Advanced Cypher

Course: Advanced Cypher — Exercises

Welcome to *Browser Guides*! This is the interface for performing exercises to this Neo4j course. They are presented in the form of self-paced exercises. Navigate through the exercises by pressing the forward button at the right of the window.

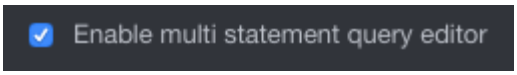
We start by walking you through some preparations and instructions. If you have used Neo4j Browser Guides before, you can skip to the last page.

Before you start

Ensure the database is started

In Neo4j Desktop, ensure that the project you have selected for use with Neo4j Browser is connected to a database that is started.

Enable multi-statement query editor

1. Click the Browser Settings button in the lower left side of Neo4j Browser  .
2. Make sure that the *Enable multi statement query editor* checkbox is selected:


Pin a window

It is useful to pin the Browser Guide to the top, so that it doesn't scroll down when you execute statements. You do this by pressing the *pin* button in the upper right corner of the page:  Pressing it once more will *unpin* the window again:  .

About the exercises

Preparations before each exercise

Each exercise starts with a *Preparations* page. It shows you how to reset the database and make sure that it will produce the expected results. This is useful, for example, if you have made a mistake in one of the previous exercises, or if you have done some testing on your own.

Writing and executing queries

1. Write the query in the query pane at the top of Neo4j Browser.
2. Click the *Run* button  in order to execute the code.

Using code blocks

Many of the exercises contain *code blocks* with runnable code. Follow these steps in order to run the code in your database:

1. Click inside of the code block. You will notice that the code is copied into the code pane at the top of Neo4j Browser.

2. Click the *Run* button  in order to execute the code.

"Taking it further"

Some exercises include optional sections called *Taking it further*. These are extra exercises that you can complete if you have more time, in order to gain more experience with Cypher.

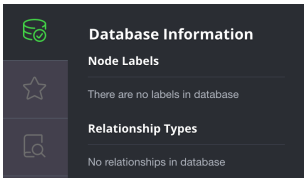
Exercises:

Exercise 1

Exercise 1: Traversing the graph (Preparations)

The database you start with should contain no nodes or relationships.

This is what you should see when you click the database icon  .



Regardless of what is in your graph, run the following script to remove all nodes and relationships from the graph and load the graph with the "mini movie" data:

```
MATCH (n) DETACH DELETE n;
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixReloaded),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixReloaded),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixReloaded),
  (LillyW)-[:DIRECTED]->(TheMatrixReloaded),
  (LanaW)-[:DIRECTED]->(TheMatrixReloaded),
  (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions', released:2003, tagline:'Everything that has a beginning has an end'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixRevolutions),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixRevolutions),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixRevolutions),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixRevolutions),
  (LillyW)-[:DIRECTED]->(TheMatrixRevolutions),
  (LanaW)-[:DIRECTED]->(TheMatrixRevolutions),
  (JoelS)-[:PRODUCED]->(TheMatrixRevolutions)

CREATE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", released:1997, tagline:'Evil has its winning ways'})
CREATE (Charlize:Person {name:'Charlize Theron', born:1975})
CREATE (Al:Person {name:'Al Pacino', born:1940})
CREATE (Taylor:Person {name:'Taylor Hackford', born:1944})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Kevin Lomax']}]->(TheDevilsAdvocate),
  (Charlize)-[:ACTED_IN {roles:['Mary Ann Lomax']}]->(TheDevilsAdvocate),
  (Al)-[:ACTED_IN {roles:['John Milton']}]->(TheDevilsAdvocate),
  (Taylor)-[:DIRECTED]->(TheDevilsAdvocate)
```

```
CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tagline:"In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth."})
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
  (TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),
  (JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),
  (DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]->(AFewGoodMen),
  (KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),
  (KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),
  (NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),
  (CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),
  (KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),
  (JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]->(AFewGoodMen),
  (JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),
  (ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),
  (AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),
  (RobR)-[:DIRECTED]->(AFewGoodMen),
  (AaronS)-[:WROTE]->(AFewGoodMen)

CREATE (TopGun:Movie {title:"Top Gun", released:1986, tagline:'I feel the need, the need for speed.'})
CREATE (KellyM:Person {name:'Kelly McGillis', born:1957})
CREATE (ValK:Person {name:'Val Kilmer', born:1959})
CREATE (AnthonyE:Person {name:'Anthony Edwards', born:1962})
CREATE (TomS:Person {name:'Tom Skerritt', born:1933})
CREATE (MegR:Person {name:'Meg Ryan', born:1961})
CREATE (TonyS:Person {name:'Tony Scott', born:1944})
CREATE (JimC:Person {name:'Jim Cash', born:1941})
CREATE
  (TomC)-[:ACTED_IN {roles:['Maverick']}]->(TopGun),
  (KellyM)-[:ACTED_IN {roles:['Charlie']}]->(TopGun),
  (ValK)-[:ACTED_IN {roles:['Iceman']}]->(TopGun),
  (AnthonyE)-[:ACTED_IN {roles:['Goose']}]->(TopGun),
  (TomS)-[:ACTED_IN {roles:['Viper']}]->(TopGun),
  (MegR)-[:ACTED_IN {roles:['Carole']}]->(TopGun),
  (TonyS)-[:DIRECTED]->(TopGun),
  (JimC)-[:WROTE]->(TopGun)

CREATE (JerryMaguire:Movie {title:'Jerry Maguire', released:2000, tagline:'The rest of his life begins now.'})
CREATE (ReneeZ:Person {name:'Renee Zellweger', born:1969})
CREATE (KellyP:Person {name:'Kelly Preston', born:1962})
CREATE (JerryO:Person {name:"Jerry O'Connell", born:1974})
CREATE (JayM:Person {name:'Jay Mohr', born:1970})
CREATE (BonnieH:Person {name:'Bonnie Hunt', born:1961})
CREATE (ReginaK:Person {name:'Regina King', born:1971})
CREATE (JonathanL:Person {name:'Jonathan Lipnicki', born:1996})
CREATE (CameronC:Person {name:'Cameron Crowe', born:1957})
CREATE
  (TomC)-[:ACTED_IN {roles:['Jerry Maguire']}]->(JerryMaguire),
  (CubaG)-[:ACTED_IN {roles:['Rod Tidwell']}]->(JerryMaguire),
  (ReneeZ)-[:ACTED_IN {roles:['Dorothy Boyd']}]->(JerryMaguire),
  (KellyP)-[:ACTED_IN {roles:['Avery Bishop']}]->(JerryMaguire),
  (JerryO)-[:ACTED_IN {roles:['Frank Cushman']}]->(JerryMaguire),
  (JayM)-[:ACTED_IN {roles:['Bob Sugar']}]->(JerryMaguire),
  (BonnieH)-[:ACTED_IN {roles:['Laurel Boyd']}]->(JerryMaguire),
  (ReginaK)-[:ACTED_IN {roles:['Marcee Tidwell']}]->(JerryMaguire),
  (JonathanL)-[:ACTED_IN {roles:['Ray Boyd']}]->(JerryMaguire),
  (CameronC)-[:DIRECTED]->(JerryMaguire),
  (CameronC)-[:PRODUCED]->(JerryMaguire),
  (CameronC)-[:WROTE]->(JerryMaguire)

CREATE (StandByMe:Movie {title:"Stand By Me", released:1986, tagline:"For some, it's the last real taste of innocence, and the first real taste of life. But for everyone, it's the time that memories are made of."})
CREATE (RiverP:Person {name:'River Phoenix', born:1970})
CREATE (CoreyF:Person {name:'Corey Feldman', born:1971})
CREATE (WilW:Person {name:'Wil Wheaton', born:1972})
CREATE (JohnC:Person {name:'John Cusack', born:1966})
CREATE (MarshallB:Person {name:'Marshall Bell', born:1942})
CREATE
```

```
  (WilW)-[:ACTED_IN {roles:['Gordie Lachance']}]->(StandByMe),
  (RiverP)-[:ACTED_IN {roles:['Chris Chambers']}]->(StandByMe),
  (JerryO)-[:ACTED_IN {roles:['Vern Tessio']}]->(StandByMe),
  (CoreyF)-[:ACTED_IN {roles:['Teddy Duchamp']}]->(StandByMe),
  (JohnC)-[:ACTED_IN {roles:['Denny Lachance']}]->(StandByMe),
  (KieferS)-[:ACTED_IN {roles:['Ace Merrill']}]->(StandByMe),
  (MarshallB)-[:ACTED_IN {roles:['Mr. Lachance']}]->(StandByMe),
  (RobR)-[:DIRECTED]->(StandByMe)

CREATE (AsGoodAsItGets:Movie {title:'As Good as It Gets', released:1997, tagline:'A comedy from the heart that goes for the throat.'})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})
CREATE (GregK:Person {name:'Greg Kinnear', born:1963})
CREATE (JamesB:Person {name:'James L. Brooks', born:1940})
CREATE
  (JackN)-[:ACTED_IN {roles:['Melvin Udall']}]->(AsGoodAsItGets),
  (HelenH)-[:ACTED_IN {roles:['Carol Connelly']}]->(AsGoodAsItGets),
  (GregK)-[:ACTED_IN {roles:['Simon Bishop']}]->(AsGoodAsItGets),
  (CubaG)-[:ACTED_IN {roles:['Frank Sachs']}]->(AsGoodAsItGets),
  (JamesB)-[:DIRECTED]->(AsGoodAsItGets)

CREATE (WhatDreamsMayCome:Movie {title:'What Dreams May Come', released:1998, tagline:'After life there is more. The end is just the
beginning.'})
CREATE (AnnabellaS:Person {name:'Annabella Sciorra', born:1960})
CREATE (MaxS:Person {name:'Max von Sydow', born:1929})
CREATE (WernerH:Person {name:'Werner Herzog', born:1942})
CREATE (Robin:Person {name:'Robin Williams', born:1951})
CREATE (VincentW:Person {name:'Vincent Ward', born:1956})
CREATE
  (Robin)-[:ACTED_IN {roles:['Chris Nielsen']}]->(WhatDreamsMayCome),
  (CubaG)-[:ACTED_IN {roles:['Albert Lewis']}]->(WhatDreamsMayCome),
  (AnnabellaS)-[:ACTED_IN {roles:['Annie Collins-Nielsen']}]->(WhatDreamsMayCome),
  (MaxS)-[:ACTED_IN {roles:['The Tracker']}]->(WhatDreamsMayCome),
  (WernerH)-[:ACTED_IN {roles:['The Face']}]->(WhatDreamsMayCome),
  (VincentW)-[:DIRECTED]->(WhatDreamsMayCome)

CREATE (SnowFallingonCedars:Movie {title:'Snow Falling on Cedars', released:1999, tagline:'First loves last. Forever.'})
CREATE (EthanH:Person {name:'Ethan Hawke', born:1970})
CREATE (RickY:Person {name:'Rick Yune', born:1971})
CREATE (JamesC:Person {name:'James Cromwell', born:1940})
CREATE (ScottH:Person {name:'Scott Hicks', born:1953})
CREATE
  (EthanH)-[:ACTED_IN {roles:['Ishmael Chambers']}]->(SnowFallingonCedars),
  (RickY)-[:ACTED_IN {roles:['Kazuo Miyamoto']}]->(SnowFallingonCedars),
  (MaxS)-[:ACTED_IN {roles:['Nels Gudmundsson']}]->(SnowFallingonCedars),
  (JamesC)-[:ACTED_IN {roles:['Judge Fielding']}]->(SnowFallingonCedars),
  (ScottH)-[:DIRECTED]->(SnowFallingonCedars)

CREATE (YouveGotMail:Movie {title:"You've Got Mail", released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (ParkerP:Person {name:'Parker Posey', born:1968})
CREATE (DaveC:Person {name:'Dave Chappelle', born:1973})
CREATE (SteveZ:Person {name:'Steve Zahn', born:1967})
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
  (MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]->(YouveGotMail),
  (GregK)-[:ACTED_IN {roles:['Frank Navasky']}]->(YouveGotMail),
  (ParkerP)-[:ACTED_IN {roles:['Patricia Eden']}]->(YouveGotMail),
  (DaveC)-[:ACTED_IN {roles:['Kevin Jackson']}]->(YouveGotMail),
  (SteveZ)-[:ACTED_IN {roles:['George Pappas']}]->(YouveGotMail),
  (NoraE)-[:DIRECTED]->(YouveGotMail)

CREATE (SleeplessInSeattle:Movie {title:'Sleepless in Seattle', released:1993, tagline:'What if someone you never met, someone you never saw,
someone you never knew was the only someone for you?'})
CREATE (RitaW:Person {name:'Rita Wilson', born:1956})
CREATE (BillPull:Person {name:'Bill Pullman', born:1953})
CREATE (VictorG:Person {name:'Victor Garber', born:1949})
CREATE (RosieO:Person {name:"Rosie O'Donnell", born:1962})
CREATE
  (TomH)-[:ACTED_IN {roles:['Sam Baldwin']}]->(SleeplessInSeattle),
  (MegR)-[:ACTED_IN {roles:['Annie Reed']}]->(SleeplessInSeattle),
  (RitaW)-[:ACTED_IN {roles:['Suzy']}]->(SleeplessInSeattle),
  (BillPull)-[:ACTED_IN {roles:['Walter']}]->(SleeplessInSeattle),
  (VictorG)-[:ACTED_IN {roles:['Greg']}]->(SleeplessInSeattle),
  (RosieO)-[:ACTED_IN {roles:['Becky']}]->(SleeplessInSeattle),
  (NoraE)-[:DIRECTED]->(SleeplessInSeattle)

CREATE (JoeVersustheVolcano:Movie {title:'Joe Versus the Volcano', released:1990, tagline:'A story of love, lava and burning desire.'})
CREATE (JohnS:Person {name:'John Patrick Stanley', born:1950})
CREATE (Nathan:Person {name:'Nathan Lane', born:1956})
```

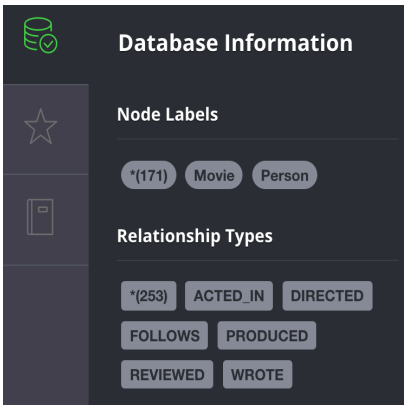
```
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Banks']}]->(JoeVersustheVolcano),
  (MegR)-[:ACTED_IN {roles:['DeDe', 'Angelica Graynamore', 'Patricia Graynamore']}]->(JoeVersustheVolcano),
  (Nathan)-[:ACTED_IN {roles:['Baw']}]->(JoeVersustheVolcano),
  (JohnS)-[:DIRECTED]->(JoeVersustheVolcano)

CREATE (WhenHarryMetSally:Movie {title:'When Harry Met Sally', released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (BillyC:Person {name:'Billy Crystal', born:1948})
CREATE (CarrieF:Person {name:'Carrie Fisher', born:1956})
CREATE (BrunoK:Person {name:'Bruno Kirby', born:1949})
CREATE
  (BillyC)-[:ACTED_IN {roles:['Harry Burns']}]->(WhenHarryMetSally),
  (MegR)-[:ACTED_IN {roles:['Sally Albright']}]->(WhenHarryMetSally),
  (CarrieF)-[:ACTED_IN {roles:['Marie']}]->(WhenHarryMetSally),
  (BrunoK)-[:ACTED_IN {roles:['Jess']}]->(WhenHarryMetSally),
  (RobR)-[:DIRECTED]->(WhenHarryMetSally),
  (RobR)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:WROTE]->(WhenHarryMetSally)

CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', released:1996, tagline:'In every life there comes a time when that thing you
dream becomes that thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
  (TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
  (LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
  (Charlize)-[:ACTED_IN {roles:['Tina']}]->(ThatThingYouDo),
  (TomH)-[:DIRECTED]->(ThatThingYouDo)

CREATE (TheReplacements:Movie {title:'The Replacements', released:2000, tagline:'Pain heals, Chicks dig scars... Glory lasts forever'})
CREATE (Brooke:Person {name:'Brooke Langton', born:1970})
CREATE (Gene:Person {name:'Gene Hackman', born:1930})
CREATE (Orlando:Person {name:'Orlando Jones', born:1968})
CREATE (Howard:Person {name:'Howard Deutch', born:1950})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Shane Falco']}]->(TheReplacements),
  (Brooke)-[:ACTED_IN {roles:['Annabelle Farrell']}]->(TheReplacements),
  (Gene)-[:ACTED_IN {roles:['Jimmy McGinty']}]->(TheReplacements),
  (Orlando)-[:ACTED_IN {roles:['Clifford Franklin']}]->(TheReplacements),
  (Howard)-[:DIRECTED]->(TheReplacements)

CREATE (RescueDawn:Movie {title:'RescueDawn', released:2006, tagline:"Based on the extraordinary true story of one man's fight for
freedom"})
CREATE (ChristianB:Person {name:'Christian Bale', born:1974})
CREATE (ZachG:Person {name:'Zach Grenier', born:1954})
CREATE
  (MarshallB)-[:ACTED_IN {roles:['Admiral']}]->(RescueDawn),
  (ChristianB)-[:ACTED_IN {roles:['Dieter Dengler']}]->(RescueDawn),
  (ZachG)-[:ACTED_IN {roles:['Squad Leader']}]->(RescueDawn),
  (SteveZ)-[:ACTED_IN {roles:['Duane']}]->(RescueDawn),
  (WernerH)-[:DIRECTED]->(RescueDawn)

CREATE (TheBirdcage:Movie {title:'The Birdcage', released:1996, tagline:'Come as you are'})
CREATE (MikeN:Person {name:'Mike Nichols', born:1931})
CREATE
  (Robin)-[:ACTED_IN {roles:['Armand Goldman']}]->(TheBirdcage),
  (Nathan)-[:ACTED_IN {roles:['Albert Goldman']}]->(TheBirdcage),
  (Gene)-[:ACTED_IN {roles:['Sen. Kevin Keeley']}]->(TheBirdcage),
  (MikeN)-[:DIRECTED]->(TheBirdcage)

CREATE (Unforgiven:Movie {title:'Unforgiven', released:1992, tagline:"It's a hell of a thing, killing a man"})
CREATE (RichardH:Person {name:'Richard Harris', born:1930})
CREATE (ClintE:Person {name:'Clint Eastwood', born:1930})
CREATE
  (RichardH)-[:ACTED_IN {roles:['English Bob']}]->(Unforgiven),
  (ClintE)-[:ACTED_IN {roles:['Bill Munny']}]->(Unforgiven),
  (Gene)-[:ACTED_IN {roles:['Little Bill Daggett']}]->(Unforgiven),
  (ClintE)-[:DIRECTED]->(Unforgiven)

CREATE (JohnnyMnemonic:Movie {title:'Johnny Mnemonic', released:1995, tagline:'The hottest data on earth. In the coolest head in town'})
CREATE (Takeshi:Person {name:'Takeshi Kitano', born:1947})
CREATE (Dina:Person {name:'Dina Meyer', born:1968})
CREATE (IceT:Person {name:'Ice-T', born:1958})
CREATE (RobertL:Person {name:'Robert Longo', born:1953})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Johnny Mnemonic']}]->(JohnnyMnemonic),
  (Takeshi)-[:ACTED_IN {roles:['Takahashi']}]->(JohnnyMnemonic),
  (Dina)-[:ACTED_IN {roles:['Jane']}]->(JohnnyMnemonic),
  (IceT)-[:ACTED_IN {roles:['J-Bone']}]->(JohnnyMnemonic),
  (RobertL)-[:DIRECTED]->(JohnnyMnemonic)
```

```
CREATE (CloudAtlas:Movie {title:'Cloud Atlas', released:2012, tagline:'Everything is connected'})
CREATE (HalleB:Person {name:'Halle Berry', born:1966})
CREATE (JimB:Person {name:'Jim Broadbent', born:1949})
CREATE (TomT:Person {name:'Tom Tykwer', born:1965})
CREATE (DavidMitchell:Person {name:'David Mitchell', born:1969})
CREATE (StefanArndt:Person {name:'Stefan Arndt', born:1961})
CREATE
  (TomH)-[:ACTED_IN {roles:['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']}]->(CloudAtlas),
  (Hugo)-[:ACTED_IN {roles:['Bill Smoke', 'Haskell Moore', 'Tadeusz Kesselring', 'Nurse Noakes', 'Boardman Mephi', 'Old Georgie']}]->
(CloudAtlas),
  (HalleB)-[:ACTED_IN {roles:['Luisa Rey', 'Jocasta Ayrs', 'Ovid', 'Meronym']}]->(CloudAtlas),
  (JimB)-[:ACTED_IN {roles:['Vyvyan Ayrs', 'Captain Molyneux', 'Timothy Cavendish']}]->(CloudAtlas),
  (TomT)-[:DIRECTED]->(CloudAtlas),
  (LillyW)-[:DIRECTED]->(CloudAtlas),
  (LanaW)-[:DIRECTED]->(CloudAtlas),
  (DavidMitchell)-[:WROTE]->(CloudAtlas),
  (StefanArndt)-[:PRODUCED]->(CloudAtlas)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006, tagline:'Break The Codes'})
CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)

CREATE (VforVendetta:Movie {title:'V for Vendetta', released:2006, tagline:'Freedom! Forever!'})
CREATE (NatalieP:Person {name:'Natalie Portman', born:1981})
CREATE (StephenR:Person {name:'Stephen Rea', born:1946})
CREATE (JohnH:Person {name:'John Hurt', born:1940})
CREATE (BenM:Person {name: 'Ben Miles', born:1967})
CREATE
  (Hugo)-[:ACTED_IN {roles:['V']}]->(VforVendetta),
  (NatalieP)-[:ACTED_IN {roles:['Evey Hammond']}]->(VforVendetta),
  (StephenR)-[:ACTED_IN {roles:['Eric Finch']}]->(VforVendetta),
  (JohnH)-[:ACTED_IN {roles:['High Chancellor Adam Sutler']}]->(VforVendetta),
  (BenM)-[:ACTED_IN {roles:['Dascomb']}]->(VforVendetta),
  (JamesM)-[:DIRECTED]->(VforVendetta),
  (LillyW)-[:PRODUCED]->(VforVendetta),
  (LanaW)-[:PRODUCED]->(VforVendetta),
  (JoelS)-[:PRODUCED]->(VforVendetta),
  (LillyW)-[:WROTE]->(VforVendetta),
  (LanaW)-[:WROTE]->(VforVendetta)

CREATE (SpeedRacer:Movie {title:'Speed Racer', released:2008, tagline:'Speed has no limits'})
CREATE (EmileH:Person {name:'Emile Hirsch', born:1985})
CREATE (JohnG:Person {name:'John Goodman', born:1960})
CREATE (SusanS:Person {name:'Susan Sarandon', born:1946})
CREATE (MatthewF:Person {name:'Matthew Fox', born:1966})
CREATE (ChristinaR:Person {name:'Christina Ricci', born:1980})
CREATE (Rain:Person {name:'Rain', born:1982})
CREATE
  (EmileH)-[:ACTED_IN {roles:['Speed Racer']}]->(SpeedRacer),
  (JohnG)-[:ACTED_IN {roles:['Pops']}]->(SpeedRacer),
  (SusanS)-[:ACTED_IN {roles:['Mom']}]->(SpeedRacer),
  (MatthewF)-[:ACTED_IN {roles:['Racer X']}]->(SpeedRacer),
  (ChristinaR)-[:ACTED_IN {roles:['Trixie']}]->(SpeedRacer),
  (Rain)-[:ACTED_IN {roles:['Taejo Togokahn']}]->(SpeedRacer),
  (BenM)-[:ACTED_IN {roles:['Cass Jones']}]->(SpeedRacer),
  (LillyW)-[:DIRECTED]->(SpeedRacer),
  (LanaW)-[:DIRECTED]->(SpeedRacer),
  (LillyW)-[:WROTE]->(SpeedRacer),
  (LanaW)-[:WROTE]->(SpeedRacer),
  (JoelS)-[:PRODUCED]->(SpeedRacer)

CREATE (NinjaAssassin:Movie {title:'Ninja Assassin', released:2009, tagline:'Prepare to enter a secret world of assassins'})
CREATE (NaomieH:Person {name:'Naomie Harris'})
CREATE
  (Rain)-[:ACTED_IN {roles:['Raizo']}]->(NinjaAssassin),
  (NaomieH)-[:ACTED_IN {roles:['Mika Coretti']}]->(NinjaAssassin),
  (RickY)-[:ACTED_IN {roles:['Takeshi']}]->(NinjaAssassin),
  (BenM)-[:ACTED_IN {roles:['Ryan Maslow']}]->(NinjaAssassin),
  (JamesM)-[:DIRECTED]->(NinjaAssassin),
  (LillyW)-[:PRODUCED]->(NinjaAssassin),
  (LanaW)-[:PRODUCED]->(NinjaAssassin),
  (JoelS)-[:PRODUCED]->(NinjaAssassin)
```

```
CREATE (TheGreenMile:Movie {title:'The Green Mile', released:1999, tagline:"Walk a mile you'll never forget."})
CREATE (MichaelD:Person {name:'Michael Clarke Duncan', born:1957})
CREATE (DavidM:Person {name:'David Morse', born:1953})
CREATE (SamR:Person {name:'Sam Rockwell', born:1968})
CREATE (GaryS:Person {name:'Gary Sinise', born:1955})
CREATE (PatriciaC:Person {name:'Patricia Clarkson', born:1959})
CREATE (FrankD:Person {name:'Frank Darabont', born:1959})
CREATE
  (TomH)-[:ACTED_IN {roles:['Paul Edgecomb']}]->(TheGreenMile),
  (MichaelD)-[:ACTED_IN {roles:['John Coffey']}]->(TheGreenMile),
  (DavidM)-[:ACTED_IN {roles:['Brutus "Brutal" Howell']}]->(TheGreenMile),
  (BonnieH)-[:ACTED_IN {roles:['Jan Edgecomb']}]->(TheGreenMile),
  (JamesC)-[:ACTED_IN {roles:['Warden Hal Moores']}]->(TheGreenMile),
  (SamR)-[:ACTED_IN {roles:['"Wild Bill" Wharton']}]->(TheGreenMile),
  (GaryS)-[:ACTED_IN {roles:['Burt Hammersmith']}]->(TheGreenMile),
  (PatriciaC)-[:ACTED_IN {roles:['Melinda Moores']}]->(TheGreenMile),
  (FrankD)-[:DIRECTED]->(TheGreenMile)

CREATE (FrostNixon:Movie {title:'Frost/Nixon', released:2008, tagline:'400 million people were waiting for the truth.'})
CREATE (FrankL:Person {name:'Frank Langella', born:1938})
CREATE (MichaelS:Person {name:'Michael Sheen', born:1969})
CREATE (OliverP:Person {name:'Oliver Platt', born:1960})
CREATE
  (FrankL)-[:ACTED_IN {roles:['Richard Nixon']}]->(FrostNixon),
  (MichaelS)-[:ACTED_IN {roles:['David Frost']}]->(FrostNixon),
  (KevinB)-[:ACTED_IN {roles:['Jack Brennan']}]->(FrostNixon),
  (OliverP)-[:ACTED_IN {roles:['Bob Zelnick']}]->(FrostNixon),
  (SamR)-[:ACTED_IN {roles:['James Reston, Jr.']}]->(FrostNixon),
  (RonH)-[:DIRECTED]->(FrostNixon)

CREATE (Hoffa:Movie {title:'Hoffa', released:1992, tagline:"He didn't want law. He wanted justice."})
CREATE (DannyD:Person {name:'Danny DeVito', born:1944})
CREATE (JohnR:Person {name:'John C. Reilly', born:1965})
CREATE
  (JackN)-[:ACTED_IN {roles:['Hoffa']}]->(Hoffa),
  (DannyD)-[:ACTED_IN {roles:['Robert "Bobby" Ciaro']}]->(Hoffa),
  (JTW)-[:ACTED_IN {roles:['Frank Fitzsimmons']}]->(Hoffa),
  (JohnR)-[:ACTED_IN {roles:['Peter "Pete" Connelly']}]->(Hoffa),
  (DannyD)-[:DIRECTED]->(Hoffa)

CREATE (Apollo13:Movie {title:'Apollo 13', released:1995, tagline:'Houston, we have a problem.'})
CREATE (EdH:Person {name:'Ed Harris', born:1950})
CREATE (BillPax:Person {name:'Bill Paxton', born:1955})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jim Lovell']}]->(Apollo13),
  (KevinB)-[:ACTED_IN {roles:['Jack Swigert']}]->(Apollo13),
  (EdH)-[:ACTED_IN {roles:['Gene Kranz']}]->(Apollo13),
  (BillPax)-[:ACTED_IN {roles:['Fred Haise']}]->(Apollo13),
  (GaryS)-[:ACTED_IN {roles:['Ken Mattingly']}]->(Apollo13),
  (RonH)-[:DIRECTED]->(Apollo13)

CREATE (Twister:Movie {title:'Twister', released:1996, tagline:"Don't Breathe. Don't Look Back."})
CREATE (PhilipH:Person {name:'Philip Seymour Hoffman', born:1967})
CREATE (JanB:Person {name:'Jan de Bont', born:1943})
CREATE
  (BillPax)-[:ACTED_IN {roles:['Bill Harding']}]->(Twister),
  (HelenH)-[:ACTED_IN {roles:['Dr. Jo Harding']}]->(Twister),
  (ZachG)-[:ACTED_IN {roles:['Eddie']}]->(Twister),
  (PhilipH)-[:ACTED_IN {roles:['Dustin "Dusty" Davis']}]->(Twister),
  (JanB)-[:DIRECTED]->(Twister)

CREATE (CastAway:Movie {title:'Cast Away', released:2000, tagline:'At the edge of the world, his journey begins.'})
CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (OneFlewOvertheCuckoosNest:Movie {title:"One Flew Over the Cuckoo's Nest", released:1975, tagline:"If he's crazy, what does that
make you?"})
CREATE (MilosF:Person {name:'Milos Forman', born:1932})
CREATE
  (JackN)-[:ACTED_IN {roles:['Randle McMurphy']}]->(OneFlewOvertheCuckoosNest),
  (DannyD)-[:ACTED_IN {roles:['Martini']}]->(OneFlewOvertheCuckoosNest),
  (MilosF)-[:DIRECTED]->(OneFlewOvertheCuckoosNest)

CREATE (SomethingsGottaGive:Movie {title:"Something's Gotta Give", released:2003})
CREATE (DianeK:Person {name:'Diane Keaton', born:1946})
CREATE (NancyM:Person {name:'Nancy Meyers', born:1949})
CREATE
```

```
  (JackN)-[:ACTED_IN {roles:['Harry Sanborn']}]->(SomethingsGottaGive),
  (DianeK)-[:ACTED_IN {roles:['Erica Barry']}]->(SomethingsGottaGive),
  (Keanu)-[:ACTED_IN {roles:['Julian Mercer']}]->(SomethingsGottaGive),
  (NancyM)-[:DIRECTED]->(SomethingsGottaGive),
  (NancyM)-[:PRODUCED]->(SomethingsGottaGive),
  (NancyM)-[:WROTE]->(SomethingsGottaGive)

CREATE (BicentennialMan:Movie {title:'Bicentennial Man', released:1999, tagline:"One robot's 200 year journey to become an ordinary man."})
CREATE (ChrisC:Person {name:'Chris Columbus', born:1958})
CREATE
  (Robin)-[:ACTED_IN {roles:['Andrew Marin']}]->(BicentennialMan),
  (OliverP)-[:ACTED_IN {roles:['Rupert Burns']}]->(BicentennialMan),
  (ChrisC)-[:DIRECTED]->(BicentennialMan)

CREATE (CharlieWilsonsWar:Movie {title:"Charlie Wilson's War", released:2007, tagline:"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire."})
CREATE (JuliaR:Person {name:'Julia Roberts', born:1967})
CREATE
  (TomH)-[:ACTED_IN {roles:['Rep. Charlie Wilson']}]->(CharlieWilsonsWar),
  (JuliaR)-[:ACTED_IN {roles:['Joanne Herring']}]->(CharlieWilsonsWar),
  (PhilipH)-[:ACTED_IN {roles:['Gust Avrakotos']}]->(CharlieWilsonsWar),
  (MikeN)-[:DIRECTED]->(CharlieWilsonsWar)

CREATE (ThePolarExpress:Movie {title:'The Polar Express', released:2004, tagline:'This Holiday Seasonâ€¦ Believe'})
CREATE
  (TomH)-[:ACTED_IN {roles:['Hero Boy', 'Father', 'Conductor', 'Hobo', 'Scrooge', 'Santa Claus']}]->(ThePolarExpress),
  (RobertZ)-[:DIRECTED]->(ThePolarExpress)

CREATE (ALeagueofTheirOwn:Movie {title:'A League of Their Own', released:1992, tagline:'Once in a lifetime you get a chance to do something different.'})
CREATE (Madonna:Person {name:'Madonna', born:1954})
CREATE (GeenaD:Person {name:'Geena Davis', born:1956})
CREATE (LoriP:Person {name:'Lori Petty', born:1963})
CREATE (PennyM:Person {name:'Penny Marshall', born:1943})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jimmy Dugan']}]->(ALeagueofTheirOwn),
  (GeenaD)-[:ACTED_IN {roles:['Dottie Hinson']}]->(ALeagueofTheirOwn),
  (LoriP)-[:ACTED_IN {roles:['Kit Keller']}]->(ALeagueofTheirOwn),
  (RosieO)-[:ACTED_IN {roles:['Doris Murphy']}]->(ALeagueofTheirOwn),
  (Madonna)-[:ACTED_IN {roles:['"All the Way" Mae Mordabito']}]->(ALeagueofTheirOwn),
  (BillPax)-[:ACTED_IN {roles:['Bob Hinson']}]->(ALeagueofTheirOwn),
  (PennyM)-[:DIRECTED]->(ALeagueofTheirOwn)

CREATE (PaulBlythe:Person {name:'Paul Blythe'})
CREATE (AngelaScope:Person {name:'Angela Scope'})
CREATE (JessicaThompson:Person {name:'Jessica Thompson'})
CREATE (JamesThompson:Person {name:'James Thompson'})

CREATE
  (JamesThompson)-[:FOLLOWS]->(JessicaThompson),
  (AngelaScope)-[:FOLLOWS]->(JessicaThompson),
  (PaulBlythe)-[:FOLLOWS]->(AngelaScope)

CREATE
  (JessicaThompson)-[:REVIEWED {summary:'An amazing journey', rating:95}]->(CloudAtlas),
  (JessicaThompson)-[:REVIEWED {summary:'Silly, but fun', rating:65}]->(TheReplacements),
  (JamesThompson)-[:REVIEWED {summary:'The coolest football movie ever', rating:100}]->(TheReplacements),
  (AngelaScope)-[:REVIEWED {summary:'Pretty funny at times', rating:62}]->(TheReplacements),
  (JessicaThompson)-[:REVIEWED {summary:'Dark, but compelling', rating:85}]->(Unforgiven),
  (JessicaThompson)-[:REVIEWED {summary:"Slapstick redeemed only by the Robin Williams and Gene Hackman's stellar performances", rating:45}]->(TheBirdcage),
  (JessicaThompson)-[:REVIEWED {summary:'A solid romp', rating:68}]->(TheDaVinciCode),
  (JamesThompson)-[:REVIEWED {summary:'Fun, but a little far fetched', rating:65}]->(TheDaVinciCode),
  (JessicaThompson)-[:REVIEWED {summary:'You had me at Jerry', rating:92}]->(JerryMaguire)
```

This is what you should see when you click the database icon  .



Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database.

Exercise 1: Traversing the graph (Overview)

In this exercise, you write some Cypher statements that help you understand traversal in Neo4j.

- **Exercise 1.1**: Determine the anchor and the number of paths traversed.
- **Exercise 1.2**: Determine the anchors and number of paths traversed.
- **Exercise 1.3:** Modify the previous query to use only one anchor.
- **Exercise 1.4:** Using WITH to chain clauses in a query.
- **Exercise 1.5:** Uniqueness of relationship path traversal.
- **Exercise 1.6:** Find the shortest path between two nodes.

Go to the next page to start this exercise.

