Course: Implementing Graph Data Models in Neo4j — 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

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

If you are using a Neo4j Sandbox, the Neo4j Browser is already connected to a started database.

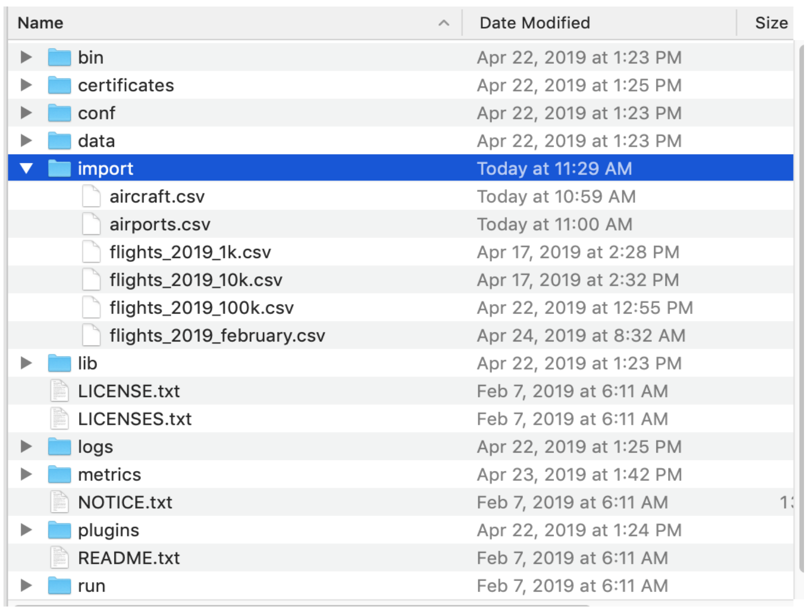
Ensure files for loading are accessible (Neo4j Desktop only)

If you are using Neo4j Desktop, you can load data either from a URL or from a local file. In preparation for this course, you should have downloaded the course data files. If you cannot load data from a URL due to a lack of an Internet connection or high bandwidth, then you can use the course data files you downloaded prior to starting this course.

Assuming you have uncompressed the downloaded files for this course, copy the neo4j-implementing-modeling*.csv files to the import folder for your database as follows:

1. Click the **Manage** tab for the database that you are using.
2. Click the **Open Folder** button. This opens a file browser for the database.
3. Copy the neo4j-implementing-modeling*.csv to the *import* folder.

The **import** folder should appear as follows:



Name	Date Modified	Size
▶ bin	Apr 22, 2019 at 1:23 PM	
▶ certificates	Apr 22, 2019 at 1:25 PM	
▶ conf	Apr 22, 2019 at 1:23 PM	
▶ data	Apr 22, 2019 at 1:23 PM	
▼ import	Today at 11:29 AM	
aircraft.csv	Today at 10:59 AM	
airports.csv	Today at 11:00 AM	
flights_2019_1k.csv	Apr 17, 2019 at 2:28 PM	
flights_2019_10k.csv	Apr 17, 2019 at 2:32 PM	
flights_2019_100k.csv	Apr 22, 2019 at 12:55 PM	
flights_2019_february.csv	Apr 24, 2019 at 8:32 AM	
▶ lib	Apr 22, 2019 at 1:23 PM	
LICENSE.txt	Feb 7, 2019 at 6:11 AM	
LICENSES.txt	Feb 7, 2019 at 6:11 AM	
▶ logs	Apr 22, 2019 at 1:25 PM	
▶ metrics	Apr 23, 2019 at 1:42 PM	
NOTICE.txt	Feb 7, 2019 at 6:11 AM	1
▶ plugins	Apr 22, 2019 at 1:24 PM	
README.txt	Feb 7, 2019 at 6:11 AM	
▶ run	Feb 7, 2019 at 6:11 AM	

Note: If you are using a Neo4j sandbox, then you cannot place any files locally for loading.

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:
   ✓ Enable multi statement query editor

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 implementing graph data models.

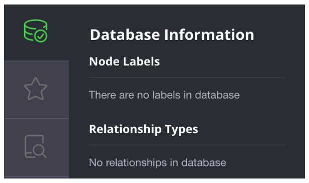
Exercises:

Exercise 1

Exercise 1: Implementing your first model (Preparations)

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

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


Database Information
Node Labels
There are no labels in database
Relationship Types
No relationships in database

If your database is not empty, you can run the Cypher below to reset it:

```
MATCH (n) DETACH DELETE n
```

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. If you are using the **Blank Neo4j Sandbox**, you should have APOC.

Exercise 1: Implementing your first model (Overview)

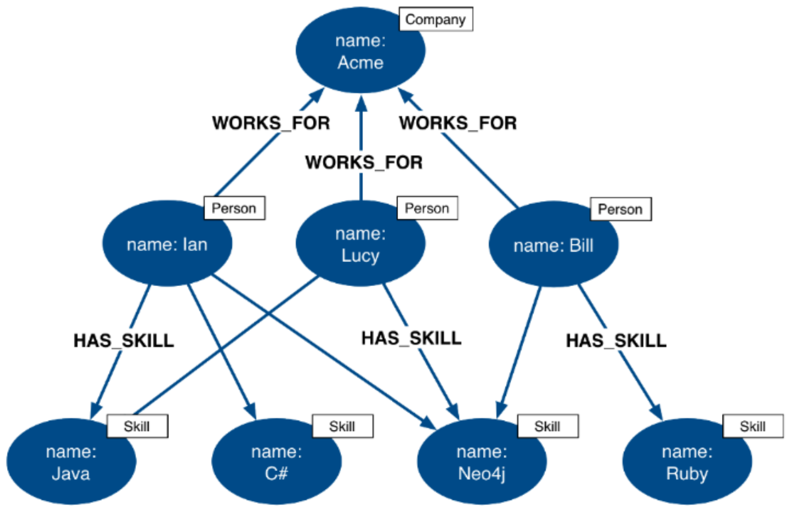
In this exercise, you will implement a simple model for people in a company and their skills and answer some questions.

- **Exercise 1.1**: Load the database with Person, Company, and Skill nodes.
- **Exercise 1.2**: Where does Ian work and what are his skills?
- **Exercise 1.3:** Where does Sarah work and what are her skills?
- **Exercise 1.4:** What people in Ian’s company have similar skills as Ian?
- **Exercise 1.5:** Modify Ian’s skills.
- **Exercise 1.6:** What are Ian’s skills?
- **Exercise 1.7:** What people in Ian’s company have similar skill as Ian?

Go to the next page to start this exercise.

