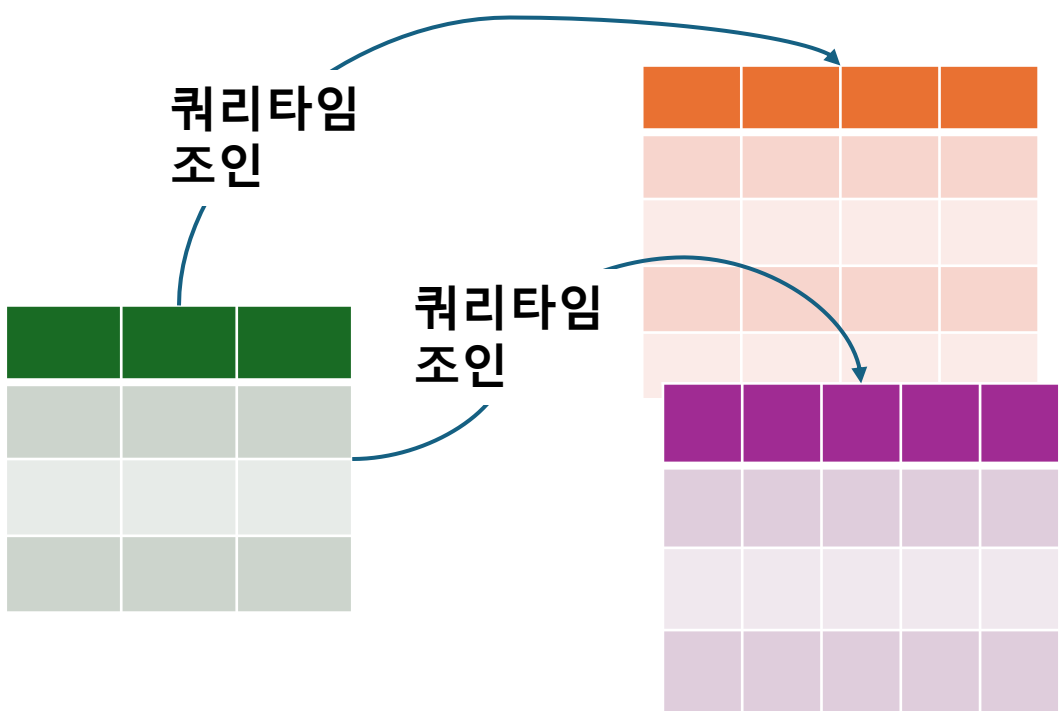
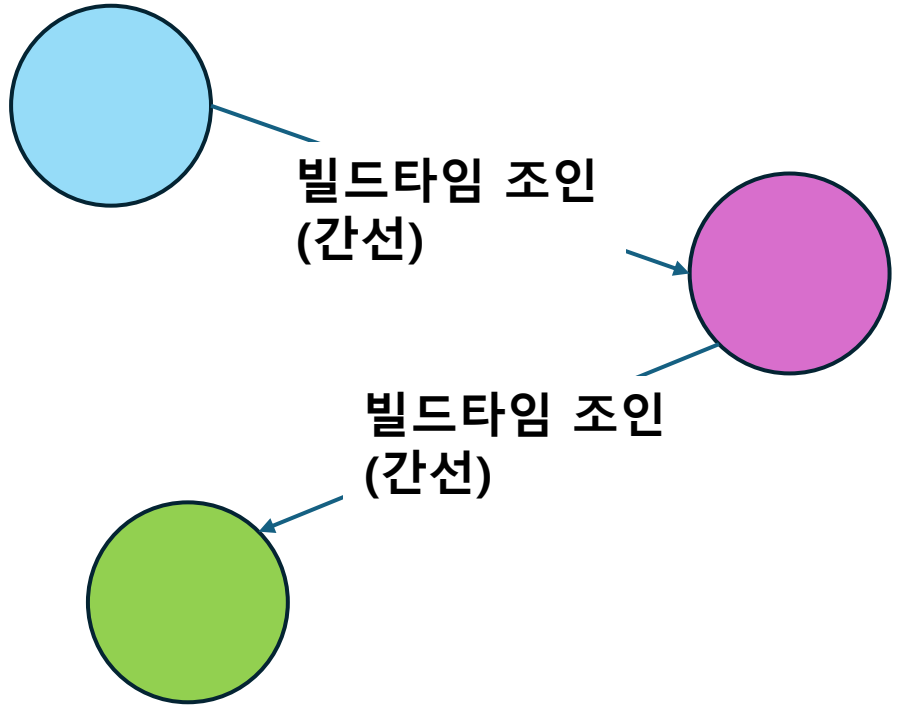
데이터사이언스를위한지식관리시스템

11주차: Neo4j(Graph Database)

그래프데이터베이스

- 관계형데이터베이스는 조인연산이 쿼리타임에 발생
- 그래프데이터베이스는 빌드타임에 조인을 미리하여 간선으로 연결
- 빌드시간은 오래걸릴 수 있지만 한번 간선을 연결해놓고 나면 쿼리타임에는 빠른 탐색





