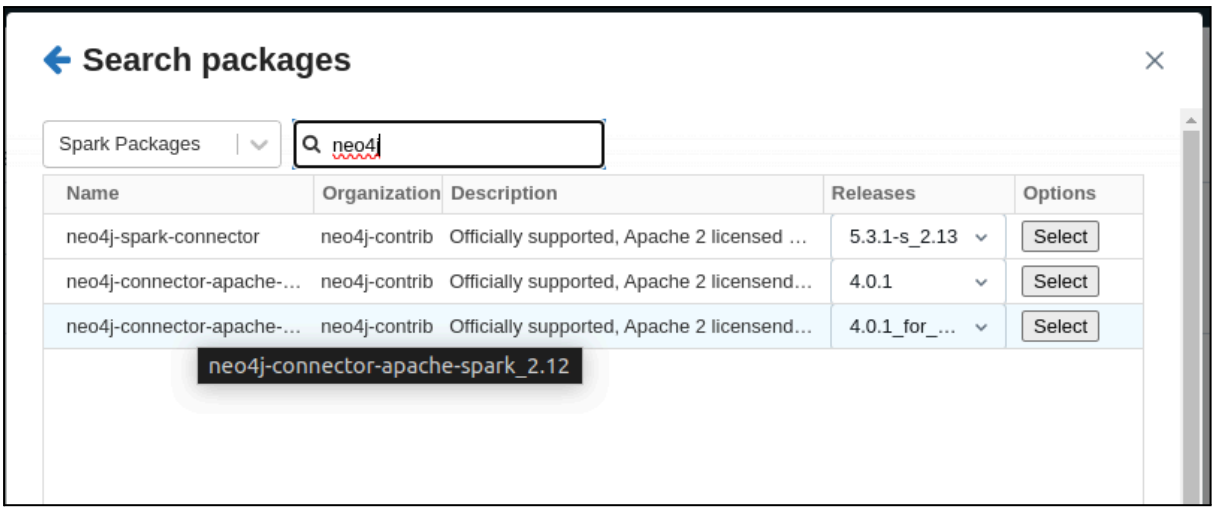
3. Depois de criado o cluster no Databricks, clicar em “Libraries” e “Install new”:



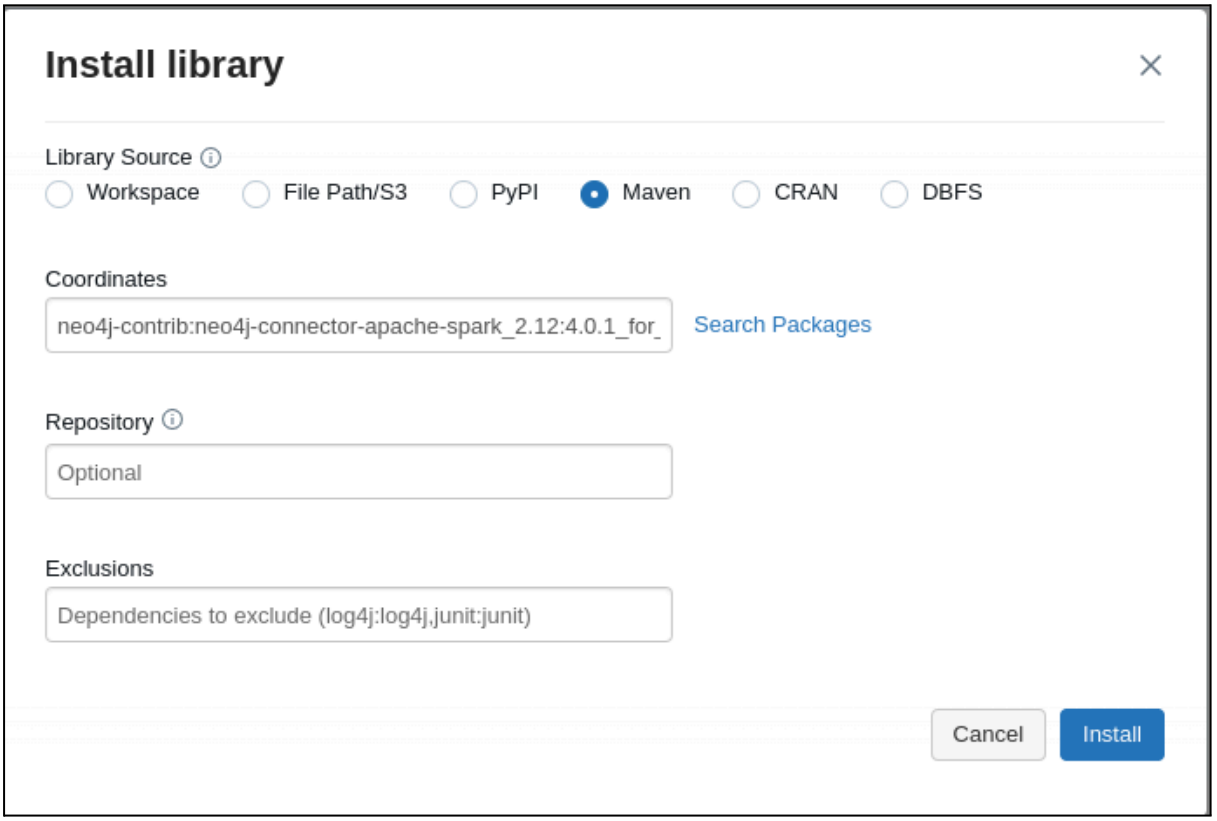
4. Clicar em “Maven” e “Search Packages”:



5. Em “Search Packages”, procurar por “neo4j”. Aparecerão algumas bibliotecas para instalação no cluster. A que se mostrou funcional é “neo4j-connector-apache-spark_2.12”.



6. Selecionar a biblioteca e então clicar em “Install”:



7. No Databricks, fazer a conexão com o Neo4j, de acordo com as informações da instância (estão no .txt baixado sobre a instância do Neo4j - bolt e password):

```
23 hours ago (15s)

#Informações necessárias - mudar de acordo com as informações da instância do neo4j
bolt = "bolt://54.204.68.245"
password = "participations-test-ounces"

df = spark.read.format("org.neo4j.spark.DataSource")\
  .option("url", bolt)\
  .option("authentication.type", "basic")\
  .option("authentication.basic.username", "neo4j")\
  .option("authentication.basic.password", password)\
  .option("labels", "Person")\
  .load()
display(df)
```

Com isso, já é suficiente ler dados do Neo4j para dentro de um dataframe do Databricks. Se quiser enviar dados para o Neo4j (como é o caso deste projeto), fazer o seguinte:

```
24 hours ago (15s)

#Informações necessárias - mudar de acordo com as informações da instância do neo4j
bolt = "bolt://54.204.68.245"
password = "participations-test-ounces"

# Relacionamentos
# Faixa - Artista (artista grava faixa)
df_faixas_artistas = df_musicas.select(
    F.col("track_name").alias("target.track"),
    F.col("artist").alias("source.artist")
).distinct()

(
    df_faixas_artistas.write
    .mode("Append")
    .format("org.neo4j.spark.DataSource")
    .option("url", bolt)
    .option("authentication.type", "basic")
    .option("authentication.basic.username", "neo4j")
    .option("authentication.basic.password", password)
    .option("relationship", "GRAVOU")
    .option("relationship.source.save.mode", "Append")
    .option("relationship.source.labels", ":Artist")
    .option("relationship.target.save.mode", "Append")
    .option("relationship.target.labels", ":Track")
    .save()
)
```

Conectar com a instância do Neo4j, definir o dataframe que será enviado, com as informações necessárias para criação dos nós e relacionamentos (os nomes das colunas precisam ser **source.<nó que será a fonte>** e **target.<nó que será o alvo>** (vide exemplo acima). Após isso, apenas definir o relacionamento entre os nós e outras configurações como visto acima. Esta parte foi criada a partir deste link presente na documentação do Neo4j.