Exercise 1.1: Load the database with Person, Company, and Skill nodes (Instructions)

Here is the data model you will work with in this exercise:



Execute this Cypher code to load the database for this model:

```
CREATE
(ben:Person{name:'Ben'}),
(charlie:Person{name:'Charlie'}),
(lucy:Person{name:'Lucy'}),
(ian:Person{name:'Ian'}),
(sarah:Person{name:'Sarah'}),
(emily:Person{name:'Emily'}),
(gordon:Person{name:'Gordon'}),
(kate:Person{name:'Kate'}),
(acme:Company{name:'Acme, Inc'}),
(startup:Company{name:'Startup, Ltd'}),
(neo4j:Skill{name:'Neo4j'}),
(rest:Skill{name:'REST'}),
(dotNet:Skill{name:'DotNet'}),
(ruby:Skill{name:'Ruby'}),
(sql:Skill{name:'SQL'}),
(architecture:Skill{name:'Architecture'}),
(java:Skill{name:'Java'}),
(python:Skill{name:'Python'}),
(javascript:Skill{name:'Javascript'}),
(clojure:Skill{name:'Clojure'}),
(ben)-[:WORKS_FOR]->(acme),
(charlie)-[:WORKS_FOR]->(acme),
(lucy)-[:WORKS_FOR]->(acme),
(ian)-[:WORKS_FOR]->(acme),
(sarah)-[:WORKS_FOR]->(startup),
(emily)-[:WORKS_FOR]->(startup),
(gordon)-[:WORKS_FOR]->(startup),
(kate)-[:WORKS_FOR]->(startup),
(ben)-[:HAS_SKILL{level:1}]->(neo4j),
(ben)-[:HAS_SKILL{level:3}]->(rest),
(charlie)-[:HAS_SKILL{level:2}]->(neo4j),
(charlie)-[:HAS_SKILL{level:1}]->(javascript),
(charlie)-[:HAS_SKILL{level:1}]->(sql),
(lucy)-[:HAS_SKILL{level:3}]->(dotNet),
(lucy)-[:HAS_SKILL{level:2}]->(architecture),
(lucy)-[:HAS_SKILL{level:1}]->(python),
(ian)-[:HAS_SKILL{level:2}]->(java),
(ian)-[:HAS_SKILL{level:3}]->(neo4j),
(ian)-[:HAS_SKILL{level:2}]->(rest),
(sarah)-[:HAS_SKILL{level:1}]->(neo4j),
(sarah)-[:HAS_SKILL{level:1}]->(java),
(sarah)-[:HAS_SKILL{level:1}]->(rest),
(sarah)-[:HAS_SKILL{level:1}]->(clojure),
(emily)-[:HAS_SKILL{level:1}]->(neo4j),
(emily)-[:HAS_SKILL{level:1}]->(dotNet),
(emily)-[:HAS_SKILL{level:1}]->(python),
(gordon)-[:HAS_SKILL{level:1}]->(dotNet),
(gordon)-[:HAS_SKILL{level:1}]->(ruby),
(kate)-[:HAS_SKILL{level:1}]->(architecture),
(kate)-[:HAS_SKILL{level:1}]->(python)
```

Exercise 1.2: Where does Ian work and what are his skills? (Instructions)

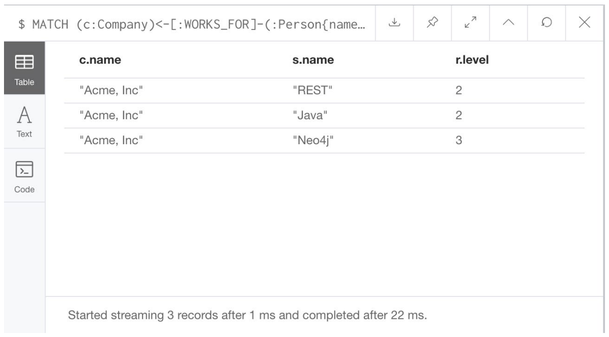
Write a query to display the company that Ian works for and his skills.

Exercise 1.2: Where does Ian work and what are his skills? (Solution)

Write a query to display the company that Ian works for and his skills.

```
MATCH (c:Company)<-[:WORKS_FOR]-(:Person{name:'Ian'})-[r:HAS_SKILL]->(s:Skill)
RETURN c.name, s.name, r.level
```

The result returned should be:


$ MATCH (c:Company)<-[:WORKS_FOR]-(:Person{name…

c.name	s.name	r.level
"Acme, Inc"	"REST"	2
"Acme, Inc"	"Java"	2
"Acme, Inc"	"Neo4j"	3

Started streaming 3 records after 1 ms and completed after 22 ms.

Exercise 1.3: Where does Sarah work and what are her skills? (Instructions)

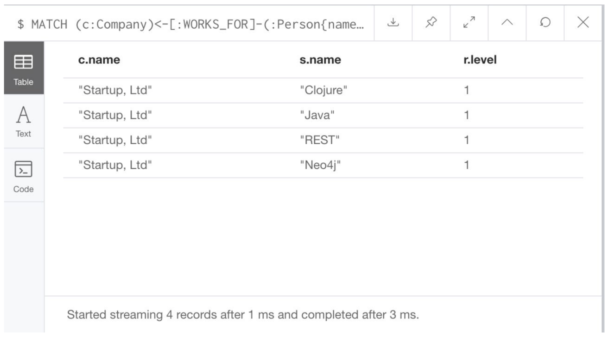
Write a query to display the company that Sarah works for and her skills.

Exercise 1.3: Where does Sarah work and what are her skills? (Solution)

Write a query to display the company that Sarah works for and her skills.

```
MATCH (c:Company)<-[:WORKS_FOR]-(:Person{name:'Sarah'})-[r:HAS_SKILL]->(s:Skill)
RETURN c.name, s.name, r.level
```

The result returned should be:


$ MATCH (c:Company)<-[:WORKS_FOR]-(:Person{name…

c.name	s.name	r.level
"Startup, Ltd"	"Clojure"	1
"Startup, Ltd"	"Java"	1
"Startup, Ltd"	"REST"	1
"Startup, Ltd"	"Neo4j"	1

Started streaming 4 records after 1 ms and completed after 3 ms.