Exercise 1.1: Determine the anchor and the number of paths traversed. (Instructions)

1. Execute this cypher code and then answer these questions:

```
MATCH (p:Person)-[:REVIEWED]->(m:Movie)<-[:REVIEWED]-(p2:Person)
WHERE p.name = 'Jessica Thompson'
RETURN p, p2, m
```

2. What is the anchor of this query?

3. How many paths are traversed during this query?

Exercise 1.1: Determine the anchor and the number of paths traversed. (Solution)

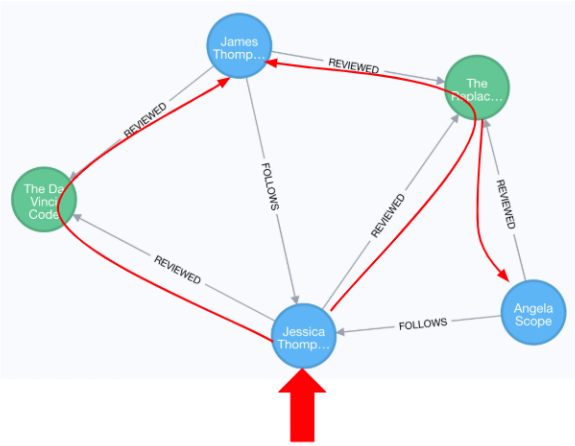
1. Execute this cypher code and then answer these questions:

2. What is the anchor of this query?

Answer: The Person node with the name property, Jessica Thompson.

3. How many paths are traversed during this query?

Answer: Three paths are traversed



Exercise 1.2: Determine the anchors and number of paths traversed. (Instructions)

1. Execute this cypher code and then answer these questions:

```
MATCH (a1:Person)-[:ACTED_IN]->(m:Movie)
MATCH (a2:Person)-[:ACTED_IN]->(m)
  WHERE a1.name = 'Rita Wilson'
RETURN m, a2
```

2. What are the anchors for each MATCH clause?

3. How many paths are traversed during this query?

Exercise 1.2: Determine the anchors and number of paths traversed. (Solution)

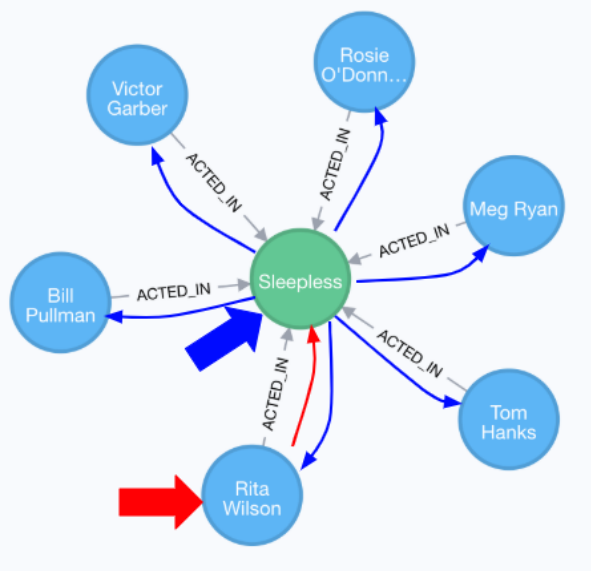
1. Execute this cypher code and then answer these questions:

2. What are the anchors for each MATCH clause?

Answer: The first MATCH uses the Person node with the name, Rita Wilson. The second MATCH uses the found movie node with the title, Sleepless in Seattle.

3. How many paths are traversed during this query?

Answer: One path is traversed for the first MATCH clause. Six paths are traversed for the second MATCH clause. Notice that the path from Rita Wilson to the movie Sleepless in Seattle is traversed twice.



Exercise 1.3: Modify the previous query to use only one anchor. (Instructions)

Modify the previous query use a single anchor and thus reduce the number of paths traversed.

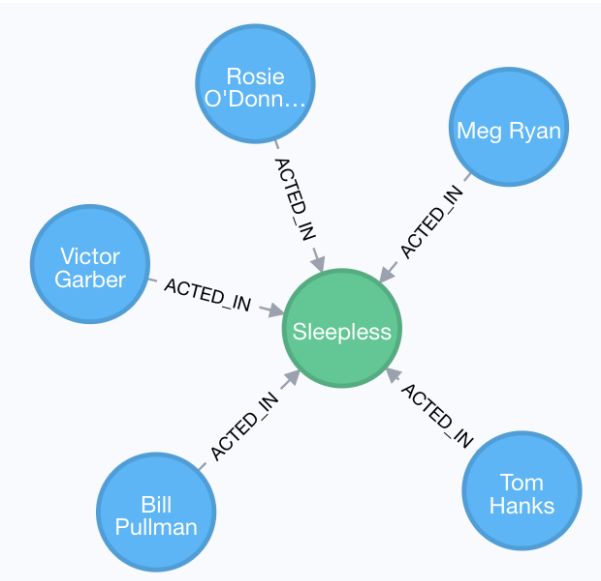
Exercise 1.3: Modify the previous query to use only one anchor. (Solution)

Modify the previous query to use a single anchor and thus reduce the number of paths traversed.

Here is the solution code:

```
MATCH (a1:Person)-[:ACTED_IN]->(m:Movie),
      (a2:Person)-[:ACTED_IN]->(m)
WHERE a1.name = 'Rita Wilson'
RETURN m, a2
```

Notice that for this query, the node, Rita Wilson is not visited twice.



Exercise 1.4: Using WITH to chain clauses in a query. (Instructions)

We want a query that finds all people who have acted in at least 5 movies. If this found person has also directed a movie, display the person's name and the title of the movie. This code is not correct.

```
MATCH (p:Person)
WITH  size((p)-[:ACTED_IN]->(:Movie)) AS movies
WHERE movies >= 5
OPTIONAL MATCH (p)-[:DIRECTED]->(m:Movie)
RETURN p.name, m.title
```

Correct this code so that it returns the correct results.

Exercise 1.4: Using WITH to chain clauses in a query. (Solution)

We want a query that finds all people who have acted in at least 5 movies. If this found person has also directed a movie, display the person's name and the title of the movie. This code is not correct.

```
MATCH (p:Person)
WITH  size((p)-[:ACTED_IN]->(:Movie)) AS movies
WHERE movies >= 5
OPTIONAL MATCH (p)-[:DIRECTED]->(m:Movie)
RETURN p.name, m.title
```

Correct this code so that it returns the correct results.

Here is the solution code. We also want to carry the p variable for the optional match. These actors acted in at least 5 movies, but only Tom Hanks also directed a movie.

```
MATCH (p:Person)
WITH  p, size((p)-[:ACTED_IN]->(:Movie)) AS movies
WHERE movies >= 5
OPTIONAL MATCH (p)-[:DIRECTED]->(m:Movie)
RETURN p.name, m.title
```

The result returned should be:

Exercise 1.5: Uniqueness of relationship path traversal. (Instructions)

Here is query to retrieve the movies and co-actors of Rita Wilson. This query does not return the Rita Wilson node as a coactor. This is because Neo4j does not traverse a relationship path more than once.

Modify this query so that the Rita Wilson node will also be returned.

```
MATCH (actor:Person {name:'Rita Wilson'})
      -[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(coActor:Person)
RETURN m.title, coActor.name
```

Exercise 1.5: Uniqueness of relationship path traversal. (Solution)

Here is query to retrieve the movies and co-actors of Rita Wilson. This query does not return the Rita Wilson node as a coactor. This is because Neo4j does not traverse a relationship path more than once.

Modify this query so that the Rita Wilson node will also be returned.

Here is one way that you can modify the query to traverse the `:ACTED_IN` relationship twice to return Rita Wilson in the set of coactors.

```
MATCH (actor:Person {name:'Rita Wilson'})-[:ACTED_IN]->(m:Movie)
MATCH  (m)<-[:ACTED_IN]-(coActor:Person)
RETURN m.title, coActor.name
```

The result returned should be:

Exercise 1.6: Find the shortest path between two nodes. (Instructions)

1. Write a query to return the shortest path between the actors Meg Ryan and Al Pacino.

2. PROFILE the query to see how the query executes.

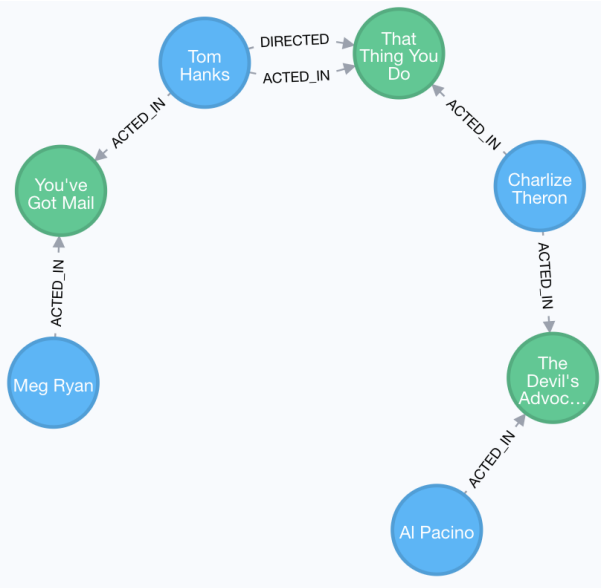
Exercise 1.6: Find the shortest path between two nodes. (Solution)

1. Write a query to return the shortest path between the actors Meg Ryan and Al Pacino.

Here is the solution code:

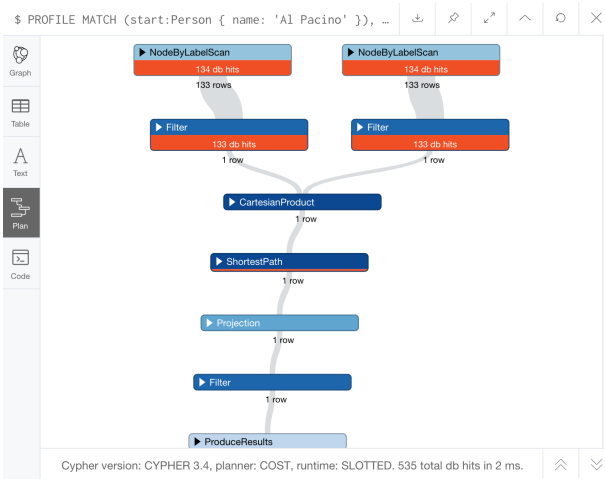
```
MATCH (start:Person { name: 'Al Pacino' }),
      (end:Person { name: 'Meg Ryan' }),
      p = shortestPath((start)-[*]-(end))
WITH p WHERE length(p) > 1
RETURN p
```

The result returned should be:



2. PROFILE the query to see how the query executes.

The PROFILE should be:



Notice that for this query a Cartesion product is specified. The Cartesion product, each with a single node is nothing to be concerned about. When Cartesian products become a problem is when each part of the query has thousands+ of nodes.

Exercise 1: Taking it further

1. Create other queries that chain clauses together with WITH.
2. Find the shortest path between two movie nodes.

Exercise 1: Implementing your first model (Summary)

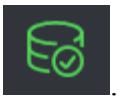
In this exercise, you performed a variety of queries to help you better understand traversals in Neo4j. You most important job as a developer will be to minimize traversals.

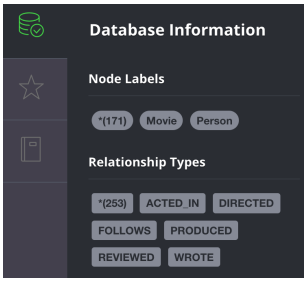
Continue to Exercise 2

Exercise 2

Exercise 2: Traversing the graph with APOC (Preparations)

The database you start with should contain the "movie" data that you loaded in the previous exercise.

This is what you should see when you click the database icon  .



1. If your database does not look like this, run the following script which removes all nodes and relationships from the graph and loads the graph with the "mini movie" data:

```
MATCH (n) DETACH DELETE n;
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixReloaded),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixReloaded),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixReloaded),
  (LillyW)-[:DIRECTED]->(TheMatrixReloaded),
  (LanaW)-[:DIRECTED]->(TheMatrixReloaded),
  (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions', released:2003, tagline:'Everything that has a beginning has an end'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixRevolutions),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixRevolutions),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixRevolutions),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixRevolutions),
  (LillyW)-[:DIRECTED]->(TheMatrixRevolutions),
  (LanaW)-[:DIRECTED]->(TheMatrixRevolutions),
  (JoelS)-[:PRODUCED]->(TheMatrixRevolutions)

CREATE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", released:1997, tagline:'Evil has its winning ways'})
CREATE (Charlize:Person {name:'Charlize Theron', born:1975})
CREATE (Al:Person {name:'Al Pacino', born:1940})
```

```
CREATE (Taylor:Person {name:'Taylor Hackford', born:1944})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Kevin Lomax']}]->(TheDevilsAdvocate),
  (Charlize)-[:ACTED_IN {roles:['Mary Ann Lomax']}]->(TheDevilsAdvocate),
  (Al)-[:ACTED_IN {roles:['John Milton']}]->(TheDevilsAdvocate),
  (Taylor)-[:DIRECTED]->(TheDevilsAdvocate)

CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tagline:"In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth."})
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
  (TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),
  (JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),
  (DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]->(AFewGoodMen),
  (KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),
  (KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),
  (NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),
  (CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),
  (KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),
  (JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]->(AFewGoodMen),
  (JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),
  (ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),
  (AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),
  (RobR)-[:DIRECTED]->(AFewGoodMen),
  (AaronS)-[:WROTE]->(AFewGoodMen)

CREATE (TopGun:Movie {title:"Top Gun", released:1986, tagline:'I feel the need, the need for speed.'})
CREATE (KellyM:Person {name:'Kelly McGillis', born:1957})
CREATE (ValK:Person {name:'Val Kilmer', born:1959})
CREATE (AnthonyE:Person {name:'Anthony Edwards', born:1962})
CREATE (TomS:Person {name:'Tom Skerritt', born:1933})
CREATE (MegR:Person {name:'Meg Ryan', born:1961})
CREATE (TonyS:Person {name:'Tony Scott', born:1944})
CREATE (JimC:Person {name:'Jim Cash', born:1941})
CREATE
  (TomC)-[:ACTED_IN {roles:['Maverick']}]->(TopGun),
  (KellyM)-[:ACTED_IN {roles:['Charlie']}]->(TopGun),
  (ValK)-[:ACTED_IN {roles:['Iceman']}]->(TopGun),
  (AnthonyE)-[:ACTED_IN {roles:['Goose']}]->(TopGun),
  (TomS)-[:ACTED_IN {roles:['Viper']}]->(TopGun),
  (MegR)-[:ACTED_IN {roles:['Carole']}]->(TopGun),
  (TonyS)-[:DIRECTED]->(TopGun),
  (JimC)-[:WROTE]->(TopGun)

CREATE (JerryMaguire:Movie {title:'Jerry Maguire', released:2000, tagline:'The rest of his life begins now.'})
CREATE (ReneeZ:Person {name:'Renee Zellweger', born:1969})
CREATE (KellyP:Person {name:'Kelly Preston', born:1962})
CREATE (JerryO:Person {name:"Jerry O'Connell", born:1974})
CREATE (JayM:Person {name:'Jay Mohr', born:1970})
CREATE (BonnieH:Person {name:'Bonnie Hunt', born:1961})
CREATE (ReginaK:Person {name:'Regina King', born:1971})
CREATE (JonathanL:Person {name:'Jonathan Lipnicki', born:1996})
CREATE (CameronC:Person {name:'Cameron Crowe', born:1957})
CREATE
  (TomC)-[:ACTED_IN {roles:['Jerry Maguire']}]->(JerryMaguire),
  (CubaG)-[:ACTED_IN {roles:['Rod Tidwell']}]->(JerryMaguire),
  (ReneeZ)-[:ACTED_IN {roles:['Dorothy Boyd']}]->(JerryMaguire),
  (KellyP)-[:ACTED_IN {roles:['Avery Bishop']}]->(JerryMaguire),
  (JerryO)-[:ACTED_IN {roles:['Frank Cushman']}]->(JerryMaguire),
  (JayM)-[:ACTED_IN {roles:['Bob Sugar']}]->(JerryMaguire),
  (BonnieH)-[:ACTED_IN {roles:['Laurel Boyd']}]->(JerryMaguire),
  (ReginaK)-[:ACTED_IN {roles:['Marcee Tidwell']}]->(JerryMaguire),
  (JonathanL)-[:ACTED_IN {roles:['Ray Boyd']}]->(JerryMaguire),
  (CameronC)-[:DIRECTED]->(JerryMaguire),
  (CameronC)-[:PRODUCED]->(JerryMaguire),
  (CameronC)-[:WROTE]->(JerryMaguire)

CREATE (StandByMe:Movie {title:"Stand By Me", released:1986, tagline:"For some, it's the last real taste of innocence, and the first real taste
```

```
of life. But for everyone, it's the time that memories are made of."})
CREATE (RiverP:Person {name:'River Phoenix', born:1970})
CREATE (CoreyF:Person {name:'Corey Feldman', born:1971})
CREATE (WilW:Person {name:'Wil Wheaton', born:1972})
CREATE (JohnC:Person {name:'John Cusack', born:1966})
CREATE (MarshallB:Person {name:'Marshall Bell', born:1942})
CREATE
  (WilW)-[:ACTED_IN {roles:['Gordie Lachance']}]->(StandByMe),
  (RiverP)-[:ACTED_IN {roles:['Chris Chambers']}]->(StandByMe),
  (JerryO)-[:ACTED_IN {roles:['Vern Tessio']}]->(StandByMe),
  (CoreyF)-[:ACTED_IN {roles:['Teddy Duchamp']}]->(StandByMe),
  (JohnC)-[:ACTED_IN {roles:['Denny Lachance']}]->(StandByMe),
  (KieferS)-[:ACTED_IN {roles:['Ace Merrill']}]->(StandByMe),
  (MarshallB)-[:ACTED_IN {roles:['Mr. Lachance']}]->(StandByMe),
  (RobR)-[:DIRECTED]->(StandByMe)

CREATE (AsGoodAsItGets:Movie {title:'As Good as It Gets', released:1997, tagline:'A comedy from the heart that goes for the throat.'})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})
CREATE (GregK:Person {name:'Greg Kinnear', born:1963})
CREATE (JamesB:Person {name:'James L. Brooks', born:1940})
CREATE
  (JackN)-[:ACTED_IN {roles:['Melvin Udall']}]->(AsGoodAsItGets),
  (HelenH)-[:ACTED_IN {roles:['Carol Connelly']}]->(AsGoodAsItGets),
  (GregK)-[:ACTED_IN {roles:['Simon Bishop']}]->(AsGoodAsItGets),
  (CubaG)-[:ACTED_IN {roles:['Frank Sachs']}]->(AsGoodAsItGets),
  (JamesB)-[:DIRECTED]->(AsGoodAsItGets)

CREATE (WhatDreamsMayCome:Movie {title:'What Dreams May Come', released:1998, tagline:'After life there is more. The end is just the
beginning.'})
CREATE (AnnabellaS:Person {name:'Annabella Sciorra', born:1960})
CREATE (MaxS:Person {name:'Max von Sydow', born:1929})
CREATE (WernerH:Person {name:'Werner Herzog', born:1942})
CREATE (Robin:Person {name:'Robin Williams', born:1951})
CREATE (VincentW:Person {name:'Vincent Ward', born:1956})
CREATE
  (Robin)-[:ACTED_IN {roles:['Chris Nielsen']}]->(WhatDreamsMayCome),
  (CubaG)-[:ACTED_IN {roles:['Albert Lewis']}]->(WhatDreamsMayCome),
  (AnnabellaS)-[:ACTED_IN {roles:['Annie Collins-Nielsen']}]->(WhatDreamsMayCome),
  (MaxS)-[:ACTED_IN {roles:['The Tracker']}]->(WhatDreamsMayCome),
  (WernerH)-[:ACTED_IN {roles:['The Face']}]->(WhatDreamsMayCome),
  (VincentW)-[:DIRECTED]->(WhatDreamsMayCome)

CREATE (SnowFallingonCedars:Movie {title:'Snow Falling on Cedars', released:1999, tagline:'First loves last. Forever.'})
CREATE (EthanH:Person {name:'Ethan Hawke', born:1970})
CREATE (RickY:Person {name:'Rick Yune', born:1971})
CREATE (JamesC:Person {name:'James Cromwell', born:1940})
CREATE (ScottH:Person {name:'Scott Hicks', born:1953})
CREATE
  (EthanH)-[:ACTED_IN {roles:['Ishmael Chambers']}]->(SnowFallingonCedars),
  (RickY)-[:ACTED_IN {roles:['Kazuo Miyamoto']}]->(SnowFallingonCedars),
  (MaxS)-[:ACTED_IN {roles:['Nels Gudmundsson']}]->(SnowFallingonCedars),
  (JamesC)-[:ACTED_IN {roles:['Judge Fielding']}]->(SnowFallingonCedars),
  (ScottH)-[:DIRECTED]->(SnowFallingonCedars)

CREATE (YouveGotMail:Movie {title:"You've Got Mail", released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (ParkerP:Person {name:'Parker Posey', born:1968})
CREATE (DaveC:Person {name:'Dave Chappelle', born:1973})
CREATE (SteveZ:Person {name:'Steve Zahn', born:1967})
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
  (MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]->(YouveGotMail),
  (GregK)-[:ACTED_IN {roles:['Frank Navasky']}]->(YouveGotMail),
  (ParkerP)-[:ACTED_IN {roles:['Patricia Eden']}]->(YouveGotMail),
  (DaveC)-[:ACTED_IN {roles:['Kevin Jackson']}]->(YouveGotMail),
  (SteveZ)-[:ACTED_IN {roles:['George Pappas']}]->(YouveGotMail),
  (NoraE)-[:DIRECTED]->(YouveGotMail)

CREATE (SleeplessInSeattle:Movie {title:'Sleepless in Seattle', released:1993, tagline:'What if someone you never met, someone you never saw,
someone you never knew was the only someone for you?'})
CREATE (RitaW:Person {name:'Rita Wilson', born:1956})
CREATE (BillPull:Person {name:'Bill Pullman', born:1953})
CREATE (VictorG:Person {name:'Victor Garber', born:1949})
CREATE (RosieO:Person {name:"Rosie O'Donnell", born:1962})
CREATE
  (TomH)-[:ACTED_IN {roles:['Sam Baldwin']}]->(SleeplessInSeattle),
  (MegR)-[:ACTED_IN {roles:['Annie Reed']}]->(SleeplessInSeattle),
  (RitaW)-[:ACTED_IN {roles:['Suzy']}]->(SleeplessInSeattle),
  (BillPull)-[:ACTED_IN {roles:['Walter']}]->(SleeplessInSeattle),
```

```
  (VictorG)-[:ACTED_IN {roles:['Greg']}]->(SleeplessInSeattle),
  (RosieO)-[:ACTED_IN {roles:['Becky']}]->(SleeplessInSeattle),
  (NoraE)-[:DIRECTED]->(SleeplessInSeattle)

CREATE (JoeVersustheVolcano:Movie {title:'Joe Versus the Volcano', released:1990, tagline:'A story of love, lava and burning desire.'})
CREATE (JohnS:Person {name:'John Patrick Stanley', born:1950})
CREATE (Nathan:Person {name:'Nathan Lane', born:1956})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Banks']}]->(JoeVersustheVolcano),
  (MegR)-[:ACTED_IN {roles:['DeDe', 'Angelica Graynamore', 'Patricia Graynamore']}]->(JoeVersustheVolcano),
  (Nathan)-[:ACTED_IN {roles:['Baw']}]->(JoeVersustheVolcano),
  (JohnS)-[:DIRECTED]->(JoeVersustheVolcano)

CREATE (WhenHarryMetSally:Movie {title:'When Harry Met Sally', released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (BillyC:Person {name:'Billy Crystal', born:1948})
CREATE (CarrieF:Person {name:'Carrie Fisher', born:1956})
CREATE (BrunoK:Person {name:'Bruno Kirby', born:1949})
CREATE
  (BillyC)-[:ACTED_IN {roles:['Harry Burns']}]->(WhenHarryMetSally),
  (MegR)-[:ACTED_IN {roles:['Sally Albright']}]->(WhenHarryMetSally),
  (CarrieF)-[:ACTED_IN {roles:['Marie']}]->(WhenHarryMetSally),
  (BrunoK)-[:ACTED_IN {roles:['Jess']}]->(WhenHarryMetSally),
  (RobR)-[:DIRECTED]->(WhenHarryMetSally),
  (RobR)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:WROTE]->(WhenHarryMetSally)

CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', released:1996, tagline:'In every life there comes a time when that thing you
dream becomes that thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
  (TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
  (LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
  (Charlize)-[:ACTED_IN {roles:['Tina']}]->(ThatThingYouDo),
  (TomH)-[:DIRECTED]->(ThatThingYouDo)

CREATE (TheReplacements:Movie {title:'The Replacements', released:2000, tagline:'Pain heals, Chicks dig scars... Glory lasts forever'})
CREATE (Brooke:Person {name:'Brooke Langton', born:1970})
CREATE (Gene:Person {name:'Gene Hackman', born:1930})
CREATE (Orlando:Person {name:'Orlando Jones', born:1968})
CREATE (Howard:Person {name:'Howard Deutch', born:1950})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Shane Falco']}]->(TheReplacements),
  (Brooke)-[:ACTED_IN {roles:['Annabelle Farrell']}]->(TheReplacements),
  (Gene)-[:ACTED_IN {roles:['Jimmy McGinty']}]->(TheReplacements),
  (Orlando)-[:ACTED_IN {roles:['Clifford Franklin']}]->(TheReplacements),
  (Howard)-[:DIRECTED]->(TheReplacements)

CREATE (RescueDawn:Movie {title:'RescueDawn', released:2006, tagline:"Based on the extraordinary true story of one man's fight for
freedom"})
CREATE (ChristianB:Person {name:'Christian Bale', born:1974})
CREATE (ZachG:Person {name:'Zach Grenier', born:1954})
CREATE
  (MarshallB)-[:ACTED_IN {roles:['Admiral']}]->(RescueDawn),
  (ChristianB)-[:ACTED_IN {roles:['Dieter Dengler']}]->(RescueDawn),
  (ZachG)-[:ACTED_IN {roles:['Squad Leader']}]->(RescueDawn),
  (SteveZ)-[:ACTED_IN {roles:['Duane']}]->(RescueDawn),
  (WernerH)-[:DIRECTED]->(RescueDawn)

CREATE (TheBirdcage:Movie {title:'The Birdcage', released:1996, tagline:'Come as you are'})
CREATE (MikeN:Person {name:'Mike Nichols', born:1931})
CREATE
  (Robin)-[:ACTED_IN {roles:['Armand Goldman']}]->(TheBirdcage),
  (Nathan)-[:ACTED_IN {roles:['Albert Goldman']}]->(TheBirdcage),
  (Gene)-[:ACTED_IN {roles:['Sen. Kevin Keeley']}]->(TheBirdcage),
  (MikeN)-[:DIRECTED]->(TheBirdcage)

CREATE (Unforgiven:Movie {title:'Unforgiven', released:1992, tagline:"It's a hell of a thing, killing a man"})
CREATE (RichardH:Person {name:'Richard Harris', born:1930})
CREATE (ClintE:Person {name:'Clint Eastwood', born:1930})
CREATE
  (RichardH)-[:ACTED_IN {roles:['English Bob']}]->(Unforgiven),
  (ClintE)-[:ACTED_IN {roles:['Bill Munny']}]->(Unforgiven),
  (Gene)-[:ACTED_IN {roles:['Little Bill Daggett']}]->(Unforgiven),
  (ClintE)-[:DIRECTED]->(Unforgiven)

CREATE (JohnnyMnemonic:Movie {title:'Johnny Mnemonic', released:1995, tagline:'The hottest data on earth. In the coolest head in town'})
CREATE (Takeshi:Person {name:'Takeshi Kitano', born:1947})
CREATE (Dina:Person {name:'Dina Meyer', born:1968})
CREATE (IceT:Person {name:'Ice-T', born:1958})
```

```
CREATE (RobertL:Person {name:'Robert Longo', born:1953})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Johnny Mnemonic']}]->(JohnnyMnemonic),
  (Takeshi)-[:ACTED_IN {roles:['Takahashi']}]->(JohnnyMnemonic),
  (Dina)-[:ACTED_IN {roles:['Jane']}]->(JohnnyMnemonic),
  (IceT)-[:ACTED_IN {roles:['J-Bone']}]->(JohnnyMnemonic),
  (RobertL)-[:DIRECTED]->(JohnnyMnemonic)

CREATE (CloudAtlas:Movie {title:'Cloud Atlas', released:2012, tagline:'Everything is connected'})
CREATE (HalleB:Person {name:'Halle Berry', born:1966})
CREATE (JimB:Person {name:'Jim Broadbent', born:1949})
CREATE (TomT:Person {name:'Tom Tykwer', born:1965})
CREATE (DavidMitchell:Person {name:'David Mitchell', born:1969})
CREATE (StefanArndt:Person {name:'Stefan Arndt', born:1961})
CREATE
  (TomH)-[:ACTED_IN {roles:['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']}]->(CloudAtlas),
  (Hugo)-[:ACTED_IN {roles:['Bill Smoke', 'Haskell Moore', 'Tadeusz Kesselring', 'Nurse Noakes', 'Boardman Mephi', 'Old Georgie']}]->
(CloudAtlas),
  (HalleB)-[:ACTED_IN {roles:['Luisa Rey', 'Jocasta Ayrs', 'Ovid', 'Meronym']}]->(CloudAtlas),
  (JimB)-[:ACTED_IN {roles:['Vyvyan Ayrs', 'Captain Molyneux', 'Timothy Cavendish']}]->(CloudAtlas),
  (TomT)-[:DIRECTED]->(CloudAtlas),
  (LillyW)-[:DIRECTED]->(CloudAtlas),
  (LanaW)-[:DIRECTED]->(CloudAtlas),
  (DavidMitchell)-[:WROTE]->(CloudAtlas),
  (StefanArndt)-[:PRODUCED]->(CloudAtlas)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006, tagline:'Break The Codes'})
CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)

CREATE (VforVendetta:Movie {title:'V for Vendetta', released:2006, tagline:'Freedom! Forever!'})
CREATE (NatalieP:Person {name:'Natalie Portman', born:1981})
CREATE (StephenR:Person {name:'Stephen Rea', born:1946})
CREATE (JohnH:Person {name:'John Hurt', born:1940})
CREATE (BenM:Person {name: 'Ben Miles', born:1967})
CREATE
  (Hugo)-[:ACTED_IN {roles:['V']}]->(VforVendetta),
  (NatalieP)-[:ACTED_IN {roles:['Evey Hammond']}]->(VforVendetta),
  (StephenR)-[:ACTED_IN {roles:['Eric Finch']}]->(VforVendetta),
  (JohnH)-[:ACTED_IN {roles:['High Chancellor Adam Sutler']}]->(VforVendetta),
  (BenM)-[:ACTED_IN {roles:['Dascomb']}]->(VforVendetta),
  (JamesM)-[:DIRECTED]->(VforVendetta),
  (LillyW)-[:PRODUCED]->(VforVendetta),
  (LanaW)-[:PRODUCED]->(VforVendetta),
  (JoelS)-[:PRODUCED]->(VforVendetta),
  (LillyW)-[:WROTE]->(VforVendetta),
  (LanaW)-[:WROTE]->(VforVendetta)

CREATE (SpeedRacer:Movie {title:'Speed Racer', released:2008, tagline:'Speed has no limits'})
CREATE (EmileH:Person {name:'Emile Hirsch', born:1985})
CREATE (JohnG:Person {name:'John Goodman', born:1960})
CREATE (SusanS:Person {name:'Susan Sarandon', born:1946})
CREATE (MatthewF:Person {name:'Matthew Fox', born:1966})
CREATE (ChristinaR:Person {name:'Christina Ricci', born:1980})
CREATE (Rain:Person {name:'Rain', born:1982})
CREATE
  (EmileH)-[:ACTED_IN {roles:['Speed Racer']}]->(SpeedRacer),
  (JohnG)-[:ACTED_IN {roles:['Pops']}]->(SpeedRacer),
  (SusanS)-[:ACTED_IN {roles:['Mom']}]->(SpeedRacer),
  (MatthewF)-[:ACTED_IN {roles:['Racer X']}]->(SpeedRacer),
  (ChristinaR)-[:ACTED_IN {roles:['Trixie']}]->(SpeedRacer),
  (Rain)-[:ACTED_IN {roles:['Taejo Togokahn']}]->(SpeedRacer),
  (BenM)-[:ACTED_IN {roles:['Cass Jones']}]->(SpeedRacer),
  (LillyW)-[:DIRECTED]->(SpeedRacer),
  (LanaW)-[:DIRECTED]->(SpeedRacer),
  (LillyW)-[:WROTE]->(SpeedRacer),
  (LanaW)-[:WROTE]->(SpeedRacer),
  (JoelS)-[:PRODUCED]->(SpeedRacer)

CREATE (NinjaAssassin:Movie {title:'Ninja Assassin', released:2009, tagline:'Prepare to enter a secret world of assassins'})
CREATE (NaomieH:Person {name:'Naomie Harris'})
CREATE
```

```
  (Rain)-[:ACTED_IN {roles:['Raizo']}]->(NinjaAssassin),
  (NaomieH)-[:ACTED_IN {roles:['Mika Coretti']}]->(NinjaAssassin),
  (RickY)-[:ACTED_IN {roles:['Takeshi']}]->(NinjaAssassin),
  (BenM)-[:ACTED_IN {roles:['Ryan Maslow']}]->(NinjaAssassin),
  (JamesM)-[:DIRECTED]->(NinjaAssassin),
  (LillyW)-[:PRODUCED]->(NinjaAssassin),
  (LanaW)-[:PRODUCED]->(NinjaAssassin),
  (JoelS)-[:PRODUCED]->(NinjaAssassin)

CREATE (TheGreenMile:Movie {title:'The Green Mile', released:1999, tagline:"Walk a mile you'll never forget."})
CREATE (MichaelD:Person {name:'Michael Clarke Duncan', born:1957})
CREATE (DavidM:Person {name:'David Morse', born:1953})
CREATE (SamR:Person {name:'Sam Rockwell', born:1968})
CREATE (GaryS:Person {name:'Gary Sinise', born:1955})
CREATE (PatriciaC:Person {name:'Patricia Clarkson', born:1959})
CREATE (FrankD:Person {name:'Frank Darabont', born:1959})
CREATE
  (TomH)-[:ACTED_IN {roles:['Paul Edgecomb']}]->(TheGreenMile),
  (MichaelD)-[:ACTED_IN {roles:['John Coffey']}]->(TheGreenMile),
  (DavidM)-[:ACTED_IN {roles:['Brutus "Brutal" Howell']}]->(TheGreenMile),
  (BonnieH)-[:ACTED_IN {roles:['Jan Edgecomb']}]->(TheGreenMile),
  (JamesC)-[:ACTED_IN {roles:['Warden Hal Moores']}]->(TheGreenMile),
  (SamR)-[:ACTED_IN {roles:['"Wild Bill" Wharton']}]->(TheGreenMile),
  (GaryS)-[:ACTED_IN {roles:['Burt Hammersmith']}]->(TheGreenMile),
  (PatriciaC)-[:ACTED_IN {roles:['Melinda Moores']}]->(TheGreenMile),
  (FrankD)-[:DIRECTED]->(TheGreenMile)

CREATE (FrostNixon:Movie {title:'Frost/Nixon', released:2008, tagline:'400 million people were waiting for the truth.'})
CREATE (FrankL:Person {name:'Frank Langella', born:1938})
CREATE (MichaelS:Person {name:'Michael Sheen', born:1969})
CREATE (OliverP:Person {name:'Oliver Platt', born:1960})
CREATE
  (FrankL)-[:ACTED_IN {roles:['Richard Nixon']}]->(FrostNixon),
  (MichaelS)-[:ACTED_IN {roles:['David Frost']}]->(FrostNixon),
  (KevinB)-[:ACTED_IN {roles:['Jack Brennan']}]->(FrostNixon),
  (OliverP)-[:ACTED_IN {roles:['Bob Zelnick']}]->(FrostNixon),
  (SamR)-[:ACTED_IN {roles:['James Reston, Jr.']}]->(FrostNixon),
  (RonH)-[:DIRECTED]->(FrostNixon)

CREATE (Hoffa:Movie {title:'Hoffa', released:1992, tagline:"He didn't want law. He wanted justice."})
CREATE (DannyD:Person {name:'Danny DeVito', born:1944})
CREATE (JohnR:Person {name:'John C. Reilly', born:1965})
CREATE
  (JackN)-[:ACTED_IN {roles:['Hoffa']}]->(Hoffa),
  (DannyD)-[:ACTED_IN {roles:['Robert "Bobby" Ciaro']}]->(Hoffa),
  (JTW)-[:ACTED_IN {roles:['Frank Fitzsimmons']}]->(Hoffa),
  (JohnR)-[:ACTED_IN {roles:['Peter "Pete" Connelly']}]->(Hoffa),
  (DannyD)-[:DIRECTED]->(Hoffa)

CREATE (Apollo13:Movie {title:'Apollo 13', released:1995, tagline:'Houston, we have a problem.'})
CREATE (EdH:Person {name:'Ed Harris', born:1950})
CREATE (BillPax:Person {name:'Bill Paxton', born:1955})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jim Lovell']}]->(Apollo13),
  (KevinB)-[:ACTED_IN {roles:['Jack Swigert']}]->(Apollo13),
  (EdH)-[:ACTED_IN {roles:['Gene Kranz']}]->(Apollo13),
  (BillPax)-[:ACTED_IN {roles:['Fred Haise']}]->(Apollo13),
  (GaryS)-[:ACTED_IN {roles:['Ken Mattingly']}]->(Apollo13),
  (RonH)-[:DIRECTED]->(Apollo13)

CREATE (Twister:Movie {title:'Twister', released:1996, tagline:"Don't Breathe. Don't Look Back."})
CREATE (PhilipH:Person {name:'Philip Seymour Hoffman', born:1967})
CREATE (JanB:Person {name:'Jan de Bont', born:1943})
CREATE
  (BillPax)-[:ACTED_IN {roles:['Bill Harding']}]->(Twister),
  (HelenH)-[:ACTED_IN {roles:['Dr. Jo Harding']}]->(Twister),
  (ZachG)-[:ACTED_IN {roles:['Eddie']}]->(Twister),
  (PhilipH)-[:ACTED_IN {roles:['Dustin "Dusty" Davis']}]->(Twister),
  (JanB)-[:DIRECTED]->(Twister)

CREATE (CastAway:Movie {title:'Cast Away', released:2000, tagline:'At the edge of the world, his journey begins.'})
CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (OneFlewOvertheCuckoosNest:Movie {title:"One Flew Over the Cuckoo's Nest", released:1975, tagline:"If he's crazy, what does that make you?"})
CREATE (MilosF:Person {name:'Milos Forman', born:1932})
```

```
CREATE
  (JackN)-[:ACTED_IN {roles:['Randle McMurphy']}]->(OneFlewOvertheCuckoosNest),
  (DannyD)-[:ACTED_IN {roles:['Martini']}]->(OneFlewOvertheCuckoosNest),
  (MilosF)-[:DIRECTED]->(OneFlewOvertheCuckoosNest)

CREATE (SomethingsGottaGive:Movie {title:"Something's Gotta Give", released:2003})
CREATE (DianeK:Person {name:'Diane Keaton', born:1946})
CREATE (NancyM:Person {name:'Nancy Meyers', born:1949})
CREATE
  (JackN)-[:ACTED_IN {roles:['Harry Sanborn']}]->(SomethingsGottaGive),
  (DianeK)-[:ACTED_IN {roles:['Erica Barry']}]->(SomethingsGottaGive),
  (Keanu)-[:ACTED_IN {roles:['Julian Mercer']}]->(SomethingsGottaGive),
  (NancyM)-[:DIRECTED]->(SomethingsGottaGive),
  (NancyM)-[:PRODUCED]->(SomethingsGottaGive),
  (NancyM)-[:WROTE]->(SomethingsGottaGive)

CREATE (BicentennialMan:Movie {title:'Bicentennial Man', released:1999, tagline:"One robot's 200 year journey to become an ordinary man."})
CREATE (ChrisC:Person {name:'Chris Columbus', born:1958})
CREATE
  (Robin)-[:ACTED_IN {roles:['Andrew Marin']}]->(BicentennialMan),
  (OliverP)-[:ACTED_IN {roles:['Rupert Burns']}]->(BicentennialMan),
  (ChrisC)-[:DIRECTED]->(BicentennialMan)

CREATE (CharlieWilsonsWar:Movie {title:"Charlie Wilson's War", released:2007, tagline:"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire."})
CREATE (JuliaR:Person {name:'Julia Roberts', born:1967})
CREATE
  (TomH)-[:ACTED_IN {roles:['Rep. Charlie Wilson']}]->(CharlieWilsonsWar),
  (JuliaR)-[:ACTED_IN {roles:['Joanne Herring']}]->(CharlieWilsonsWar),
  (PhilipH)-[:ACTED_IN {roles:['Gust Avrakotos']}]->(CharlieWilsonsWar),
  (MikeN)-[:DIRECTED]->(CharlieWilsonsWar)

CREATE (ThePolarExpress:Movie {title:'The Polar Express', released:2004, tagline:'This Holiday Seasonâ€¦ Believe'})
CREATE
  (TomH)-[:ACTED_IN {roles:['Hero Boy', 'Father', 'Conductor', 'Hobo', 'Scrooge', 'Santa Claus']}]->(ThePolarExpress),
  (RobertZ)-[:DIRECTED]->(ThePolarExpress)

CREATE (ALeagueofTheirOwn:Movie {title:'A League of Their Own', released:1992, tagline:'Once in a lifetime you get a chance to do something different.'})
CREATE (Madonna:Person {name:'Madonna', born:1954})
CREATE (GeenaD:Person {name:'Geena Davis', born:1956})
CREATE (LoriP:Person {name:'Lori Petty', born:1963})
CREATE (PennyM:Person {name:'Penny Marshall', born:1943})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jimmy Dugan']}]->(ALeagueofTheirOwn),
  (GeenaD)-[:ACTED_IN {roles:['Dottie Hinson']}]->(ALeagueofTheirOwn),
  (LoriP)-[:ACTED_IN {roles:['Kit Keller']}]->(ALeagueofTheirOwn),
  (RosieO)-[:ACTED_IN {roles:['Doris Murphy']}]->(ALeagueofTheirOwn),
  (Madonna)-[:ACTED_IN {roles:['"All the Way" Mae Mordabito']}]->(ALeagueofTheirOwn),
  (BillPax)-[:ACTED_IN {roles:['Bob Hinson']}]->(ALeagueofTheirOwn),
  (PennyM)-[:DIRECTED]->(ALeagueofTheirOwn)

CREATE (PaulBlythe:Person {name:'Paul Blythe'})
CREATE (AngelaScope:Person {name:'Angela Scope'})
CREATE (JessicaThompson:Person {name:'Jessica Thompson'})
CREATE (JamesThompson:Person {name:'James Thompson'})

CREATE
  (JamesThompson)-[:FOLLOWS]->(JessicaThompson),
  (AngelaScope)-[:FOLLOWS]->(JessicaThompson),
  (PaulBlythe)-[:FOLLOWS]->(AngelaScope)

CREATE
  (JessicaThompson)-[:REVIEWED {summary:'An amazing journey', rating:95}]->(CloudAtlas),
  (JessicaThompson)-[:REVIEWED {summary:'Silly, but fun', rating:65}]->(TheReplacements),
  (JamesThompson)-[:REVIEWED {summary:'The coolest football movie ever', rating:100}]->(TheReplacements),
  (AngelaScope)-[:REVIEWED {summary:'Pretty funny at times', rating:62}]->(TheReplacements),
  (JessicaThompson)-[:REVIEWED {summary:'Dark, but compelling', rating:85}]->(Unforgiven),
  (JessicaThompson)-[:REVIEWED {summary:"Slapstick redeemed only by the Robin Williams and Gene Hackman's stellar performances", rating:45}]->(TheBirdcage),
  (JessicaThompson)-[:REVIEWED {summary:'A solid romp', rating:68}]->(TheDaVinciCode),
  (JamesThompson)-[:REVIEWED {summary:'Fun, but a little far fetched', rating:65}]->(TheDaVinciCode),
  (JessicaThompson)-[:REVIEWED {summary:'You had me at Jerry', rating:92}]->(JerryMaguire)
```

2. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
```

```
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 2: Traversing the graph with APOC (Overview)

In this exercise, you write some Cypher statements that use APOC for graph traversal.

- **Exercise 2.1**: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher.
- **Exercise 2.2**: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher with APOC.
- **Exercise 2.3**: Modify a query to use APOC.

Go to the next page to start this exercise.

Exercise 2.1: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher. (Instructions)

Write the Cypher code to return all movies that are up to 3 hops away from the actor Hugo Weaving.

Exercise 2.1: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher. (Solution)

Write the Cypher code to return all movies that are up to 3 hops away from the actor Hugo Weaving.

Here is the solution code:

```
MATCH (p:Person {name: 'Hugo Weaving'})-[*3]-(m:Movie)
WITH DISTINCT m
RETURN  m.title
```

The results returned should look like this:

Exercise 2.2: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher with APOC. (Instructions)

Write the Cypher code that uses APOC to return all movies that are up to 3 hops away from the actor Hugo Weaving.

Exercise 2.2: Obtain movies that are up to 3 hops away from the actor Hugo Weaving using Cypher with APOC. (Solution)

Write the Cypher code that uses APOC to return all movies that are up to 3 hops away from the actor Hugo Weaving.

Here is the solution code:

```
MATCH (p:Person {name: 'Hugo Weaving'})
WITH p
CALL apoc.path.subgraphNodes(p, {labelFilter:'>Movie',maxLevel:3}) YIELD node AS movie
RETURN  movie.title
```

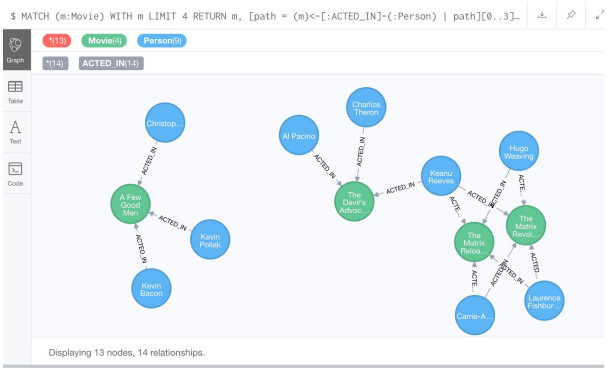
The results returned should look like this:

Exercise 2.3: Modify a query to use APOC. (Instructions)

Here is a Cypher query that retrieves four movies and for each movie, return the :ACTED_IN path to at most three actors.

```
MATCH (m:Movie) WITH m LIMIT 4
RETURN m, [path = (m)<-[:ACTED_IN]-(:Person) | path][0..3] as paths
```

1. Run this query. The results should be:



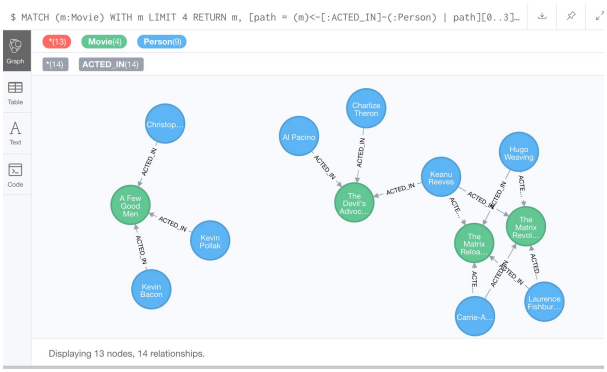
2. Rewrite this query to use APOC.

Exercise 2.3: Modify a query to use APOC. (Solution)

Here is a Cypher query that retrieves four movies and for each movie, return the :ACTED_IN path to at most three actors.

```
MATCH (m:Movie) WITH m LIMIT 4
RETURN m, [path = (m)<-[:ACTED_IN]-(:Person) | path][0..3] as paths
```

1. Run this query. The results should be:

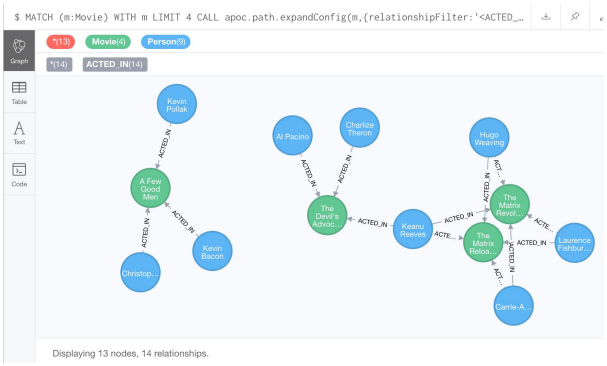


2. Rewrite this query to use APOC.

Here is the solution code:

```
MATCH (m:Movie) WITH m LIMIT 4
CALL apoc.path.expandConfig(m,{relationshipFilter:'<ACTED_IN',labelFilter:'/Person',limit:3,maxLevel:1}) YIELD path
RETURN path
```

The results should be:



Exercise 2: Taking it further

1. Explore the APOC documentation.
2. Write code to perform different types of graph traversal.

Exercise 2: Traversing the graph with APOC (Summary)

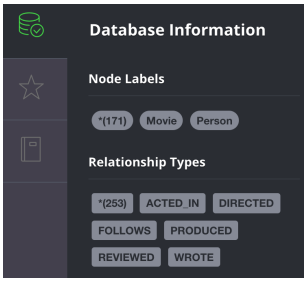
In this exercise, you performed and compared the execution of queries using Cypher for path traversal with the same queries using APOC procedures.

Continue to Exercise 3

Exercise 3: Working with UNWIND, pattern and list comprehension (Preparations)

The database you start with should contain the "movie" data that you loaded in the first exercise.

This is what you should see when you click the database icon  .



If your database does not look like this, run the following script which removes all nodes and relationships from the graph and loads the graph with the "mini movie" data:

```
MATCH (n) DETACH DELETE n;
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixReloaded),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixReloaded),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixReloaded),
  (LillyW)-[:DIRECTED]->(TheMatrixReloaded),
  (LanaW)-[:DIRECTED]->(TheMatrixReloaded),
  (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions', released:2003, tagline:'Everything that has a beginning has an end'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixRevolutions),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixRevolutions),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixRevolutions),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixRevolutions),
  (LillyW)-[:DIRECTED]->(TheMatrixRevolutions),
  (LanaW)-[:DIRECTED]->(TheMatrixRevolutions),
  (JoelS)-[:PRODUCED]->(TheMatrixRevolutions)

CREATE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", released:1997, tagline:'Evil has its winning ways'})
CREATE (Charlize:Person {name:'Charlize Theron', born:1975})
CREATE (Al:Person {name:'Al Pacino', born:1940})
```

```
CREATE (Taylor:Person {name:'Taylor Hackford', born:1944})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Kevin Lomax']}]->(TheDevilsAdvocate),
  (Charlize)-[:ACTED_IN {roles:['Mary Ann Lomax']}]->(TheDevilsAdvocate),
  (Al)-[:ACTED_IN {roles:['John Milton']}]->(TheDevilsAdvocate),
  (Taylor)-[:DIRECTED]->(TheDevilsAdvocate)

CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tagline:"In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth."})
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
  (TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),
  (JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),
  (DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]->(AFewGoodMen),
  (KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),
  (KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),
  (NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),
  (CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),
  (KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),
  (JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]->(AFewGoodMen),
  (JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),
  (ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),
  (AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),
  (RobR)-[:DIRECTED]->(AFewGoodMen),
  (AaronS)-[:WROTE]->(AFewGoodMen)

CREATE (TopGun:Movie {title:"Top Gun", released:1986, tagline:'I feel the need, the need for speed.'})
CREATE (KellyM:Person {name:'Kelly McGillis', born:1957})
CREATE (ValK:Person {name:'Val Kilmer', born:1959})
CREATE (AnthonyE:Person {name:'Anthony Edwards', born:1962})
CREATE (TomS:Person {name:'Tom Skerritt', born:1933})
CREATE (MegR:Person {name:'Meg Ryan', born:1961})
CREATE (TonyS:Person {name:'Tony Scott', born:1944})
CREATE (JimC:Person {name:'Jim Cash', born:1941})
CREATE
  (TomC)-[:ACTED_IN {roles:['Maverick']}]->(TopGun),
  (KellyM)-[:ACTED_IN {roles:['Charlie']}]->(TopGun),
  (ValK)-[:ACTED_IN {roles:['Iceman']}]->(TopGun),
  (AnthonyE)-[:ACTED_IN {roles:['Goose']}]->(TopGun),
  (TomS)-[:ACTED_IN {roles:['Viper']}]->(TopGun),
  (MegR)-[:ACTED_IN {roles:['Carole']}]->(TopGun),
  (TonyS)-[:DIRECTED]->(TopGun),
  (JimC)-[:WROTE]->(TopGun)

CREATE (JerryMaguire:Movie {title:'Jerry Maguire', released:2000, tagline:'The rest of his life begins now.'})
CREATE (ReneeZ:Person {name:'Renee Zellweger', born:1969})
CREATE (KellyP:Person {name:'Kelly Preston', born:1962})
CREATE (JerryO:Person {name:"Jerry O'Connell", born:1974})
CREATE (JayM:Person {name:'Jay Mohr', born:1970})
CREATE (BonnieH:Person {name:'Bonnie Hunt', born:1961})
CREATE (ReginaK:Person {name:'Regina King', born:1971})
CREATE (JonathanL:Person {name:'Jonathan Lipnicki', born:1996})
CREATE (CameronC:Person {name:'Cameron Crowe', born:1957})
CREATE
  (TomC)-[:ACTED_IN {roles:['Jerry Maguire']}]->(JerryMaguire),
  (CubaG)-[:ACTED_IN {roles:['Rod Tidwell']}]->(JerryMaguire),
  (ReneeZ)-[:ACTED_IN {roles:['Dorothy Boyd']}]->(JerryMaguire),
  (KellyP)-[:ACTED_IN {roles:['Avery Bishop']}]->(JerryMaguire),
  (JerryO)-[:ACTED_IN {roles:['Frank Cushman']}]->(JerryMaguire),
  (JayM)-[:ACTED_IN {roles:['Bob Sugar']}]->(JerryMaguire),
  (BonnieH)-[:ACTED_IN {roles:['Laurel Boyd']}]->(JerryMaguire),
  (ReginaK)-[:ACTED_IN {roles:['Marcee Tidwell']}]->(JerryMaguire),
  (JonathanL)-[:ACTED_IN {roles:['Ray Boyd']}]->(JerryMaguire),
  (CameronC)-[:DIRECTED]->(JerryMaguire),
  (CameronC)-[:PRODUCED]->(JerryMaguire),
  (CameronC)-[:WROTE]->(JerryMaguire)

CREATE (StandByMe:Movie {title:"Stand By Me", released:1986, tagline:"For some, it's the last real taste of innocence, and the first real taste
```

```
of life. But for everyone, it's the time that memories are made of."})
CREATE (RiverP:Person {name:'River Phoenix', born:1970})
CREATE (CoreyF:Person {name:'Corey Feldman', born:1971})
CREATE (WilW:Person {name:'Wil Wheaton', born:1972})
CREATE (JohnC:Person {name:'John Cusack', born:1966})
CREATE (MarshallB:Person {name:'Marshall Bell', born:1942})
CREATE
  (WilW)-[:ACTED_IN {roles:['Gordie Lachance']}]->(StandByMe),
  (RiverP)-[:ACTED_IN {roles:['Chris Chambers']}]->(StandByMe),
  (JerryO)-[:ACTED_IN {roles:['Vern Tessio']}]->(StandByMe),
  (CoreyF)-[:ACTED_IN {roles:['Teddy Duchamp']}]->(StandByMe),
  (JohnC)-[:ACTED_IN {roles:['Denny Lachance']}]->(StandByMe),
  (KieferS)-[:ACTED_IN {roles:['Ace Merrill']}]->(StandByMe),
  (MarshallB)-[:ACTED_IN {roles:['Mr. Lachance']}]->(StandByMe),
  (RobR)-[:DIRECTED]->(StandByMe)

CREATE (AsGoodAsItGets:Movie {title:'As Good as It Gets', released:1997, tagline:'A comedy from the heart that goes for the throat.'})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})
CREATE (GregK:Person {name:'Greg Kinnear', born:1963})
CREATE (JamesB:Person {name:'James L. Brooks', born:1940})
CREATE
  (JackN)-[:ACTED_IN {roles:['Melvin Udall']}]->(AsGoodAsItGets),
  (HelenH)-[:ACTED_IN {roles:['Carol Connelly']}]->(AsGoodAsItGets),
  (GregK)-[:ACTED_IN {roles:['Simon Bishop']}]->(AsGoodAsItGets),
  (CubaG)-[:ACTED_IN {roles:['Frank Sachs']}]->(AsGoodAsItGets),
  (JamesB)-[:DIRECTED]->(AsGoodAsItGets)

CREATE (WhatDreamsMayCome:Movie {title:'What Dreams May Come', released:1998, tagline:'After life there is more. The end is just the
beginning.'})
CREATE (AnnabellaS:Person {name:'Annabella Sciorra', born:1960})
CREATE (MaxS:Person {name:'Max von Sydow', born:1929})
CREATE (WernerH:Person {name:'Werner Herzog', born:1942})
CREATE (Robin:Person {name:'Robin Williams', born:1951})
CREATE (VincentW:Person {name:'Vincent Ward', born:1956})
CREATE
  (Robin)-[:ACTED_IN {roles:['Chris Nielsen']}]->(WhatDreamsMayCome),
  (CubaG)-[:ACTED_IN {roles:['Albert Lewis']}]->(WhatDreamsMayCome),
  (AnnabellaS)-[:ACTED_IN {roles:['Annie Collins-Nielsen']}]->(WhatDreamsMayCome),
  (MaxS)-[:ACTED_IN {roles:['The Tracker']}]->(WhatDreamsMayCome),
  (WernerH)-[:ACTED_IN {roles:['The Face']}]->(WhatDreamsMayCome),
  (VincentW)-[:DIRECTED]->(WhatDreamsMayCome)

CREATE (SnowFallingonCedars:Movie {title:'Snow Falling on Cedars', released:1999, tagline:'First loves last. Forever.'})
CREATE (EthanH:Person {name:'Ethan Hawke', born:1970})
CREATE (RickY:Person {name:'Rick Yune', born:1971})
CREATE (JamesC:Person {name:'James Cromwell', born:1940})
CREATE (ScottH:Person {name:'Scott Hicks', born:1953})
CREATE
  (EthanH)-[:ACTED_IN {roles:['Ishmael Chambers']}]->(SnowFallingonCedars),
  (RickY)-[:ACTED_IN {roles:['Kazuo Miyamoto']}]->(SnowFallingonCedars),
  (MaxS)-[:ACTED_IN {roles:['Nels Gudmundsson']}]->(SnowFallingonCedars),
  (JamesC)-[:ACTED_IN {roles:['Judge Fielding']}]->(SnowFallingonCedars),
  (ScottH)-[:DIRECTED]->(SnowFallingonCedars)

CREATE (YouveGotMail:Movie {title:"You've Got Mail", released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (ParkerP:Person {name:'Parker Posey', born:1968})
CREATE (DaveC:Person {name:'Dave Chappelle', born:1973})
CREATE (SteveZ:Person {name:'Steve Zahn', born:1967})
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
  (MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]->(YouveGotMail),
  (GregK)-[:ACTED_IN {roles:['Frank Navasky']}]->(YouveGotMail),
  (ParkerP)-[:ACTED_IN {roles:['Patricia Eden']}]->(YouveGotMail),
  (DaveC)-[:ACTED_IN {roles:['Kevin Jackson']}]->(YouveGotMail),
  (SteveZ)-[:ACTED_IN {roles:['George Pappas']}]->(YouveGotMail),
  (NoraE)-[:DIRECTED]->(YouveGotMail)

CREATE (SleeplessInSeattle:Movie {title:'Sleepless in Seattle', released:1993, tagline:'What if someone you never met, someone you never saw,
someone you never knew was the only someone for you?'})
CREATE (RitaW:Person {name:'Rita Wilson', born:1956})
CREATE (BillPull:Person {name:'Bill Pullman', born:1953})
CREATE (VictorG:Person {name:'Victor Garber', born:1949})
CREATE (RosieO:Person {name:"Rosie O'Donnell", born:1962})
CREATE
  (TomH)-[:ACTED_IN {roles:['Sam Baldwin']}]->(SleeplessInSeattle),
  (MegR)-[:ACTED_IN {roles:['Annie Reed']}]->(SleeplessInSeattle),
  (RitaW)-[:ACTED_IN {roles:['Suzy']}]->(SleeplessInSeattle),
  (BillPull)-[:ACTED_IN {roles:['Walter']}]->(SleeplessInSeattle),
```

```
  (VictorG)-[:ACTED_IN {roles:['Greg']}]->(SleeplessInSeattle),
  (RosieO)-[:ACTED_IN {roles:['Becky']}]->(SleeplessInSeattle),
  (NoraE)-[:DIRECTED]->(SleeplessInSeattle)

CREATE (JoeVersustheVolcano:Movie {title:'Joe Versus the Volcano', released:1990, tagline:'A story of love, lava and burning desire.'})
CREATE (JohnS:Person {name:'John Patrick Stanley', born:1950})
CREATE (Nathan:Person {name:'Nathan Lane', born:1956})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Banks']}]->(JoeVersustheVolcano),
  (MegR)-[:ACTED_IN {roles:['DeDe', 'Angelica Graynamore', 'Patricia Graynamore']}]->(JoeVersustheVolcano),
  (Nathan)-[:ACTED_IN {roles:['Baw']}]->(JoeVersustheVolcano),
  (JohnS)-[:DIRECTED]->(JoeVersustheVolcano)

CREATE (WhenHarryMetSally:Movie {title:'When Harry Met Sally', released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (BillyC:Person {name:'Billy Crystal', born:1948})
CREATE (CarrieF:Person {name:'Carrie Fisher', born:1956})
CREATE (BrunoK:Person {name:'Bruno Kirby', born:1949})
CREATE
  (BillyC)-[:ACTED_IN {roles:['Harry Burns']}]->(WhenHarryMetSally),
  (MegR)-[:ACTED_IN {roles:['Sally Albright']}]->(WhenHarryMetSally),
  (CarrieF)-[:ACTED_IN {roles:['Marie']}]->(WhenHarryMetSally),
  (BrunoK)-[:ACTED_IN {roles:['Jess']}]->(WhenHarryMetSally),
  (RobR)-[:DIRECTED]->(WhenHarryMetSally),
  (RobR)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:WROTE]->(WhenHarryMetSally)

CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', released:1996, tagline:'In every life there comes a time when that thing you
dream becomes that thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
  (TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
  (LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
  (Charlize)-[:ACTED_IN {roles:['Tina']}]->(ThatThingYouDo),
  (TomH)-[:DIRECTED]->(ThatThingYouDo)

CREATE (TheReplacements:Movie {title:'The Replacements', released:2000, tagline:'Pain heals, Chicks dig scars... Glory lasts forever'})
CREATE (Brooke:Person {name:'Brooke Langton', born:1970})
CREATE (Gene:Person {name:'Gene Hackman', born:1930})
CREATE (Orlando:Person {name:'Orlando Jones', born:1968})
CREATE (Howard:Person {name:'Howard Deutch', born:1950})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Shane Falco']}]->(TheReplacements),
  (Brooke)-[:ACTED_IN {roles:['Annabelle Farrell']}]->(TheReplacements),
  (Gene)-[:ACTED_IN {roles:['Jimmy McGinty']}]->(TheReplacements),
  (Orlando)-[:ACTED_IN {roles:['Clifford Franklin']}]->(TheReplacements),
  (Howard)-[:DIRECTED]->(TheReplacements)

CREATE (RescueDawn:Movie {title:'RescueDawn', released:2006, tagline:"Based on the extraordinary true story of one man's fight for
freedom"})
CREATE (ChristianB:Person {name:'Christian Bale', born:1974})
CREATE (ZachG:Person {name:'Zach Grenier', born:1954})
CREATE
  (MarshallB)-[:ACTED_IN {roles:['Admiral']}]->(RescueDawn),
  (ChristianB)-[:ACTED_IN {roles:['Dieter Dengler']}]->(RescueDawn),
  (ZachG)-[:ACTED_IN {roles:['Squad Leader']}]->(RescueDawn),
  (SteveZ)-[:ACTED_IN {roles:['Duane']}]->(RescueDawn),
  (WernerH)-[:DIRECTED]->(RescueDawn)

CREATE (TheBirdcage:Movie {title:'The Birdcage', released:1996, tagline:'Come as you are'})
CREATE (MikeN:Person {name:'Mike Nichols', born:1931})
CREATE
  (Robin)-[:ACTED_IN {roles:['Armand Goldman']}]->(TheBirdcage),
  (Nathan)-[:ACTED_IN {roles:['Albert Goldman']}]->(TheBirdcage),
  (Gene)-[:ACTED_IN {roles:['Sen. Kevin Keeley']}]->(TheBirdcage),
  (MikeN)-[:DIRECTED]->(TheBirdcage)

CREATE (Unforgiven:Movie {title:'Unforgiven', released:1992, tagline:"It's a hell of a thing, killing a man"})
CREATE (RichardH:Person {name:'Richard Harris', born:1930})
CREATE (ClintE:Person {name:'Clint Eastwood', born:1930})
CREATE
  (RichardH)-[:ACTED_IN {roles:['English Bob']}]->(Unforgiven),
  (ClintE)-[:ACTED_IN {roles:['Bill Munny']}]->(Unforgiven),
  (Gene)-[:ACTED_IN {roles:['Little Bill Daggett']}]->(Unforgiven),
  (ClintE)-[:DIRECTED]->(Unforgiven)

CREATE (JohnnyMnemonic:Movie {title:'Johnny Mnemonic', released:1995, tagline:'The hottest data on earth. In the coolest head in town'})
CREATE (Takeshi:Person {name:'Takeshi Kitano', born:1947})
CREATE (Dina:Person {name:'Dina Meyer', born:1968})
CREATE (IceT:Person {name:'Ice-T', born:1958})
```

```
CREATE (RobertL:Person {name:'Robert Longo', born:1953})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Johnny Mnemonic']}]->(JohnnyMnemonic),
  (Takeshi)-[:ACTED_IN {roles:['Takahashi']}]->(JohnnyMnemonic),
  (Dina)-[:ACTED_IN {roles:['Jane']}]->(JohnnyMnemonic),
  (IceT)-[:ACTED_IN {roles:['J-Bone']}]->(JohnnyMnemonic),
  (RobertL)-[:DIRECTED]->(JohnnyMnemonic)

CREATE (CloudAtlas:Movie {title:'Cloud Atlas', released:2012, tagline:'Everything is connected'})
CREATE (HalleB:Person {name:'Halle Berry', born:1966})
CREATE (JimB:Person {name:'Jim Broadbent', born:1949})
CREATE (TomT:Person {name:'Tom Tykwer', born:1965})
CREATE (DavidMitchell:Person {name:'David Mitchell', born:1969})
CREATE (StefanArndt:Person {name:'Stefan Arndt', born:1961})
CREATE
  (TomH)-[:ACTED_IN {roles:['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']}]->(CloudAtlas),
  (Hugo)-[:ACTED_IN {roles:['Bill Smoke', 'Haskell Moore', 'Tadeusz Kesselring', 'Nurse Noakes', 'Boardman Mephi', 'Old Georgie']}]->
(CloudAtlas),
  (HalleB)-[:ACTED_IN {roles:['Luisa Rey', 'Jocasta Ayrs', 'Ovid', 'Meronym']}]->(CloudAtlas),
  (JimB)-[:ACTED_IN {roles:['Vyvyan Ayrs', 'Captain Molyneux', 'Timothy Cavendish']}]->(CloudAtlas),
  (TomT)-[:DIRECTED]->(CloudAtlas),
  (LillyW)-[:DIRECTED]->(CloudAtlas),
  (LanaW)-[:DIRECTED]->(CloudAtlas),
  (DavidMitchell)-[:WROTE]->(CloudAtlas),
  (StefanArndt)-[:PRODUCED]->(CloudAtlas)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006, tagline:'Break The Codes'})
CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)

CREATE (VforVendetta:Movie {title:'V for Vendetta', released:2006, tagline:'Freedom! Forever!'})
CREATE (NatalieP:Person {name:'Natalie Portman', born:1981})
CREATE (StephenR:Person {name:'Stephen Rea', born:1946})
CREATE (JohnH:Person {name:'John Hurt', born:1940})
CREATE (BenM:Person {name: 'Ben Miles', born:1967})
CREATE
  (Hugo)-[:ACTED_IN {roles:['V']}]->(VforVendetta),
  (NatalieP)-[:ACTED_IN {roles:['Evey Hammond']}]->(VforVendetta),
  (StephenR)-[:ACTED_IN {roles:['Eric Finch']}]->(VforVendetta),
  (JohnH)-[:ACTED_IN {roles:['High Chancellor Adam Sutler']}]->(VforVendetta),
  (BenM)-[:ACTED_IN {roles:['Dascomb']}]->(VforVendetta),
  (JamesM)-[:DIRECTED]->(VforVendetta),
  (LillyW)-[:PRODUCED]->(VforVendetta),
  (LanaW)-[:PRODUCED]->(VforVendetta),
  (JoelS)-[:PRODUCED]->(VforVendetta),
  (LillyW)-[:WROTE]->(VforVendetta),
  (LanaW)-[:WROTE]->(VforVendetta)

CREATE (SpeedRacer:Movie {title:'Speed Racer', released:2008, tagline:'Speed has no limits'})
CREATE (EmileH:Person {name:'Emile Hirsch', born:1985})
CREATE (JohnG:Person {name:'John Goodman', born:1960})
CREATE (SusanS:Person {name:'Susan Sarandon', born:1946})
CREATE (MatthewF:Person {name:'Matthew Fox', born:1966})
CREATE (ChristinaR:Person {name:'Christina Ricci', born:1980})
CREATE (Rain:Person {name:'Rain', born:1982})
CREATE
  (EmileH)-[:ACTED_IN {roles:['Speed Racer']}]->(SpeedRacer),
  (JohnG)-[:ACTED_IN {roles:['Pops']}]->(SpeedRacer),
  (SusanS)-[:ACTED_IN {roles:['Mom']}]->(SpeedRacer),
  (MatthewF)-[:ACTED_IN {roles:['Racer X']}]->(SpeedRacer),
  (ChristinaR)-[:ACTED_IN {roles:['Trixie']}]->(SpeedRacer),
  (Rain)-[:ACTED_IN {roles:['Taejo Togokahn']}]->(SpeedRacer),
  (BenM)-[:ACTED_IN {roles:['Cass Jones']}]->(SpeedRacer),
  (LillyW)-[:DIRECTED]->(SpeedRacer),
  (LanaW)-[:DIRECTED]->(SpeedRacer),
  (LillyW)-[:WROTE]->(SpeedRacer),
  (LanaW)-[:WROTE]->(SpeedRacer),
  (JoelS)-[:PRODUCED]->(SpeedRacer)

CREATE (NinjaAssassin:Movie {title:'Ninja Assassin', released:2009, tagline:'Prepare to enter a secret world of assassins'})
CREATE (NaomieH:Person {name:'Naomie Harris'})
CREATE
```

```
  (Rain)-[:ACTED_IN {roles:['Raizo']}]->(NinjaAssassin),
  (NaomieH)-[:ACTED_IN {roles:['Mika Coretti']}]->(NinjaAssassin),
  (RickY)-[:ACTED_IN {roles:['Takeshi']}]->(NinjaAssassin),
  (BenM)-[:ACTED_IN {roles:['Ryan Maslow']}]->(NinjaAssassin),
  (JamesM)-[:DIRECTED]->(NinjaAssassin),
  (LillyW)-[:PRODUCED]->(NinjaAssassin),
  (LanaW)-[:PRODUCED]->(NinjaAssassin),
  (JoelS)-[:PRODUCED]->(NinjaAssassin)

CREATE (TheGreenMile:Movie {title:'The Green Mile', released:1999, tagline:"Walk a mile you'll never forget."})
CREATE (MichaelD:Person {name:'Michael Clarke Duncan', born:1957})
CREATE (DavidM:Person {name:'David Morse', born:1953})
CREATE (SamR:Person {name:'Sam Rockwell', born:1968})
CREATE (GaryS:Person {name:'Gary Sinise', born:1955})
CREATE (PatriciaC:Person {name:'Patricia Clarkson', born:1959})
CREATE (FrankD:Person {name:'Frank Darabont', born:1959})
CREATE
  (TomH)-[:ACTED_IN {roles:['Paul Edgecomb']}]->(TheGreenMile),
  (MichaelD)-[:ACTED_IN {roles:['John Coffey']}]->(TheGreenMile),
  (DavidM)-[:ACTED_IN {roles:['Brutus "Brutal" Howell']}]->(TheGreenMile),
  (BonnieH)-[:ACTED_IN {roles:['Jan Edgecomb']}]->(TheGreenMile),
  (JamesC)-[:ACTED_IN {roles:['Warden Hal Moores']}]->(TheGreenMile),
  (SamR)-[:ACTED_IN {roles:['"Wild Bill" Wharton']}]->(TheGreenMile),
  (GaryS)-[:ACTED_IN {roles:['Burt Hammersmith']}]->(TheGreenMile),
  (PatriciaC)-[:ACTED_IN {roles:['Melinda Moores']}]->(TheGreenMile),
  (FrankD)-[:DIRECTED]->(TheGreenMile)

CREATE (FrostNixon:Movie {title:'Frost/Nixon', released:2008, tagline:'400 million people were waiting for the truth.'})
CREATE (FrankL:Person {name:'Frank Langella', born:1938})
CREATE (MichaelS:Person {name:'Michael Sheen', born:1969})
CREATE (OliverP:Person {name:'Oliver Platt', born:1960})
CREATE
  (FrankL)-[:ACTED_IN {roles:['Richard Nixon']}]->(FrostNixon),
  (MichaelS)-[:ACTED_IN {roles:['David Frost']}]->(FrostNixon),
  (KevinB)-[:ACTED_IN {roles:['Jack Brennan']}]->(FrostNixon),
  (OliverP)-[:ACTED_IN {roles:['Bob Zelnick']}]->(FrostNixon),
  (SamR)-[:ACTED_IN {roles:['James Reston, Jr.']}]->(FrostNixon),
  (RonH)-[:DIRECTED]->(FrostNixon)

CREATE (Hoffa:Movie {title:'Hoffa', released:1992, tagline:"He didn't want law. He wanted justice."})
CREATE (DannyD:Person {name:'Danny DeVito', born:1944})
CREATE (JohnR:Person {name:'John C. Reilly', born:1965})
CREATE
  (JackN)-[:ACTED_IN {roles:['Hoffa']}]->(Hoffa),
  (DannyD)-[:ACTED_IN {roles:['Robert "Bobby" Ciaro']}]->(Hoffa),
  (JTW)-[:ACTED_IN {roles:['Frank Fitzsimmons']}]->(Hoffa),
  (JohnR)-[:ACTED_IN {roles:['Peter "Pete" Connelly']}]->(Hoffa),
  (DannyD)-[:DIRECTED]->(Hoffa)

CREATE (Apollo13:Movie {title:'Apollo 13', released:1995, tagline:'Houston, we have a problem.'})
CREATE (EdH:Person {name:'Ed Harris', born:1950})
CREATE (BillPax:Person {name:'Bill Paxton', born:1955})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jim Lovell']}]->(Apollo13),
  (KevinB)-[:ACTED_IN {roles:['Jack Swigert']}]->(Apollo13),
  (EdH)-[:ACTED_IN {roles:['Gene Kranz']}]->(Apollo13),
  (BillPax)-[:ACTED_IN {roles:['Fred Haise']}]->(Apollo13),
  (GaryS)-[:ACTED_IN {roles:['Ken Mattingly']}]->(Apollo13),
  (RonH)-[:DIRECTED]->(Apollo13)

CREATE (Twister:Movie {title:'Twister', released:1996, tagline:"Don't Breathe. Don't Look Back."})
CREATE (PhilipH:Person {name:'Philip Seymour Hoffman', born:1967})
CREATE (JanB:Person {name:'Jan de Bont', born:1943})
CREATE
  (BillPax)-[:ACTED_IN {roles:['Bill Harding']}]->(Twister),
  (HelenH)-[:ACTED_IN {roles:['Dr. Jo Harding']}]->(Twister),
  (ZachG)-[:ACTED_IN {roles:['Eddie']}]->(Twister),
  (PhilipH)-[:ACTED_IN {roles:['Dustin "Dusty" Davis']}]->(Twister),
  (JanB)-[:DIRECTED]->(Twister)

CREATE (CastAway:Movie {title:'Cast Away', released:2000, tagline:'At the edge of the world, his journey begins.'})
CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (OneFlewOvertheCuckoosNest:Movie {title:"One Flew Over the Cuckoo's Nest", released:1975, tagline:"If he's crazy, what does that make you?"})
CREATE (MilosF:Person {name:'Milos Forman', born:1932})
```

```
CREATE
  (JackN)-[:ACTED_IN {roles:['Randle McMurphy']}]->(OneFlewOvertheCuckoosNest),
  (DannyD)-[:ACTED_IN {roles:['Martini']}]->(OneFlewOvertheCuckoosNest),
  (MilosF)-[:DIRECTED]->(OneFlewOvertheCuckoosNest)

CREATE (SomethingsGottaGive:Movie {title:"Something's Gotta Give", released:2003})
CREATE (DianeK:Person {name:'Diane Keaton', born:1946})
CREATE (NancyM:Person {name:'Nancy Meyers', born:1949})
CREATE
  (JackN)-[:ACTED_IN {roles:['Harry Sanborn']}]->(SomethingsGottaGive),
  (DianeK)-[:ACTED_IN {roles:['Erica Barry']}]->(SomethingsGottaGive),
  (Keanu)-[:ACTED_IN {roles:['Julian Mercer']}]->(SomethingsGottaGive),
  (NancyM)-[:DIRECTED]->(SomethingsGottaGive),
  (NancyM)-[:PRODUCED]->(SomethingsGottaGive),
  (NancyM)-[:WROTE]->(SomethingsGottaGive)

CREATE (BicentennialMan:Movie {title:'Bicentennial Man', released:1999, tagline:"One robot's 200 year journey to become an ordinary man."})
CREATE (ChrisC:Person {name:'Chris Columbus', born:1958})
CREATE
  (Robin)-[:ACTED_IN {roles:['Andrew Marin']}]->(BicentennialMan),
  (OliverP)-[:ACTED_IN {roles:['Rupert Burns']}]->(BicentennialMan),
  (ChrisC)-[:DIRECTED]->(BicentennialMan)

CREATE (CharlieWilsonsWar:Movie {title:"Charlie Wilson's War", released:2007, tagline:"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire."})
CREATE (JuliaR:Person {name:'Julia Roberts', born:1967})
CREATE
  (TomH)-[:ACTED_IN {roles:['Rep. Charlie Wilson']}]->(CharlieWilsonsWar),
  (JuliaR)-[:ACTED_IN {roles:['Joanne Herring']}]->(CharlieWilsonsWar),
  (PhilipH)-[:ACTED_IN {roles:['Gust Avrakotos']}]->(CharlieWilsonsWar),
  (MikeN)-[:DIRECTED]->(CharlieWilsonsWar)

CREATE (ThePolarExpress:Movie {title:'The Polar Express', released:2004, tagline:'This Holiday Seasonâ€¦ Believe'})
CREATE
  (TomH)-[:ACTED_IN {roles:['Hero Boy', 'Father', 'Conductor', 'Hobo', 'Scrooge', 'Santa Claus']}]->(ThePolarExpress),
  (RobertZ)-[:DIRECTED]->(ThePolarExpress)

CREATE (ALeagueofTheirOwn:Movie {title:'A League of Their Own', released:1992, tagline:'Once in a lifetime you get a chance to do something different.'})
CREATE (Madonna:Person {name:'Madonna', born:1954})
CREATE (GeenaD:Person {name:'Geena Davis', born:1956})
CREATE (LoriP:Person {name:'Lori Petty', born:1963})
CREATE (PennyM:Person {name:'Penny Marshall', born:1943})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jimmy Dugan']}]->(ALeagueofTheirOwn),
  (GeenaD)-[:ACTED_IN {roles:['Dottie Hinson']}]->(ALeagueofTheirOwn),
  (LoriP)-[:ACTED_IN {roles:['Kit Keller']}]->(ALeagueofTheirOwn),
  (RosieO)-[:ACTED_IN {roles:['Doris Murphy']}]->(ALeagueofTheirOwn),
  (Madonna)-[:ACTED_IN {roles:['"All the Way" Mae Mordabito']}]->(ALeagueofTheirOwn),
  (BillPax)-[:ACTED_IN {roles:['Bob Hinson']}]->(ALeagueofTheirOwn),
  (PennyM)-[:DIRECTED]->(ALeagueofTheirOwn)

CREATE (PaulBlythe:Person {name:'Paul Blythe'})
CREATE (AngelaScope:Person {name:'Angela Scope'})
CREATE (JessicaThompson:Person {name:'Jessica Thompson'})
CREATE (JamesThompson:Person {name:'James Thompson'})

CREATE
  (JamesThompson)-[:FOLLOWS]->(JessicaThompson),
  (AngelaScope)-[:FOLLOWS]->(JessicaThompson),
  (PaulBlythe)-[:FOLLOWS]->(AngelaScope)

CREATE
  (JessicaThompson)-[:REVIEWED {summary:'An amazing journey', rating:95}]->(CloudAtlas),
  (JessicaThompson)-[:REVIEWED {summary:'Silly, but fun', rating:65}]->(TheReplacements),
  (JamesThompson)-[:REVIEWED {summary:'The coolest football movie ever', rating:100}]->(TheReplacements),
  (AngelaScope)-[:REVIEWED {summary:'Pretty funny at times', rating:62}]->(TheReplacements),
  (JessicaThompson)-[:REVIEWED {summary:'Dark, but compelling', rating:85}]->(Unforgiven),
  (JessicaThompson)-[:REVIEWED {summary:"Slapstick redeemed only by the Robin Williams and Gene Hackman's stellar performances", rating:45}]->(TheBirdcage),
  (JessicaThompson)-[:REVIEWED {summary:'A solid romp', rating:68}]->(TheDaVinciCode),
  (JamesThompson)-[:REVIEWED {summary:'Fun, but a little far fetched', rating:65}]->(TheDaVinciCode),
  (JessicaThompson)-[:REVIEWED {summary:'You had me at Jerry', rating:92}]->(JerryMaguire)
```

Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 3: Working with UNWIND, pattern and list comprehension (Overview)

In this exercise, you write some Cypher statements that utilize UNWIND and pattern/list comprehensions to extract data:

- **Exercise 3.1**: Write a query to return actors as a list.
- **Exercise 3.2**: Use UNWIND to process the list of actors.
- **Exercise 3.3**: Use pattern comprehension to extract data for lists.
- **Exercise 3.4**: Use list comprehension to create another list.

Go to the next page to start this exercise.

Exercise 3.1: Write a query to return actors as a list. (Instructions)

Write the Cypher code to return a list of actors in the movie, The Matrix.

Exercise 3.1: Write a query to return actors as a list. (Solution)

Write the Cypher code to return a list of actorsin the movie, The Matrix.

Here is the code:

```
MATCH (movie:Movie {title:'The Matrix'})<-[:ACTED_IN]-(actor:Person)
RETURN collect(actor) AS MatrixActors
```

The results returned should look like this:

Exercise 3.2: Use UNWIND to process the list of actors. (Instructions)

Rather than return the list of actors, unwind this list to find all movies that these actors acted in and return list of the distinct titles for these movies.

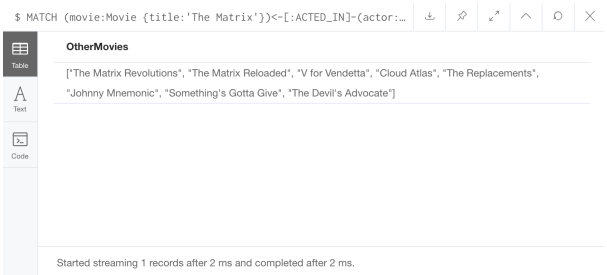
Exercise 3.2: Use UNWIND to process the list of actors. (Solution)

Rather than return the list of actors, unwind this list to find all movies that these actors acted in and return a list of the distinct titles for these movies.

Here is the code:

```
MATCH (movie:Movie {title:'The Matrix'})<-[:ACTED_IN]-(actor:Person)
WITH movie, collect(actor) AS MatrixActors
UNWIND MatrixActors as a
WITH movie, a MATCH (a)-[:ACTED_IN]->(m:Movie)
WHERE movie.title <> m.title
RETURN collect(DISTINCT m.title) as OtherMovies
```

The results returned should look like this:



Exercise 3.3: Use pattern comprehension to extract data for lists. (Instructions)

Here is a Cypher query that retrieves all movies and returns the list of distinct actor names and director names for each movie.

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
```

```
RETURN m.title AS Movie ,collect(DISTINCT a.name) AS Actors, collect(DISTINCT d.name) AS Directors
```

1. Run this query.

2. Rewrite to return the same results using pattern comprehension to extract the data for the Actors and Directors lists.

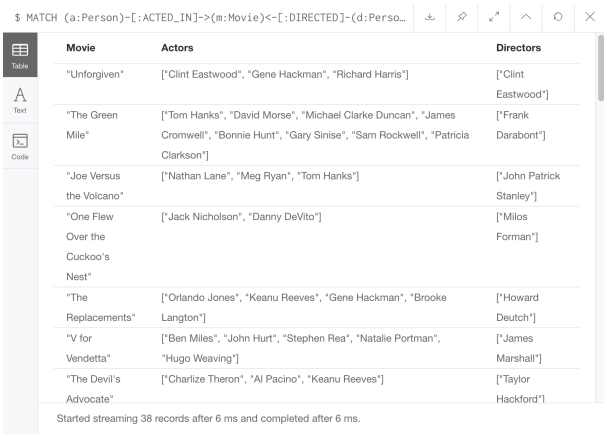
Exercise 3.3: Use pattern comprehension to extract data for lists. (Solution)

Here is a Cypher query that retrieves all movies and returns the list of distinct actor names and director names for each movie.

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
RETURN m.title AS Movie ,collect(DISTINCT a.name) AS Actors, collect(DISTINCT d.name) AS Directors
```

1. Run this query.

The results should be:



Movie	Actors	Directors
"Unforgiven"	["Clint Eastwood", "Gene Hackman", "Richard Harris"]	["Clint Eastwood"]
"The Green Mile"	["Tom Hanks", "David Morse", "Michael Clarke Duncan", "James Cromwell", "Bonnie Hunt", "Gary Sinise", "Sam Rockwell", "Patricia Clarkson"]	["Frank Darabont"]
"Joe Versus the Volcano"	["Nathan Lane", "Meg Ryan", "Tom Hanks"]	["John Patrick Stanley"]
"One Flew Over the Cuckoo's Nest"	["Jack Nicholson", "Danny DeVito"]	["Milos Forman"]
"The Replacements"	["Orlando Jones", "Keanu Reeves", "Gene Hackman", "Brooke Langton"]	["Howard Deutch"]
"V for Vendetta"	["Ben Miles", "John Hurt", "Stephen Rea", "Natalie Portman", "Hugo Weaving"]	["James Marshall"]
"The Devil's Advocate"	["Charlize Theron", "Al Pacino", "Keanu Reeves"]	["Taylor Hackford"]

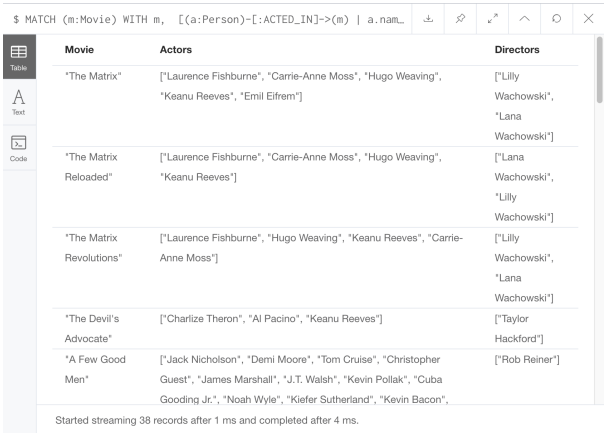
Started streaming 38 records after 6 ms and completed after 6 ms.

2. Rewrite to return the same results using pattern comprehension to extract the data for the Actors and Directors lists.

Here is the solution code:

```
MATCH (m:Movie)
WITH m,
[(a:Person)-[:ACTED_IN]->(m) | a.name] as Actors,
[(m)<-[:DIRECTED]-(d:Person) | d.name] as Directors
RETURN m.title AS Movie , Actors, Directors
```

The results should be:



Exercise 3.4: Use list comprehension to create another list. (Instructions)

Write a Cypher query that retrieves all Movie titles and year released and then returns a list of all movies and how many years ago they were released. Use list comprehension to extract the year released from the list and calculate the number of years ago that the movie was released.

Hint: Use `date().year` for the current year.

Exercise 3.4: Use list comprehension to create another list. (Solution)

Write a Cypher query that retrieves all Movie titles and year released and then returns a list of all movies and how many years ago they were released. Use list comprehension to extract the year released from the list and calculate the number of years ago that the movie was released.

Hint: Use `date().year` for the current year.

Here is the solution code:

```
MATCH (m:Movie)
WITH collect(m.title) AS Movies, collect (m.released) AS Released
WITH Movies, Released,
[x IN Released | date().year - x + 1] AS YearsAgo
RETURN Movies, YearsAgo
```

The result returned should be:

Exercise 3: Taking it further

1. Profile the queries you have written to see whether using pattern comprehension performs better than using explicit `collect()` calls.

2. Use list projection to return the ages of all of the actors.

Exercise 3: Working with UNWIND, pattern and list comprehension (Summary)

In this exercise, you gained some experience with `UNWIND`, as well as pattern comprehension and list comprehension in your queries.

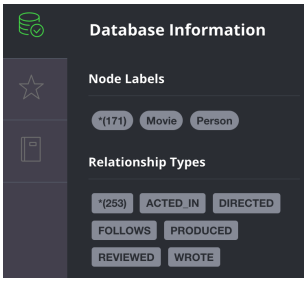
Continue to Exercise 4

Exercise 4

Exercise 4: Combining query results (Preparations)

The database you start with should contain the "movie" data that you loaded in the first exercise.

This is what you should see when you click the database icon  .



If your database does not look like this, run the following script which removes all nodes and relationships from the graph and loads the graph with the "mini movie" data:

```
MATCH (n) DETACH DELETE n;
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixReloaded),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixReloaded),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixReloaded),
  (LillyW)-[:DIRECTED]->(TheMatrixReloaded),
  (LanaW)-[:DIRECTED]->(TheMatrixReloaded),
  (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions', released:2003, tagline:'Everything that has a beginning has an end'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixRevolutions),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixRevolutions),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixRevolutions),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixRevolutions),
  (LillyW)-[:DIRECTED]->(TheMatrixRevolutions),
  (LanaW)-[:DIRECTED]->(TheMatrixRevolutions),
  (JoelS)-[:PRODUCED]->(TheMatrixRevolutions)

CREATE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", released:1997, tagline:'Evil has its winning ways'})
CREATE (Charlize:Person {name:'Charlize Theron', born:1975})
CREATE (Al:Person {name:'Al Pacino', born:1940})
```

```
CREATE (Taylor:Person {name:'Taylor Hackford', born:1944})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Kevin Lomax']}]->(TheDevilsAdvocate),
  (Charlize)-[:ACTED_IN {roles:['Mary Ann Lomax']}]->(TheDevilsAdvocate),
  (Al)-[:ACTED_IN {roles:['John Milton']}]->(TheDevilsAdvocate),
  (Taylor)-[:DIRECTED]->(TheDevilsAdvocate)

CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tagline:"In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth."})
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
  (TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),
  (JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),
  (DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]->(AFewGoodMen),
  (KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),
  (KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),
  (NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),
  (CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),
  (KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),
  (JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]->(AFewGoodMen),
  (JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),
  (ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),
  (AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),
  (RobR)-[:DIRECTED]->(AFewGoodMen),
  (AaronS)-[:WROTE]->(AFewGoodMen)

CREATE (TopGun:Movie {title:"Top Gun", released:1986, tagline:'I feel the need, the need for speed.'})
CREATE (KellyM:Person {name:'Kelly McGillis', born:1957})
CREATE (ValK:Person {name:'Val Kilmer', born:1959})
CREATE (AnthonyE:Person {name:'Anthony Edwards', born:1962})
CREATE (TomS:Person {name:'Tom Skerritt', born:1933})
CREATE (MegR:Person {name:'Meg Ryan', born:1961})
CREATE (TonyS:Person {name:'Tony Scott', born:1944})
CREATE (JimC:Person {name:'Jim Cash', born:1941})
CREATE
  (TomC)-[:ACTED_IN {roles:['Maverick']}]->(TopGun),
  (KellyM)-[:ACTED_IN {roles:['Charlie']}]->(TopGun),
  (ValK)-[:ACTED_IN {roles:['Iceman']}]->(TopGun),
  (AnthonyE)-[:ACTED_IN {roles:['Goose']}]->(TopGun),
  (TomS)-[:ACTED_IN {roles:['Viper']}]->(TopGun),
  (MegR)-[:ACTED_IN {roles:['Carole']}]->(TopGun),
  (TonyS)-[:DIRECTED]->(TopGun),
  (JimC)-[:WROTE]->(TopGun)

CREATE (JerryMaguire:Movie {title:'Jerry Maguire', released:2000, tagline:'The rest of his life begins now.'})
CREATE (ReneeZ:Person {name:'Renee Zellweger', born:1969})
CREATE (KellyP:Person {name:'Kelly Preston', born:1962})
CREATE (JerryO:Person {name:"Jerry O'Connell", born:1974})
CREATE (JayM:Person {name:'Jay Mohr', born:1970})
CREATE (BonnieH:Person {name:'Bonnie Hunt', born:1961})
CREATE (ReginaK:Person {name:'Regina King', born:1971})
CREATE (JonathanL:Person {name:'Jonathan Lipnicki', born:1996})
CREATE (CameronC:Person {name:'Cameron Crowe', born:1957})
CREATE
  (TomC)-[:ACTED_IN {roles:['Jerry Maguire']}]->(JerryMaguire),
  (CubaG)-[:ACTED_IN {roles:['Rod Tidwell']}]->(JerryMaguire),
  (ReneeZ)-[:ACTED_IN {roles:['Dorothy Boyd']}]->(JerryMaguire),
  (KellyP)-[:ACTED_IN {roles:['Avery Bishop']}]->(JerryMaguire),
  (JerryO)-[:ACTED_IN {roles:['Frank Cushman']}]->(JerryMaguire),
  (JayM)-[:ACTED_IN {roles:['Bob Sugar']}]->(JerryMaguire),
  (BonnieH)-[:ACTED_IN {roles:['Laurel Boyd']}]->(JerryMaguire),
  (ReginaK)-[:ACTED_IN {roles:['Marcee Tidwell']}]->(JerryMaguire),
  (JonathanL)-[:ACTED_IN {roles:['Ray Boyd']}]->(JerryMaguire),
  (CameronC)-[:DIRECTED]->(JerryMaguire),
  (CameronC)-[:PRODUCED]->(JerryMaguire),
  (CameronC)-[:WROTE]->(JerryMaguire)

CREATE (StandByMe:Movie {title:"Stand By Me", released:1986, tagline:"For some, it's the last real taste of innocence, and the first real taste
```

```
of life. But for everyone, it's the time that memories are made of."})
CREATE (RiverP:Person {name:'River Phoenix', born:1970})
CREATE (CoreyF:Person {name:'Corey Feldman', born:1971})
CREATE (WilW:Person {name:'Wil Wheaton', born:1972})
CREATE (JohnC:Person {name:'John Cusack', born:1966})
CREATE (MarshallB:Person {name:'Marshall Bell', born:1942})
CREATE
  (WilW)-[:ACTED_IN {roles:['Gordie Lachance']}]->(StandByMe),
  (RiverP)-[:ACTED_IN {roles:['Chris Chambers']}]->(StandByMe),
  (JerryO)-[:ACTED_IN {roles:['Vern Tessio']}]->(StandByMe),
  (CoreyF)-[:ACTED_IN {roles:['Teddy Duchamp']}]->(StandByMe),
  (JohnC)-[:ACTED_IN {roles:['Denny Lachance']}]->(StandByMe),
  (KieferS)-[:ACTED_IN {roles:['Ace Merrill']}]->(StandByMe),
  (MarshallB)-[:ACTED_IN {roles:['Mr. Lachance']}]->(StandByMe),
  (RobR)-[:DIRECTED]->(StandByMe)

CREATE (AsGoodAsItGets:Movie {title:'As Good as It Gets', released:1997, tagline:'A comedy from the heart that goes for the throat.'})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})
CREATE (GregK:Person {name:'Greg Kinnear', born:1963})
CREATE (JamesB:Person {name:'James L. Brooks', born:1940})
CREATE
  (JackN)-[:ACTED_IN {roles:['Melvin Udall']}]->(AsGoodAsItGets),
  (HelenH)-[:ACTED_IN {roles:['Carol Connelly']}]->(AsGoodAsItGets),
  (GregK)-[:ACTED_IN {roles:['Simon Bishop']}]->(AsGoodAsItGets),
  (CubaG)-[:ACTED_IN {roles:['Frank Sachs']}]->(AsGoodAsItGets),
  (JamesB)-[:DIRECTED]->(AsGoodAsItGets)

CREATE (WhatDreamsMayCome:Movie {title:'What Dreams May Come', released:1998, tagline:'After life there is more. The end is just the
beginning.'})
CREATE (AnnabellaS:Person {name:'Annabella Sciorra', born:1960})
CREATE (MaxS:Person {name:'Max von Sydow', born:1929})
CREATE (WernerH:Person {name:'Werner Herzog', born:1942})
CREATE (Robin:Person {name:'Robin Williams', born:1951})
CREATE (VincentW:Person {name:'Vincent Ward', born:1956})
CREATE
  (Robin)-[:ACTED_IN {roles:['Chris Nielsen']}]->(WhatDreamsMayCome),
  (CubaG)-[:ACTED_IN {roles:['Albert Lewis']}]->(WhatDreamsMayCome),
  (AnnabellaS)-[:ACTED_IN {roles:['Annie Collins-Nielsen']}]->(WhatDreamsMayCome),
  (MaxS)-[:ACTED_IN {roles:['The Tracker']}]->(WhatDreamsMayCome),
  (WernerH)-[:ACTED_IN {roles:['The Face']}]->(WhatDreamsMayCome),
  (VincentW)-[:DIRECTED]->(WhatDreamsMayCome)

CREATE (SnowFallingonCedars:Movie {title:'Snow Falling on Cedars', released:1999, tagline:'First loves last. Forever.'})
CREATE (EthanH:Person {name:'Ethan Hawke', born:1970})
CREATE (RickY:Person {name:'Rick Yune', born:1971})
CREATE (JamesC:Person {name:'James Cromwell', born:1940})
CREATE (ScottH:Person {name:'Scott Hicks', born:1953})
CREATE
  (EthanH)-[:ACTED_IN {roles:['Ishmael Chambers']}]->(SnowFallingonCedars),
  (RickY)-[:ACTED_IN {roles:['Kazuo Miyamoto']}]->(SnowFallingonCedars),
  (MaxS)-[:ACTED_IN {roles:['Nels Gudmundsson']}]->(SnowFallingonCedars),
  (JamesC)-[:ACTED_IN {roles:['Judge Fielding']}]->(SnowFallingonCedars),
  (ScottH)-[:DIRECTED]->(SnowFallingonCedars)

CREATE (YouveGotMail:Movie {title:"You've Got Mail", released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (ParkerP:Person {name:'Parker Posey', born:1968})
CREATE (DaveC:Person {name:'Dave Chappelle', born:1973})
CREATE (SteveZ:Person {name:'Steve Zahn', born:1967})
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
  (MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]->(YouveGotMail),
  (GregK)-[:ACTED_IN {roles:['Frank Navasky']}]->(YouveGotMail),
  (ParkerP)-[:ACTED_IN {roles:['Patricia Eden']}]->(YouveGotMail),
  (DaveC)-[:ACTED_IN {roles:['Kevin Jackson']}]->(YouveGotMail),
  (SteveZ)-[:ACTED_IN {roles:['George Pappas']}]->(YouveGotMail),
  (NoraE)-[:DIRECTED]->(YouveGotMail)

CREATE (SleeplessInSeattle:Movie {title:'Sleepless in Seattle', released:1993, tagline:'What if someone you never met, someone you never saw,
someone you never knew was the only someone for you?'})
CREATE (RitaW:Person {name:'Rita Wilson', born:1956})
CREATE (BillPull:Person {name:'Bill Pullman', born:1953})
CREATE (VictorG:Person {name:'Victor Garber', born:1949})
CREATE (RosieO:Person {name:"Rosie O'Donnell", born:1962})
CREATE
  (TomH)-[:ACTED_IN {roles:['Sam Baldwin']}]->(SleeplessInSeattle),
  (MegR)-[:ACTED_IN {roles:['Annie Reed']}]->(SleeplessInSeattle),
  (RitaW)-[:ACTED_IN {roles:['Suzy']}]->(SleeplessInSeattle),
  (BillPull)-[:ACTED_IN {roles:['Walter']}]->(SleeplessInSeattle),
```

```
(VictorG)-[:ACTED_IN {roles:['Greg']}]->(SleeplessInSeattle),
(RosieO)-[:ACTED_IN {roles:['Becky']}]->(SleeplessInSeattle),
(NoraE)-[:DIRECTED]->(SleeplessInSeattle)

CREATE (JoeVersustheVolcano:Movie {title:'Joe Versus the Volcano', released:1990, tagline:'A story of love, lava and burning desire.'})
CREATE (JohnS:Person {name:'John Patrick Stanley', born:1950})
CREATE (Nathan:Person {name:'Nathan Lane', born:1956})
CREATE
(TomH)-[:ACTED_IN {roles:['Joe Banks']}]->(JoeVersustheVolcano),
(MegR)-[:ACTED_IN {roles:['DeDe', 'Angelica Graynamore', 'Patricia Graynamore']}]->(JoeVersustheVolcano),
(Nathan)-[:ACTED_IN {roles:['Baw']}]->(JoeVersustheVolcano),
(JohnS)-[:DIRECTED]->(JoeVersustheVolcano)

CREATE (WhenHarryMetSally:Movie {title:'When Harry Met Sally', released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (BillyC:Person {name:'Billy Crystal', born:1948})
CREATE (CarrieF:Person {name:'Carrie Fisher', born:1956})
CREATE (BrunoK:Person {name:'Bruno Kirby', born:1949})
CREATE
(BillyC)-[:ACTED_IN {roles:['Harry Burns']}]->(WhenHarryMetSally),
(MegR)-[:ACTED_IN {roles:['Sally Albright']}]->(WhenHarryMetSally),
(CarrieF)-[:ACTED_IN {roles:['Marie']}]->(WhenHarryMetSally),
(BrunoK)-[:ACTED_IN {roles:['Jess']}]->(WhenHarryMetSally),
(RobR)-[:DIRECTED]->(WhenHarryMetSally),
(RobR)-[:PRODUCED]->(WhenHarryMetSally),
(NoraE)-[:PRODUCED]->(WhenHarryMetSally),
(NoraE)-[:WROTE]->(WhenHarryMetSally)

CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', released:1996, tagline:'In every life there comes a time when that thing you
dream becomes that thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
(TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
(LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
(Charlize)-[:ACTED_IN {roles:['Tina']}]->(ThatThingYouDo),
(TomH)-[:DIRECTED]->(ThatThingYouDo)

CREATE (TheReplacements:Movie {title:'The Replacements', released:2000, tagline:'Pain heals, Chicks dig scars... Glory lasts forever'})
CREATE (Brooke:Person {name:'Brooke Langton', born:1970})
CREATE (Gene:Person {name:'Gene Hackman', born:1930})
CREATE (Orlando:Person {name:'Orlando Jones', born:1968})
CREATE (Howard:Person {name:'Howard Deutch', born:1950})
CREATE
(Keanu)-[:ACTED_IN {roles:['Shane Falco']}]->(TheReplacements),
(Brooke)-[:ACTED_IN {roles:['Annabelle Farrell']}]->(TheReplacements),
(Gene)-[:ACTED_IN {roles:['Jimmy McGinty']}]->(TheReplacements),
(Orlando)-[:ACTED_IN {roles:['Clifford Franklin']}]->(TheReplacements),
(Howard)-[:DIRECTED]->(TheReplacements)

CREATE (RescueDawn:Movie {title:'RescueDawn', released:2006, tagline:"Based on the extraordinary true story of one man's fight for
freedom"})
CREATE (ChristianB:Person {name:'Christian Bale', born:1974})
CREATE (ZachG:Person {name:'Zach Grenier', born:1954})
CREATE
(MarshallB)-[:ACTED_IN {roles:['Admiral']}]->(RescueDawn),
(ChristianB)-[:ACTED_IN {roles:['Dieter Dengler']}]->(RescueDawn),
(ZachG)-[:ACTED_IN {roles:['Squad Leader']}]->(RescueDawn),
(SteveZ)-[:ACTED_IN {roles:['Duane']}]->(RescueDawn),
(WernerH)-[:DIRECTED]->(RescueDawn)

CREATE (TheBirdcage:Movie {title:'The Birdcage', released:1996, tagline:'Come as you are'})
CREATE (MikeN:Person {name:'Mike Nichols', born:1931})
CREATE
(Robin)-[:ACTED_IN {roles:['Armand Goldman']}]->(TheBirdcage),
(Nathan)-[:ACTED_IN {roles:['Albert Goldman']}]->(TheBirdcage),
(Gene)-[:ACTED_IN {roles:['Sen. Kevin Keeley']}]->(TheBirdcage),
(MikeN)-[:DIRECTED]->(TheBirdcage)

CREATE (Unforgiven:Movie {title:'Unforgiven', released:1992, tagline:"It's a hell of a thing, killing a man"})
CREATE (RichardH:Person {name:'Richard Harris', born:1930})
CREATE (ClintE:Person {name:'Clint Eastwood', born:1930})
CREATE
(RichardH)-[:ACTED_IN {roles:['English Bob']}]->(Unforgiven),
(ClintE)-[:ACTED_IN {roles:['Bill Munny']}]->(Unforgiven),
(Gene)-[:ACTED_IN {roles:['Little Bill Daggett']}]->(Unforgiven),
(ClintE)-[:DIRECTED]->(Unforgiven)

CREATE (JohnnyMnemonic:Movie {title:'Johnny Mnemonic', released:1995, tagline:'The hottest data on earth. In the coolest head in town'})
CREATE (Takeshi:Person {name:'Takeshi Kitano', born:1947})
CREATE (Dina:Person {name:'Dina Meyer', born:1968})
CREATE (IceT:Person {name:'Ice-T', born:1958})
```

```
CREATE (RobertL:Person {name:'Robert Longo', born:1953})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Johnny Mnemonic']}]->(JohnnyMnemonic),
  (Takeshi)-[:ACTED_IN {roles:['Takahashi']}]->(JohnnyMnemonic),
  (Dina)-[:ACTED_IN {roles:['Jane']}]->(JohnnyMnemonic),
  (IceT)-[:ACTED_IN {roles:['J-Bone']}]->(JohnnyMnemonic),
  (RobertL)-[:DIRECTED]->(JohnnyMnemonic)

CREATE (CloudAtlas:Movie {title:'Cloud Atlas', released:2012, tagline:'Everything is connected'})
CREATE (HalleB:Person {name:'Halle Berry', born:1966})
CREATE (JimB:Person {name:'Jim Broadbent', born:1949})
CREATE (TomT:Person {name:'Tom Tykwer', born:1965})
CREATE (DavidMitchell:Person {name:'David Mitchell', born:1969})
CREATE (StefanArndt:Person {name:'Stefan Arndt', born:1961})
CREATE
  (TomH)-[:ACTED_IN {roles:['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']}]->(CloudAtlas),
  (Hugo)-[:ACTED_IN {roles:['Bill Smoke', 'Haskell Moore', 'Tadeusz Kesselring', 'Nurse Noakes', 'Boardman Mephi', 'Old Georgie']}]->
(CloudAtlas),
  (HalleB)-[:ACTED_IN {roles:['Luisa Rey', 'Jocasta Ayrs', 'Ovid', 'Meronym']}]->(CloudAtlas),
  (JimB)-[:ACTED_IN {roles:['Vyvyan Ayrs', 'Captain Molyneux', 'Timothy Cavendish']}]->(CloudAtlas),
  (TomT)-[:DIRECTED]->(CloudAtlas),
  (LillyW)-[:DIRECTED]->(CloudAtlas),
  (LanaW)-[:DIRECTED]->(CloudAtlas),
  (DavidMitchell)-[:WROTE]->(CloudAtlas),
  (StefanArndt)-[:PRODUCED]->(CloudAtlas)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006, tagline:'Break The Codes'})
CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)

CREATE (VforVendetta:Movie {title:'V for Vendetta', released:2006, tagline:'Freedom! Forever!'})
CREATE (NatalieP:Person {name:'Natalie Portman', born:1981})
CREATE (StephenR:Person {name:'Stephen Rea', born:1946})
CREATE (JohnH:Person {name:'John Hurt', born:1940})
CREATE (BenM:Person {name: 'Ben Miles', born:1967})
CREATE
  (Hugo)-[:ACTED_IN {roles:['V']}]->(VforVendetta),
  (NatalieP)-[:ACTED_IN {roles:['Evey Hammond']}]->(VforVendetta),
  (StephenR)-[:ACTED_IN {roles:['Eric Finch']}]->(VforVendetta),
  (JohnH)-[:ACTED_IN {roles:['High Chancellor Adam Sutler']}]->(VforVendetta),
  (BenM)-[:ACTED_IN {roles:['Dascomb']}]->(VforVendetta),
  (JamesM)-[:DIRECTED]->(VforVendetta),
  (LillyW)-[:PRODUCED]->(VforVendetta),
  (LanaW)-[:PRODUCED]->(VforVendetta),
  (JoelS)-[:PRODUCED]->(VforVendetta),
  (LillyW)-[:WROTE]->(VforVendetta),
  (LanaW)-[:WROTE]->(VforVendetta)

CREATE (SpeedRacer:Movie {title:'Speed Racer', released:2008, tagline:'Speed has no limits'})
CREATE (EmileH:Person {name:'Emile Hirsch', born:1985})
CREATE (JohnG:Person {name:'John Goodman', born:1960})
CREATE (SusanS:Person {name:'Susan Sarandon', born:1946})
CREATE (MatthewF:Person {name:'Matthew Fox', born:1966})
CREATE (ChristinaR:Person {name:'Christina Ricci', born:1980})
CREATE (Rain:Person {name:'Rain', born:1982})
CREATE
  (EmileH)-[:ACTED_IN {roles:['Speed Racer']}]->(SpeedRacer),
  (JohnG)-[:ACTED_IN {roles:['Pops']}]->(SpeedRacer),
  (SusanS)-[:ACTED_IN {roles:['Mom']}]->(SpeedRacer),
  (MatthewF)-[:ACTED_IN {roles:['Racer X']}]->(SpeedRacer),
  (ChristinaR)-[:ACTED_IN {roles:['Trixie']}]->(SpeedRacer),
  (Rain)-[:ACTED_IN {roles:['Taejo Togokahn']}]->(SpeedRacer),
  (BenM)-[:ACTED_IN {roles:['Cass Jones']}]->(SpeedRacer),
  (LillyW)-[:DIRECTED]->(SpeedRacer),
  (LanaW)-[:DIRECTED]->(SpeedRacer),
  (LillyW)-[:WROTE]->(SpeedRacer),
  (LanaW)-[:WROTE]->(SpeedRacer),
  (JoelS)-[:PRODUCED]->(SpeedRacer)

CREATE (NinjaAssassin:Movie {title:'Ninja Assassin', released:2009, tagline:'Prepare to enter a secret world of assassins'})
CREATE (NaomieH:Person {name:'Naomie Harris'})
CREATE
```

```
  (Rain)-[:ACTED_IN {roles:['Raizo']}]->(NinjaAssassin),
  (NaomieH)-[:ACTED_IN {roles:['Mika Coretti']}]->(NinjaAssassin),
  (RickY)-[:ACTED_IN {roles:['Takeshi']}]->(NinjaAssassin),
  (BenM)-[:ACTED_IN {roles:['Ryan Maslow']}]->(NinjaAssassin),
  (JamesM)-[:DIRECTED]->(NinjaAssassin),
  (LillyW)-[:PRODUCED]->(NinjaAssassin),
  (LanaW)-[:PRODUCED]->(NinjaAssassin),
  (JoelS)-[:PRODUCED]->(NinjaAssassin)

CREATE (TheGreenMile:Movie {title:'The Green Mile', released:1999, tagline:"Walk a mile you'll never forget."})
CREATE (MichaelD:Person {name:'Michael Clarke Duncan', born:1957})
CREATE (DavidM:Person {name:'David Morse', born:1953})
CREATE (SamR:Person {name:'Sam Rockwell', born:1968})
CREATE (GaryS:Person {name:'Gary Sinise', born:1955})
CREATE (PatriciaC:Person {name:'Patricia Clarkson', born:1959})
CREATE (FrankD:Person {name:'Frank Darabont', born:1959})
CREATE
  (TomH)-[:ACTED_IN {roles:['Paul Edgecomb']}]->(TheGreenMile),
  (MichaelD)-[:ACTED_IN {roles:['John Coffey']}]->(TheGreenMile),
  (DavidM)-[:ACTED_IN {roles:['Brutus "Brutal" Howell']}]->(TheGreenMile),
  (BonnieH)-[:ACTED_IN {roles:['Jan Edgecomb']}]->(TheGreenMile),
  (JamesC)-[:ACTED_IN {roles:['Warden Hal Moores']}]->(TheGreenMile),
  (SamR)-[:ACTED_IN {roles:['"Wild Bill" Wharton']}]->(TheGreenMile),
  (GaryS)-[:ACTED_IN {roles:['Burt Hammersmith']}]->(TheGreenMile),
  (PatriciaC)-[:ACTED_IN {roles:['Melinda Moores']}]->(TheGreenMile),
  (FrankD)-[:DIRECTED]->(TheGreenMile)

CREATE (FrostNixon:Movie {title:'Frost/Nixon', released:2008, tagline:'400 million people were waiting for the truth.'})
CREATE (FrankL:Person {name:'Frank Langella', born:1938})
CREATE (MichaelS:Person {name:'Michael Sheen', born:1969})
CREATE (OliverP:Person {name:'Oliver Platt', born:1960})
CREATE
  (FrankL)-[:ACTED_IN {roles:['Richard Nixon']}]->(FrostNixon),
  (MichaelS)-[:ACTED_IN {roles:['David Frost']}]->(FrostNixon),
  (KevinB)-[:ACTED_IN {roles:['Jack Brennan']}]->(FrostNixon),
  (OliverP)-[:ACTED_IN {roles:['Bob Zelnick']}]->(FrostNixon),
  (SamR)-[:ACTED_IN {roles:['James Reston, Jr.']}]->(FrostNixon),
  (RonH)-[:DIRECTED]->(FrostNixon)

CREATE (Hoffa:Movie {title:'Hoffa', released:1992, tagline:"He didn't want law. He wanted justice."})
CREATE (DannyD:Person {name:'Danny DeVito', born:1944})
CREATE (JohnR:Person {name:'John C. Reilly', born:1965})
CREATE
  (JackN)-[:ACTED_IN {roles:['Hoffa']}]->(Hoffa),
  (DannyD)-[:ACTED_IN {roles:['Robert "Bobby" Ciaro']}]->(Hoffa),
  (JTW)-[:ACTED_IN {roles:['Frank Fitzsimmons']}]->(Hoffa),
  (JohnR)-[:ACTED_IN {roles:['Peter "Pete" Connelly']}]->(Hoffa),
  (DannyD)-[:DIRECTED]->(Hoffa)

CREATE (Apollo13:Movie {title:'Apollo 13', released:1995, tagline:'Houston, we have a problem.'})
CREATE (EdH:Person {name:'Ed Harris', born:1950})
CREATE (BillPax:Person {name:'Bill Paxton', born:1955})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jim Lovell']}]->(Apollo13),
  (KevinB)-[:ACTED_IN {roles:['Jack Swigert']}]->(Apollo13),
  (EdH)-[:ACTED_IN {roles:['Gene Kranz']}]->(Apollo13),
  (BillPax)-[:ACTED_IN {roles:['Fred Haise']}]->(Apollo13),
  (GaryS)-[:ACTED_IN {roles:['Ken Mattingly']}]->(Apollo13),
  (RonH)-[:DIRECTED]->(Apollo13)

CREATE (Twister:Movie {title:'Twister', released:1996, tagline:"Don't Breathe. Don't Look Back."})
CREATE (PhilipH:Person {name:'Philip Seymour Hoffman', born:1967})
CREATE (JanB:Person {name:'Jan de Bont', born:1943})
CREATE
  (BillPax)-[:ACTED_IN {roles:['Bill Harding']}]->(Twister),
  (HelenH)-[:ACTED_IN {roles:['Dr. Jo Harding']}]->(Twister),
  (ZachG)-[:ACTED_IN {roles:['Eddie']}]->(Twister),
  (PhilipH)-[:ACTED_IN {roles:['Dustin "Dusty" Davis']}]->(Twister),
  (JanB)-[:DIRECTED]->(Twister)

CREATE (CastAway:Movie {title:'Cast Away', released:2000, tagline:'At the edge of the world, his journey begins.'})
CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (OneFlewOvertheCuckoosNest:Movie {title:"One Flew Over the Cuckoo's Nest", released:1975, tagline:"If he's crazy, what does that make you?"})
CREATE (MilosF:Person {name:'Milos Forman', born:1932})
```

```
CREATE
  (JackN)-[:ACTED_IN {roles:['Randle McMurphy']}]->(OneFlewOvertheCuckoosNest),
  (DannyD)-[:ACTED_IN {roles:['Martini']}]->(OneFlewOvertheCuckoosNest),
  (MilosF)-[:DIRECTED]->(OneFlewOvertheCuckoosNest)

CREATE (SomethingsGottaGive:Movie {title:"Something's Gotta Give", released:2003})
CREATE (DianeK:Person {name:'Diane Keaton', born:1946})
CREATE (NancyM:Person {name:'Nancy Meyers', born:1949})
CREATE
  (JackN)-[:ACTED_IN {roles:['Harry Sanborn']}]->(SomethingsGottaGive),
  (DianeK)-[:ACTED_IN {roles:['Erica Barry']}]->(SomethingsGottaGive),
  (Keanu)-[:ACTED_IN {roles:['Julian Mercer']}]->(SomethingsGottaGive),
  (NancyM)-[:DIRECTED]->(SomethingsGottaGive),
  (NancyM)-[:PRODUCED]->(SomethingsGottaGive),
  (NancyM)-[:WROTE]->(SomethingsGottaGive)

CREATE (BicentennialMan:Movie {title:'Bicentennial Man', released:1999, tagline:"One robot's 200 year journey to become an ordinary man."})
CREATE (ChrisC:Person {name:'Chris Columbus', born:1958})
CREATE
  (Robin)-[:ACTED_IN {roles:['Andrew Marin']}]->(BicentennialMan),
  (OliverP)-[:ACTED_IN {roles:['Rupert Burns']}]->(BicentennialMan),
  (ChrisC)-[:DIRECTED]->(BicentennialMan)

CREATE (CharlieWilsonsWar:Movie {title:"Charlie Wilson's War", released:2007, tagline:"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire."})
CREATE (JuliaR:Person {name:'Julia Roberts', born:1967})
CREATE
  (TomH)-[:ACTED_IN {roles:['Rep. Charlie Wilson']}]->(CharlieWilsonsWar),
  (JuliaR)-[:ACTED_IN {roles:['Joanne Herring']}]->(CharlieWilsonsWar),
  (PhilipH)-[:ACTED_IN {roles:['Gust Avrakotos']}]->(CharlieWilsonsWar),
  (MikeN)-[:DIRECTED]->(CharlieWilsonsWar)

CREATE (ThePolarExpress:Movie {title:'The Polar Express', released:2004, tagline:'This Holiday Seasonâ€¦ Believe'})
CREATE
  (TomH)-[:ACTED_IN {roles:['Hero Boy', 'Father', 'Conductor', 'Hobo', 'Scrooge', 'Santa Claus']}]->(ThePolarExpress),
  (RobertZ)-[:DIRECTED]->(ThePolarExpress)

CREATE (ALeagueofTheirOwn:Movie {title:'A League of Their Own', released:1992, tagline:'Once in a lifetime you get a chance to do something different.'})
CREATE (Madonna:Person {name:'Madonna', born:1954})
CREATE (GeenaD:Person {name:'Geena Davis', born:1956})
CREATE (LoriP:Person {name:'Lori Petty', born:1963})
CREATE (PennyM:Person {name:'Penny Marshall', born:1943})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jimmy Dugan']}]->(ALeagueofTheirOwn),
  (GeenaD)-[:ACTED_IN {roles:['Dottie Hinson']}]->(ALeagueofTheirOwn),
  (LoriP)-[:ACTED_IN {roles:['Kit Keller']}]->(ALeagueofTheirOwn),
  (RosieO)-[:ACTED_IN {roles:['Doris Murphy']}]->(ALeagueofTheirOwn),
  (Madonna)-[:ACTED_IN {roles:['"All the Way" Mae Mordabito']}]->(ALeagueofTheirOwn),
  (BillPax)-[:ACTED_IN {roles:['Bob Hinson']}]->(ALeagueofTheirOwn),
  (PennyM)-[:DIRECTED]->(ALeagueofTheirOwn)

CREATE (PaulBlythe:Person {name:'Paul Blythe'})
CREATE (AngelaScope:Person {name:'Angela Scope'})
CREATE (JessicaThompson:Person {name:'Jessica Thompson'})
CREATE (JamesThompson:Person {name:'James Thompson'})

CREATE
  (JamesThompson)-[:FOLLOWS]->(JessicaThompson),
  (AngelaScope)-[:FOLLOWS]->(JessicaThompson),
  (PaulBlythe)-[:FOLLOWS]->(AngelaScope)

CREATE
  (JessicaThompson)-[:REVIEWED {summary:'An amazing journey', rating:95}]->(CloudAtlas),
  (JessicaThompson)-[:REVIEWED {summary:'Silly, but fun', rating:65}]->(TheReplacements),
  (JamesThompson)-[:REVIEWED {summary:'The coolest football movie ever', rating:100}]->(TheReplacements),
  (AngelaScope)-[:REVIEWED {summary:'Pretty funny at times', rating:62}]->(TheReplacements),
  (JessicaThompson)-[:REVIEWED {summary:'Dark, but compelling', rating:85}]->(Unforgiven),
  (JessicaThompson)-[:REVIEWED {summary:"Slapstick redeemed only by the Robin Williams and Gene Hackman's stellar performances", rating:45}]->(TheBirdcage),
  (JessicaThompson)-[:REVIEWED {summary:'A solid romp', rating:68}]->(TheDaVinciCode),
  (JamesThompson)-[:REVIEWED {summary:'Fun, but a little far fetched', rating:65}]->(TheDaVinciCode),
  (JessicaThompson)-[:REVIEWED {summary:'You had me at Jerry', rating:92}]->(JerryMaguire)
```

Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 4: Combining query results (Overview)

In this exercise, you write some Cypher statements that combine the results of two separate queries:

- **Exercise 4.1**: Write a query to combine query results with `UNION`.
- **Exercise 4.2**: USE APOC to sort the combined query results that use `UNION`.
- **Exercise 4.3**: Use `UNWIND` to sort the combined query results.

Go to the next page to start this exercise.

Exercise 4.1: Write a query to combine query results with `UNION`. (Instructions)

Write the Cypher code to combine the results of a retrieval of all actors that were born after 1960 and a retrieval of all writers that were born after 1960. Each query will return the name of the person, their age, and their Role (either Actor or Writer).

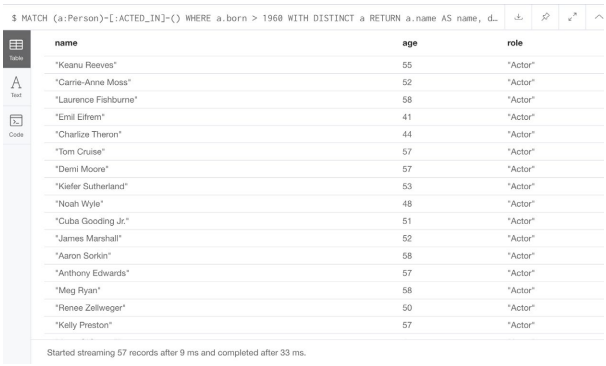
Exercise 4.1: Write a query to combine query results with `UNION`. (Solution)

Write the Cypher code to combine the results of a retrieval of all actors that were born after 1960 and a retrieval of all writers that were born after 1960. Each query will return the name of the person, their age, and their Role (either Actor or Writer).

Here is the solution code:

```
MATCH (a:Person)-[:ACTED_IN]-()
WHERE a.born  >  1960
WITH DISTINCT a
RETURN a.name AS name, date().year - a.born AS age, 'Actor' AS role
UNION
MATCH (w:Person)-[:WROTE]-()
WHERE w.born > 1960
WITH DISTINCT w
RETURN  w.name AS name, date().year - w.born AS age, 'Writer' AS role
```

The results returned should look like this:



Exercise 4.2: USE APOC to sort the combined query results that use UNION. (Instructions)

Modify the previous query to be executed using APOC so that you can sort the results returned by age.

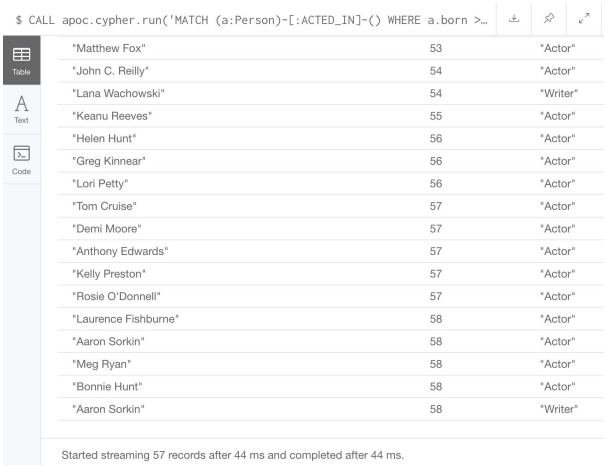
Exercise 4.2: USE APOC to sort the combined query results that use UNION. (Solution)

Modify the previous query to be executed using APOC so that you can sort the results returned by age.

Here is the solution code:

```
CALL apoc.cypher.run('MATCH (a:Person)-[:ACTED_IN]-()
WHERE a.born  >  1960
WITH DISTINCT a
RETURN  a.name AS name, date().year - a.born AS age, "Actor" AS role
UNION
MATCH (w:Person)-[:WROTE]-()
WHERE w.born > 1960
WITH DISTINCT w
RETURN  w.name AS name, date().year - w.born AS age, "Writer" AS role',{}) YIELD value
WITH value
RETURN value.name as name, value.age as age, value.role as role ORDER BY age
```

The results returned should look like this:



Exercise 4.3: Use `UNWIND` to sort the combined query results. (Instructions)

Rewrite the query to not use APOC and to use `collect()` and `UNWIND` to sort the results, rather than `UNION`.

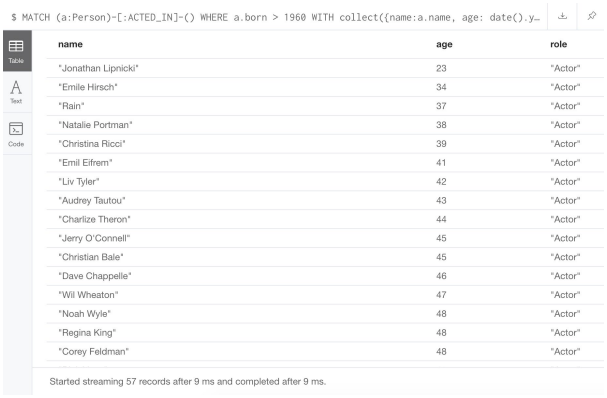
Exercise 4.3: Use `UNWIND` to sort the combined query results. (Solution)

Rewrite the query to not use APOC and to use `collect()` and `UNWIND` to sort the results, rather than `UNION`.

Here is the solution code:

```
MATCH (a:Person)-[:ACTED_IN]-()
WHERE a.born  >  1960
WITH collect({name:a.name, age: date().year - a.born, role: 'Actor'}) as result1
MATCH (w:Person)-[:WROTE]-()
WHERE w.born > 1960
WITH result1 + collect({name:w.name, age: date().year - w.born, role: 'Writer'}) AS allResults
UNWIND allResults as result
WITH result.name as name, result.age as age, result.role as role RETURN DISTINCT name, age, role ORDER BY age
```

The results should be:



Exercise 4: Taking it further

1. Profile the queries you have written to see which query performs best.
2. Create a query where using `UNION` is not better than the alternatives.

Exercise 4: Combining query results (Summary)

In this exercise, you have written queries that combine results using `UNION` and `collect()`/`UNWIND`. The most performant solution you use for your application will depend on the data model and the amount of data in the graph.

Continue to Exercise 5

Exercise 5

Exercise 5: Loading normalized data (Preparations)

Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 5: Loading normalized data (Overview)

In this exercise, you will first remove all data and indexes from the database so you can start fresh, then you will load normalized data into the database.

- **Exercise 5.1**: Remove all nodes, relationships, and indexes from the database.
- **Exercise 5.2**: Determine how large the datasets are that you will be loading.
- **Exercise 5.3**: Examine the data you will be loading.
- **Exercise 5.4**: Write Cypher code to transform the data you will be loading.
- **Exercise 5.5**: Create the constraints.
- **Exercise 5.6**: Load the `Movie` data.
- **Exercise 5.7**: Load the `Person` data.
- **Exercise 5.8**: Load the data to create the `:DIRECTED` relationships.
- **Exercise 5.9**: Load the data to create the `:ACTED_IN` relationships.
- **Exercise 5.10**: Create the indexes.
- **Exercise 5.11**: Create the Genre nodes and `:IS_GENRE` relationships.

Go to the next page to start this exercise.

Exercise 5.1: Remove all nodes, relationships, and indexes from the database. (Instructions)

To start with a database that contains no data or indexes, run this code:

```
// Remove all indexes and constraints
CALL apoc.schema.assert({},{},true);
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
MATCH (n:Genre) DETACH DELETE n;
```

Note: Although your database does not contain nodes of all of these types, these are the types you will be working with for the remainder of this course and the same "reset" code will be useful later in this course.

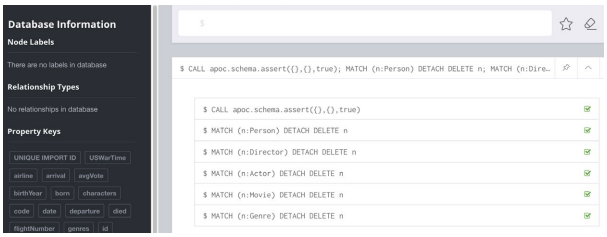
Exercise 5.1: Remove all nodes, relationships, and indexes from the database. (Solution)

To start with a database that contains no data or indexes, run this code:

```
// Remove all indexes and constraints
CALL apoc.schema.assert({},{},true);
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
MATCH (n:Genre) DETACH DELETE n;
```

Note: Although your database does not contain nodes of all of these types, these are the types you will be working with for the remainder of this course and the same "reset" code will be useful later in this course.

The results returned should look like this:



The database should have no nodes and relationships.

Exercise 5.2: Determine how large the datasets are that you will be loading. (Instructions)

The datasets containing the normalized data are at these locations:

https://data.neo4j.com/advanced-cypher/movies1.csv
https://data.neo4j.com/advanced-cypher/people.csv
https://data.neo4j.com/advanced-cypher/roles.csv
https://data.neo4j.com/advanced-cypher/directors.csv

Write Cypher code to return the number of lines in each of these CSV files.

Exercise 5.2: Determine how large the datasets are that you will be loading. (Solution)

The datasets containing the normalized data are at these locations:

https://data.neo4j.com/advanced-cypher/movies1.csv
https://data.neo4j.com/advanced-cypher/people.csv
https://data.neo4j.com/advanced-cypher/roles.csv
https://data.neo4j.com/advanced-cypher/directors.csv

Write Cypher code to return the number of lines in each of these CSV files.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies1.csv' AS rows
WITH count(rows) as MoviesRows
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/people.csv' AS rows
WITH MoviesRows, count(rows) as PeopleRows
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/roles.csv' AS rows
WITH MoviesRows, PeopleRows, count(rows) as RolesRows
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/directors.csv' AS rows
RETURN MoviesRows, PeopleRows, RolesRows, count(rows) as DirectorsRows
```

The results returned should look like this:

The number of rows in these files is < 100K so we should not need any special loading options (like `USING PERIODIC COMMIT`).

Exercise 5.3: Examine the data you will be loading. (Instructions)

Write queries to return the first five rows of each CSV file. Make a note of the header names and if IDs are being used to uniquely identify people and movies.

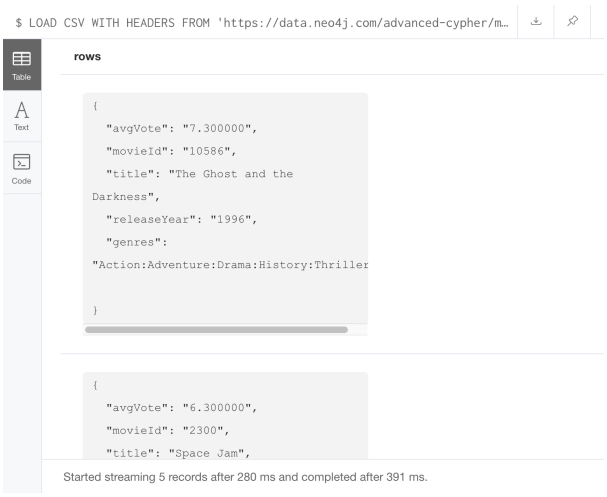
Exercise 5.3: Examine the data you will be loading. (Solution)

Write queries to return the first five rows of each CSV file. Make a note of the header names and if IDs are being used to uniquely identify people and movies.

Here is the solution code for the **movies1.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies1.csv' AS rows
RETURN rows LIMIT 5
```

The results should be:



Note here that each row represents a movie with a unique ID, `movieId`.

Here is the solution code for the **people.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/people.csv' AS rows
RETURN rows LIMIT 5
```

The results should be:

Note hear that each row represents a person with a unique ID, `personId`.

Here is the solution code for the **roles.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/roles.csv' AS rows
RETURN rows LIMIT 5
```

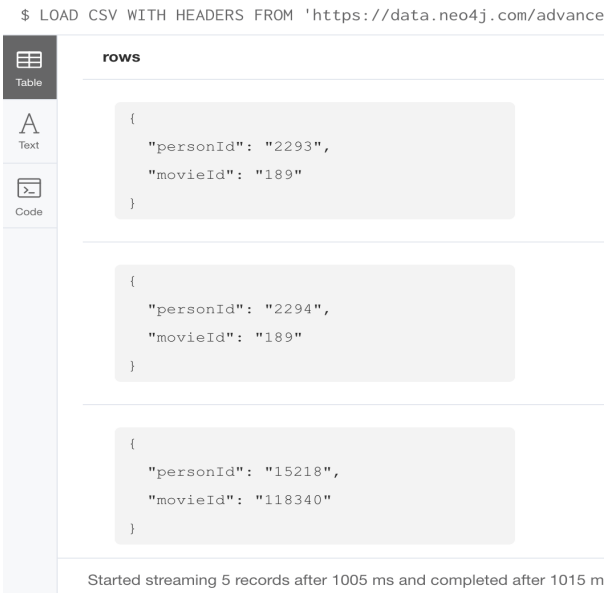
The results should be:

Note here that each row has data for a person, personId and a movie, `movieId`. It is with this **roles.csv** file that the `:ACTED_IN` relationship between a person and a movie will be created in the database.

Here is the solution code for the **directors.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/directors.csv' AS rows
RETURN rows LIMIT 5
```

The results should be:



Note here that each row has data for a person, `personId` and a movie, `movieId`. It is with this **directors.csv** file that the `:DIRECTED` relationship between a person and a movie will be created in the database.

Exercise 5.4: Write Cypher code to transform the data you will be loading. (Instructions)

In examining the data in these CSV files, we want to transform data as follows before adding it to the database:

- In **movies1.csv**: `avgVote` is of type float.
- In **movies1.csv**: `releaseYear` is of type integer.
- In **movies1.csv**: `genres` is is a list of string values.
- In **people.csv**: `birthYear` is of type integer.
- In **people.csv**: `deathYear` is of type integer.

Write Cypher code to transform these values and return the data in the new format. Use `LIMIT 5` again to show the transformation for the first five rows.

Exercise 5.4: Write Cypher code to transform the data you will be loading. (Solution)

In examining the data in these CSV files, we want to transform data as follows before adding it to the database:

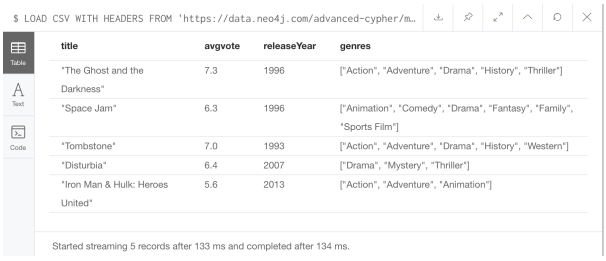
- In **movies1.csv**: `avgVote` is of type float.
- In **movies1.csv**: `releaseYear` is of type integer.
- In **movies1.csv**: `genres` is is a list of string values.
- In **people.csv**: `birthYear` is of type integer.
- In **people.csv**: `deathYear` is of type integer.

Write Cypher code to transform these values and return the data in the new format. Use `LIMIT 5` again to show the transformation for the first five rows.

Here is the solution code for the **movies1.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies1.csv' AS rows
RETURN rows.title as title,
       toFloat(rows.avgVote) as avgvote,
       toInteger(rows.releaseYear) as releaseYear,
       split(rows.genres,":") as genres
       LIMIT 5
```

The results should be:



Here is the solution code for the **people.csv** file:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/people.csv' AS rows
RETURN rows.name as name,
       toInteger(rows.birthYear) as born,
```

```
        toInteger(rows.deathYear) as died
        LIMIT 5
```

The results should be:

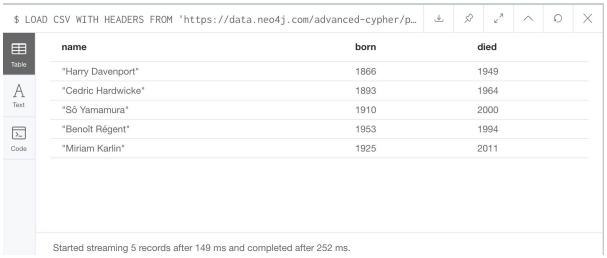
Notice that for the first five rows, these people do not have data for `deathYear`.

Do a query against the dataset to see if there are any people with a value for `deathYear`.

Here is the code:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/people.csv' AS rows
WITH rows WHERE exists(rows.deathYear)
RETURN rows.name as name,
       toInteger(rows.birthYear) as born,
       toInteger(rows.deathYear) as died
       LIMIT 5
```

The results should be:



Exercise 5.5: Create the constraints. (Instructions)

The **movies1.csv** fields will be mapped to `Movie` node properties as follows:

row field	**property**
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

The **people.csv** fields will be mapped to `Person` node properties as follows:

row field	**property**
personId	id
name	name

birthYear	born
deathYear	died

To improve loading when nodes are created using `MERGE`, add uniqueness constraints as follows:

- **Uniqueness constraint on the `id` property of a `Movie` node.**
- **Uniqueness constraint on the `id` property of a `Person` node.**

Exercise 5.5: Create the constraints. (Solution)

The **movies1.csv** fields will be mapped to `Movie` node properties as follows:

row field	**property**
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

The **people.csv** fields will be mapped to `Person` node properties as follows:

row field	**property**
personId	id
name	name
birthYear	born
deathYear	died

To improve loading when nodes are created using `MERGE`, add uniqueness constraints as follows:

- **Uniqueness constraint on the `id` property of a `Movie` node.**
- **Uniqueness constraint on the `id` property of a `Person` node.**

Here is the solution code:

```
CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE
```

The results returned should look like this:

Exercise 5.6: Load the `Movie` data. (Instructions)

The **movies1.csv** fields will be mapped to `Movie` node properties as follows:

row field	property
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Load the movies1.csv file to create the `Movie` nodes in the database.

Exercise 5.6: Load the `Movie` data. (Solution)

The **movies1.csv** fields will be mapped to `Movie` node properties as follows:

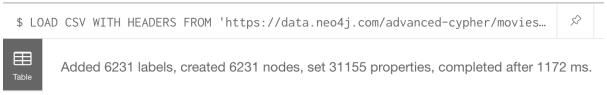
row field	property
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Load the movies1.csv file to create the `Movie` nodes in the database.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies1.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
    ON CREATE SET
          m.title = row.title,
          m.avgVote = toFloat(row.avgVote),
          m.releaseYear = toInteger(row.releaseYear),
          m.genres = split(row.genres,":")
```

The results returned should look like this:



Exercise 5.7: Load the `Person` data. (Instructions)

The **people.csv** fields will be mapped to `Person` node properties as follows:

row field	property
personId	id
name	name
birthYear	born
deathYear	died

Load the people.csv file to create the `Person` nodes in the database.

Exercise 5.7: Load the `Person` data. (Solution)

The **people.csv** fields will be mapped to `Person` node properties as follows:

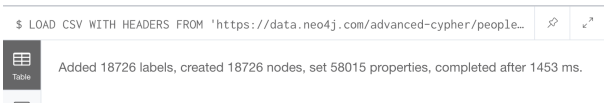
row field	property
personId	id
name	name
birthYear	born
deathYear	died

Load the people.csv file to create the `Person` nodes in the database.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/people.csv' as row

MERGE(person:Person {id: toInteger(row.personId)})
ON CREATE SET person.name = row.name,
              person.born = toInteger(row.birthYear),
              person.died = toInteger(row.deathYear)
```

The results returned should look like this:



Exercise 5.8: Load the data to create the `:DIRECTED` relationships. (Instructions)

Load the directors.csv file to create the relationship between a `Person` node and a `Movie` node in the database. In addition, add the `Director` label to each `Person` node with the `:DIRECTED` relationship.

Exercise 5.8: Load the data to create the `:DIRECTED` relationships. (Solution)

Load the directors.csv file to create the relationship between a `Person` node and a `Movie` node in the database. In addition, add the `Director` label to each `Person` node with the `:DIRECTED` relationship.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/directors.csv' as row

MATCH (movie:Movie {id:toInteger(row.movieId)})
MATCH (person:Person {id: toInteger(row.personId)})
MERGE (person)-[:DIRECTED]->(movie)
ON CREATE SET person:Director
```

The results returned should look like this:

Exercise 5.9: Load the data to create the `:ACTED_IN` relationships. (Instructions)

Load the roles.csv file to create the relationship between a `Person` node and a `Movie` node in the database. In addition, set the `roles` property for the relationship to have the list of characters for the actor. Finally, add the `Actor` label to each `Person` node with the `:ACTED_IN` relationship.

Exercise 5.9: Load the data to create the `:ACTED_IN` relationships. (Solution)

Load the roles.csv file to create the relationship between a `Person` node and a `Movie` node in the database. In addition, set the `roles` property for the relationship to have the list of characters for the actor. Finally, add the `Actor` label to each `Person` node with the `:ACTED_IN` relationship.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/roles.csv' AS row

MATCH  (movie:Movie  {id: toInteger(row.movieId) })
MATCH  (person:Person {id: toInteger(row.personId) })
MERGE  (person)-[r:ACTED_IN]->(movie) ON CREATE SET r.roles = split(coalesce(row.characters,""),
":")
ON CREATE SET person:Actor
```

The results returned should look like this:

Exercise 5.10: Create the indexes. (Instructions)

To improve retrieval performance, add indexes as follows:

- **Index on the `name` property of a `Person` node.**
- **Index on the `title` property of a `Movie` node.**

Exercise 5.5: Create the indexes. (Solution)

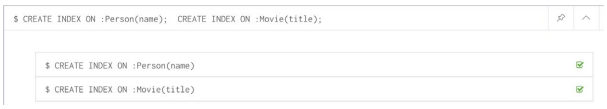
To improve retrieval performance, add indexes as follows:

- **Index on the `name` property of a `Person` node.**
- **Index on the `title` property of a `Movie` node.**

Here is the solution code:

```
CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title)
```

The results returned should look like this:



Exercise 5.11: Create the `Genre` nodes and `:IS_GENRE` relationships. (Instructions)

Although the `Movie` nodes have a property, `genres`, we want a separate node of type `Genre`. Every `Movie` will have a `:IS_GENRE` relationship with one or more `Genre` nodes. A `Genre` node will have a single property, `name`.

First, create a uniqueness constraint for the `name` property for nodes of type `Genre`. Then use the data in the graph to create `Genre` nodes from the `Movie` nodes and add the `:IS_GENRE` relationships between `Movie` nodes and `Genre` nodes. In addition, remove the `genres` property from the `Movie` nodes.

Exercise 5.11: Create the `Genre` nodes and `:IS_GENRE` relationships. (Solution)

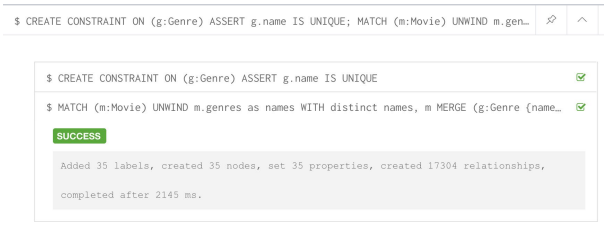
Although the `Movie` nodes have a property, `genres`, we want a separate node of type `Genre`. Every `Movie` will have a `:IS_GENRE` relationship with one or more `Genre` nodes. A `Genre` node will have a single property, `name`.