7. Execução e explicação das consultas - Neo4j

Observação: as consultas foram realizadas levando em consideração um subset de 5000 músicas, uma vez que a conexão entre Neo4j e Databricks se mostra relativamente lenta para conjuntos muito grandes de dados. Como o foco do projeto é a conexão entre as tecnologias e não tanto a utilização de datasets imensos, demos mais importância para o sistema de recomendação a partir de dados do Databricks ao invés da utilização de muitos dados.

Consulta 1: Dadas as músicas: ‘21 guns’ - Bailey Jehl, ‘I will wait’ - The Mayries e ‘Say something’ - A Great Big World, me recomende 10 novas músicas relacionadas.

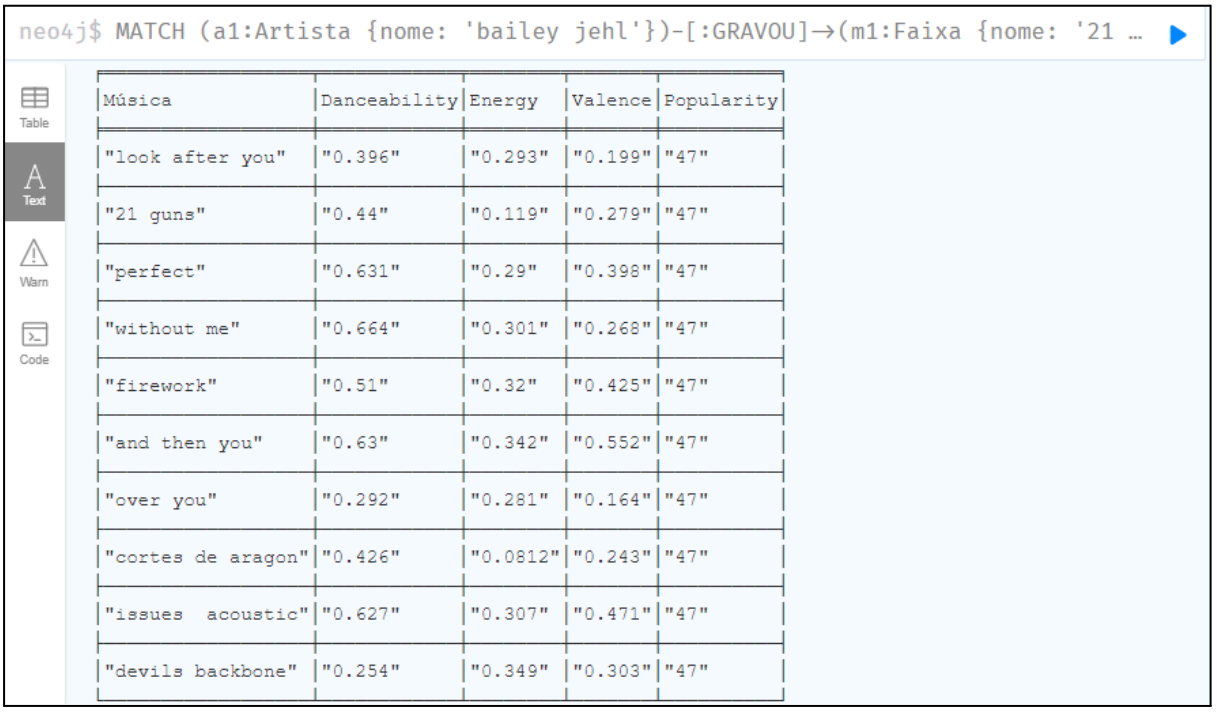
Explicação da Consulta 1: Essa consulta deverá retornar o nome das 10 músicas recomendadas, levando em consideração as características: *‘danceability’*, *‘energy’*, *‘valence’* e *‘popularity’* em comum das músicas: ‘21 guns’ - Bailey Jehl, ‘I will wait’ - The Mayries e ‘Say something’ - A Great Big World. As 10 músicas recomendadas devem possuir características semelhantes também às músicas mencionadas.