Exercise 1.4: What people in Ian’s company have similar skills as Ian? (Instructions)

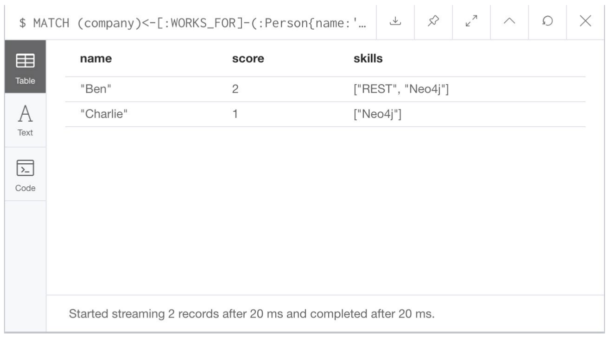
Write a query to display people in Ian’s company that have the same skills as Ian.

Exercise 1.4: What people in Ian’s company have similar skills as Ian? (Solution)

Write a query to display people in Ian’s company that have the same skills as Ian.

```
MATCH (company)<-[:WORKS_FOR]-(:Person{name:'Ian'})
                -[:HAS_SKILL]->(skill),
(company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
RETURN colleague.name AS name,
count(skill) AS score,
collect(skill.name) AS skills
ORDER BY score DESC
```

The result returned should be:



```
$ MATCH (company)<-[:WORKS_FOR]-(:Person{name:'…
```

name	score	skills
"Ben"	2	["REST", "Neo4j"]
"Charlie"	1	["Neo4j"]

Started streaming 2 records after 20 ms and completed after 20 ms.

Exercise 1.5: Modify Ian’s skills. (Instructions)

Write Cypher that updates Ian’s skills to include Java, Python, REST, and Neo4j.

Hint: Use MERGE to avoid duplication of nodes and relationships.

Exercise 1.5: Modify Ian’s skills. (Solution)

Write Cypher that updates Ian’s skills to include Java, Python, REST, and Neo4j.

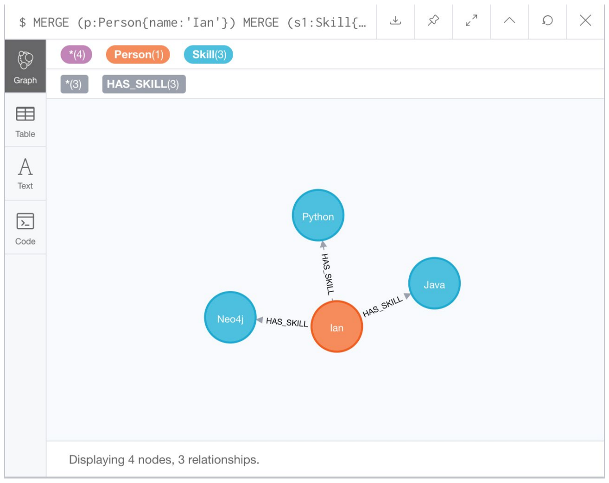
Hint: Use MERGE to avoid duplication of nodes and relationships.

You can use MERGE to update the nodes and relationships to reflect the skills for Ian as follows:

```
MERGE (p:Person{name:'Ian'})
MERGE (s1:Skill{name:'Java'})
MERGE (s2:Skill{name:'Python'})
MERGE (s3:Skill{name:'Neo4j'})
MERGE (c)<-[:WORKS_FOR]-(p)
MERGE (p)-[r1:HAS_SKILL]->(s1)
MERGE (p)-[r2:HAS_SKILL]->(s2)
MERGE (p)-[r3:HAS_SKILL]->(s3)
SET r1.level = 2
SET r2.level = 2
SET r3.level = 3
RETURN   p, s1, s2, s3
```

Here we use MERGE to ensure duplicate nodes and relationships are not created. For example, we already have nodes and relationships for Java, Neo4j, and REST so some nodes will not be updated. The only data that is added to the graph is the relationship for the Python skill.

The result returned should be:


```
$ MERGE (p:Person{name:'Ian'}) MERGE (s1:Skill{…
```

Displaying 4 nodes, 3 relationships.

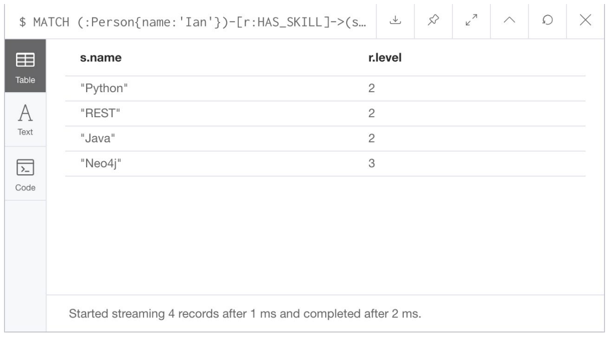
Exercise 1.6: What are Ian’s skills? (Instructions)

Repeat the query to display Ian’s skills.

```
MATCH (:Person{name:'Ian'})-[r:HAS_SKILL]->(s:Skill)
RETURN  s.name, r.level
```

You should see a different set of skills since you modified Ian’s skills.

The result returned should be:


```
$ MATCH (:Person{name:'Ian'})-[r:HAS_SKILL]->(s…
```

s.name	r.level
"Python"	2
"REST"	2
"Java"	2
"Neo4j"	3

Started streaming 4 records after 1 ms and completed after 2 ms.

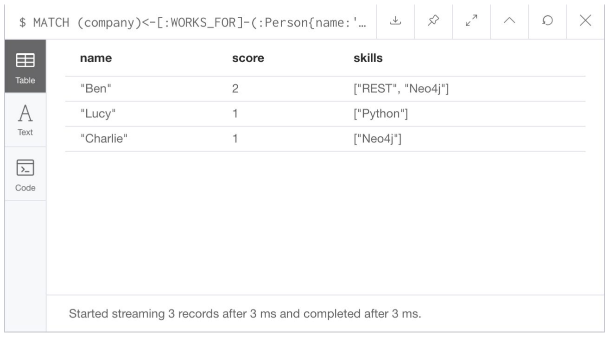
Exercise 1.7: What people in Ian’s company have similar skill as Ian? (Instructions)

Repeat the query to display people in Ian’s company that have the same skills as Ian.

```
MATCH (company)<-[:WORKS_FOR]-(:Person{name:'Ian'})
                -[:HAS_SKILL]->(skill),
 (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
RETURN colleague.name AS name,
count(skill) AS score,
collect(skill.name) AS skills
ORDER BY score DESC
```

You should see a different result here because you have modified Ian’s skills.

The result returned should be:


```
$ MATCH (company)<-[:WORKS_FOR]-(:Person{name:'…
```

name	score	skills
"Ben"	2	["REST", "Neo4j"]
"Lucy"	1	["Python"]
"Charlie"	1	["Neo4j"]

Started streaming 3 records after 3 ms and completed after 3 ms.