First, create a uniqueness constraint for the `name` property for nodes of type `Genre`. Then use the data in the graph to create `Genre` nodes from the `Movie` nodes and add the `:IS_GENRE` relationships between `Movie` nodes and `Genre` nodes. In addition, remove the `genres` property from the `Movie` nodes.

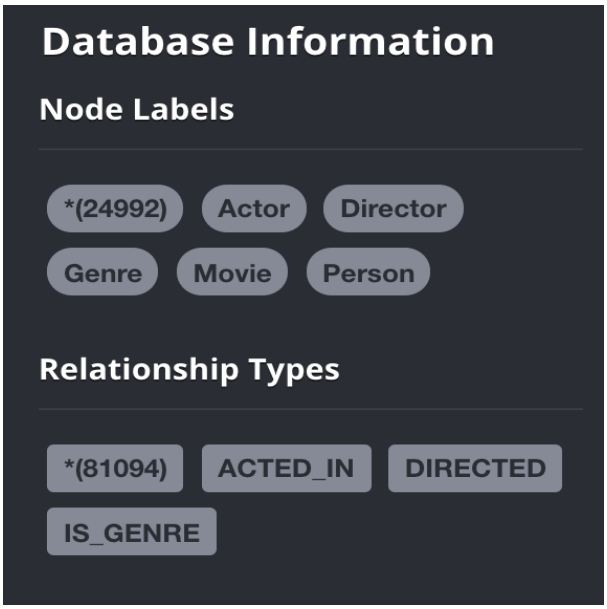
Here is the solution code:

```
CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
SET m.genres = null
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

The results returned should look like this:



Your database should now be as follows:



Exercise 5: Taking it further

1. Perform all of the steps in this exercise as a set of statements (including resetting the database at the beginning).
2. Perform some queries to become familiar with the newly-loaded data.

Exercise 5: Loading normalized data (Summary)

In this exercise, you have written code to load normalized data into a graph and also create nodes from data in the graph.

Continue to Exercise 6

Exercise 6

Exercise 6: Loading denormalized data (Preparations)

Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 6: Loading denormalized data (Overview)

In this exercise, you will first remove all data and indexes from the database so you can start fresh, then you will load denormalized data into the database.

- **Exercise 6.1**: Remove all nodes, relationships, and indexes from the database.
- **Exercise 6.2**: Determine how large the dataset is that you will be loading.
- **Exercise 6.3**: Examine the data you will be loading.
- **Exercise 6.4**: Create the constraints.
- **Exercise 6.5**: Load the movie and person data by merging the movie data from the row and then the person data.
- **Exercise 6.6**: Remove all nodes and relationships from the database and then profile the same load.
- **Exercise 6.7**: Remove all nodes and relationships from the database.
- **Exercise 6.8**: Load and merge the movie data while collecting the person data, then use unwind to merge the person data.
- **Exercise 6.9**: Create the indexes.
- **Exercise 6.10**: Create the `Genre` nodes and `:IS_GENRE` relationships.

Go to the next page to start this exercise.

Exercise 6.1: Remove all nodes, relationships, and indexes from the database. (Instructions)

To start with a database that contains no data or indexes, run this code just like you did in the previous exercise:

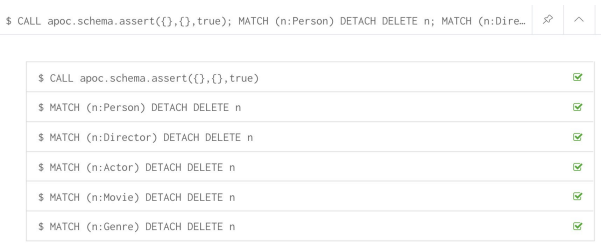
```
// Remove all indexes and constraints
CALL apoc.schema.assert({},{},true);
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
MATCH (n:Genre) DETACH DELETE n
```

Exercise 6.1: Remove all nodes, relationships, and indexes from the database. (Solution)

To start with a database that contains no data or indexes, run this code just like you did in the previous exercise:

```
// Remove all indexes and constraints
CALL apoc.schema.assert({},{},true);
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
MATCH (n:Genre) DETACH DELETE n
```

The results returned should look like this:



The database should have no nodes and relationships.

Exercise 6.2: Determine how large the dataset is that you will be loading. (Instructions)

The dataset containing the denormalized data is found here:

https://data.neo4j.com/advanced-cypher/movies2.csv

Write Cypher code to return the number of lines in this file.

Exercise 6.2: Determine how large the dataset is that you will be loading. (Solution)

The dataset containing the denormalized data is found here:

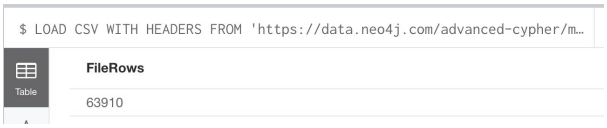
https://data.neo4j.com/advanced-cypher/movies2.csv

Write Cypher code to return the number of lines in this file.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS rows
RETURN count(rows) as FileRows
```

The results returned should look like this:



The number of rows in this file is < 100K so we should not need any special loading options (like `USING PERIODIC COMMIT`).

Exercise 6.3: Examine the data you will be loading. (Instructions)

Since this is denormalized data, you will need to examine more rows to understand how the data has been normalized.

Write a query to return the first 50 rows of the CSV file. Make a note of the header names and if IDs are being used to uniquely identify people and movies.

Exercise 6.3: Examine the data you will be loading. (Solution)

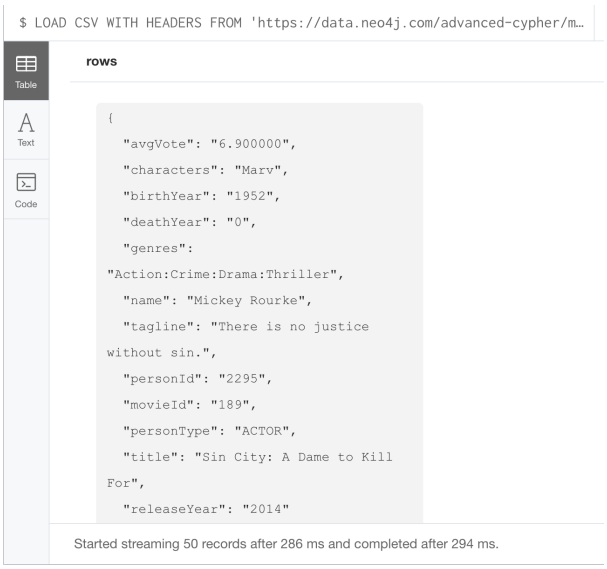
Since this is denormalized data, you will need to examine more rows to understand how the data has been normalized.

Write a query to return the first 50 rows of the CSV file. Make a note of the header names and if IDs are being used to uniquely identify people and movies.

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS rows
RETURN rows LIMIT 50
```

The results should be:



Notice that each row has movie data and person data. Each row uses a `movieId` and `personId` to uniquely identify a movie or person. A row also has a field, `personType`, where the field will either have a value, "ACTOR" or a value, "DIRECTOR".

Exercise 6.4: Create the constraints. (Instructions)

The **movies2.csv** fields will be mapped to `Movie` and `Person` node properties as follows:

For `Movie` nodes:

row field	**property**
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Note: The tagline data will not be loaded.

For `Person` nodes:

row field	**property**
personId	id
name	name
birthYear	born
deathYear	died

To improve loading when nodes are created using `MERGE`, add uniqueness constraints as follows, just as you did for the normalized data:

- **Uniqueness constraint on the `id` property of a `Movie` node.**

- **Uniqueness constraint on the `id` property of a `Person` node.**

Exercise 6.4: Create the constraints. (Solution)

The **movies2.csv** fields will be mapped to `Movie` and `Person` node properties as follows:

For `Movie` nodes:

row field	property
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Note: The tagline data will not be loaded.

For `Person` nodes:

row field	property
personId	id
name	name
birthYear	born
deathYear	died

To improve loading when nodes are created using `MERGE`, add uniqueness constraints as follows, just as you did for the normalized data:

- **Uniqueness constraint on the `id` property of a `Movie` node.**
- **Uniqueness constraint on the `id` property of a `Person` node.**

Here is the solution code:

```
CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE
```

The results returned should look like this:

Exercise 6.5: Load the movie and person data by merging the movie data from the row and then the person data. (Instructions)

The **movies2.csv** fields will be mapped to `Movie` and `Person` node properties as follows:

For `Movie` nodes:

row field	property
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Note: The tagline data will not be loaded.

For `Person` nodes:

row field	property
personId	id
name	name
birthYear	born
deathYear	died

Load the movies2.csv file to:

1. **Use `MERGE` to create the `Movie` node.**
2. **Use `MERGE` to create the `Person` node.**
3. **Use conditional processing to create the relationships, `:DIRECTED` and `:ACTED_IN` (using apoc.do.when).**

Exercise 6.5: Load the movie and person data by merging the movie data from the row and then the person data. (Solution)

The **movies2.csv** fields will be mapped to `Movie` and `Person` node properties as follows:

For `Movie` nodes:

row field	property
movieId	id
title	title
avgVote	avgVote
releaseYear	releaseYear
genres	genres

Note: The tagline data will not be loaded.

For `Person` nodes:

row field	property
personId	id
name	name
birthYear	born

deathYear died

Load the movies2.csv file to:

1. **Use `MERGE` to create the `Movie` node.**
2. **Use `MERGE` to create the `Person` node.**
3. **Use conditional processing to create the relationships, `:DIRECTED` and `:ACTED_IN` (using apoc.do.when).**

Here is the solution code:

```
LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
      p.died = toInteger(row.deathYear)
WITH row, m, p
CALL apoc.do.when(row.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: split(coalesce(row.characters,''), ':')}]->(m)
          ON CREATE SET p:Actor",
     "MERGE (p)-[:DIRECTED]->(m)
          ON CREATE SET p:Director",
      {row:row, m:m, p:p}) YIELD value AS value
SET p:Person  // cannot end query with APOC call
```

The results returned should look like this:

Exercise 6.6: Remove all nodes and relationships from the database and then profile the same load. (Instructions)

1. Execute this code to remove all nodes and relationships in the database:

```
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n
```

2. Profile the previously executed load.

Exercise 6.6: Remove all nodes and relationships from the database and then profile the same load. (Solution)

1. Execute this code to remove all nodes and relationships in the database:

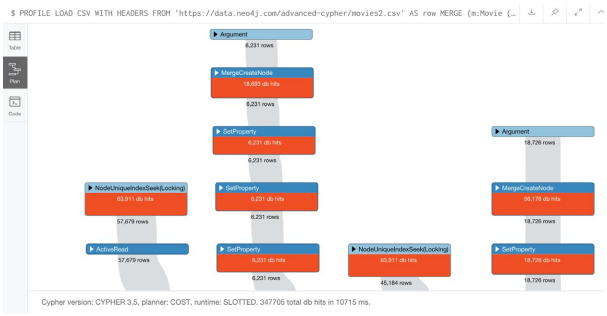
```
MATCH (n:Person) DETACH DELETE n;
```

```
MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n
```

2. Profile the previously executed load.

```
PROFILE LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
      p.died = toInteger(row.deathYear)
WITH row, m, p
CALL apoc.do.when(row.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: split(coalesce(row.characters,''), ':')}]->(m)
          ON CREATE SET p:Actor",
     "MERGE (p)-[:DIRECTED]->(m)
          ON CREATE SET p:Director",
      {row:row, m:m, p:p}) YIELD value AS value
SET p:Person  // cannot end query with APOC call
```

The results returned should look like this:



This load required 347,703 DB hits.

Exercise 6.7: Remove all nodes and relationships from the database. (Instructions)

Next, you will try another alternative for loading the denormalized data so you should execute this code the remove all existing nodes and relationships:

```
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n
```

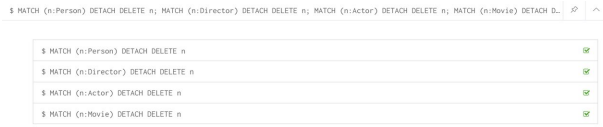
Exercise 6.7: Remove all nodes and relationships from the database. (Solution)

Next, you will try another alternative for loading the denormalized data so you should execute this code the remove all existing nodes and relationships:

```
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
```

```
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n
```

The results returned should look like this:



The database should have no nodes and relationships.

Exercise 6.8: Load and merge the movie data while collecting the person data, then use unwind to merge the person data. (Instructions)

Load the movie data while collecting the person data, then use unwind to merge the person data. Just like you did previously, `CALL apoc.doc.when()` to add the relationships. Profile this load.

Exercise 6.8: Load and merge the movie data while collecting the person data, then use unwind to merge the person data. (Solution)

Load the movie data while collecting the person data, then use unwind to merge the person data. Just like you did previously, `CALL apoc.doc.when()` to add the relationships. Profile this load.

Here is the solution code:

```
PROFILE LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId, row.title as title, row.genres as genres,
toInteger(row.releaseYear) as releaseYear, toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear),
died:toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count()  // cannot end query with APOC call
```

The results returned should look like this:

This method of loading the data required 290,026 DB hits, which is better than the previous method. Collecting the results and unwinding them is much more efficient.

Exercise 6.9: Create the indexes. (Instructions)

To improve retrieval performance, create indexes as follows, just as you did for the normalized data:

- **Index on the `name` property of a `Person` node.**
- **Index on the `title` property of a `Movie` node.**

Exercise 6.9: Create the indexes and constraints. (Solution)

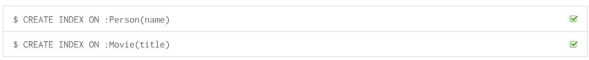
To improve retrieval performance, create indexes as follows, just as you did for the normalized data:

- **Index on the `name` property of a `Person` node.**
- **Index on the `title` property of a `Movie` node.**

Here is the solution code:

```
CREATE INDEX ON :Person(name);
CREATE INDEX ON :Movie(title)
```

The results returned should look like this:



Exercise 6.10: Create the `Genre` nodes and `:IS_GENRE` relationships. (Instructions)

Just as you did for the load of the normalized data, create a uniqueness constraint for the `name` property for nodes of type `Genre`. Then use the data in the graph to create `Genre` nodes from the `Movie` nodes and add the `:IS_GENRE` relationships between `Movie` nodes and `Genre` nodes. In addition, remove the `genres` property from the `Movie` nodes.

Exercise 6.10: Create the `Genre` nodes and `:IS_GENRE` relationships. (Solution)

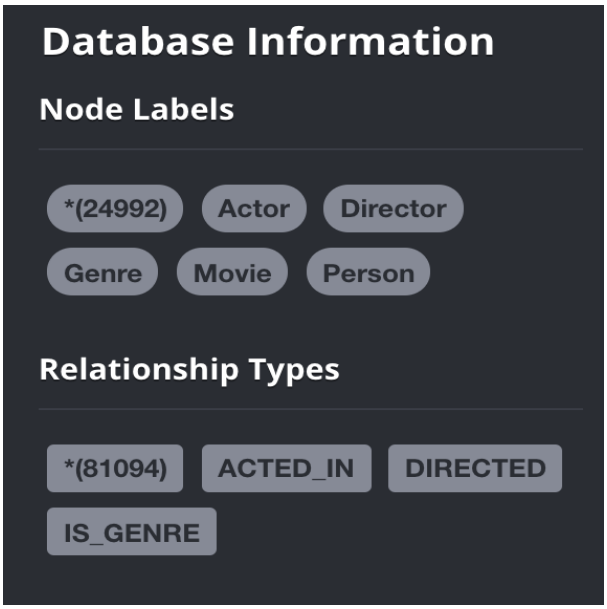
Just as you did for the load of the normalized data, create a uniqueness constraint for the `name` property for nodes of type `Genre`. Then use the data in the graph to create `Genre` nodes from the `Movie` nodes and add the `:IS_GENRE` relationships between `Movie` nodes and `Genre` nodes. In addition, remove the `genres` property from the `Movie` nodes.

Here is the solution code:

```
CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
SET m.genres = null
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

The results returned should look like this:

Your database should now be as follows:



Exercise 6: Taking it further

Perform some queries to become familiar with the newly-loaded data.

Exercise 6: Loading denormalized data (Summary)

In this exercise, you have written code to load denormalized data into a graph and also create nodes from data in the graph. You have seen that one way that you can optimize the load is to save the data during one pass into a collection and unwind the data for processing.

Continue to Exercise 7

Exercise 7

Exercise 7: Loading large datasets (Preparations)

Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

Exercise 7: Loading large datasets (Overview)

In the previous exercises you loaded both normalized and denormalized data. The data that you loaded was well within the size range of data that Neo4j can handle (less than 100K rows). Rather than attempting to load an even larger dataset, in this exercise, you will reduce the default virtual memory used by Neo4j. The effect of lowering this virtual memory setting for Neo4j will allow you to simulate an out of memory condition that you would likely run into when loading a large dataset.

In this exercise, you will:

- **Exercise 7.1**: Remove all nodes and relationships from the database.
- **Exercise 7.2**: Temporarily modify the virtual memory used by Neo4j.
- **Exercise 7.3**: Attempt to load the denormalized data for movies and people.
- **Exercise 7.4**: Profile this query to use `PERIODIC COMMIT` to load the data.
- **Exercise 7.5**: Attempt to load the data using `PERIODIC COMMIT`, but with an eager operator.
- **Exercise 7.6**: Increase the virtual memory to show the profile of the query.

Go to the next page to start this exercise.

Exercise 7.1: Remove all nodes and relationships from the database. (Instructions only)

To start with a database that contains no data, run this code just like you did in the previous exercise:

```
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
MATCH (n:Genre) DETACH DELETE n
```

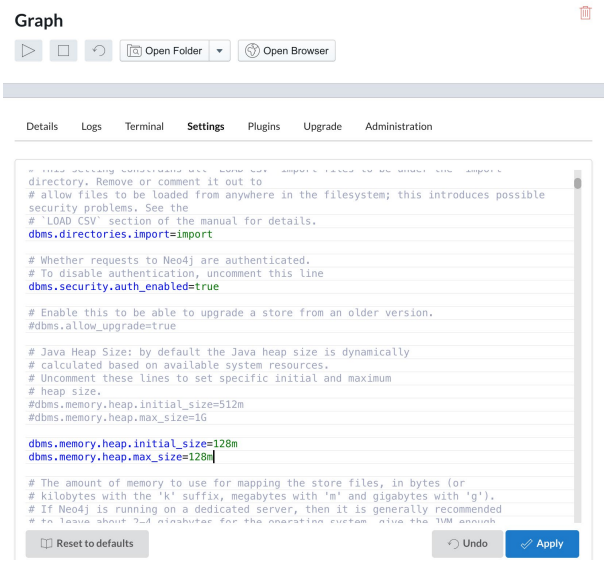
The database should have no nodes and relationships.

Exercise 7.2: Temporarily modify the virtual memory used by Neo4j. (Instructions only)

For this exercise, you will set the virtual memory used by Neo4j to an artificially low value. This will enable you to simulate an out of memory error.

1. **Stop the database in Neo4j Desktop.**
2. **Click the Manage button for the database.**
3. **Click the Settings for the database. This opens an editable view of the neo4j.conf file for the database.**
4. **The default virtual memory for Neo4j is:**
    a. **dbms.memory.heap.initial_size=512m**
    b. **dbms.memory.heap.max_size=1G**
5. **Modify both of these values to be 128m.**
6. **Click Apply.**
7. **Start the database**

The settings should look like this:



Exercise 7.3: Attempt to load the denormalized data for movies and people. (Instructions)

Run the Cypher code you wrote to load the denormalized data as two `MERGE` clauses with no collection of data. Do not create the relationships.

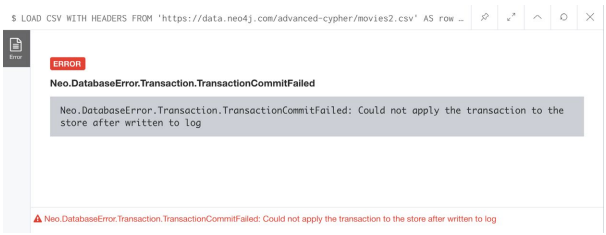
Exercise 7.3: Attempt to load the denormalized data for movies and people. (Solution)

Run the Cypher code you wrote to load the denormalized data as two `MERGE` clauses with no collection of data. Do not create the relationships.

Here is the code:

```
LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
      p.died = toInteger(row.deathYear)
```

The results should be:



This load which you were able to do previously with the default virtual memory settings now fails due to out of memory. No data is written to the database.

Exercise 7.4: Profile this query to use `PERIODIC COMMIT` to load the data. (Instructions)

Repeat the previous execution, but add the `PERIODIC COMMIT` clause and profile the query.

Note: You may receive an error that the database needs to be restarted due to the previous error, in which case you should restart the database.

Exercise 7.4: Profile this query to use `PERIODIC COMMIT` to load the data. (Solution)

Repeat the previous execution, but add the `PERIODIC COMMIT` clause and profile the query.

Note: You may receive an error that the database needs to be restarted due to the previous error, in which case you should restart the database.

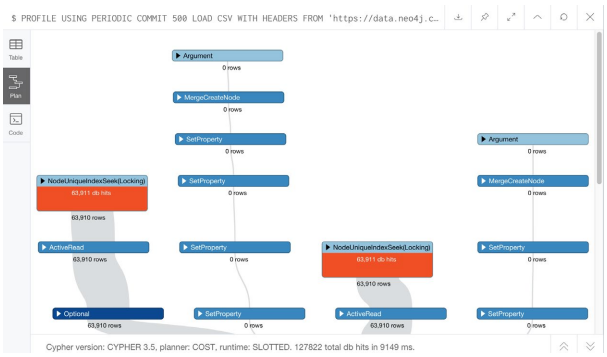
Here is the solution code:

```
PROFILE USING PERIODIC COMMIT 500 LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-
cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
```

```
    p.died = toInteger(row.deathYear)
```

This load is successful because we have specified `USING PERIODIC COMMIT` which will enable the transactions to succeed with a lower virtual memory available.

The results returned should look like this:



Exercise 7.5: Attempt to load the data using `PERIODIC COMMIT`, but with an eager operator. (Instructions)

Since you will be loading the data again, delete all of the data in the database by executing this code:

```
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
```

The database should have no nodes and relationships.

Repeat the last load statement with a profile, but add a `RETURN` clause to return the movie titles and order them by title.

Exercise 7.5: Attempt to load the data using `PERIODIC COMMIT`, but with an eager operator. (Solution)

Since you will be loading the data again, delete all of the data in the database by executing this code:

```
// Remove all nodes/relationships
MATCH (n:Person) DETACH DELETE n;
MATCH (n:Director) DETACH DELETE n;
MATCH (n:Actor) DETACH DELETE n;
MATCH (n:Movie) DETACH DELETE n;
```

The database should have no nodes and relationships.

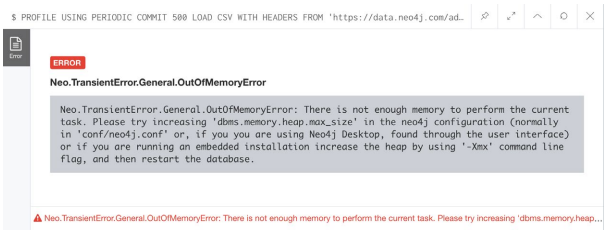
Repeat the last load statement with a profile, but add a `RETURN` clause to return the movie titles and order them by title.

Here is the solution code:

```
PROFILE USING PERIODIC COMMIT 500 LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-
cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
```

```
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
      p.died = toInteger(row.deathYear)
RETURN m.title ORDER BY m.title
```

The results returned should look like this:



Here we see that even with `USING PERIODIC COMMIT`, the load fails. This is because the `ORDER BY` in the `RETURN` clause requires eager operators to sort the data before returning. `PERIODIC COMMIT` was disabled.

Exercise 7.6: Increase the virtual memory to show the profile of the query. (Instructions)

1. **Stop the database in Neo4j Desktop.**
2. **Click the Manage button for the database.**
3. **Click the Settings for the database. This opens an editable view of the neo4j.conf file for the database.**
4. **Set the virtual memory back to its default:**
    a. **dbms.memory.heap.initial_size=512m**
    b. **dbms.memory.heap.max_size=1G**
5. **Click Apply.**
6. **Start the database**
7. **Profile the previously executed load.**

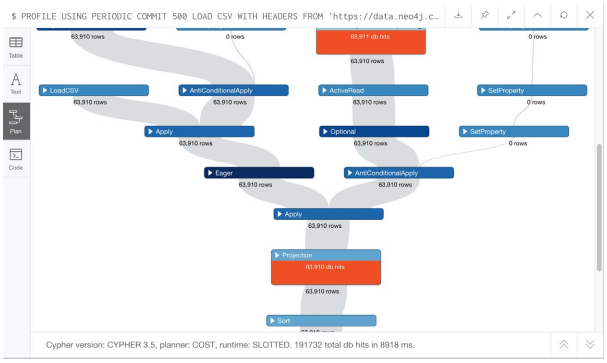
Exercise 7.6: Increase the virtual memory to show the profile of the query. (Solution)

1. **Stop the database in Neo4j Desktop.**
2. **Click the Manage button for the database.**
3. **Click the Settings for the database. This opens an editable view of the neo4j.conf file for the database.**
4. **Set the virtual memory back to its default:**
    a. **dbms.memory.heap.initial_size=512m**
    b. **dbms.memory.heap.max_size=1G**

5. **Click Apply.**
6. **Start the database**
7. **Profile the previously executed load.**

```
PROFILE USING PERIODIC COMMIT 500 LOAD CSV WITH HEADERS FROM 'https://data.neo4j.com/advanced-
cypher/movies2.csv' AS row
MERGE (m:Movie {id:toInteger(row.movieId)})
   ON CREATE SET m.title=row.title, m.avgVote=toFloat(row.avgVote),
      m.releaseYear=toInteger(row.releaseYear), m.genres=split(row.genres,":")
MERGE (p:Person {id: toInteger(row.personId)})
   ON CREATE SET p.name = row.name, p.born = toInteger(row.birthYear),
      p.died = toInteger(row.deathYear)
RETURN m.title ORDER BY m.title
```

The results returned should look like this:



The load was successful because the virtual memory was increased. `USING PERIODIC COMMIT` was not used due to the eager operator for sorting the results.

Exercise 7: Taking it further

Profile the load using `apoc.periodic.iterate` and compare it with your other profile results.

Exercise 7: Loading large datasets (Summary)

In this exercise, you lowered the virtual memory required for Neo4j to emulate what an out of memory error would be if you were to attempt to load a large dataset. For large dataset, you must either use PERIODIC COMMIT or you must use apoc.periodic.iterate() combined with apoc.load.csv() to load large datasets. Continue to Exercise 8

Exercise 8

Exercise 8: Aggregating data (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

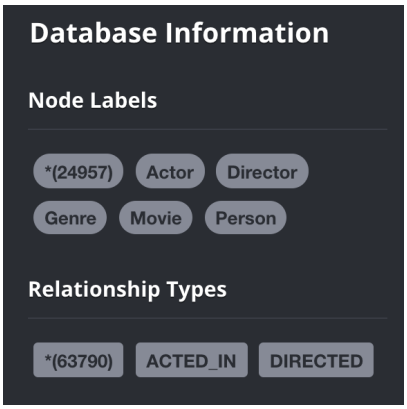
2. Ensure that the virtual memory for Neo4j is set back to its defaults: dbms.memory.heap.initial_size=512m dbms.memory.heap.max_size=1G

3. This exercise assumes that you have a fully-loaded movie database without the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
```

```
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
               ON CREATE SET p:Actor",
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
```

This is what you should see when you click the database icon  .



Exercise 8: Aggregating data (Overview)

In this exercise, you will gain more experience aggregating data and using aggregated data with Cypher.

In this exercise, you will:

- **Exercise 8.1**: Profile the collection of data.
- **Exercise 8.2**: Use APOC to work with collections.
- **Exercise 8.3**: Use the properties of a relationship to create a map.
- **Exercise 8.4**: Use pattern comprehension to customize data returned.
- **Exercise 8.5**: Use APOC to group data returned.

Go to the next page to start this exercise.

Exercise 8.1: Profile the collection of data. (Instructions)

1. Profile and write a query to return any five movies and their ratings (avgVote).

2. Profile and write a query to return the top five movies based upon their ratings. Do not use APOC.

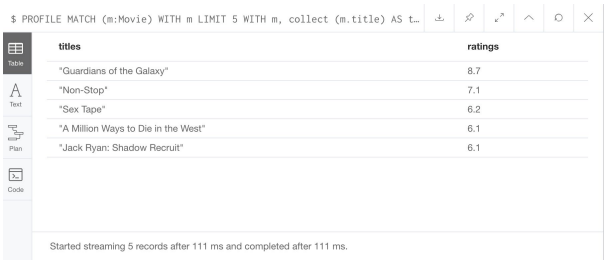
Exercise 8.1: Profile the collection of data. (Solution)

1. Profile and write a query to return any five movies and their ratings (avgVote).

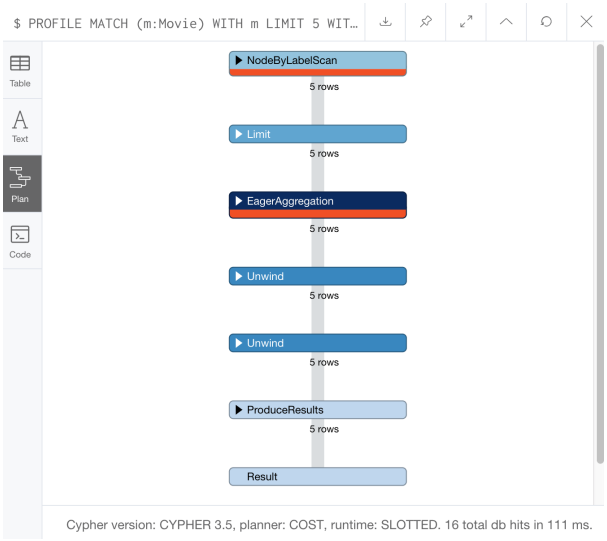
Here is the solution code:

```
PROFILE MATCH (m:Movie)
WITH m LIMIT 5
WITH m, collect (m.title) AS t, collect(m.avgVote) AS r
UNWIND t as titles
UNWIND r as ratings
RETURN titles, ratings
```

The result of performing this code should look something like this:



The profile is:



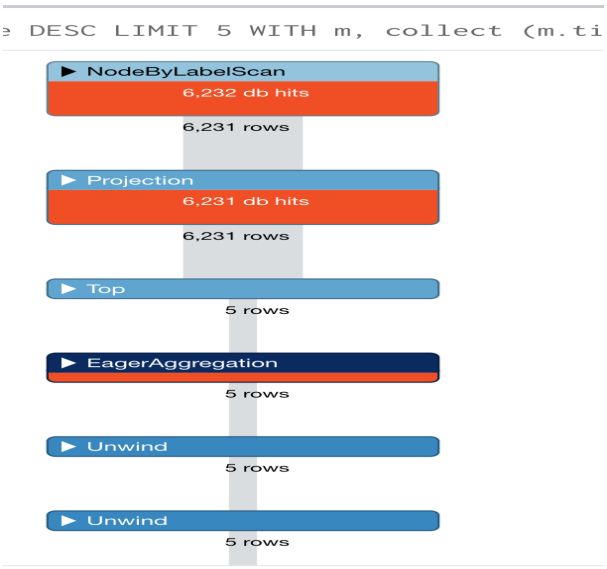
2. Profile and write a query to return the top five movies based upon their ratings. Do not use APOC.

Here is the solution code:

```
PROFILE MATCH (m:Movie)
WITH m ORDER BY m.avgVote DESC LIMIT 5
WITH m, collect (m.title) AS t, collect(m.avgVote) AS r
UNWIND t as titles
UNWIND r as ratings
RETURN titles, ratings
```

The result of performing this code should be:

The profile is:



Because the data needed to be sorted, all records needed to be read before they could be limited to 5.

Exercise 8.2: Use APOC to work with collections. (Instructions)

Here is Cypher code to collect the list of genres from each movie.

```
MATCH (m:Movie)
WITH collect(m.genres) AS genres
RETURN genres, size(genres)
```

1. Execute this code.

2. Rewrite this code to use APOC to remove the embedded lists and also remove duplicates.

Exercise 8.2: Use APOC to work with collections. (Solution)

Here is Cypher code to collect the list of genres from each movie.

```
MATCH (m:Movie)
WITH collect(m.genres) AS genres
RETURN genres, size(genres)
```

1. Execute this code.

The result returned should be:

2. Rewrite this code to use APOC to remove the embedded lists and also remove duplicates.

Here is the solution code:

```
MATCH (m:Movie)
WITH collect(m.genres) AS genres
// remove embedded lists
WITH apoc.coll.flatten(genres) AS allGenres
// remove duplicates
WITH apoc.coll.toSet(allGenres) AS genreList
RETURN genreList, size(genreList)
```

The result returned should be:

Exercise 8.3: Use the properties of a relationship to create a map. (Instructions)

Suppose you want to create a map for every Actor which looks as follows:

```
{
  name: <name of actor>,
  born: <year actor born>,
  roles: [roles that this actor has played in all of his/her movies]
  movies: [titles of the movies this actor has acted in]
}
```

Write a Cypher query to return this data as a map for every actor who was born after the year 1970.

Exercise 8.3: Use the properties of a relationship to create a map. (Solution)

Suppose you want to create a map for every Actor which looks as follows:

```
{
  name: <name of actor>,
```

```
  born: <year actor born>,
  roles: [roles that this actor has played in all of his/her movies]
  movies: [titles of the movies this actor has acted in]
}
```

Write a Cypher query to return this data as a map for every actor who was born after the year 1970.

Here is the solution code:

```
MATCH (m:Movie)<-[act:ACTED_IN]-(p:Person)
WHERE p.born > 1970
WITH p, collect(act.roles) AS roleList, collect(m) AS movieList
WITH p, movieList, apoc.coll.flatten(roleList) AS allRoles
// remove duplicates
WITH p,  movieList, apoc.coll.toSet(allRoles) AS trimmedRoleList
RETURN p { .name, .born,  roles: trimmedRoleList, movies: [x in movieList | x.title] } AS actor
```

The results should be:

This load which you were able to do previously with the default virtual memory settings now fails due to out of memory. No data is written to the database.

Exercise 8.4: Use pattern comprehension to customize data returned. (Instructions)

1. Execute this code to add the `Genre` nodes and relationships to the graph:

```
MATCH (m:Movie)
UNWIND m.genres as names
WITH DISTINCT names, m
SET m.genres = null
MERGE (g:Genre {name:names})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

2. Write a query that retrieves all Genre nodes and returns the following map:

```
{
Genre: <genre name>,
Movies: [<titles of movies for this genre released in the year 2010>]
```

```
}
{
<next genre>}
}
```

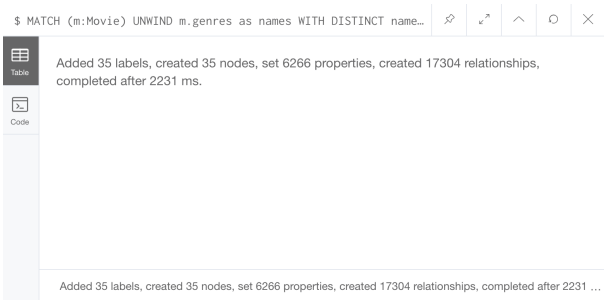
Use pattern comprehension for the `:IS_GENRE` relationship to assemble the list of movie titles with the release year of 2010.