Neo4j

- https://neo4j.com
- MongoDB에서 Atlas로 관리형 서비스를 제공하듯이 Neo4j에서도 AuraDB로 관리형 서비스 제공
- 우리는 구축형으로 사용합니다
- https://neo4j.com/docs/operations-manual/current/installation/
- 그 중에서도 Ubuntu의 경우는 https://neo4j.com/docs/operations-manual/current/installation/linux/debian/ 를 따릅니다
- 여기서 기존 java 버전의 충돌이 발생할 수 있습니다
- java 버전이 요구사항과 맞지 않으면 아래와 같은 일이 발생합니다

```
(base) ubuntu@uos-bigdata:~$ sudo systemctl status neo4j
○ neo4j.service - Neo4j Graph Database
     Loaded: loaded (/usr/lib/systemd/system/neo4j.service; enabled; preset: enabled)
     Active: inactive (dead)

Nov 14 14:39:56 uos-bigdata systemd[1]: Started neo4j.service - Neo4j Graph Database.
Nov 14 14:39:56 uos-bigdata neo4j[9783]: Unsupported Java 11.0.28 detected. Please use Java(TM) 21 or Java(TM) 25 to run Neo4j S
Nov 14 14:39:56 uos-bigdata systemd[1]: neo4j.service: Deactivated successfully.
```

한지붕 두JAVA 사용하기(JAVA_HOME)

- kafka의 경우 실행 스크립트에서 JAVA_HOME이라는 환경변수를 확인하고 그것이 설정되어 있으면 그 경로에 있는 java를 사용합니다
- 따라서 JAVA_HOME으로 분리하는 것이 한지붕 두JAVA를 사용하는 방법의 하나 입니다

```
# Which java to use
if [ -z "$JAVA_HOME" ]; then
  JAVA="java"
else
  JAVA="$JAVA_HOME/bin/java"
fi
```

리눅스의 심볼릭 링크

- 한지붕 두JAVA를 사용하는 또 하나의 방법입니다

```
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ which java
/usr/bin/java
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ ll /usr/bin/java
lrwxrwxrwx 1 root root 22 Jul 17 20:44 /usr/bin/java -> /etc/alternatives/java*
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ ll /etc/alternatives/java
lrwxrwxrwx 1 root root 43 Nov 14 14:00 /etc/alternatives/java -> /usr/lib/jvm/java-21-openjdk-amd64/bin/java*
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$
```

- 그러나 아래와 같이 변경하면 기존 사용중인 다른 애플리케이션이 영향을 받습니다

```
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ sudo rm /etc/alternatives/java
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ sudo ln -s /usr/lib/jvm/java-11-openjdk-amd64/bin/java /etc/alternatives/java
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$ java --version
openjdk 11.0.28 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
(base) ubuntu@uos-bigdata:~/kafka_2.13-3.9.1$
```

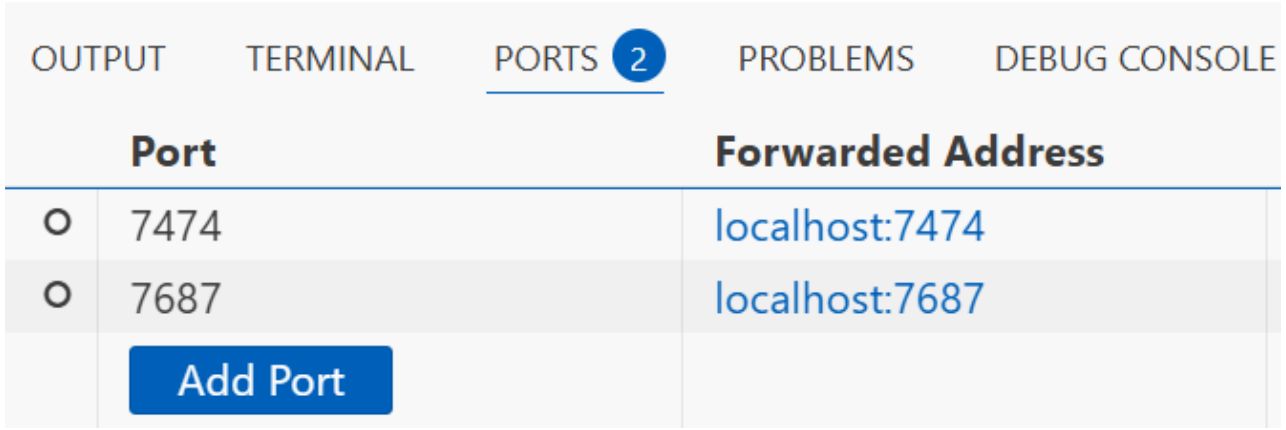
Neo4j 시작하기

- https://neo4j.com/docs/operations-manual/current/installation/linux/debian/ 를 따라 Neo4j가 설치되고 나면 2개의 포트를 수신합니다
- 하나는 Neo4j DB의 입출력을 담당하는 포트 입니다(bolt)
- 다른 하나는 Neo4j DB의 클라이언트를 담당하는 포트 입니다(http)

```
(base) ubuntu@uos-bigdata:~$ sudo netstat -nap | grep java
tcp6       0      0 127.0.0.1:7474          :::*                    LISTEN      10081/java
tcp6       0      0 127.0.0.1:7687          :::*                    LISTEN      10081/java
unix  3      [ ]         STREAM     CONNECTED     44220    10052/java
unix  3      [ ]         STREAM     CONNECTED     45213    10081/java
unix  3      [ ]         STREAM     CONNECTED     45214    10081/java
unix  2      [ ]         STREAM     CONNECTED     45264    10081/java
(base) ubuntu@uos-bigdata:~$
```