Exercise 1: Implementing your first model (Summary)

In this exercise, you added `Person`, `Skill`, and `Company` nodes to a graph and added the relationships between the nodes. This graph is a very simple model to review the basic process for implementing a model by creating or merging nodes and relationships. In the remaining exercises, you will start with a simple model and evolve it by refactoring the data in the graph. As you add data to a graph, you will most often use MERGE so that duplicate nodes and relationships are not created in the graph.

Continue to Exercise 2

Exercise 2

Exercise 2: Getting started with the airport graph data model (Preparations)

You will now work with the airport graph data model.

Execute the following Cypher statement to delete nodes and relationships in the graph:

```
MATCH (n) DETACH DELETE n
```

Exercise 2: Getting started with the airport graph data model (Overview)

For the remaining exercises in this course, you will be working with data related to flights and airports. In this exercise, you will create nodes and relationships that reflect our very simple model that has an airport entity.

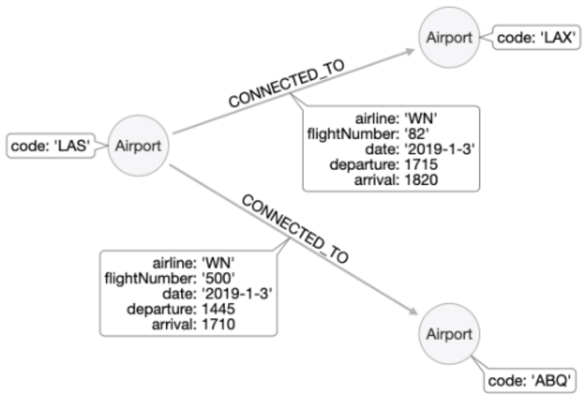
Here are the tasks for this exercise:

- **Exercise 2.1**: Implement the starting model containing `Airport` nodes.
- **Exercise 2.2**: Display the nodes and relationships in the graph.
- **Exercise 2.3**: Query the graph.

Go to the next page to start this exercise.

Exercise 2.1: Implement the starting model containing `Airport` nodes (Instructions)

Here is the starting airport model:

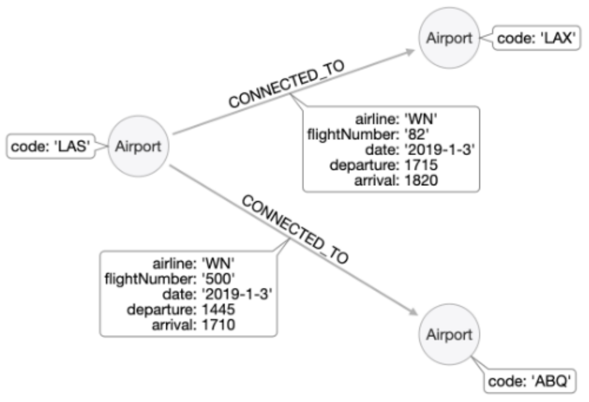


Write and execute the Cypher code to create the nodes and relationships for this simple airport model.

Hint: Since you have a small amount of data here, you can use CREATE if you ensure the nodes and relationships you create are unique.

Exercise 2.1: Implement the starting model containing `Airport` nodes (Solution)

Here is the starting airport model:

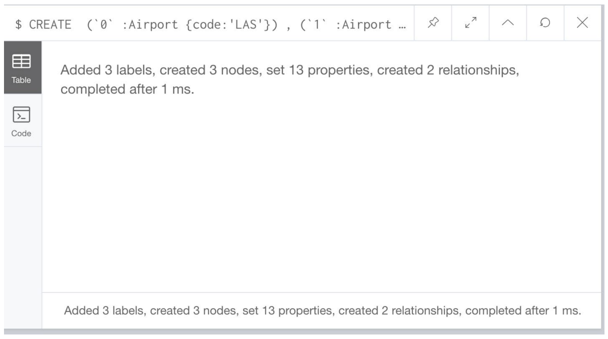


Write and execute the Cypher code to create the nodes and relationships for this simple airport model.

Hint: Since you have a small amount of data here, you can use CREATE if you ensure the nodes and relationships you create are unique.

```
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`)
```

The result returned should be:



```
$ CREATE  (`0` :Airport {code:'LAS'}) , (`1` :Airport …
```

```
Added 3 labels, created 3 nodes, set 13 properties, created 2 relationships,
completed after 1 ms.

Added 3 labels, created 3 nodes, set 13 properties, created 2 relationships, completed after 1 ms.
```

Exercise 2.2: Display the newly-created nodes (Instructions)

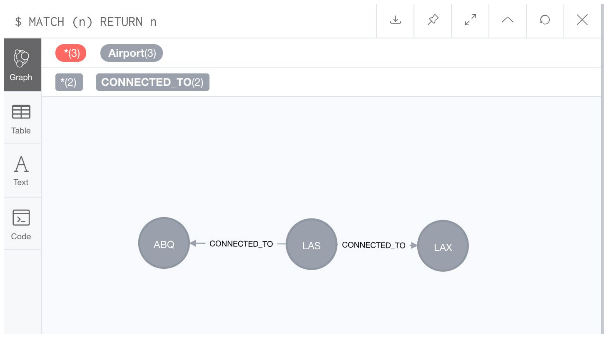
Write and execute a Cypher query to return all nodes in the graph.

Exercise 2.2: Display the newly-created nodes (Solution)

Write and execute a Cypher query to return all nodes in the graph.

```
MATCH (n)
RETURN n
```

The result returned should be:


```
$ MATCH (n) RETURN n
```

Exercise 2.3: Query the graph (Instructions)

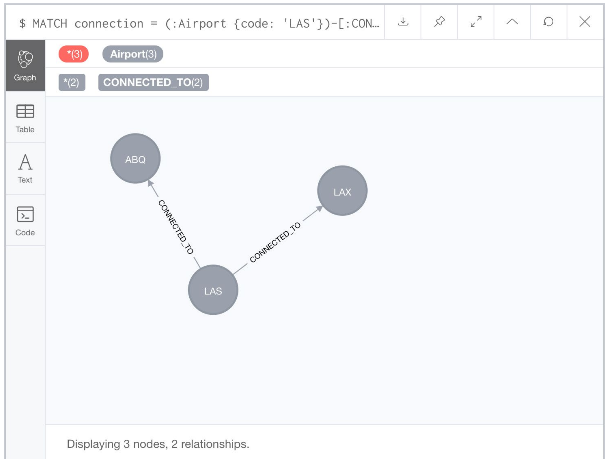
Write and execute a Cypher query to return all connections leaving LAS.