Consulta em Cypher:

```
MATCH (a1:Artista {nome: 'bailey jehl'})-[:GRAVOU]->(m1:Faixa {nome:
'21 guns'}),

      (a2:Artista {nome: 'the mayries'})-[:GRAVOU]->(m2:Faixa {nome:
'i will wait'}),

       (a3:Artista {nome: 'a great big world'})-[:GRAVOU]->(m3:Faixa
{nome: 'say something'})

WITH avg(toFloat(m1.danceability)) AS mediaDanceability,

     avg(toFloat(m1.energy)) AS mediaEnergy,

     avg(toFloat(m1.valence)) AS mediaValence,

     avg(toFloat(m1.popularity)) AS mediaPopularity

MATCH (r:Faixa)

WHERE abs(toFloat(r.danceability) - mediaDanceability) < 0.3

  AND abs(toFloat(r.energy) - mediaEnergy) < 0.3

  AND abs(toFloat(r.valence) - mediaValence) < 0.3

  AND abs(toFloat(r.popularity) - mediaPopularity) < 0.3

RETURN r.nome AS Música,

       r.danceability AS Danceability,
```

```
        r.energy AS Energy,
        r.valence AS Valence,
        r.popularity AS Popularity
LIMIT 10;
```

Resultado Obtido:



neo4j$ MATCH (a1:Artista {nome: 'bailey jehl'})-[:GRAVOU]→(m1:Faixa {nome: '21 …

Table | Text | Warn | Code

Música	Danceability	Energy	Valence	Popularity
"look after you"	"0.396"	"0.293"	"0.199"	"47"
"21 guns"	"0.44"	"0.119"	"0.279"	"47"
"perfect"	"0.631"	"0.29"	"0.398"	"47"
"without me"	"0.664"	"0.301"	"0.268"	"47"
"firework"	"0.51"	"0.32"	"0.425"	"47"
"and then you"	"0.63"	"0.342"	"0.552"	"47"
"over you"	"0.292"	"0.281"	"0.164"	"47"
"cortes de aragon"	"0.426"	"0.0812"	"0.243"	"47"
"issues acoustic"	"0.627"	"0.307"	"0.471"	"47"
"devils backbone"	"0.254"	"0.349"	"0.303"	"47"

Explicação consulta em Cypher:

A consulta localiza três faixas específicas e calcula a média das propriedades '*danceability'*, *'energy'*, *'valence'* e *'popularity'* dessas faixas. Com base nessas médias, a consulta então procura outras faixas no banco de dados que tenham valores semelhantes para essas mesmas propriedades, com uma margem de diferença de até 0.3. O resultado é uma lista de até 10 faixas que compartilham características similares às faixas originais, retornando o nome das faixas e seus valores para '*danceability'*, *'energy'*, *'valence'* e *'popularity'*.

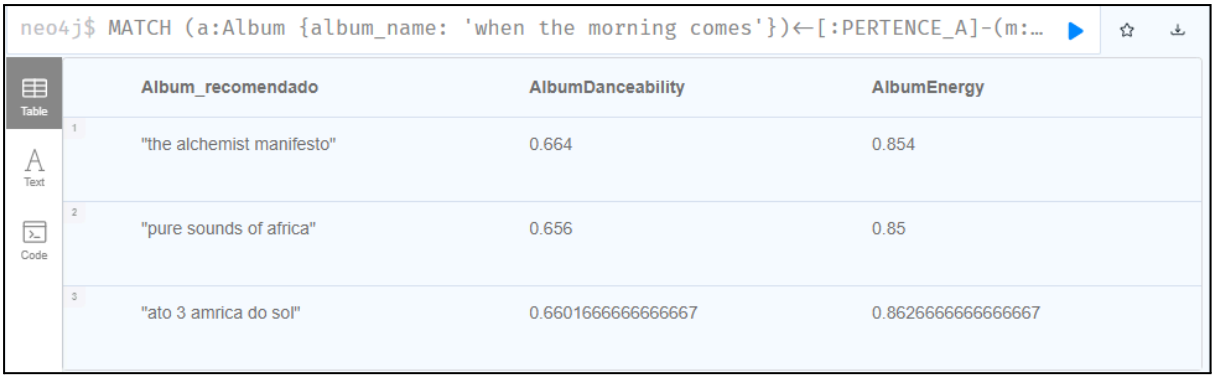
Consulta 2: Dado o álbum 'When the morning comes', me recomende outros 3 álbuns com características relacionadas.