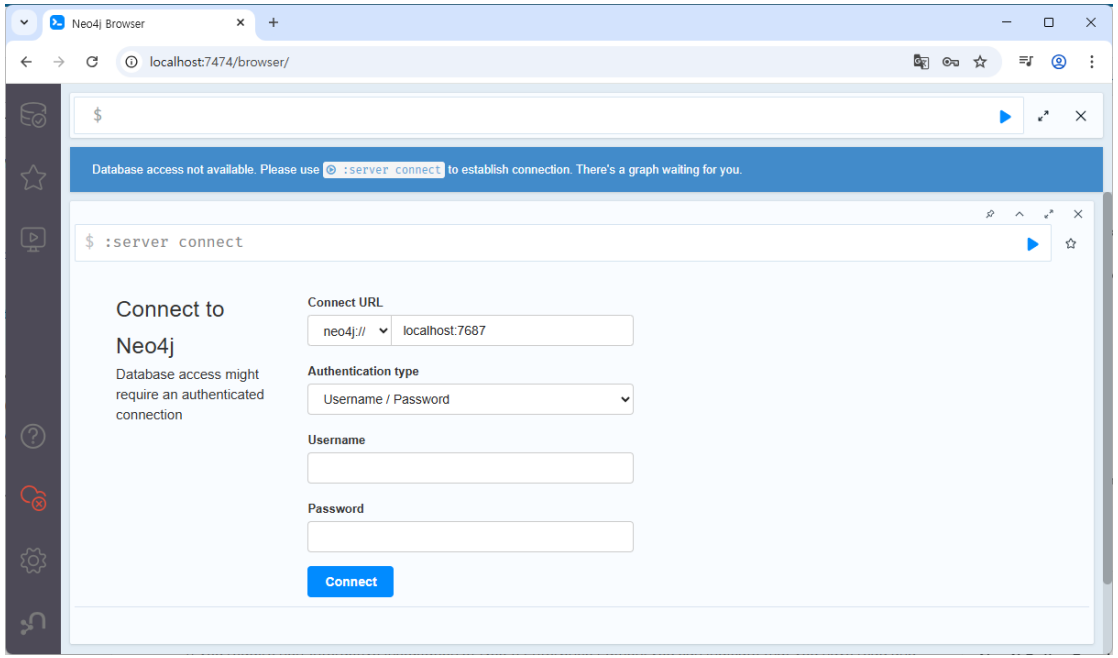
Neo4j 접속하기

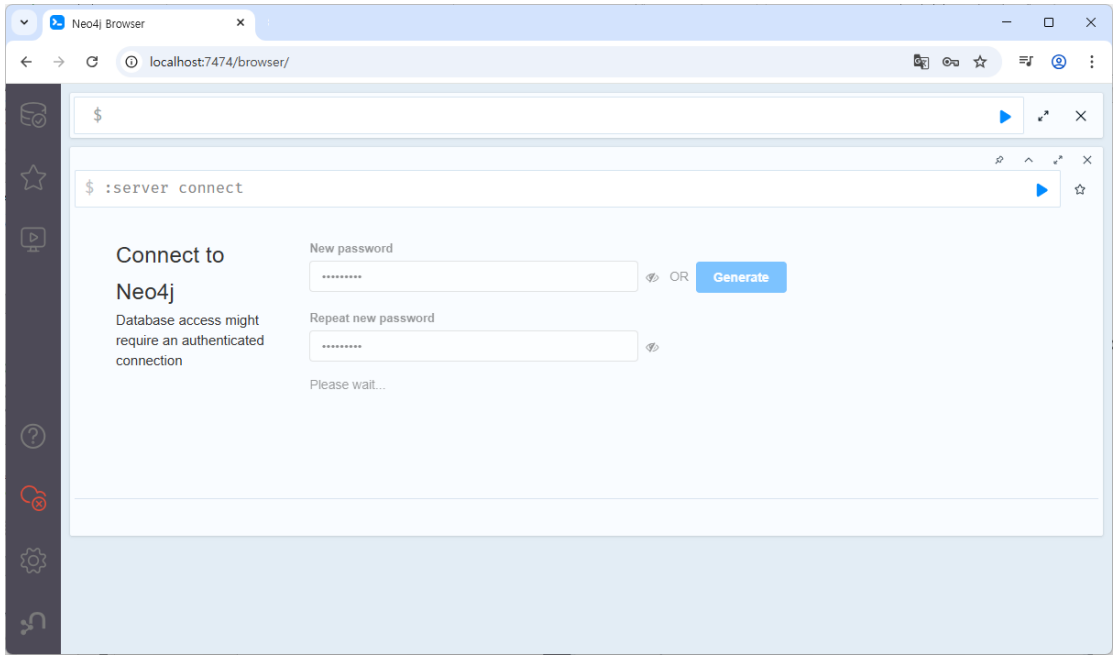
- Neo4j에서는 자체적으로 웹기반의 클라이언트를 제공합니다
- 웹기반 클라이언트를 사용한 방법과 python 클라이언트를 사용한 방법의 두가지로 접근해봅니다
- 따라서 아래와 같이 2개의 포트포워딩 설정합니다(SSH 터널링)