Exercise 2.3: Query the graph (Solution)

Write and execute a Cypher query to return all connections leaving LAS.

```
MATCH connection = (:Airport {code: 'LAS'})-[:CONNECTED_TO]->(:Airport)
RETURN connection
```

The result returned should be:


```
$ MATCH connection = (:Airport {code: 'LAS'})-[:CON…
```

Displaying 3 nodes, 2 relationships.

Exercise 2: Getting started with the airport graph data model (Summary)

In this exercise, you created the initial graph for the airport graph data model that you will be working with. This graph is just a start. In the next exercise you will load more data into the graph.

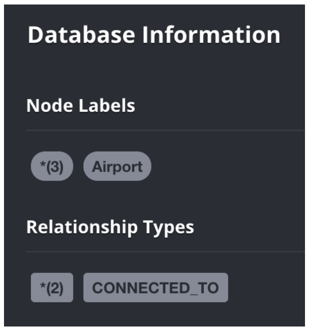
Continue to Exercise 3

Exercise 3

Exercise 3: Loading airport data (Preparations)

Before you begin this exercise, make sure that you have created the three `Airport` nodes in the previous exercise.

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


Database Information

Node Labels

*(3) Airport

Relationship Types

*(2) CONNECTED_TO

If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise:

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`)
```

Exercise 3: Loading airport data (Overview)

As part of our modeling workflow, we want to answer this initial question:

As an air travel enthusiast, I want to know how airports are connected so that I can find the busiest ones.

In this exercise you will load airport and flight data for 1000 flights. This will be the data that you use to begin the implementation of the graph model and the refactoring that will occur in the graph. The data that you load is in CSV format with headers. Once you have loaded the data, you will explore the data.

- **Exercise 3.1**: Preview the data.
- **Exercise 3.2**: Load the airport data.
- **Exercise 3.3**: Find the airports with most outgoing connections.
- **Exercise 3.4**: Find the airports with the most incoming connections.
- **Exercise 3.5**: Find all the connections into Las Vegas (LAS).

- **Exercise 3.6**: Find all the connections from Las Vegas (LAS) to Los Angeles (LAX).
- **Exercise 3.7**: Find a specific connection.

Go to the next page to start this exercise.

Exercise 3.1: Preview the data. (Instructions)

Let’s preview what the data we will be loading looks like.

Execute the following query to retrieve the first five rows of data to see what data we’ve got to work with:

```
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
RETURN row
LIMIT 5
```

Note: If your course data files are placed in the **import** folder for the database, then you would use the following syntax to perform the query without needing to use the URL for the data:

```
LOAD CSV WITH HEADERS FROM  'file:///flights_2019_1k.csv' AS row
RETURN row
LIMIT 5
```

This query:

- Loads the file 'flights__2019_1k'
- Iterates over the file, referring to each line as the variable `row`.
- Returns the first 5 lines in the file.

We have many different fields in this CSV file, but the ones that will be helpful for answering our question are:

- Origin
- Dest
- FlightNum

Exercise 3.2: Load the airport data. (Instructions)

Execute the following query to create nodes and relationships for these connections:

```
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime)
```

Note: If your course data files are placed in the **import** folder for the database, then you would use the following syntax to perform the query without needing to use the URL for the data:

```
LOAD CSV WITH HEADERS FROM  'file:///flights_2019_1k.csv' AS row
```

```
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime)
```

This Cypher code:

- Iterates through each row in the file.
- Creates nodes with the `Airport` label for the origin and destination airports if they don't already exist.
- Creates a `CONNECTED_TO` relationship between origin and destination airports for each row in the file.

By default properties will be stored as strings. We know that `year`, `month`, and `day` are actually numeric values so we coerce them using the `toInteger` function.

Exercise 3.3: Find the airports with the most outgoing connections. (Instructions)

Write and execute a Cypher query that returns the top ten airports that have the most outgoing connections.

Exercise 3.3: Find the airports with the most outgoing connections. (Solution)

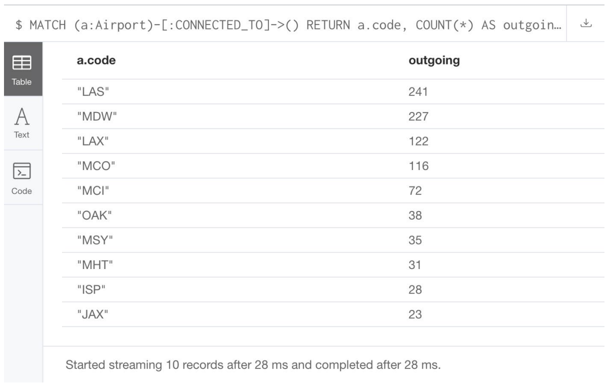
Write and execute a Cypher query that returns the top ten airports that have the most outgoing connections.

```
MATCH (a:Airport)-[:CONNECTED_TO]->()
RETURN a.code, COUNT(*) AS outgoing
ORDER BY outgoing DESC
LIMIT 10
```

This query:

- Finds every node with the `Airport` label.
- Finds all outgoing `CONNECTED_TO` relationships.
- Counts them up grouped by airport.
- Returns the `Airport` nodes and the `outgoing` count in descending order by `outgoing`.
- Limits the number of airports returned to 10.

The result returned should be:


```
$ MATCH (a:Airport)-[:CONNECTED_TO]->() RETURN a.code, COUNT(*) AS outgoin…
```

a.code	outgoing
"LAS"	241
"MDW"	227
"LAX"	122
"MCO"	116
"MCI"	72
"OAK"	38
"MSY"	35
"MHT"	31
"ISP"	28
"JAX"	23

Started streaming 10 records after 28 ms and completed after 28 ms.

Exercise 3.4: Find the airports with the most incoming connections. (Instructions)

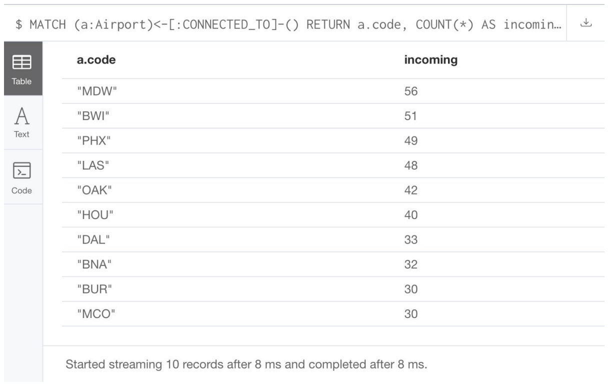
Write and execute a Cypher query that returns the top ten airports that have the most incoming connections.

Exercise 3.4: Find the airports with the most incoming connections. (Solution)

Write and execute a Cypher query that returns the top ten airports that have the most incoming connections.