Explicação da Consulta 2: Essa consulta deverá retornar o nome de 3 álbuns que tenham características relacionadas ao álbum 'When the morning comes'. Para isso, será considerado a média de dançabilidade e energia das músicas contidas nos álbuns, e então, comparar quais são os 3 álbuns que possuem a média mais próxima do álbum 'When the morning comes'.

Consulta em Cypher:

```
MATCH (a:Album {album_name: 'when the morning comes'})<-[:PERTENCE_A]-(m:Faixa)

WITH avg(toFloat(m.danceability)) AS mediaDanceability,

     avg(toFloat(m.energy)) AS mediaEnergy

MATCH (a2:Album)<-[:PERTENCE_A]-(m2:Faixa)

WHERE a2.album_name <> 'when the morning comes'

WITH a2,

     mediaDanceability, mediaEnergy,

     avg(toFloat(m2.danceability)) AS albumMediaDanceability,

     avg(toFloat(m2.energy)) AS albumMediaEnergy

WITH a2.album_name AS AlbumName,

     albumMediaDanceability,

     albumMediaEnergy,

          abs(albumMediaDanceability - mediaDanceability) AS DiferencaDanceability,

     abs(albumMediaEnergy - mediaEnergy) AS DiferencaEnergy,

          (abs(albumMediaDanceability - mediaDanceability) + abs(albumMediaEnergy - mediaEnergy)) AS similaridade

ORDER BY similaridade ASC

LIMIT 3

RETURN AlbumName AS Album_recomendado,

       albumMediaDanceability AS AlbumDanceability,

       albumMediaEnergy AS AlbumEnergy
```

Resultado Obtido:



	Album_recomendado	AlbumDanceability	AlbumEnergy
1	"the alchemist manifesto"	0.664	0.854
2	"pure sounds of africa"	0.656	0.85
3	"ato 3 amrica do sol"	0.6601666666666667	0.8626666666666667

Obs: ‘Danceability’ e ‘Energy’ do álbum: ‘When the morning comes’:

	Album	AverageDanceability	AverageEnergy
1	"when the morning comes"	0.6595	0.8545

Explicação consulta em Cypher:

A consulta identifica o álbum "when the morning comes" e calcula as médias das propriedades danceability e energy de suas faixas. Em seguida, ela busca todos os outros álbuns no banco de dados e calcula as médias dessas mesmas propriedades para as faixas de cada álbum. Para cada álbum, a consulta calcula a diferença absoluta entre as médias de danceability e energy desse álbum e as médias do álbum "when the morning comes". A soma dessas diferenças é usada como uma medida de similaridade, e os três álbuns mais similares são selecionados. Por fim, a consulta retorna os nomes desses álbuns recomendados, juntamente com suas médias de danceability e energy.

Consulta 3: Entre os artistas Angelina Cruz, Jason Mraz e Andrew Foy, qual deles possui as músicas mais dançantes e populares?

Explicação da Consulta 3: Essa consulta deverá retornar o nome do artista que possui as músicas mais dançantes e populares, de acordo com os parâmetros *‘danceability’* e *‘popularity’*. Para isso, deverá ser feita a média das músicas de cada um dos artistas mencionados, e em seguida, comparar as médias para analisar qual deles possui em sua totalidade as músicas mais populares e dançantes.