Neo4j 접속하기

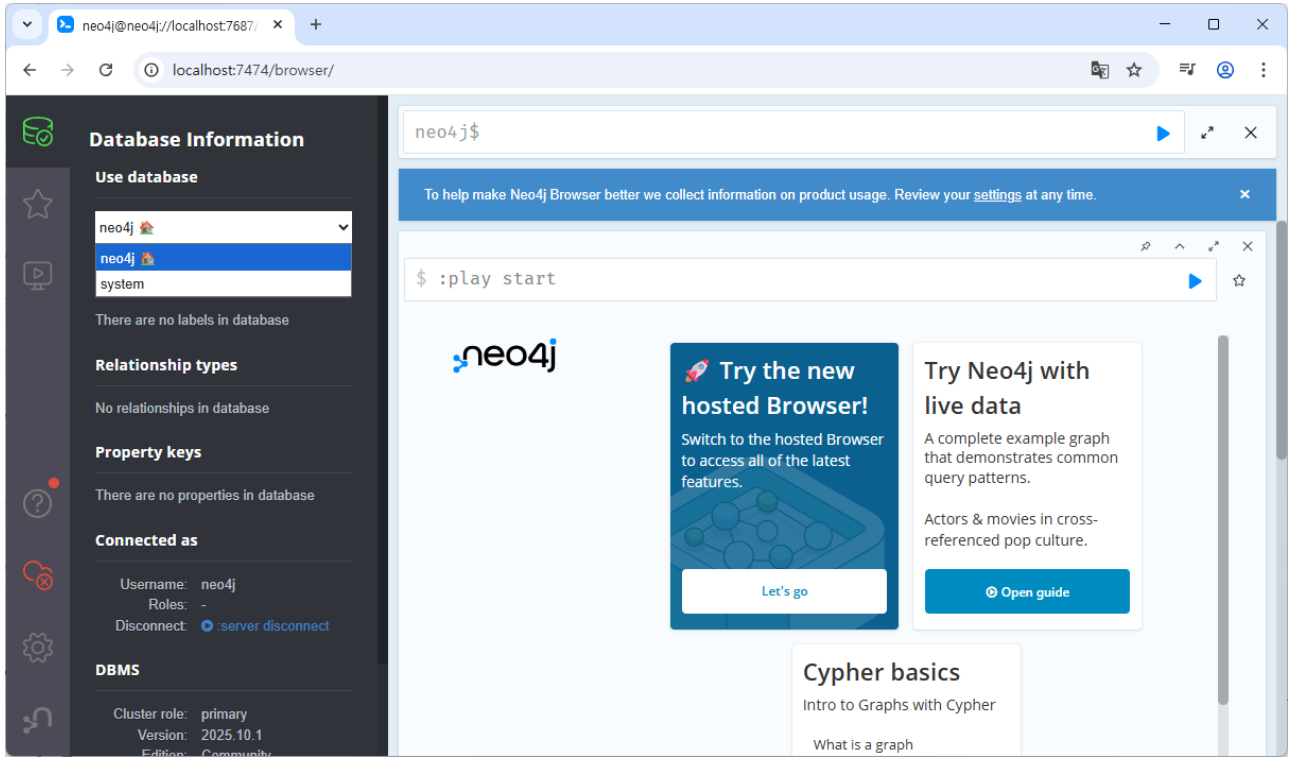
- http://localhost:7474
- URL scheme의 의미: neo4j:// 는 AuraDB나 클러스터 모드에서 사용하며 bolt://는 단일 서버로 구성시 사용합니다
- 이 부분은 mongodb와 유사합니다
- 초기 ID와 PW는 neo4j/neo4j이며 이후 재설정을 진행하게 됩니다





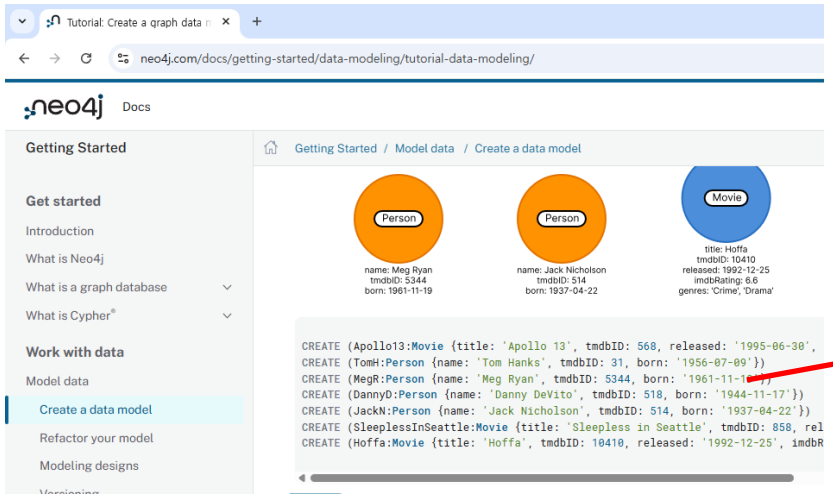
Neo4j 의 각 구성요소: database

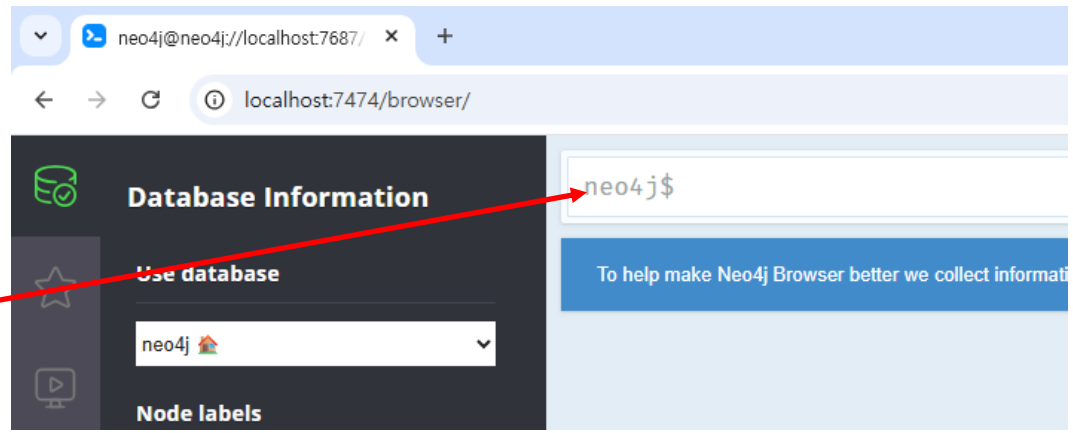
- 마찬가지로 Database라는 논리적 네임스페이스를 제공합니다
- 단, Neo4j community edition에서는 단일 Database(neo4j)만 사용 가능합니다
- MariaDB, MongoDB에서는 개별 Database를 만들었지만 이번에는 단일 Database에서 진행합니다