```
MATCH (a:Airport)<-[:CONNECTED_TO]-()
RETURN a.code, COUNT(*) AS incoming
ORDER BY incoming DESC
LIMIT 10
```

The result returned should be:


```
$ MATCH (a:Airport)<-[:CONNECTED_TO]-() RETURN a.code, COUNT(*) AS incomin…
```

a.code	incoming
"MDW"	56
"BWI"	51
"PHX"	49
"LAS"	48
"OAK"	42
"HOU"	40
"DAL"	33
"BNA"	32
"BUR"	30
"MCO"	30

Started streaming 10 records after 8 ms and completed after 8 ms.

Exercise 3.5: Find all the connections into Las Vegas (LAS). (Instructions)

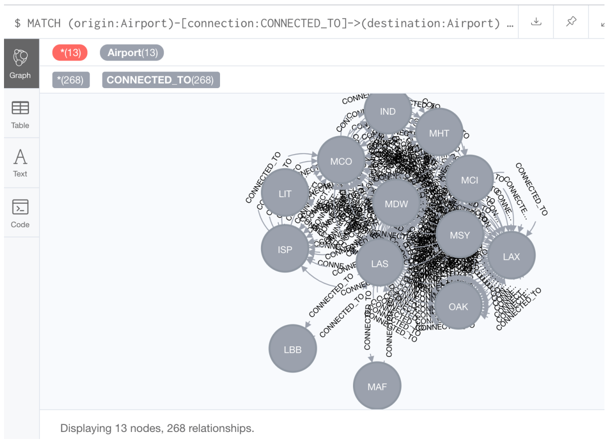
Write and execute a Cypher query that returns all of the incoming connections to Las Vegas (LAS).

Exercise 3.5: Find the airports with the most incoming connections. (Solution)

Write and execute a Cypher query that returns all of the incoming connections to Las Vegas (LAS).

```
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, connection
```

The result returned should be:



```
$ MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport) …
```

Displaying 13 nodes, 268 relationships.

Exercise 3.6: Find all the connections from Las Vegas (LAS) to Los Angeles (LAX). (Instructions)

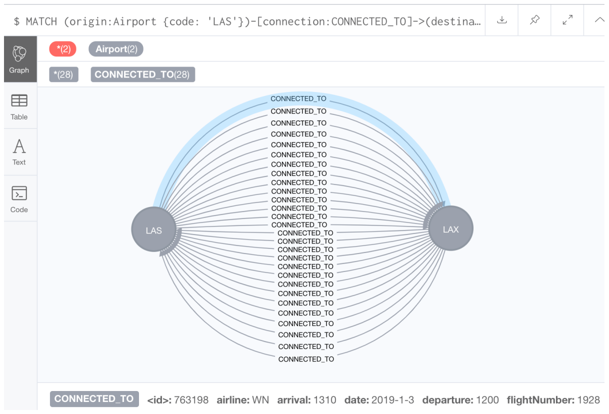
Write and execute a Cypher query that returns all of the connections from Las Vegas (LAS) to Los Angeles (LAX).

Exercise 3.6: Find all the connections from Las Vegas (LAS) to Los Angeles (LAX). (Solution)

Write and execute a Cypher query that returns all of the connections from Las Vegas (LAS) to Los Angeles (LAX).

```
  MATCH (origin:Airport {code: 'LAS'})-[connection:CONNECTED_TO]->(destination:Airport {code:
 'LAX'})
  RETURN origin, destination, connection
```

The result returned should be:



```
$ MATCH (origin:Airport {code: 'LAS'})-[connection:CONNECTED_TO]->(destina…
```

CONNECTED_TO <id>: 763198 airline: WN arrival: 1310 date: 2019-1-3 departure: 1200 flightNumber: 1928

Exercise 3.7: Find a specific flight. (Instructions)

Write and execute a Cypher query that returns connection information about a particular flight where the airline is 'WN' and the flight number is '1016'

Exercise 3.7: Find a specific flight. (Solution)

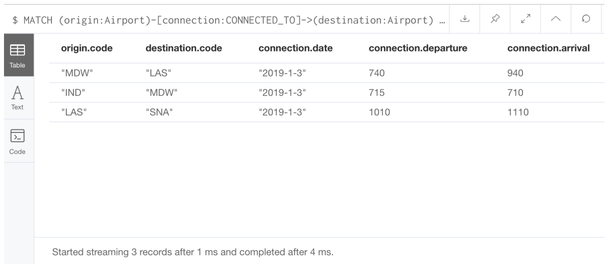
Write and execute a Cypher query that returns connection information about a particular flight where the

airline is 'WN' and the flight number is '1016'

```
MATCH  (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
WHERE connection.airline = 'WN' AND connection.flightNumber = '1016'
RETURN origin.code, destination.code, connection.date, connection.departure, connection.arrival
```

This query is reasonably fast because we only have 1,000 connections between airports, but under the covers we're actually doing a lot of unnecessary work.

The result returned should be:



```
$ MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport) …
```

origin.code	destination.code	connection.date	connection.departure	connection.arrival
"MDW"	"LAS"	"2019-1-3"	740	940
"IND"	"MDW"	"2019-1-3"	715	710
"LAS"	"SNA"	"2019-1-3"	1010	1110

Started streaming 3 records after 1 ms and completed after 4 ms.

Exercise 3: Loading airport Data (Summary)

In this exercise, you added more data to the graph representing more airports and 1000 connections between them (flights). This graph is still small, but it has enough data that we can start to examine the performance of queries to determine if we can evolve the model (refactor) to yield better query performance.

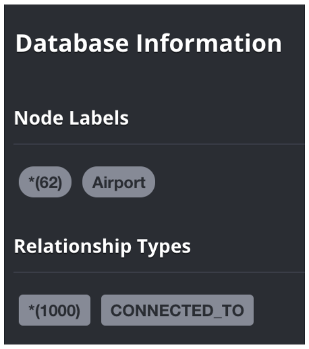
Continue to Exercise 4

Exercise 4

Exercise 4: Profiling queries (Preparations)

Before you begin this exercise, make sure that you have loaded the **flights_2019_1k** data in the previous exercise.

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



If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise:

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

Exercise 4: Profile queries (Overview)

As part of our modeling workflow, we must analyze queries for our domain and understand their performance with a set of data. We currently have data for 1000 connections between airports (flights) and we want to see the cost our queries against the existing data. Later in these exercises, you will add more data to the graph, but for now we will analyze queries for the existing data.

Note: The PROFILE numbers you see in this browser guide may differ slightly from what you see on your system, because of differences in versions of Neo4j.

In this exercise you will profile and understand the behavior of a couple of queries:

- **Exercise 4.1**: What is the cost of finding all flights that land in Las Vegas (LAS)?
- **Exercise 4.2**: What is the cost of finding all flights for airline 'WN' with flight number '1016'?

Go to the next page to start this exercise.

Exercise 4.1: What is the cost of finding all flights that land in Las Vegas (LAS)? (Instructions)

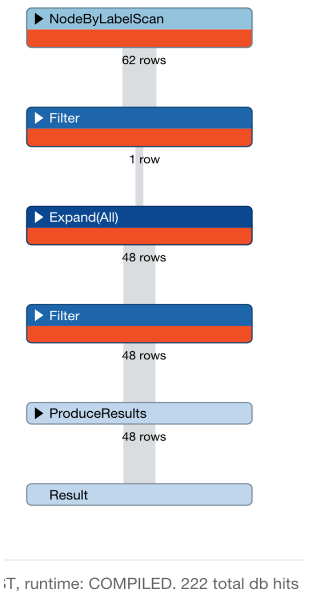
Write and execute the Cypher code to profile the query to retrieve all flights that land in Las Vegas (LAS)?

Exercise 4.1: What is the cost of finding all flights that land in Las Vegas (LAS)? (Solution)

Write and execute the Cypher code to profile the query to retrieve all flights that land in Las Vegas (LAS)?

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c
```

The result returned should be:



Expand each section of the profile shown. One thing that we could do to reduce the number of db hits is to create an index on `Airport.code` and the current query is reading all `Airport` node labels to find the airport for LAS. This query does return 13 nodes and 268 relationships so getting the number of db hits lower than 160 (222 - 62) is probably all we can do with this model.

Exercise 4.2: What is the cost of finding all flights for airline 'WN' with flight number '1016'? (Instructions)

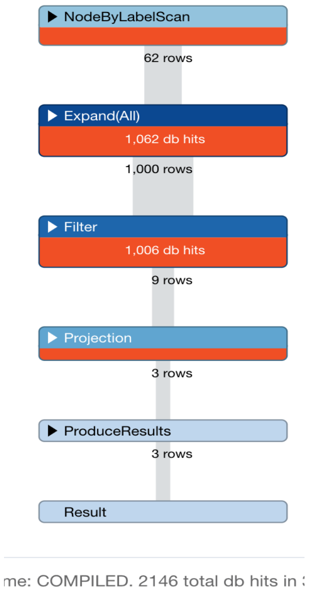
Write and execute the Cypher code to profile the query to retrieve all flights for airline 'WN' with flight number '1016'?

Exercise 4.2: What is the cost of finding all flights for airline 'WN' with flight number '1016'? (Solution)

Write and execute the Cypher code to profile the query to retrieve all flights for airline 'WN' with flight number '1016'?

```
PROFILE
MATCH  (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
WHERE connection.airline = 'WN' AND connection.flightNumber = '1016'
RETURN origin.code, destination.code, connection.date, connection.departure, connection.arrival
```

The result returned should be:



Expand each section of the profile shown. Again, creating an index on Airport.code will reduce the number of db hits by 62. The query has to read all data for all `CONNECTED_TO` relationships to find the correct flights. The number of db hits of 2146 is too high. What we are going to do is change (refactor) our current model so that we do not have to visit all `CONNECTED_TO` relationships to satisfy the query. We will do this by creating a `Flight` node from a relationship.

Exercise 4: Profiling queries (Summary)

In this exercise, you profiled the performance of a couple of queries and found that we need to at a minimum:

- Add an index to the `Airport.code` property.
- Model a flight as a `Flight` node.

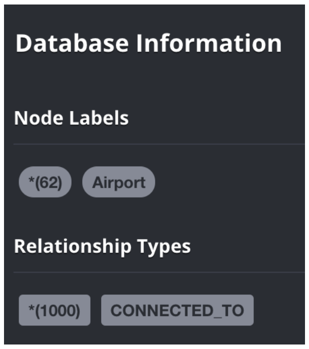
Continue to Exercise 5

Exercise 5

Exercise 5: Creating the `Flight` node from the `CONNECTED_TO` relationships (Preparations)

Before you begin this exercise, make sure that you have loaded the **flights_2019_1k** data in Exercise 3.

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



If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise.

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

Exercise 5: Creating the `Flight` node from the `CONNECTED_TO` relationships (Overview)

You have seen from profiling a few queries that the current model needs refactoring. You will refactor the current graph to fit the new model and then you will profile the queries again.

Here are the tasks you will perform:

- **Exercise 5.1**: Add an index to the `Airport.code` property.
- **Exercise 5.2**: Add a unique constraint to the `Flight.flightId` property.

- **Exercise 5.3**: Add an index to the `Flight.number` property.
- **Exercise 5.4**: Refactor the graph to create `Flight` nodes from the `CONNECTED_TO` relationships.
- **Exercise 5.5**: Profile our first query that finds all flights that land in Las Vegas.
- **Exercise 5.6**: Profile our second query that finds all flights for airline 'WN' with flight number '1016'.
- **Exercise 5.7**: Delete the `CONNECTED_TO` relationship from the graph.

Go to the next page to start this exercise.

Exercise 5.1: Add an index to the `Airport.code` property. (Instructions)

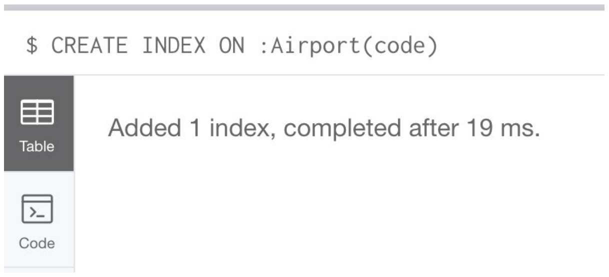
Write and execute the Cypher statement to add an index to the Airport nodes based upon the code property.

Exercise 5.1: Add an index to the `Airport.code` property. (Solution)

Write and execute the Cypher statement to add an index to the `Airport` nodes based upon the code property.

```
CREATE INDEX ON :Airport(code)
```

The result returned should be:



```
$ CREATE INDEX ON :Airport(code)
```

Added 1 index, completed after 19 ms.