Consulta em Cypher:

```
MATCH (a:Artista)-[:GRAVOU]->(m:Faixa)

WHERE a.nome IN ['angelina cruz', 'jason mraz', 'andrew foy']

WITH a.nome AS ArtistName,

     avg(toFloat(m.danceability)) AS AvgDanceability,
```

```
     avg(toFloat(m.popularity)) AS AvgPopularity
RETURN ArtistName AS Artist,
       AvgDanceability AS MediaDanceabilty,
       AvgPopularity AS MediaPopularity
ORDER BY AvgDanceability DESC, AvgPopularity DESC
```

Resultado Obtido:

```
2  MATCH (a:Artista)-[:GRAVOU]→(m:Faixa)
3  WHERE a.nome IN ['angelina cruz', 'jason mraz', 'andrew foy']
4  WITH a.nome AS ArtistName,
5       avg(toFloat(m.danceability)) AS AvgDanceability,
6       avg(toFloat(m.popularity)) AS AvgPopularity
7  RETURN ArtistName AS Artist,
8         AvgDanceability AS MediaDanceabilty,
9         AvgPopularity AS MediaPopularity
10 ORDER BY AvgDanceability DESC, AvgPopularity DESC
11
```

Table | Text | Code

	Artist	MediaDanceabilty	MediaPopularity
1	"andrew foy"	0.6407692307692308	39.846153846153854
2	"jason mraz"	0.6018787878787879	35.787878787878796
3	"angelina cruz"	0.59	38.0

Explicação consulta em Cypher:

A consulta busca as faixas gravadas pelos artistas "angelina cruz", "jason mraz" e "andrew foy" no banco de dados e calcula a média das propriedades danceability e popularity para as faixas de cada artista. Os resultados são ordenados primeiro pela média de danceability em ordem decrescente e, em seguida, pela média de popularity, também em ordem decrescente. A consulta retorna o nome de cada artista, juntamente com as médias de danceability e popularity de suas faixas.

Consulta 4: Quais músicas do artista Eddie Vedder são as mais positivas e energéticas?

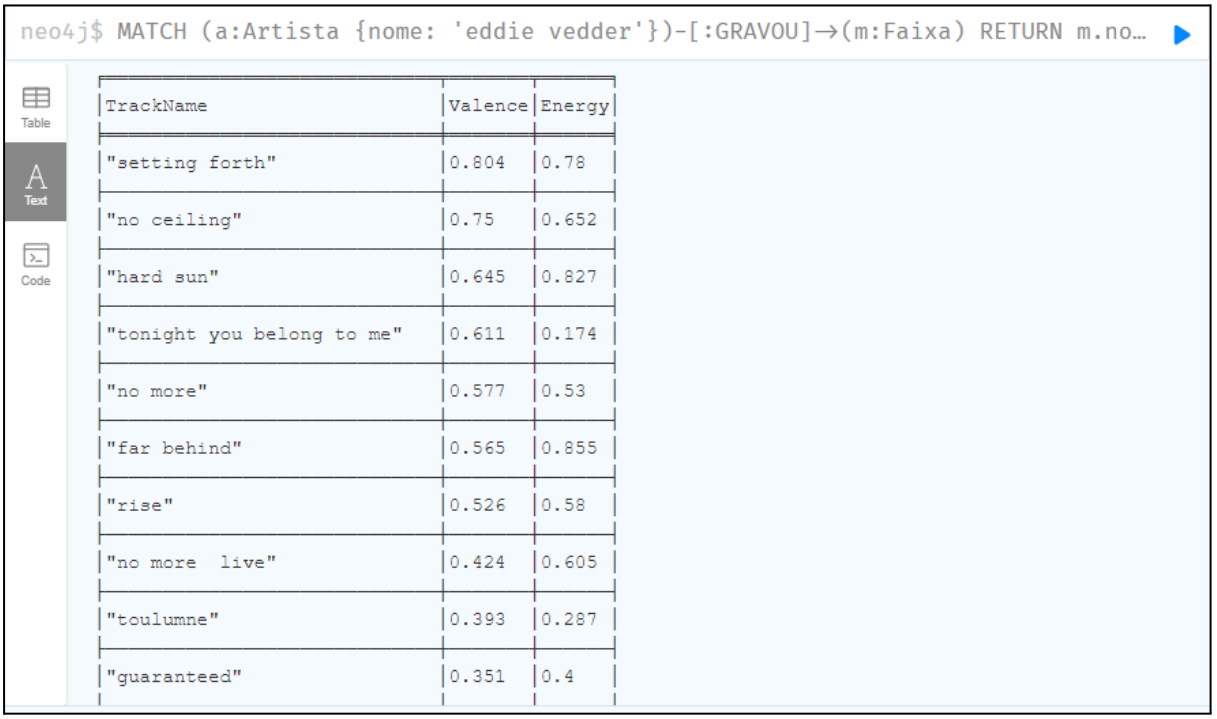
Explicação da Consulta 4: Essa consulta deverá retornar uma ordenação das músicas mais positivas e enérgicas do artista Eddie Vedder, levando em consideração os campos *'valence'* e *'energy'* de suas músicas.