Neo4j 데이터베이스의 각 구성요소: Cypher

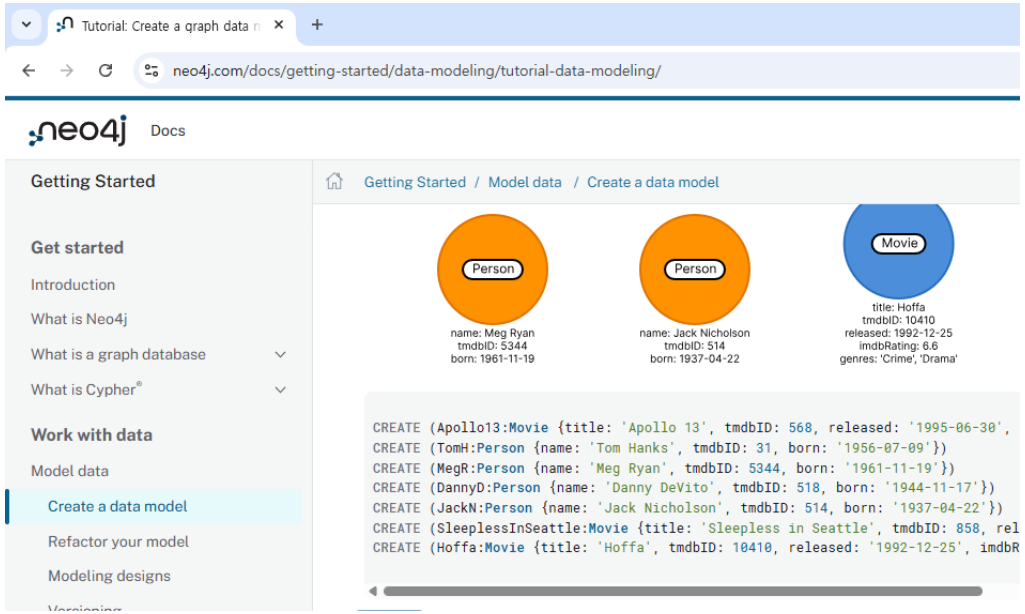
- https://neo4j.com/docs/getting-started/data-modeling/tutorial-data-modeling/ 의 예제를 진행해봅니다
- 위 URL에서 Instance model 을 검색합니다
- 이후 우측과 같이 CREATE... 이라는 문장이 여러 개 있는 것을 복사하여 아래와 같이 붙여넣고 실행해봅니다
- 이것을 Cypher라고 합니다
- MariaDB나 MongoDB의 Query와 비슷한 것입니다
- 즉, Neo4j에서 데이터를 조작하거나 질의하는데 사용하는 명령문을 Cypher라고 합니다