Exercise 8.4: Use pattern comprehension to customize data returned. (Solution)

1. Execute this code to add the `Genre` nodes and relationships to the graph:

```
MATCH (m:Movie)
UNWIND m.genres as names
WITH DISTINCT names, m
SET m.genres = null
MERGE (g:Genre {name:names})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

The results returned should look like this:



2. Write a query that retrieves all Genre nodes and returns the following map:

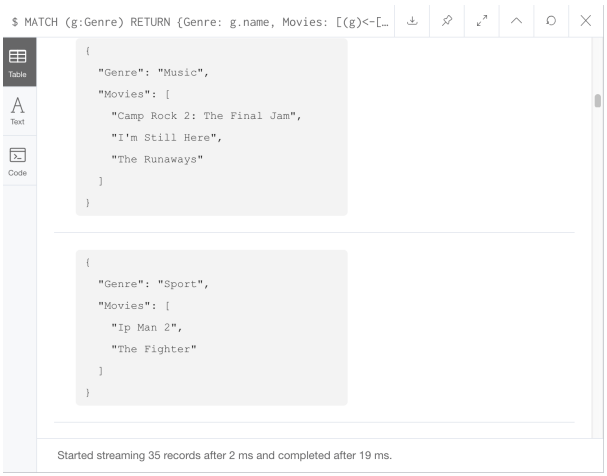
```
{
Genre: <genre name>,
Movies: [<titles of movies for this genre released in the year 2010>]
}
{
<next genre>}
}
```

Use pattern comprehension for the `:IS_GENRE` relationship to assemble the list of movie titles with the release year of 2010.

Here is the solution code:

```
MATCH (g:Genre)
RETURN  {Genre: g.name, Movies: [(g)<-[:IS_GENRE]-(m) WHERE m.releaseYear = 2010| m.title]}
```

Here are the results returned:



Exercise 8.5: Use APOC to group data returned. (Instructions)

Write a query that retrieves all directors born after 1980 and returns their properties grouped by the year they were born.

Exercise 8.5: Use APOC to group data returned. (Solution)

Write a query that retrieves all directors born after 1980 and returns their properties grouped by the year they were born.

Here is the solution code:

```
MATCH (d:Director)
WHERE d.born > 1980
WITH collect(properties(d)) AS directors
RETURN apoc.map.groupByMulti(directors,"born")
```

The results returned should look like this:

Exercise 8: Taking it further

1. Explore and write other queries that use APOC map functions.
2. PROFILE some of the queries you have written and see if you can improve them.

Exercise 8: Aggregating data (Summary)

In this exercise, you gained more experience aggregating data and using aggregated data with Cypher. Continue to Exercise 9

Exercise 9

Exercise 9: Iteration and conditional processing (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

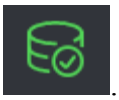
If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

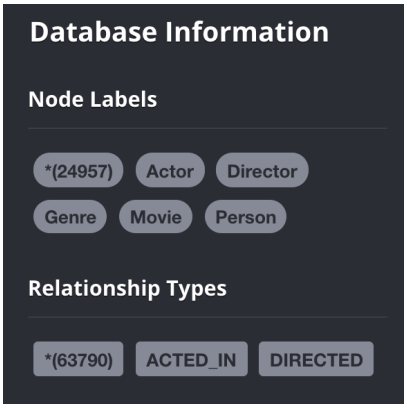
1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database without the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
```

This is what you should see when you click the database icon  .



Exercise 9: Iteration and conditional processing (Overview)

In this exercise, you will gain experience with iterating and writing code to perform conditional processing.

In this exercise, you will:

- **Exercise 9.1**: Collect genres data from the database to create the Genre nodes using `FOREACH`.
- **Exercise 9.2**: Use `CASE` for conditionally returning a value.
- **Exercise 9.3**: Use `CASE` and `FOREACH` to perform conditional updates.

Go to the next page to start this exercise.

Exercise 9.1: Collect genres data from the database to create the Genre nodes using `FOREACH`. (Instructions)

Previously, you wrote code to create the `Genre` nodes as follows:

```
MATCH (m:Movie)
UNWIND m.genres as names
WITH DISTINCT names, m
SET m.genres = null
MERGE (g:Genre {name:names})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

This code created the names collection. It then processed each distinct name for a genre to create the `Genre` nodes and the `:IS_GENRE` relationships.

Rewrite this code to use `FOREACH` to process each `Movie` node, create the `Genre` nodes, and the `:IS_GENRE`

relationships.

Exercise 9.1: Collect genres data from the database to create the Genre nodes using FOREACH. (Solution)

Previously, you wrote code to create the `Genre` nodes as follows:

```
MATCH (m:Movie)
UNWIND m.genres as names
WITH DISTINCT names, m
SET m.genres = null
MERGE (g:Genre {name:names})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

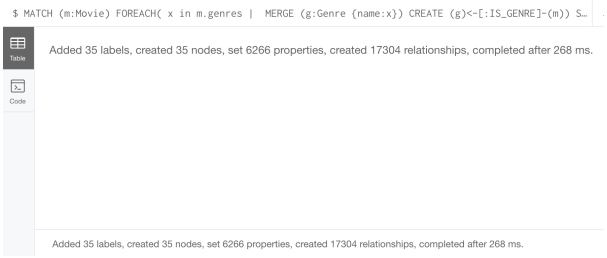
This code created the names collection. It then processed each distinct name for a genre to create the `Genre` nodes and the `:IS_GENRE` relationships.

Rewrite this code to use FOREACH to process each Movie node, create the Genre nodes, and the :IS_GENRE relationships.

Here is the solution code:

```
MATCH (m:Movie)
FOREACH( x in m.genres | MERGE (g:Genre {name:x}) CREATE (g)<-[:IS_GENRE]-(m))
SET m.genres = null
```

The result of executing this code should be:



Exercise 9.2: Use CASE for conditionally returning a value. (Instructions)

You want to modify every movie in the database to add a property, `multiDirectors` which will have a value of `true` or `false`.

1. There are many ways to do this. Write the code to transform each Movie node using a CASE clause where you test the number of directors for each Movie node.

2. Query the database using this new property to determine how many movies have multiple directors.

Exercise 9.2: Use CASE for conditionally returning a value. (Solution)

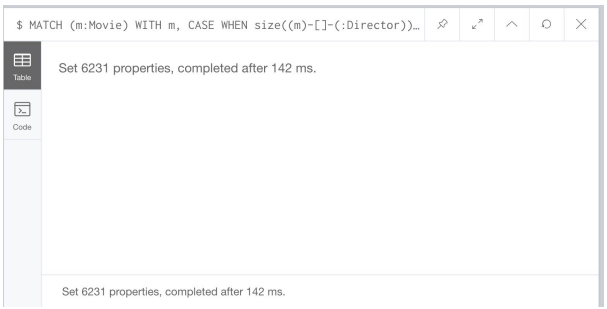
You want to modify every movie in the database to add a property, `multiDirectors` which will have a value of `true` or `false`.

1. There are many ways to do this. Write the code to transform each Movie node using a CASE clause where you test the number of directors for each Movie node.

Here is the solution code:

```
MATCH (m:Movie)
WITH m,
CASE
   WHEN size((m)-[]-(:Director)) > 1 THEN true
   ELSE false
END
   AS md
WITH m, md SET m.multiDirectors = md
```

The result of executing this code should be:

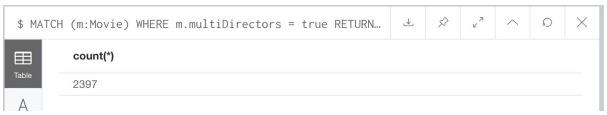


2. Query the database using this new property to determine how many movies have multiple directors.

Here is the solution code:

```
MATCH (m:Movie)
WHERE m.multiDirectors = true
RETURN count(*)
```

The result of executing this code should be:



Exercise 9.3: Use `CASE` and `FOREACH` to perform conditional updates. (Instructions)

1. Rewrite the previous code to use `CASE` and `FOREACH` together to update the `Movie` node in the `FOREACH` clause. Set the values to `"yes"`/`"no"` rather than `true`/`false`.

2. Query the database to determine how many movies have multiple directors.

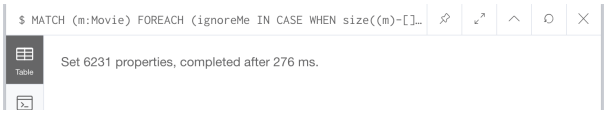
Exercise 9.3: Use `CASE` and `FOREACH` to perform conditional updates. (Solution)

1. Rewrite the previous code to use `CASE` and `FOREACH` together to update the `Movie` node in the `FOREACH` clause. Set the values to `"yes"`/`"no"` rather than `true`/`false`.

Here is the solution code:

```
MATCH (m:Movie)
FOREACH (ignoreMe IN CASE WHEN size((m)-[]-(:Director)) > 1 THEN [1] ELSE [] END |
   SET m.multiDirectors = "yes")
FOREACH (ignoreMe IN CASE WHEN size((m)-[]-(:Director)) <= 1 THEN [1] ELSE [] END |
   SET m.multiDirectors = "no")
```

The result should be:

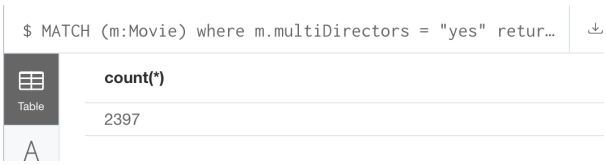


2. Query the database to determine how many movies have multiple directors.

Here is the solution code:

```
MATCH (m:Movie)
WHERE m.multiDirectors = "yes"
RETURN count(*)
```

The result of executing this code should be:



Exercise 9: Taking it further

1. Try using `UNWIND` as an alternative to `FOREACH`
2. Profile your code to see which alternatives are better in terms of DB hits.

Exercise 9: Iteration and conditional processing (Summary)

In this exercise, you gained experience performing conditional processing with Cypher. Continue to Exercise 10

Exercise 10

Exercise 10: Working with DB stats and the count store (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

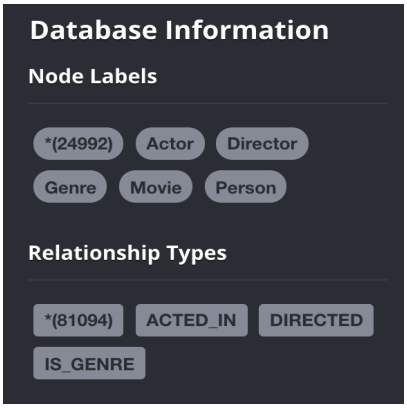
1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

This is what you should see when you click the database icon  .



Exercise 10: Working with DB stats and the count store (Overview)

In this exercise, you will view DB stats and use the count store in your queries.

In this exercise, you will:

- **Exercise 10.1**: Modify a node `MATCH` query to use the count store.
- **Exercise 10.2**: Modify a relationship match query to use the count store.
- **Exercise 10.3**: Modify a query to use `size()` instead of `count()`.
- **Exercise 10.4**: Query the count stores in the database.

Go to the next page to start this exercise.

Exercise 10.1: Modify a node `MATCH` query to use the count store. (Instructions)

Here is a query to count the number of nodes that are `Person` and `Actor` nodes.

```
MATCH (:Actor:Person) RETURN count(*)
```

Profile it and then modify this query to use the count store.

Exercise 10.1: Modify a node `MATCH` query to use the count store. (Solution)

Here is a query to count the number of nodes that are `Person` and `Actor` nodes.

```
MATCH (:Actor:Person) RETURN count(*)
```

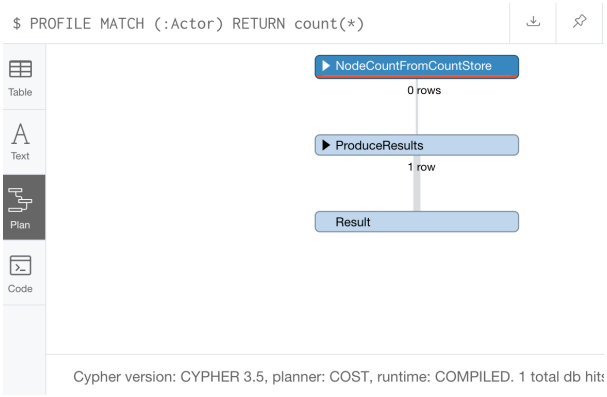
Profile it and then modify this query to use the count store.

Since a `Person` node will always have at least one other type associated with it, you can simply get the count of the nodes with the `Actor` label to take advantage of using the count store.

Here is the solution code:

```
PROFILE MATCH (:Actor) RETURN count(*)
```

The result of executing this code should be:



Exercise 10.2: Modify a relationship match query to use the count store. (Instructions)

Here is a query to count the number of `:DIRECTED` relationships in the graph.

```
MATCH ()-[:DIRECTED]-() RETURN count(*)
```

Profile it and then modify this query to use the count store.

Exercise 10.2: Modify a relationship match query to use the count store. (Solution)

Here is a query to count the number of `:DIRECTED` relationships in the graph.

```
MATCH ()-[:DIRECTED]-() RETURN count(*)
```

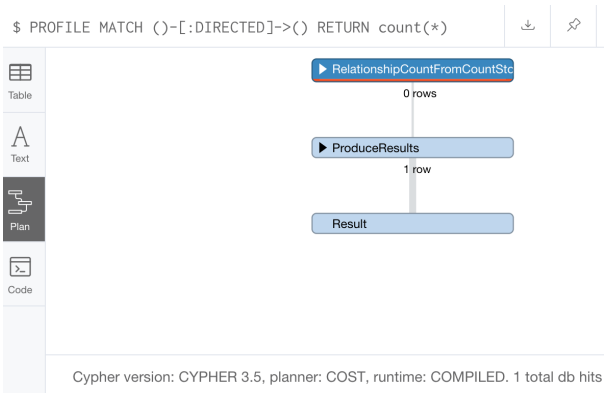
Profile it and then modify this query to use the count store.

In order to take advantage of the count store in a path, you must specify the direction of the relationship.

Here is the solution code:

```
PROFILE MATCH ()-[:DIRECTED]->() RETURN count(*)
```

The result of executing this code should be:



Exercise 10.3: Modify a query to use `size()` instead of `count()`. (Instructions)

Here is a query to count the number of `:IS_GENRE` relationships per movie in the graph.

```
MATCH (m:Movie)-[:IS_GENRE]->()
RETURN m, count(*) AS genreCount ORDER BY genreCount DESC
```

Profile it and then modify this query to use `size()` rather than `count()`.

Exercise 10.3: Modify a query to use `size()` instead of `count()`. (Solution)

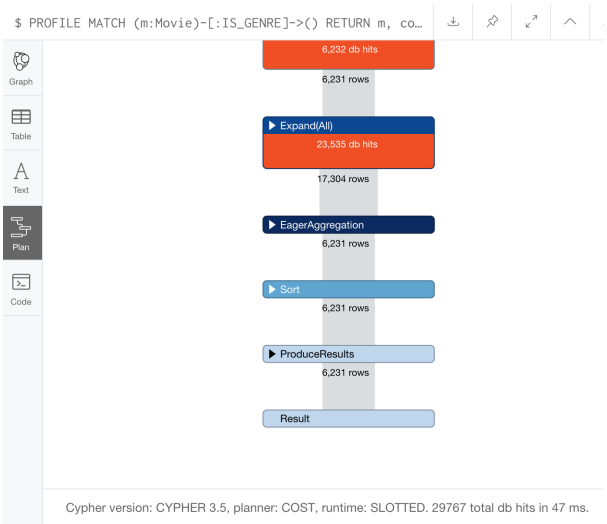
Here is a query to count the number of `:IS_GENRE` relationships per movie in the graph.

```
MATCH (m:Movie)-[:IS_GENRE]->()
RETURN m, count(*) AS genreCount ORDER BY genreCount DESC
```

Profile it and then modify this query to use `size()` rather than `count()`.

When you profile this query, you see 29,767 DB hits:

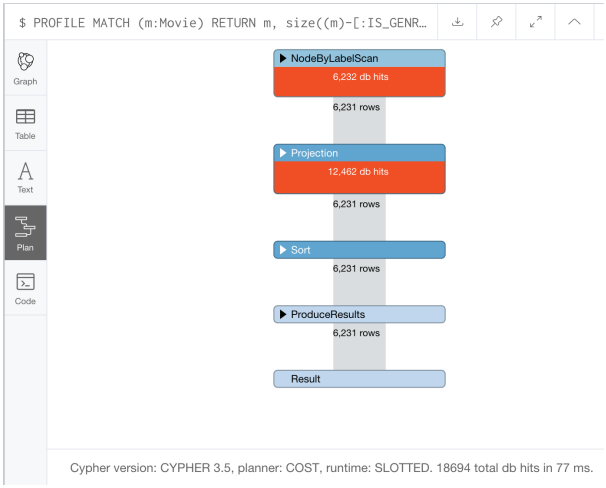
The result of executing this code should be:



Here is the solution code:

```
PROFILE MATCH (m:Movie)
RETURN m, size((m)-[:IS_GENRE]->()) as genreCount ORDER BY genreCount DESC
```

The result of executing this code should be:



Here we see fewer DB hits of 18,694.

Exercise 10.4: Query the count stores in the database. (Instructions)

Write a query to return the values in the count store for the database.

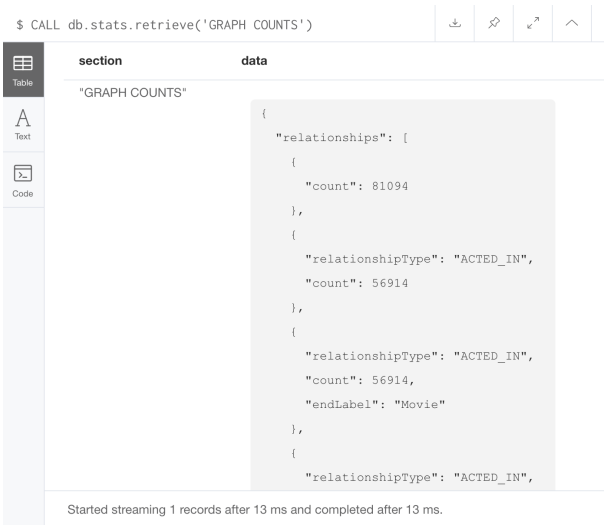
Exercise 10.4: Query the count stores in the database. (Solution)

Write a query to return the values in the count store for the database.

Here is the solution code:

```
CALL db.stats.retrieve('GRAPH COUNTS')
```

The result of executing this code should be:



Exercise 10: Taking it further

1. Perform queries the utilize the count store and compare them to count store values returned from the call to `db.stats.retrieve()`.
2. Look at other types of data returned from `db.stats.retrieve()`.
3. Use APOC to retrieve stats from the database.

Exercise 10: Working with DB stats and the count store (Summary)

In this exercise, you gained experience viewing DB stats and using the count store in your queries. Continue to Exercise 11

Exercise 11

Exercise 11: Preparing for query tuning (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

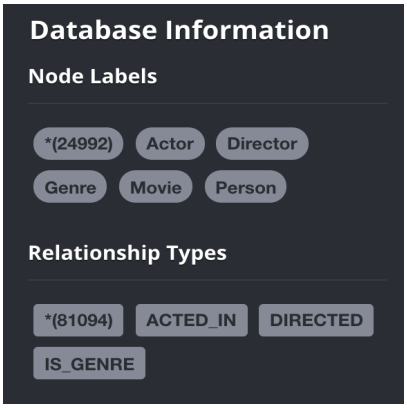
1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

This is what you should see when you click the database icon  .



Exercise 11: Preparing for query tuning (Overview)

In this exercise, you prepare your development environment for query tuning.

In this exercise, you will:

- **Exercise 11.1**: Modify the configuration for maximum virtual memory and a pagecache size that matches your database size.
- **Exercise 11.2**: Warm up the pagecache.

Go to the next page to start this exercise.

Exercise 11.1: Modify the configuration for maximum virtual memory and a pagecache size that matches your database size. (Instructions)

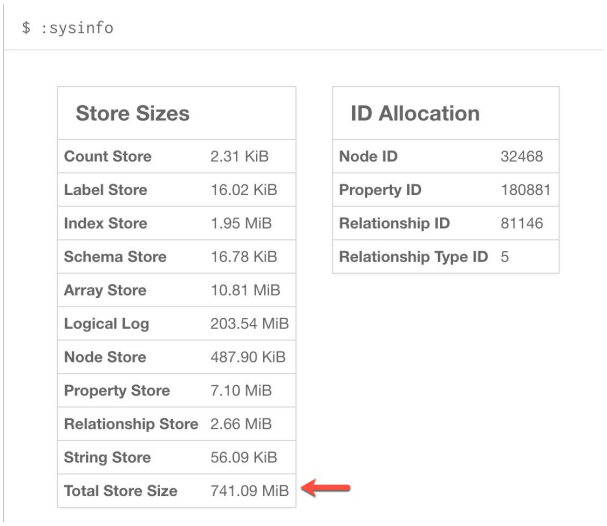
1. Determine the size of your database.

2. Set the virtual memory and pagecache sizes in the configuration for your database, then restart the database.

Exercise 11.1: Modify the configuration for maximum virtual memory and a pagecache size that matches your database size. (Solution)

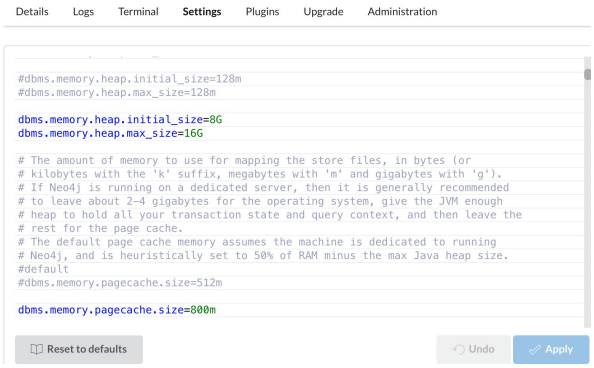
1. Determine the size of your database.

You should determine the total size of your database by using the `:sysinfo` command in Neo4j Browser:



2. Set the virtual memory and pagecache sizes in the configuration for your database, then restart the database.

Make these changes to the configuration in the settings tab as follows:



Exercise 11.2: Warm up the pagecache. (Instructions)

Execute Cypher code to retrieve the entire database into the pagecache.

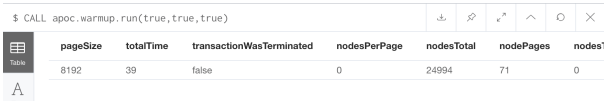
Exercise 11.2: Warm up the pagecache. (Solution)

Execute Cypher code to retrieve the entire database into the pagecache.

Here is the solution code:

```
CALL apoc.warmup.run(true,true,true)
```

The result of executing this code should be:



Exercise 11: Taking it further

1. Create a set of queries (from past queries you have done) that contain parameters rather than string constants.
2. Set parameters that will be used for the parameterized queries. The best reuse of the query cache is to use parameters.

Exercise 11: Preparing for query tuning (Summary)

In this exercise, you set up your development environment for query tuning. Continue to Exercise 12

Exercise 12

Exercise 12: Optimizing queries (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. Ensure that the virtual memory and pagecache settings have been adjusted to be:

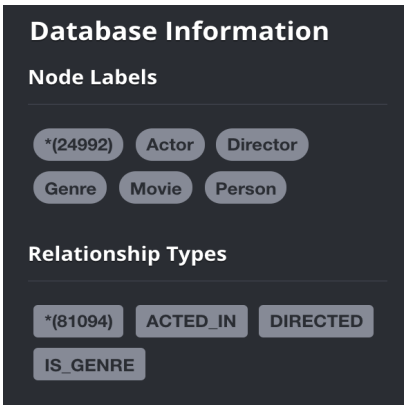
- dbms.memory.heap.initial_size=8G
- dbms.memory.heap.max_size=16G
- dbms.memory.pagecache.size=800m*

3. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
```

```
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

This is what you should see when you click the database icon  .



Exercise 12: Optimizing queries (Overview)

In this exercise, you profile queries and optimize them using some best practices.

In this exercise, you will:

- **Exercise 12.1**: Reduce rows processed by collecting intermediate results during the query.
- **Exercise 12.2**: Reduce cardinalities for variable-length paths.
- **Exercise 12.3**: Improve a query that returns unrelated data.
- **Exercise 12.4**: Rewrite a query to collect data using pattern comprehension.
- **Exercise 12.5**: Rewrite a query that limits results.
- **Exercise 12.6**: Defer property access.

Go to the next page to start this exercise.

Exercise 12.1: Reduce rows processed by collecting intermediate results during the query. (Instructions)

Here is a query that retrieves the genres and directors for a movie.

```
PROFILE MATCH (movie:Movie {title:'Up'})-[:IS_GENRE]-(genre)
MATCH (movie)<-[:DIRECTED]-(director)
RETURN genre, director, movie
```

1. Execute this query and note how many rows are processed.

2. Rewrite this query to reduce the number of rows processed by collection results during the query.

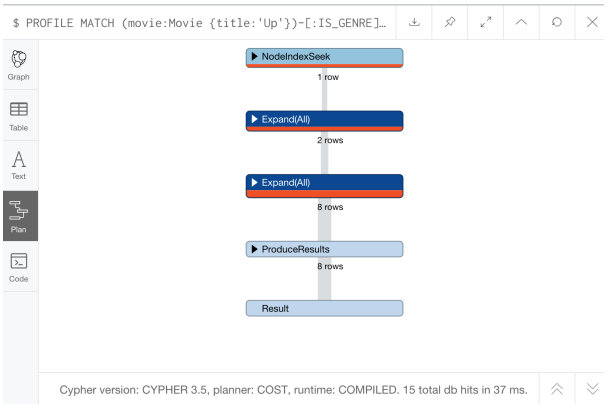
Exercise 12.1: Reduce rows processed by collecting intermediate results during the query. (Solution)

Here is a query that retrieves the genres and directors for a movie.

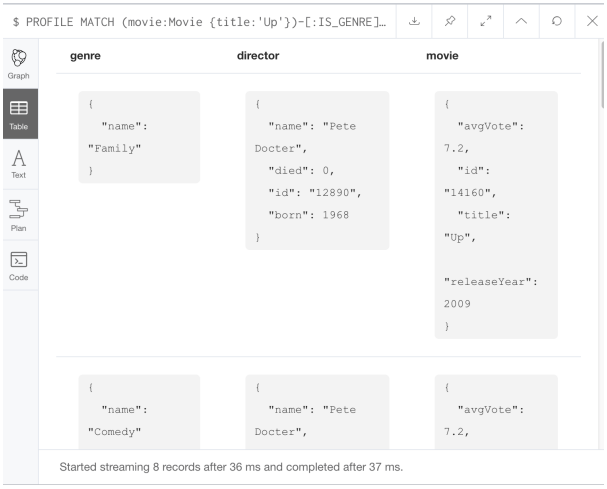
```
PROFILE MATCH (movie:Movie {title:'Up'})-[:IS_GENRE]-(genre)
MATCH (movie)<-[:DIRECTED]-(director)
RETURN genre, director, movie
```

1. Execute this query and note how many rows are processed.

The profile when executing this code should be:



The result of executing this code should be:

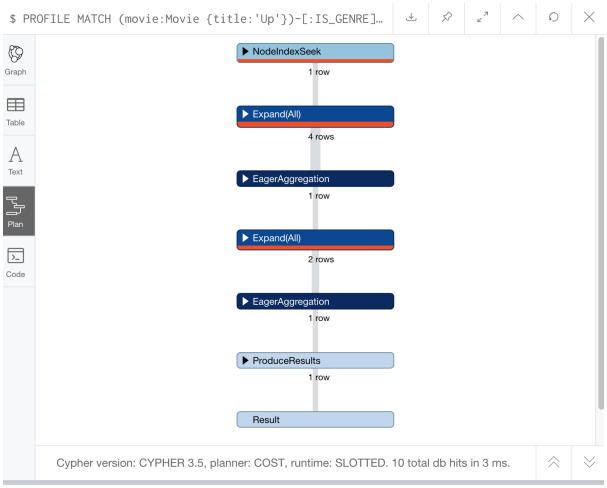


2. Rewrite this query to reduce the number of rows processed by collection results during the query.

Here is the solution code:

```
PROFILE MATCH (movie:Movie {title:'Up'})-[:IS_GENRE]->(g)
WITH movie, collect(g) as genres
MATCH (movie)<-[:DIRECTED]-(director)
WITH movie, genres, collect(director) as directors
RETURN genres, movie, directors
```

The profile when executing this code should show fewer rows processed:



The result of executing this code should be:

Exercise 12.2: Reduce cardinalities for variable-length paths. (Instructions)

1. Execute this Cypher code to retrieve the titles of all movies that are up to 5 hops away from Tom Hanks.

```
PROFILE
MATCH (:Person{name:'Tom Hanks'})-[:ACTED_IN*..5]-(movie)
RETURN movie.title
```

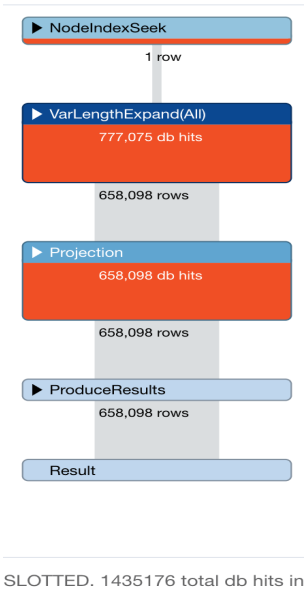
2. Modify this query to reduce the number of rows processed.

Exercise 12.2: Reduce cardinalities for variable-length paths. (Solution)

1. Execute this Cypher code to retrieve the titles of all movies that are up to 5 hops away from Tom Hanks.

```
PROFILE
MATCH (:Person{name:'Tom Hanks'})-[:ACTED_IN*..5]-(movie)
RETURN movie.title
```

The result of executing this code should be:

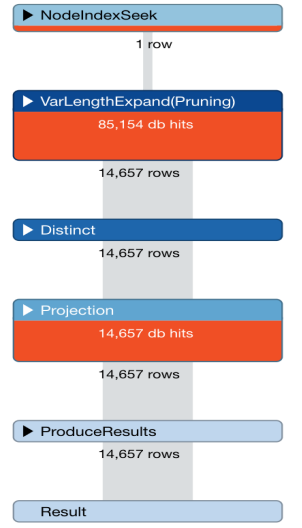


2. Modify this query to reduce the number of rows processed.

Here is the solution code:

```
PROFILE
MATCH (:Person{name:'Tom Hanks'})-[:ACTED_IN*..5]-(movie)
WITH DISTINCT movie
RETURN movie.title
```

The result of executing this code should be:



SLOTTED. 99814 total db hits in 2

Exercise 12.3: Improve a query that returns unrelated data. (Instructions)

1. Execute this Cypher code to retrieve the titles of all movies released before 1950 and people born before 1920.

```
PROFILE MATCH (p:Person), (m:Movie)
WHERE  p.born < 1920 AND m.releaseYear < 1950
RETURN p, m
```

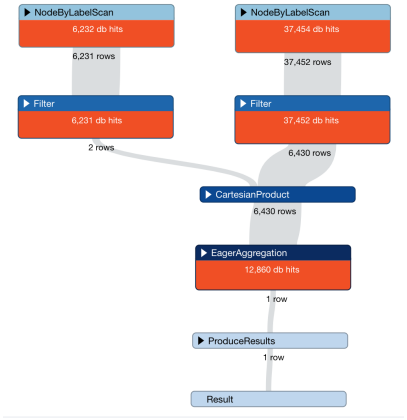
2. What is wrong with this query? Modify this query to optimize it.

Exercise 12.3: Improve a query that returns unrelated data. (Solution)

1. Execute this Cypher code to retrieve the titles of all movies released before 1920 and people born before 1920.

```
PROFILE MATCH (p:Person), (m:Movie)
WHERE  p.born < 1920 AND m.releaseYear < 1920
WITH collect(p.name) as people, collect(m.title) as movies
RETURN people, movies
```

The result of executing this query should be the following where we see a cartesian product:



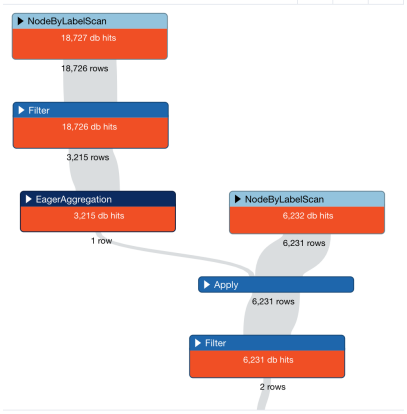
HER 3.5, planner: COST, runtime: SLOTTED. 100229 total db hits in 538 ms.

2. What is wrong with this query? Modify this query to optimize it.

Here is the solution code:

```
PROFILE MATCH (p:Person)
WHERE  p.born < 1920
WITH collect(p.name) AS people
MATCH (m:Movie)
WHERE  m.releaseYear < 1920
WITH collect(m.title) as movies, people
RETURN people, movies
```

The result of executing this code should be:



CYPHER 3.5, planner: COST, runtime: SLOTTED. 53133 total db hits in 159 ms.

Exercise 12.4: Rewrite a query to collect data using pattern comprehension. (Instructions)

1. Execute this Cypher code to retrieve the titles of of movies that an young actor (born after 1990) has acted in in the three year period of 2010 to 2012.

```
PROFILE MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)
WHERE 2010 <= m.releaseYear <= 2012 AND a.born > 1990
RETURN a.name as actor, a.born as born,
collect(DISTINCT m.title) AS titles ORDER BY actor
```

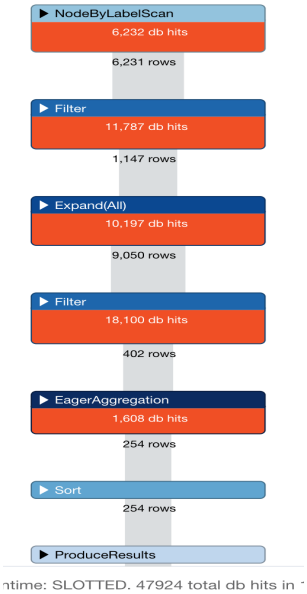
2. Rewrite this query to collect data using pattern comprehension to collect the titles.

Exercise 12.4: Rewrite a query to collect data using pattern comprehension. (Solution)

1. Execute this Cypher code to retrieve the titles of of movies that an young actor (born after 1990) has acted in in the three year period of 2010 to 2012.

```
PROFILE MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)
WHERE 2010 <= m.releaseYear <= 2012 AND a.born > 1990
RETURN a.name as actor, a.born as born,
collect(DISTINCT m.title) AS titles ORDER BY actor
```

The result of executing this query should be the following:

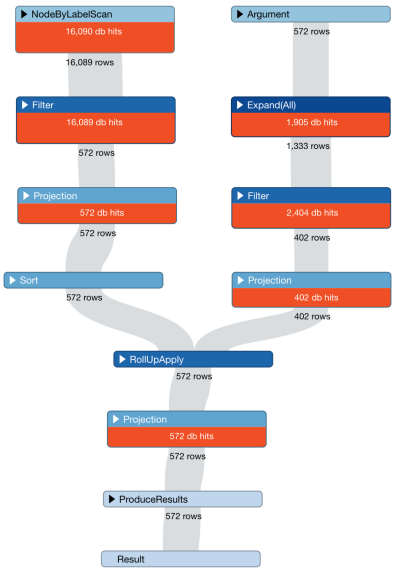


2. Rewrite this query to collect data using pattern comprehension to collect the titles.

Here is the solution code:

```
PROFILE MATCH (a:Actor)
WHERE a.born > 1990
RETURN a.name AS actor,  a.born AS born,
[(a)-->(x) WHERE 2010 <= x.releaseYear <= 2012 | x.title] AS titles ORDER BY actor
```

The result of executing this query should be the following:



If you look at the rows returned with this code, you will notice that empty titles collections are produced which is not the case in the previous query.

Exercise 12.5: Rewrite a query that limits results. (Instructions)

1. Execute this Cypher code to return a movie released in 2015 along with its actors.

```
PROFILE MATCH (movie:Movie)
WHERE movie.releaseYear = 2015
OPTIONAL MATCH (movie)<-[:ACTED_IN]-(actor)
WITH movie, collect(actor.name) as actors
RETURN movie.title, actors LIMIT 1
```

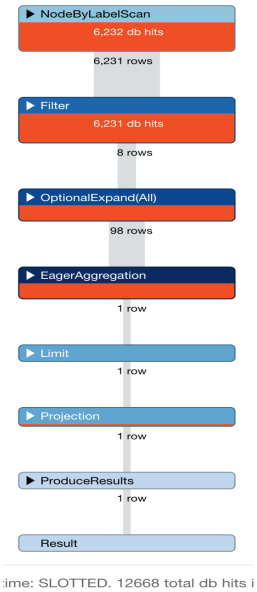
2. Rewrite this query with a small change to perform better.