Consulta em Cypher:

```
MATCH (a:Artista {nome: 'eddie vedder'})-[:GRAVOU]->(m:Faixa)
```

```
RETURN m.nome AS TrackName,
       toFloat(m.valence) AS Valence,
       toFloat(m.energy) AS Energy
ORDER BY Valence DESC, Energy DESC
```

Resultado Obtido:



TrackName	Valence	Energy
"setting forth"	0.804	0.78
"no ceiling"	0.75	0.652
"hard sun"	0.645	0.827
"tonight you belong to me"	0.611	0.174
"no more"	0.577	0.53
"far behind"	0.565	0.855
"rise"	0.526	0.58
"no more live"	0.424	0.605
"toulumne"	0.393	0.287
"guaranteed"	0.351	0.4

Explicação consulta em Cypher:

A consulta recupera todas as faixas gravadas pelo artista "eddie vedder" no banco de dados e retorna o nome de cada faixa, juntamente com os valores das propriedades valence e energy convertidos para o tipo float. As faixas são ordenadas em ordem decrescente, primeiro pelo valor de valence e, em seguida, pelo valor de energy.

8. Conclusão e Dificuldades Encontradas

Durante o desenvolvimento deste projeto, algumas dificuldades foram enfrentadas, conforme descrito a seguir:

Conexão entre Databricks e Neo4j: Estabelecer a conexão entre o Databricks e o Neo4j foi uma tarefa que demandou certo esforço, especialmente devido às diferenças nas versões das bibliotecas e do Spark. O conector do spark com o neo4j precisava de uma compatibilidade de versões que demoramos para encontrar.

Criação dos Relacionamentos Entre os Nós no Neo4j: Após estabelecida a conexão entre o Databricks e o Neo4j, tivemos uma certa dificuldade em conseguir criar os relacionamentos entre Album, Artista e Faixa. Os nós estavam sendo criados porém o relacionamentos entre eles não. Nesse sentido, a documentação fornecida no site do Neo4j foi essencial para que conseguíssemos criar corretamente os relacionamentos.

Desempenho da Conexão: Durante as etapas de leitura e escrita de dados entre o Databricks e o Neo4j, observamos que a conexão era relativamente lenta, especialmente ao lidar com grandes volumes de dados. Essa lentidão impactou a execução das consultas e a transferência de dados, forçando-nos a limitar o conjunto de dados utilizado para testes a um subset de 5000 músicas. Esse fator limitou a capacidade de testar a aplicação com volumes maiores de dados, que seriam mais representativos de um cenário real de streaming de música.

Apesar dessas dificuldades, as soluções encontradas permitiram a continuidade do projeto e o desenvolvimento de um sistema de recomendação funcional, ainda que com algumas limitações em termos de escalabilidade e performance.

Bibliografia

https://neo4j.com/docs/cypher/

https://neo4j.com/docs/import/

https://neo4j.com/docs/cypher-manual/current/queries/basic/

https://neo4j.com/docs/cypher-manual/current/queries/expressions/

https://neo4j.com/docs/cypher-manual/current/clauses/

https://spark.apache.org/docs/latest/api/python/getting_started/index.html

https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html

https://spark.apache.org/docs/latest/api/python/getting_started/testing_pyspark.html

https://neo4j.com/docs/spark/current/

https://docs.github.com/pt/actions/about-github-actions/understanding-github-actions

https://towardsdatascience.com/using-neo4j-with-pyspark-on-databricks-eb3d127f2245

https://neo4j.com/docs/spark/current/writing/