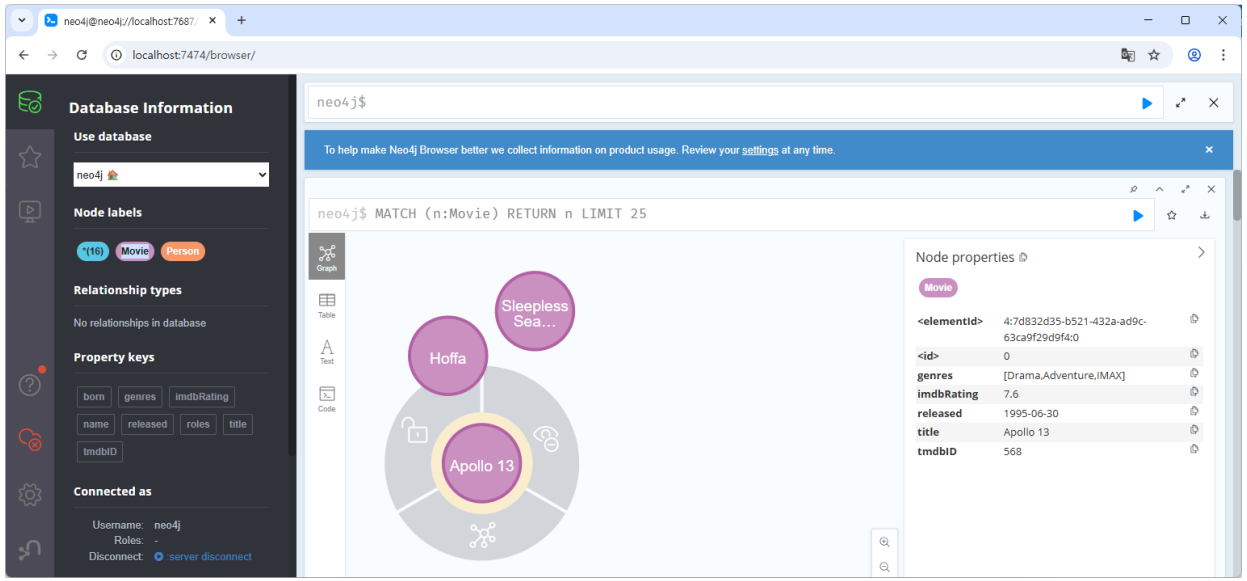
Neo4j 데이터베이스의 각 구성요소: Node

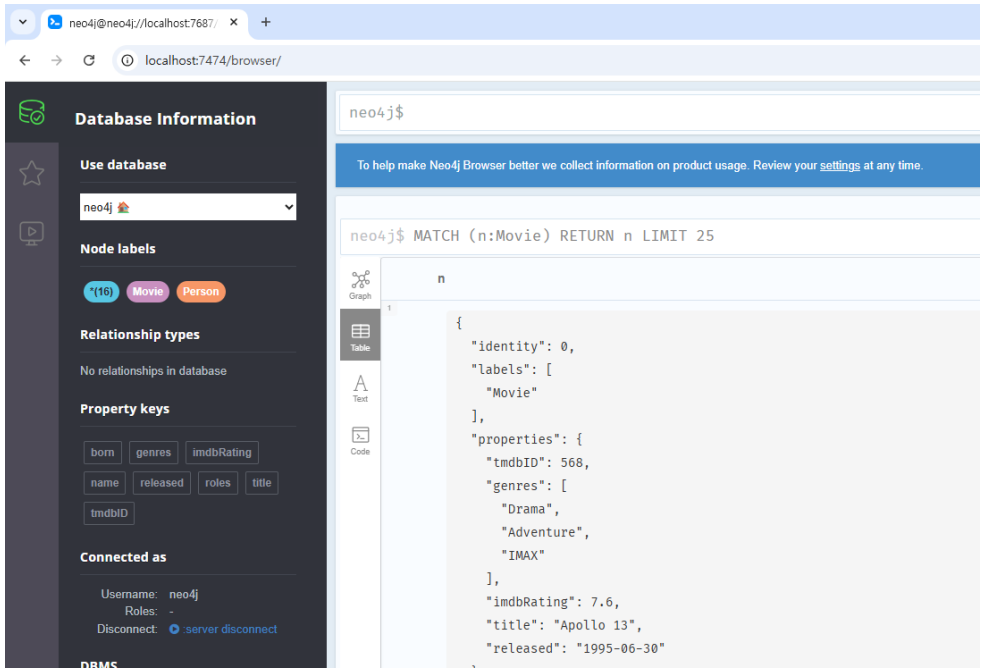
- https://neo4j.com/docs/getting-started/data-modeling/tutorial-data-modeling/ 의 예제를 진행해봅니다
- 위 URL에서 Instance model 을 검색합니다
- 이후 우측과 같이 CREATE… 이라는 문장이 여러 개 있는 것을 복사합니다
- 이 때 Node label을 개별로 바꿔서 진행해보겠습니다
- Node label을 독립적으로 바꾸면 논리적 분리가 가능합니다
- 우측의 예제는 위 예제를 그대로 실행한 것입니다
- Movie와 Person이라는 Node label이 있습니다
- 그런데 우리는 Movie1과 Person1, Movie2와 Person2로 Node label을 분리할 수 있습니다



Neo4j 데이터베이스의 각 구성요소: Property

- 각 노드에 속해 있는 속성입니다
- 그러니까 노드가 마치 MariaDB에서의 하나의 레코드와 비슷해 보입니다
- JSON이나 List형태의 타입이 지원되어서 MongoDB의 document와도 비슷해 보입니다
- Relationship을 연결하기 전까지는 그러해 보입니다