Exercise 12.5: Rewrite a query that limits results. (Solution)

1. Execute this Cypher code to return a movie released in 2015 along with its actors.

```
PROFILE MATCH (movie:Movie)
WHERE movie.releaseYear = 2015
OPTIONAL MATCH (movie)<-[:ACTED_IN]-(actor)
WITH movie, collect(actor.name) as actors
RETURN movie.title, actors LIMIT 1
```

Here is the profile when executing this query:

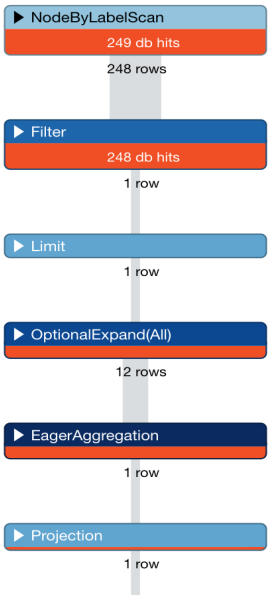


2. Rewrite this query with a small change to perform better.

Here is the solution code:

```
PROFILE MATCH (movie:Movie)
WHERE movie.releaseYear = 2015
WITH movie LIMIT 1
OPTIONAL MATCH (movie)<-[:ACTED_IN]-(actor)
WITH movie, collect(actor.name) as actors
RETURN movie.title, actors
```

The profile for executing this rewritten query should be the following:



Exercise 12.6: Defer property access. (Instructions)

1. Execute this Cypher code to return the top ten movies with the largest casts.

```
PROFILE
MATCH (a)-[:ACTED_IN]->(m:Movie)
```

```
RETURN m.title as title, count(a) as numActors
ORDER BY numActors DESC LIMIT 10
```

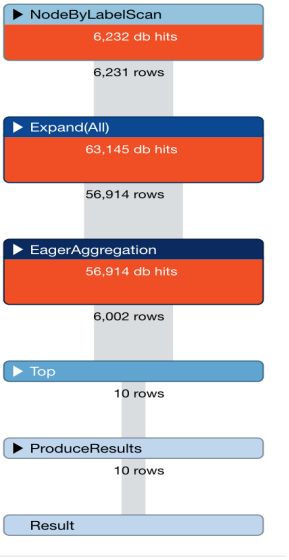
2. Rewrite this query with a small change to perform better by deferring property access.

Exercise 12.6: Defer property access. (Solution)

1. Execute this Cypher code to return the top ten movies with the largest casts.

```
PROFILE
MATCH (a)-[:ACTED_IN]->(m:Movie)
RETURN m.title as title, count(a) as numActors
ORDER BY numActors DESC LIMIT 10
```

Here is the profile when executing this query:



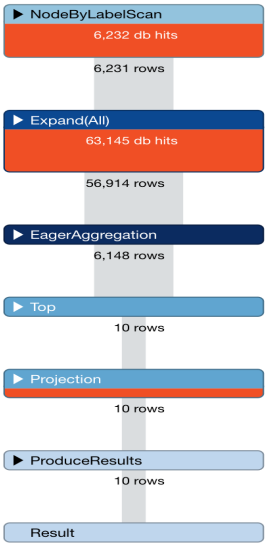
:ime: COMPILED. 126291 total db hi

2. Rewrite this query with a small change to perform better by deferring property access.

Here is the solution code:

```
PROFILE
MATCH (a)-[:ACTED_IN]->(m:Movie)
WITH m, count(a) as numActors ORDER by numActors DESC LIMIT 10
RETURN m.title, numActors
```

The profile for executing this rewritten query should be the following:



time: COMPILED. 69387 total db hit

Exercise 12: Taking it further

1. Use parameters in some of the queries you have worked on in this exercise.
2. In Exercise 12.4, the revised query was:

```
PROFILE MATCH (a:Actor)
WHERE a.born > 1990
RETURN a.name AS actor,  a.born AS born,
[(a)-->(x) WHERE 2010 <= x.releaseYear <= 2012 | x.title] AS titles ORDER BY actor
```

Modify this query so that empty collections are not returned.

Exercise 12: Preparing for query tuning (Summary)

In this exercise, you modified queries to perform better. Continue to Exercise 13

Exercise 13

Exercise 13: Using indexes (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

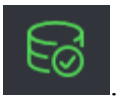
If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

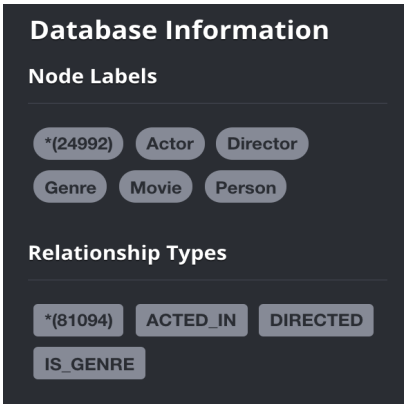
1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

This is what you should see when you click the database icon  .



Exercise 13: Using indexes (Overview)

In this exercise, you gain more experience with indexes.

In this exercise, you will:

- **Exercise 13.1**: Improve a query and observe how indexes are used.
- **Exercise 13.2**: Specify the use of two indexes for a query.
- **Exercise 13.3**: Create and use a fulltext schema index on a node property.
- **Exercise 13.4**: Create and use a fulltext schema index on a relationship.

Go to the next page to start this exercise.

Exercise 13.1: Improve a query and observe how indexes are used. (Instructions)

Here is a query that retrieves the name for the actor, Meg Ryan and the movies she has acted in that contain the word "Love".

```
PROFILE
MATCH (a)-[:ACTED_IN]->(m:Movie)
WHERE a.name = 'Meg Ryan' AND m.title CONTAINS 'Love'
RETURN m.title ORDER BY m.title DESC
```

1. Execute this query and note whether an index is used.

2. Rewrite this query to improve its performance.

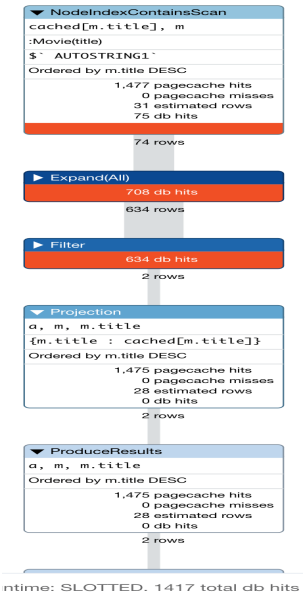
Exercise 13.1: Improve a query and observe how indexes are used. (Solution)

Here is a query that retrieves the name for the actor, Meg Ryan and the movies she has acted in that contain the word "Love".

```
PROFILE
MATCH (a)-[:ACTED_IN]->(m:Movie)
WHERE a.name = 'Meg Ryan' AND m.title CONTAINS 'Love'
RETURN m.title ORDER BY m.title DESC
```

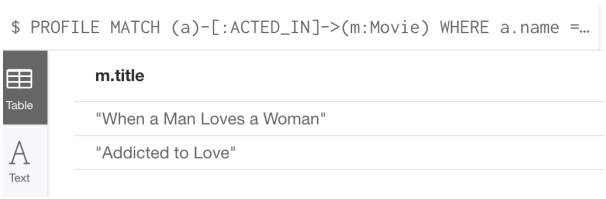
1. Execute this query and note whether an index is used.

The profile when executing this code should be:



Notice that the index is not used to retrieve the person node. You must specify :Person in the path for the query to use the index.

The result of executing this code should be:

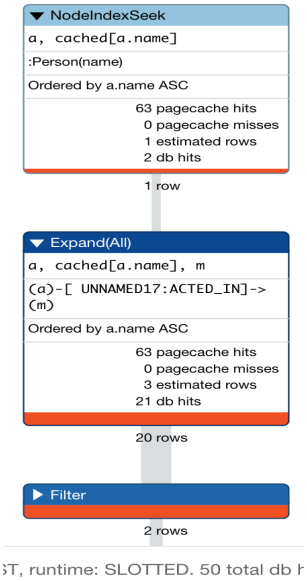


2. Rewrite this query to improve its performance.

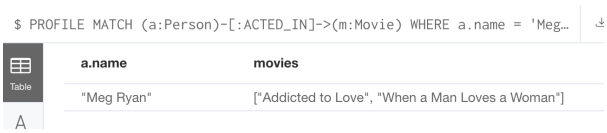
Here is the solution code:

```
PROFILE
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name = 'Meg Ryan' and m.title CONTAINS "Love"
WITH a, m ORDER BY m.title
WITH a, collect(m.title) as movies
RETURN  a.name, movies
```

The profile when executing this code should show fewer rows processed:



The result of executing this code should be:



Exercise 13.2: Specify the use of two indexes for a query. (Instructions)

1. Execute this Cypher code to retrieve the titles of movies that Tom Cruise acted in and Steven Spielberg directed.

```
PROFILE
MATCH (p1:Person)-[:ACTED_IN]->(m)<-[:DIRECTED]-(p2:Person)
WHERE p1.name = 'Tom Cruise'
AND p2.name = 'Steven Spielberg'
RETURN m.title
```

2. Modify this query to perform better by using two indexes.

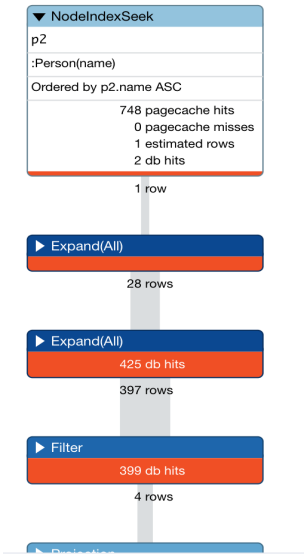
Exercise 13.2: Specify the use of two indexes for a query. (Solution)

1. Execute this Cypher code to retrieve the titles of movies that Tom Cruise acted in and Steven Spielberg directed.

```
PROFILE
MATCH (p1:Person)-[:ACTED_IN]->(m)<-[:DIRECTED]-(p2:Person)
WHERE p1.name = 'Tom Cruise'
AND p2.name = 'Steven Spielberg'
```

```
RETURN m.title
```

The profile for executing this code should be:



ST, runtime: COMPILED. 857 total db hi

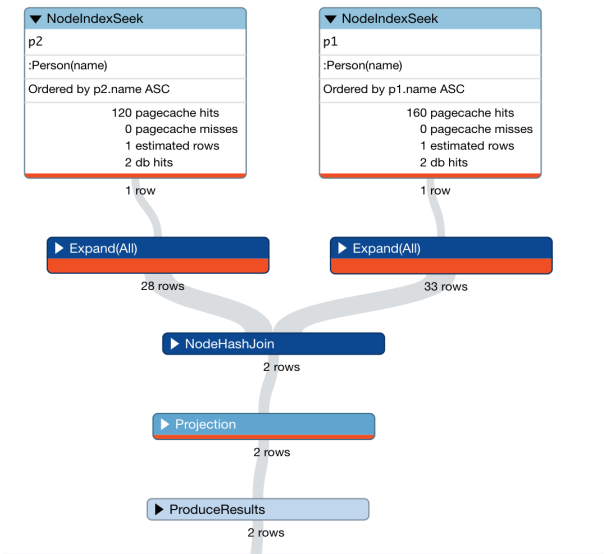
Notice that the index is used for p2.

2. Modify this query to perform better by using two indexes.

Here is the solution code:

```
PROFILE MATCH (p1:Person)-[:ACTED_IN]->(m)<-[:DIRECTED]-(p2:Person)
USING INDEX p1:Person(name)
USING INDEX p2:Person(name)
WHERE p1.name = 'Tom Cruise'
AND p2.name = 'Steven Spielberg'
RETURN m.title
```

The profile for executing this code should be:



HER 3.5, planner: COST, runtime: COMPILED. 69 total db hits in 679 ms.

Exercise 13.3: Create and use a fulltext schema index on a node property. (Instructions)

1. Write and execute a query using the existing index on the `Movie.title` property to retrieve all movie titles that contain these strings:

- **'Part Two'**
- **'Part II'**
- **'Part 2'**

2. Create a fulltext schema index on the `Movie.title` property.

3. Perform the same query using the fulltext schema index.

Exercise 13.3: Create and use a fulltext schema index on a node property. (Solution)

1. Write and execute a query using the existing index on the `Movie.title` property to retrieve all movie titles that contain these strings:

- **'Part Two'**
- **'Part II'**
- **'Part 2'**

Here is the solution code:

```
MATCH (m:Movie)
WHERE m.title CONTAINS 'Part II' OR m.title CONTAINS 'Part Two' OR m.title Contains 'Part 2'
RETURN m.title
```

The result of executing this query should be the following:

2. Create a fulltext schema index on the `Movie.title` property.

Here is the solution code:

```
CALL db.index.fulltext.createNodeIndex('MovieTitle',['Movie'], ['title'])
```

3. Perform the same query using the fulltext schema index.

Here is the solution code:

```
CALL db.index.fulltext.queryNodes("MovieTitle", '"Part Two" OR "Part II" OR "Part 2"') YIELD
node, score
RETURN node.title,  score
```

The result of executing this query should be the following:

Exercise 13.4: Create and use a fulltext schema index on a relationship. (Instructions)

In the current database you are working with, the `:ACTED_IN` relationship has a single property, `roles` which is an array of strings. You cannot use a fulltext schema index on an array type so in order to create an index on a relationship for searching, you will first add a property to all `:ACTED_IN` relationships that creates a new property, `roleList`.

1. Execute this Cypher code to create this new property for all `:ACTED_IN` relationships:

```
MATCH (a:Actor)-[rel:ACTED_IN]-(m)
WITH rel, REDUCE(s = HEAD(rel.roles), n IN TAIL(rel.roles) | s + ', ' + n) AS result
SET rel.roleList = result
```

2. Create a fulltext schema index on the roleList property of the `:ACTED_IN` relationships.

3. Write and execute the query to find all `:ACTED_IN` relationships in the database containing the role `"Batman"` using fulltext schema index search and return the names of the actors and the movies for these roles.

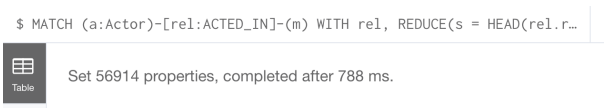
Exercise 13.4: Create and use a fulltext schema index on a relationship. (Solution)

In the current database you are working with, the `:ACTED_IN` relationship has a single property, `roles` which is an array of strings. You cannot use a fulltext schema index on an array type so in order to create an index on a relationship for searching, you will first add a property to all `:ACTED_IN` relationships that creates a new property, `roleList`.

1. Execute this Cypher code to create this new property for all `:ACTED_IN` relationships:

```
MATCH (a:Actor)-[rel:ACTED_IN]-(m)
WITH rel, REDUCE(s = HEAD(rel.roles), n IN TAIL(rel.roles) | s + ', ' + n) AS result
SET rel.roleList = result
```

Here is the result of running this code:



2. Create a fulltext schema index on the roleList property of the `:ACTED_IN` relationships.

Here is the solution code:

```
CALL db.index.fulltext.createRelationshipIndex('Roles',['ACTED_IN'], ['roleList'])
```

3. Write and execute the query to find all `:ACTED_IN` relationships in the database containing the role `"Batman"` using fulltext schema index search and return the names of the actors and the movies for these roles.

Here is the solution code:

```
CALL db.index.fulltext.queryRelationships('Roles', 'Batman') YIELD relationship, score
WITH relationship
MATCH (a:Actor)-[relationship]->(m:Movie)
RETURN a.name, relationship.roleList, m.title
```

Here is the result of running this code:

Exercise 13: Taking it further

1. Use parameters in some of the queries you have worked on in this exercise.
2. Perform some profiling to see if you can improve any of the queries in this exercise.

Exercise 13: Preparing for query tuning (Summary)

In this exercise, gained experience with index use in Cypher, including fulltext schema indexes. Continue to Exercise 14

Exercise 14

Exercise 14: Monitoring queries (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

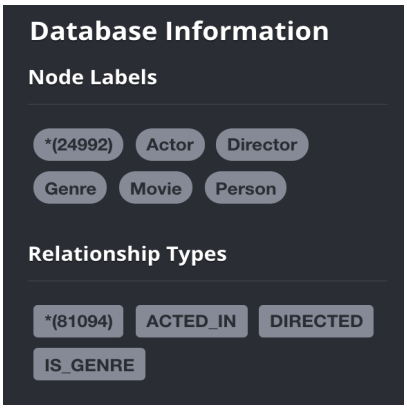
```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
     "MERGE (p)-[:DIRECTED]->(m)
         ON CREATE SET p:Director",
     {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

Note: This reset script will fail if you have fulltext schema indexes in the database. Before you run this script, you must run this code to drop the fulltext schema indexes:

```
call db.index.fulltext.drop('Roles');
call db.index.fulltext.drop('MovieTitle')
```

This is what you should see when you click the database icon  .



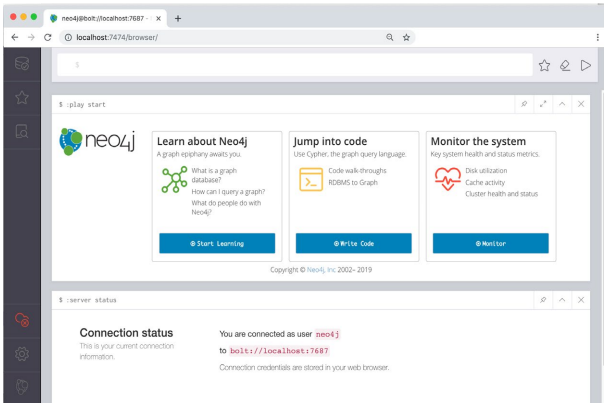
3. In this exercise, you will be running queries as you have done so in your Neo4j Browser. In order to monitor queries, you must open a Web browser and start another Neo4j Browser session. In most cases, you should be able to enter this address to connect to your started database:

https://localhost:7474

If this port has changed, it will be specified in the settings for the database and you should use those settings.

If you can successfully open Neo4j Browser at a particular port, simply enter the password for the ndo4j user and you should be ready to monitor any queries you are executing in your other Neo4j Browser window.

Here is what a Web browser should appear as if you have correctly connected to the started database:



Exercise 14: Monitoring queries (Overview)

In this exercise, you gain more experience monitoring queries and using the Query Log Analyzer tool.

In this exercise, you will:

- **Exercise 14.1**: Monitor and kill a long-running query.
- **Exercise 14.2**: Prepare for query logging.
- **Exercise 14.3**: Install the Query Log Analyzer Tool.
- **Exercise 14.4**: Execute some queries that will be logged.
- **Exercise 14.5**: View the queries logged using the Query Log Analyzer Tool.

Go to the next page to start this exercise.

Exercise 14.1: Monitor and kill a long-running query. (Instructions)

Here is a query that will run for a long time:

```
MATCH (a), (b), (c), (d), (e) RETURN count(id(a))
```

1. Execute this query in the Neo4j Browser session (main session) you have used throughout this training.

2. In the monitoring Neo4j Browser session you started in a different window, monitor the queries using the `:queries` command.

3. In the monitoring Neo4j Browser session, kill the query that you started above in the Neo4j Browser display shown for `:queries` command.

4. Repeat the query again that runs for a long time in your main Neo4j Browser session.

5. In the monitoring Neo4j Browser session, call the cypher procedure to list the queries and return the query and the query ID.

6. In the monitoring Neo4j Browser session, call the cypher procedure to kill the query.

Exercise 14.1: Monitor and kill a long-running query. (Solution)

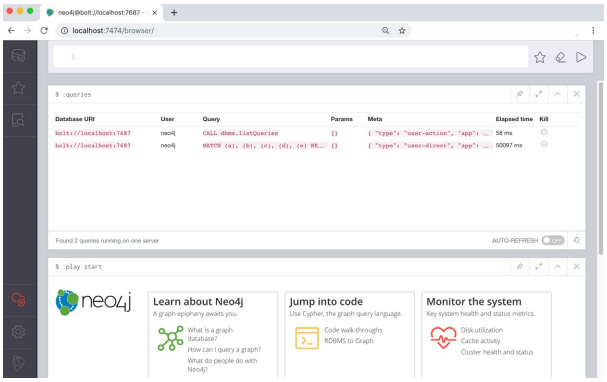
Here is a query that will run for a long time:

```
MATCH (a), (b), (c), (d), (e) RETURN count(id(a))
```

1. Execute this query in the Neo4j Browser session (main session) you have used throughout this training.

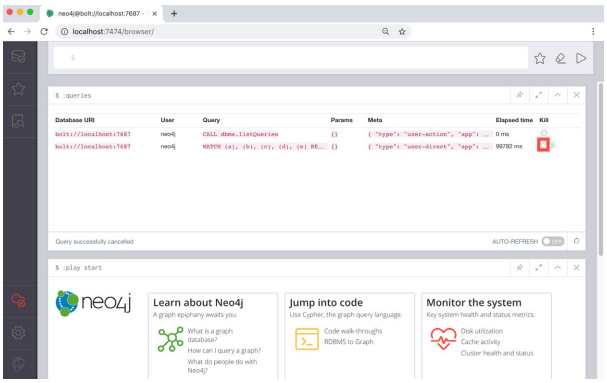
2. In the monitoring Neo4j Browser session you started in a different window, monitor the queries using the `:queries` command.

This is what you should see:

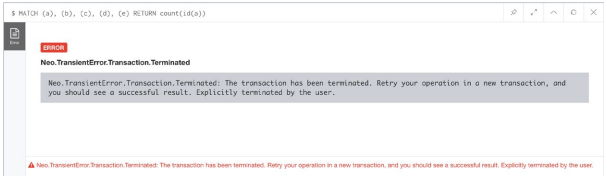


3. In the monitoring Neo4j Browser session, kill the query that you started above in the Neo4j Browser display shown for `:queries` command.

You should select the query to kill and kill it as shown here:



You should see this in your main Neo4j Browser session:



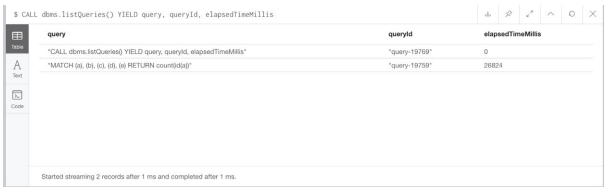
4. Repeat the query again that runs for a long time in your main Neo4j Browser session.

5. In the monitoring Neo4j Browser session, call the cypher procedure to list the queries and return the query and the query ID.

Here is the solution code:

```
CALL dbms.listQueries() YIELD query, queryId, elapsedTimeMillis
```

The result of executing this code should be:



6. In the monitoring Neo4j Browser session, call the cypher procedure to kill the query.

Here is the solution code:

```
CALL dbms.listQueries() YIELD query, queryId, elapsedTimeMillis
WHERE query STARTS WITH 'MATCH' AND elapsedTimeMillis > 5000
CALL dbms.killQuery(queryId) YIELD queryId as qid, message
RETURN "Query: " + " " +  qid +" " + message as result
```

The result of executing this code should be:

And again you should see the failure in your main Neo4j Browser session.

Exercise 14.2: Prepare for query logging. (Instructions)

Add these settings to enable query logging:

Hint: You can add them to the end of the settings file.

```
dbms.logs.query.enabled=true
# If the execution of query takes more time than this threshold,
# the query is logged. If set to zero then all queries
dbms.logs.query.threshold=100ms
dbms.logs.query.parameter_logging_enabled=true
dbms.logs.query.time_logging_enabled=true
dbms.logs.query.allocation_logging_enabled=true
dbms.logs.query.page_logging_enabled=true
dbms.track_query_cpu_time=true
dbms.track_query_allocation=true
```

Exercise 14.2: Prepare for query logging. (Solution)

Add these settings to enable query logging:

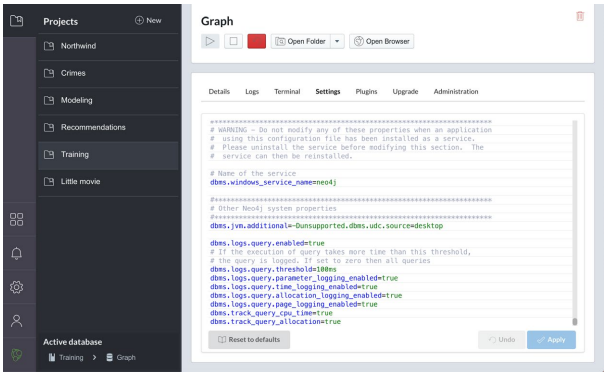
Hint: You can add them to the end of the settings file.

1. In Neo4j Desktop click **Manage** for the currently running database.
2. Click the **Settings** tab.
3. Add the following code to the end of this file.

```
dbms.logs.query.enabled=true
# If the execution of query takes more time than this threshold,
# the query is logged. If set to zero then all queries
dbms.logs.query.threshold=100ms
dbms.logs.query.parameter_logging_enabled=true
dbms.logs.query.time_logging_enabled=true
dbms.logs.query.allocation_logging_enabled=true
dbms.logs.query.page_logging_enabled=true
dbms.track_query_cpu_time=true
dbms.track_query_allocation=true
```

4. Click **Apply**. You need not restart the database yet as you will be restarting it after you install the Query Log Analyzer tool.

You should now see this in your Neo4j Desktop:



Exercise 14.3: Install the Query Log Analyzer Tool. (Instructions)

From Neo4j Desktop install the Query Log Analyzer Tool for your project.

Note: You must have a recent version of Neo4j Desktop. Make sure your version is up-to-date.

Exercise 14.3: Install the Query Log Analyzer Tool. (Solution)

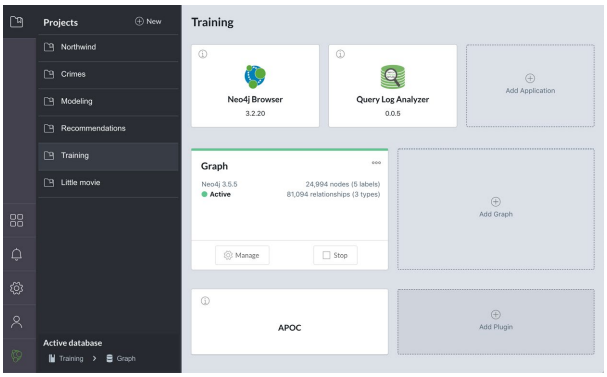
From Neo4j Desktop install the Query Log Analyzer Tool for your project.

Note: You must have a recent version of Neo4j Desktop. Make sure your version is up-to-date.

1. In Neo4j Desktop, for the project your are working with for this course, click **Add Application**.
2. Click **Add** for the Query Log Analyzer.
3. Click **OK**.

4. Restart the database.

You should now see this in your Neo4j Desktop:



You are now ready to analyze queries that are logged. (Completed queries).

Exercise 14.4: Execute some queries that will be logged. (Instructions)

1. From your main Neo4j Browser session, type `:history` to view many of your past queries. Select a set of them to execute.

2. Confirm that the queries have been written to the query.log file.

3. Make a note of the location of the query.log file as you will be opening it in the Query Log Analyzer Tool.

Exercise 14.4: Execute some queries that will be logged. (Solution)

1. From your main Neo4j Browser session, type `:history` to view many of your past queries. Select a set of them to execute.

2. Confirm that the queries have been written to the query.log file.

a. Click the **Manage** button in Neo4j Desktop.

b. Click the **Open Folder** button.

c. Ensure that **query.log** exists in the **log** folder.

3. Make a note of the location of the query.log file as you will be opening it in the Query Log Analyzer Tool.

Exercise 14.5: View the queries logged using the Query Log Analyzer Tool. (Instructions)

1. From Neo4j Desktop, open the Neo4j Query Log Analyzer Tool.

2. Select the query.log file.

3. Click Analyze File.

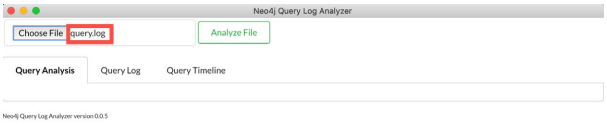
4. Explore the queries logged.

Exercise 14.5: View the queries logged using the Query Log Analyzer Tool. (Solution)

1. From Neo4j Desktop, open the Neo4j Query Log Analyzer Tool.

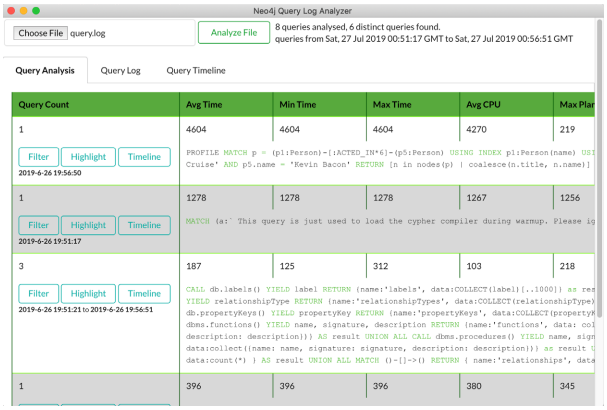
2. Select the query.log file.

You should now see this the Query Log Analyzer Tool UI where you have selected the query.log file:



3. Click Analyze File.

Depending on what queries were logged, you should now see this:



4. Explore the queries logged.

Exercise 14: Taking it further

1. Write custom code to kill specific types of queries.
2. Analyze a larger set of queries.

Exercise 14: Monitoring queries (Summary)

In this exercise, you gained experience with monitoring and killing long-running queries and you got your feet wet with logging queries and monitoring them with the Query Log Analyzer Tool. Continue to Exercise 15

Exercise 15

Exercise 15: Monitoring locking (Preparations)

1. Verify that your Neo4j Browser session has access to the APOC library by executing the Cypher below:

```
CALL dbms.procedures()
YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN name ORDER BY name ASC
```

If this code does not return the list of APOC procedures, then you must ensure that the APOC library is available by installing the plugin (Neo4j Desktop) and restarting the database as follows:

1. Make sure Neo4j Desktop is online.
2. In Neo4j Desktop for the project you are working with, click **Add Plugin**.
3. Select the install button for APOC.
4. Click the Install button.
5. Close the Add Plugin window.
6. Start or restart the database.

2. This exercise assumes that you have a fully-loaded movie database with the `Genre` nodes. Execute this code to reset the database:

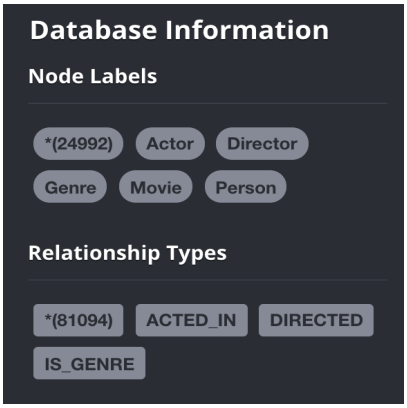
```
CALL apoc.schema.assert({},{},true);

MATCH (n:Person) DETACH DELETE n;

MATCH (n:Director) DETACH DELETE n;

MATCH (n:Actor) DETACH DELETE n;

MATCH (n:Movie) DETACH DELETE n;

MATCH (n:Genre) DETACH DELETE n;

CREATE CONSTRAINT ON (m:Movie)
ASSERT m.id IS UNIQUE;

CREATE CONSTRAINT ON (p:Person)
ASSERT p.id IS UNIQUE;

LOAD CSV WITH HEADERS FROM
     'https://data.neo4j.com/advanced-cypher/movies2.csv' AS row
WITH row.movieId as movieId,
row.title as title,
row.genres as genres,
toInteger(row.releaseYear) as releaseYear,
toFloat(row.avgVote) as avgVote,
collect({id: row.personId, name:row.name, born: toInteger(row.birthYear), died:
toInteger(row.deathYear),personType: row.personType, roles:
split(coalesce(row.characters,""),':')}) as people
MERGE (m:Movie {id:movieId})
   ON CREATE SET m.title=title, m.avgVote=avgVote,
      m.releaseYear=releaseYear, m.genres=split(genres,":")
WITH *
UNWIND people as person
MERGE (p:Person {id: person.id})
   ON CREATE SET p.name = person.name, p.born = person.born, p.died = person.died
WITH  m, person, p
CALL apoc.do.when(person.personType = 'ACTOR',
     "MERGE (p)-[:ACTED_IN {roles: person.roles}]->(m)
                ON CREATE SET p:Actor",
```

```
      "MERGE (p)-[:DIRECTED]->(m)
          ON CREATE SET p:Director",
      {m:m, p:p, person:person}) YIELD value AS value
RETURN count(*);  // cannot end query with APOC call

CREATE INDEX ON :Person(name);

CREATE INDEX ON :Movie(title);

CREATE CONSTRAINT ON (g:Genre) ASSERT g.name IS UNIQUE;
MATCH (m:Movie)
UNWIND m.genres as name
WITH DISTINCT name, m
MERGE (g:Genre {name:name})
WITH g, m
MERGE (g)<-[:IS_GENRE]-(m)
```

Note: This reset script will fail if you have fulltext schema indexes in the database. Before you run this script, you must run this code to drop the fulltext schema indexes:

```
call db.index.fulltext.drop('Roles');
call db.index.fulltext.drop('MovieTitle')
```

This is what you should see when you click the database icon  .



3. In this exercise, you will be running queries as you have done so in your main Neo4j Browser session. In order to monitor locking, you must open a Web browser and start another Neo4j Browser session. If you have done this for the previous exercise, you should be all set and you can ignore the rest of these setup instructions.

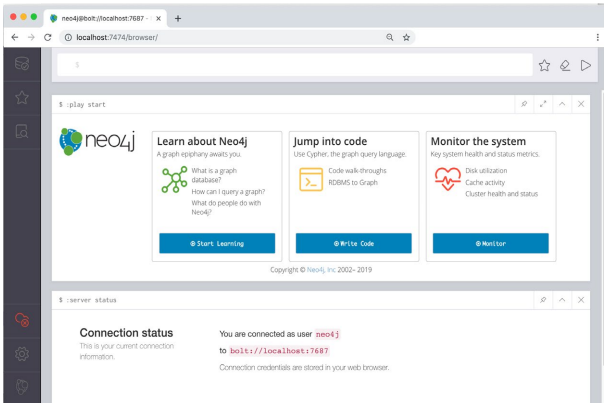
In most cases, you should be able to enter this address to connect to your started database:

https://localhost:7474

If this port has changed, it will be specified in the settings for the database and you should use those settings.

If you can successfully open Neo4j Browser at a particular port, simply enter the password for the ndo4j user and you should be ready to monitor any queries you are executing in your other Neo4j Browser window.

Here is what a Web browser should appear as if you have correctly connected to the started database:



Exercise 15: Monitoring locking (Overview)

In this exercise, you gain more experience monitoring locking in a multi-user environment.

In this exercise, you will:

- **Exercise 15.1**: Execute code that holds a write lock on a node.
- **Exercise 15.2**: Execute code that attempts to lock the same node.
- **Exercise 15.3**: Write code to monitor locking.

Go to the next page to start this exercise.

Exercise 15.1: Execute code that holds a write lock on a node. (Instructions)

Here is code that holds a lock on a node for a long time (200 seconds).

```
MATCH (a:Actor)
WHERE a.name = 'Doris Day'
SET a.died= 2019
WITH a
call apoc.util.sleep(200000)
RETURN a.died
```

Execute this code in your main Neo4j Browser session.

Exercise 15.1: Execute code that holds a write lock on a node. (Solution)

Here is code that holds a lock on a node for a long time (200 seconds).

```
MATCH (a:Actor)
WHERE a.name = 'Doris Day'
SET a.died= 2019
WITH a
call apoc.util.sleep(200000)
RETURN a.died
```

Execute this code in your main Neo4j Browser session.

This is what you should see:

Exercise 15.2: Execute code that attempts to lock the same node. (Instructions)

1. In the monitoring Neo4j Browser session, execute this code that attempts to lock the same node:

```
MATCH (a:Actor)
WHERE a.name = 'Doris Day'
SET a.died= 2000
RETURN a.died
```

When you run this code, it will hang because it is waiting for the other client to release the lock on the node.

2. Terminate this query in the monitoring Neo4j Browser session by simply click the "X" in the top right corner of the window.

Exercise 15.2: Execute code that attempts to lock the same node. (Solution)

1. In the monitoring Neo4j Browser session, execute this code that attempts to lock the same node:

```
MATCH (a:Actor)
WHERE a.name = 'Doris Day'
SET a.died= 2000
RETURN a.died
```

When you run this code, it will hang because it is waiting for the other client to release the lock on the node.

2. Terminate this query in the monitoring Neo4j Browser session by simply click the "X" in the top right corner of the window.

Exercise 15.3: Write code to monitor locking. (Instructions)

The update should complete in the main Neo4j Browser window. Next, you will prepare to monitor locking.

1. In the monitoring Neo4j Browser session, write code to monitor locking.

2. In the main Neo4j Browser window, recall the last statement and re-execute it to hold the write lock for 200 seconds.

3. In the monitoring Neo3j Browser session, execute the code to monitor locking.

Exercise 15.3: Write code to monitor locking. (Solution)

The update should complete in the main Neo4j Browser window. Next, you will prepare to monitor locking.

1. In the monitoring Neo4j Browser session, write code to monitor locking.

Here is the solution code:

```
CALL dbms.listTransactions() YIELD currentQueryId , currentQuery
WHERE currentQuery STARTS WITH "MATCH (a:Actor)"
CALL dbms.listActiveLocks(currentQueryId) YIELD mode, resourceType
RETURN mode, resourceType
```

2. In the main Neo4j Browser window, recall the last statement and re-execute it to hold the write lock for 200 seconds.

3. In the monitoring Neo3j Browser session, execute the code to monitor locking.

You should see something like this:

Exercise 15: Taking it further

1. Try locking contention with three users.
2. Write code to kill the query found with the lock.

Exercise 15: Monitoring queries (Summary)

In this exercise, you gained experience with monitoring locks.

Congratulations! You have completed the exercises for this course.