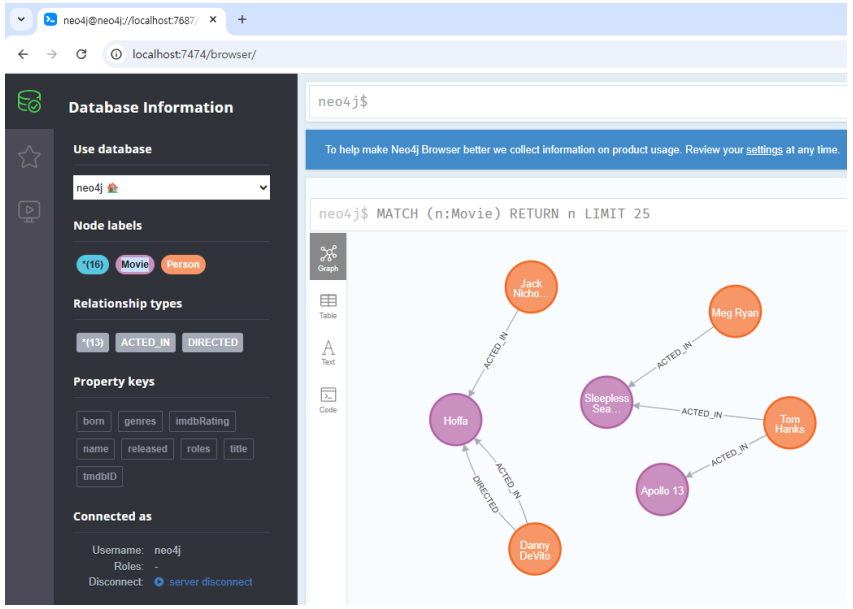
Neo4j 데이터베이스의 각 구성요소: Relationship

- Relationship
- 노드간의 관계를 서술하고 실제로 엣지를 연결합니다
- 이 부분이 관계형 데이터베이스와의 차이 입니다
- 쿼리타임에 엣지가 연결되는 것이 아니라 미리 만들어 두는 것입니다
- Neo4j의 튜토리얼을 그대로 사용시 동작하지 않으므로 아래 내용을 사용합니다

```
MATCH (TomH:Person {name: 'Tom Hanks'})
MATCH (MegR:Person {name: 'Meg Ryan'})
MATCH (DannyD:Person {name: 'Danny DeVito'})
MATCH (JackN:Person {name: 'Jack Nicholson'})

MATCH (Apollo13:Movie {title: 'Apollo 13'})
MATCH (SleeplessInSeattle:Movie {title: 'Sleepless in Seattle'})
MATCH (Hoffa:Movie {title: 'Hoffa'})

MERGE (TomH)-[:ACTED_IN]->(Apollo13)
MERGE (TomH)-[:ACTED_IN]->(SleeplessInSeattle)
MERGE (MegR)-[:ACTED_IN]->(SleeplessInSeattle)
MERGE (DannyD)-[:ACTED_IN]->(Hoffa)
MERGE (DannyD)-[:DIRECTED]->(Hoffa)
MERGE (JackN)-[:ACTED_IN]->(Hoffa);
```

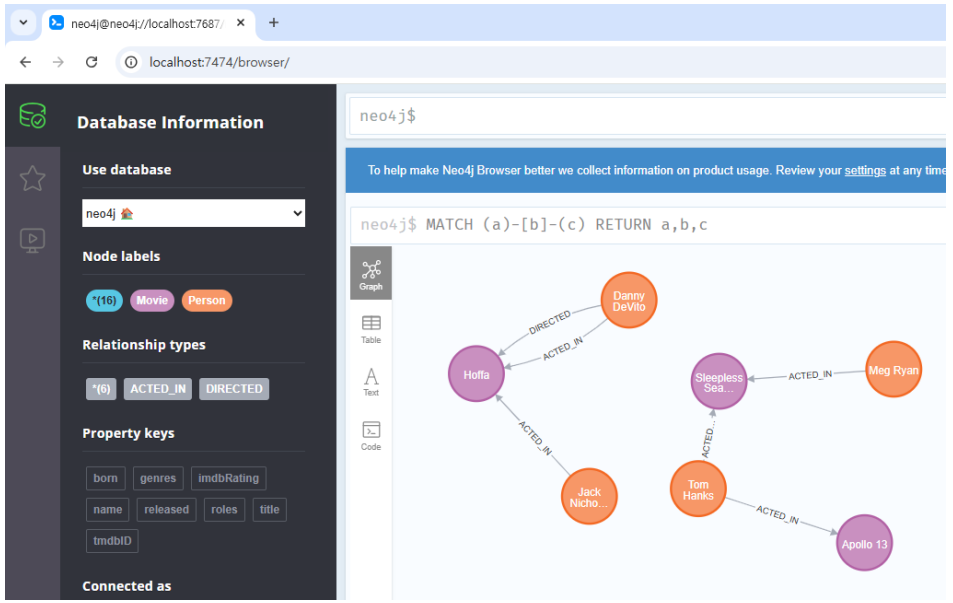


Neo4j 데이터베이스 탐색하기

- Node를 탐색할 때에는 ( ) 기호를 사용합니다
- Relationship을 탐색할 때에는 [ ] 기호를 사용합니다
- Relationship의 방향성을 표시할 때에는 -> 기호를 사용합니다
- 방향성을 생략할 때에는 - 기호를 사용합니다

- (a) - [b] - (c)
- 여기서 a와 c는 node, b는 relationship 입니다
- 그리고 어떠한 제약조건도 서술하지 않았습니다
- 따라서 node2개가 relationship으로 연결된 모든 경우입니다
- 이 때 왼쪽의 node를 a, 오른쪽의 node를 b라고 지칭합니다
- 그리고 relationship은 c라고 지칭합니다
- 이들 셋을 반환합니다

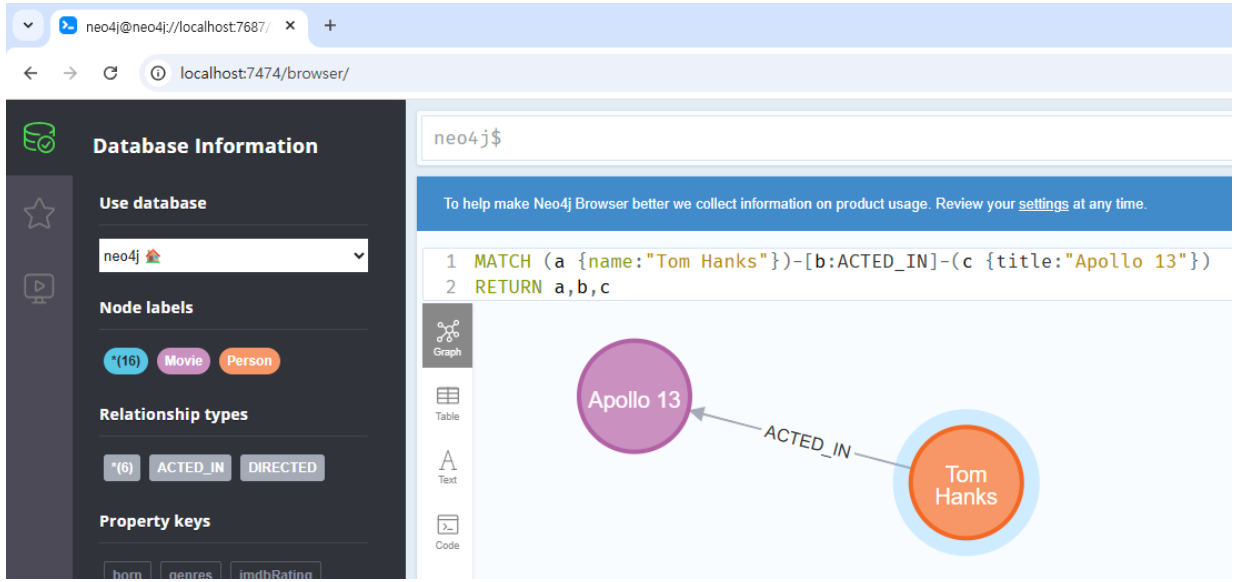
```
MATCH (a)-[b]-(c)
RETURN a,b,c
```



Neo4j 데이터베이스 탐색하기

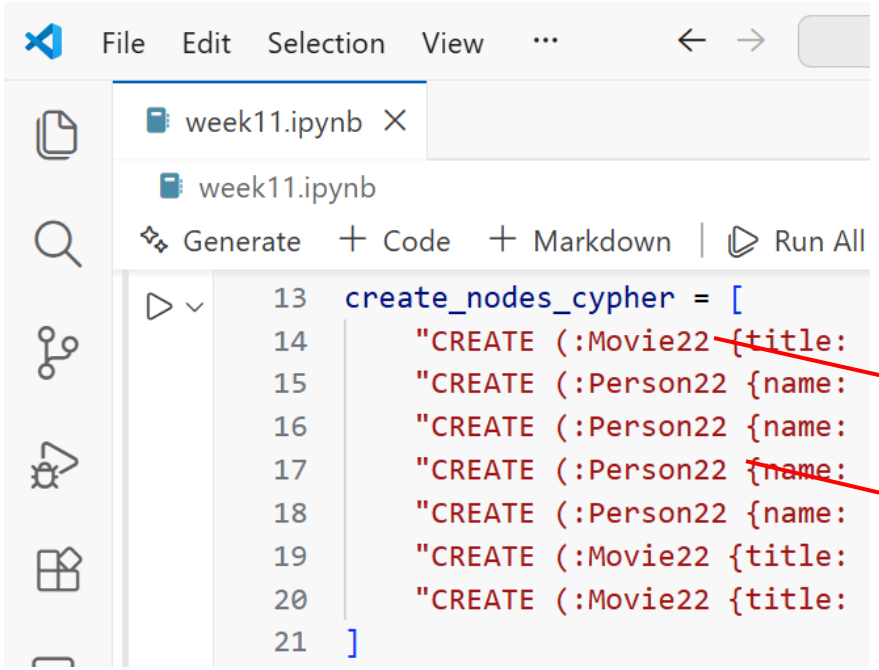
- 이번에는 조건을 추가해봅니다

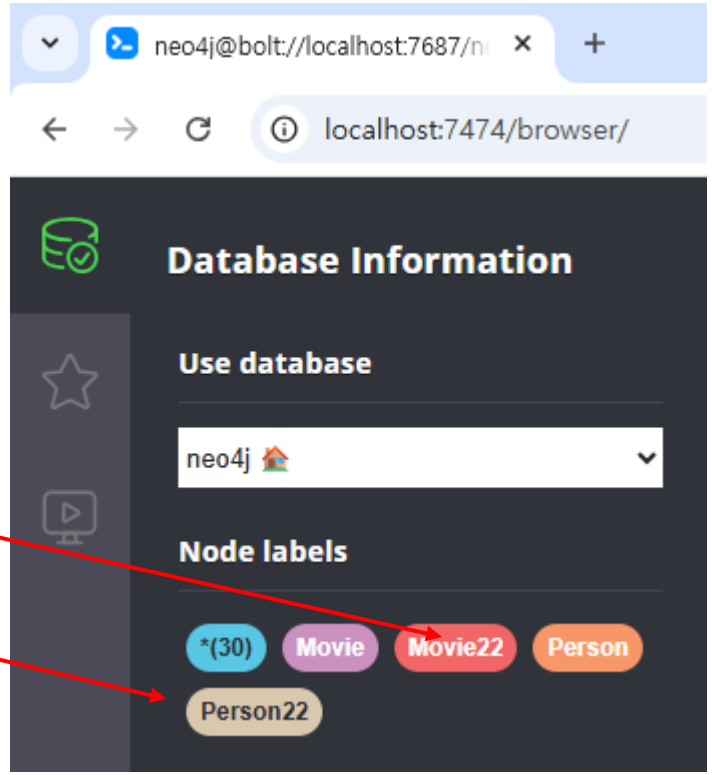
```
MATCH (a {name:"Tom Hanks"})-[b:ACTED_IN]-(c {title:"Apollo 13"})
RETURN a,b,c
```



Python에서 예제 진행하기

- 이제 온라인강의실에서 week11.ipynb를 받아서 실행해봅니다
- 코드 내의 cypher를 변경하여 실행해보고 UI상에서 보여지는 Node, Relationship의 변화를 확인해봅니다





정리

- MariaDB, MongoDB에서 사용하던 데이터를 Neo4j에서도 동일하게 표현할 수 있습니다
- 그러나 그래프 지향 데이터베이스에서는 데이터 간의 관계를 미리 계산하여 기록한다는 것이 결정적인 차이 입니다
- 쿼리타임에는 이를 따라가기만 하기 때문에 탐색 비용은 발생하지만 JOIN 비용이 발생하지 않습니다