Exercise 5.2: Add a unique constraint to the `Flight.flightId` property. (Instructions)

You will be creating nodes of type `Flight`. One of the properties you will adding to a `Flight` node is an ID that makes each `Flight` node unique. The name of this property will be `flightId`.

Write and execute the Cypher statement to add a unique constraint to the `Flight` nodes based upon the `flightId` property.

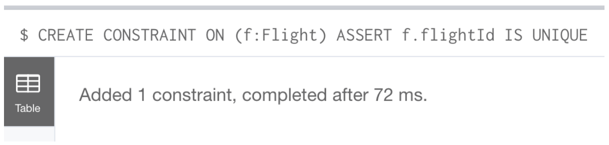
Exercise 5.2: Add a unique constraint to the `Flight.flightId` property. (Solution)

You will be creating nodes of type `Flight`. One of the properties you will adding to a `Flight` node is an ID that makes each `Flight` node unique. The name of this property will be `flightId`.

Write and execute the Cypher statement to add a unique constraint to the `Flight` nodes based upon the `flightId` property.

```
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE
```

The result returned should be:


```
$ CREATE CONSTRAINT ON (f:Flight) ASSERT f.flightId IS UNIQUE
```

Added 1 constraint, completed after 72 ms.

Exercise 5.3: Add an index to the `Flight.number` property. (Instructions)

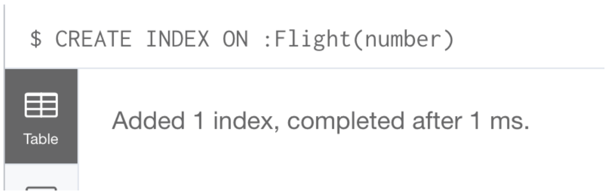
Write and execute the Cypher statement to add an index to the `Flight` nodes based upon the `number` property.

Exercise 5.3: Add an index to the `Flight.number` property. (Solution)

Write and execute the Cypher statement to add an index to the `Flight` nodes based upon the `number` property.

```
CREATE INDEX ON :Flight(number)
```

The result returned should be:


```
$ CREATE INDEX ON :Flight(number)
```

Added 1 index, completed after 1 ms.

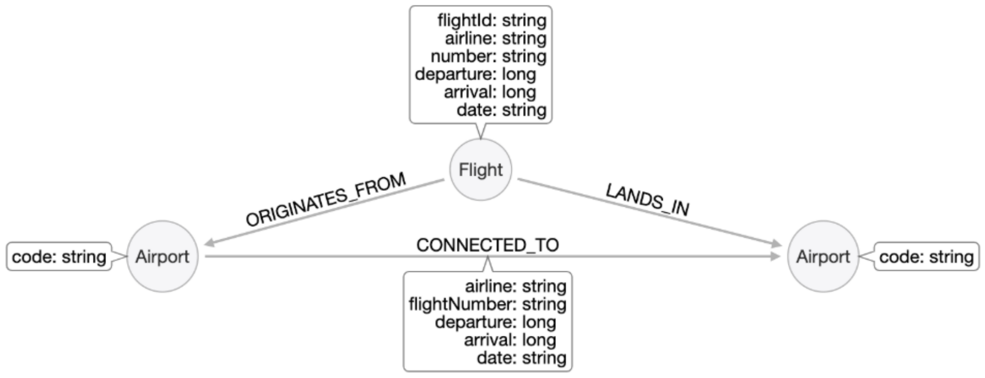
Exercise 5.4: Refactor the graph to create Flight nodes from the CONNECTED_TO relationships. (Instructions)

Flight nodes will be created from existing CONNECTED_TO relationships where you must keep track of the origin airport and the destination airport. For example: (origin:Airport)-[CONNECTED_TO]→(destination:Airport). The properties for the Flight nodes will be set as follows:

- `flightId`: connection.airline + connection.flightNumber + `''` + *connection.date* + `''` + origin.code + `'_'` + destination.code
- `date`: connection.date
- `airline`: connection.airline
- `number`: connection.flightNumber
- `departure`: connection.departure
- `arrival`: connection.arrival

In addition, the new relationships between Flight and Airport nodes will be ORIGINATES_FROMand LANDS_IN.

Here is the model you will refactor to:



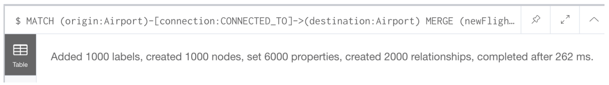
Write and execute the Cypher statement traverse all `CONNECTED_TO` relationships to create the `Flight` nodes.

Exercise 5.4: Refactor the graph to create `Flight` nodes from the `CONNECTED_TO` relationships. (Solution)

Write and execute the Cypher statement traverse all `CONNECTED_TO` relationships to create the `Flight` nodes.

```
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight { flightId: connection.airline +  connection.flightNumber +
       '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination)
```

The result returned should be:



```
$ MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport) MERGE (newFligh…
Added 1000 labels, created 1000 nodes, set 6000 properties, created 2000 relationships, completed after 262 ms.
```

Exercise 5.5: Profile our first query that finds all flights that land in Las Vegas. (Instructions)

The previous query was:

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c
```

Modify and execute this query to use the new model and compare your results for this query profile after the refactoring

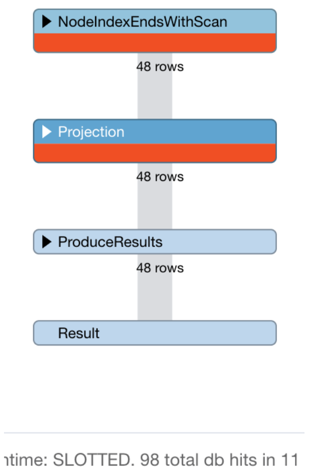
Exercise 5.5: Profile our first query that finds all flights that land in Las Vegas. (Solution)

Modify and execute this query to use the new model and compare your results for this query profile after the refactoring

```
PROFILE
```

```
MATCH (f:Flight)
WHERE f.flightId ENDS WITH 'LAS'
RETURN f.flightId
```

The result returned should be:



ıtime: SLOTTED. 98 total db hits in 11

The original query had 222 db hits. The revised query with the refactored model is better with 98 db hits.

Exercise 5.6: Profile our second query that finds all flights for airline 'WN' with flight number '1016'. (Instructions)

The previous query was:

```
PROFILE
MATCH  (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
WHERE connection.airline = 'WN' AND connection.flightNumber = '1016'
RETURN origin.code, destination.code, connection.date, connection.departure, connection.arrival
```

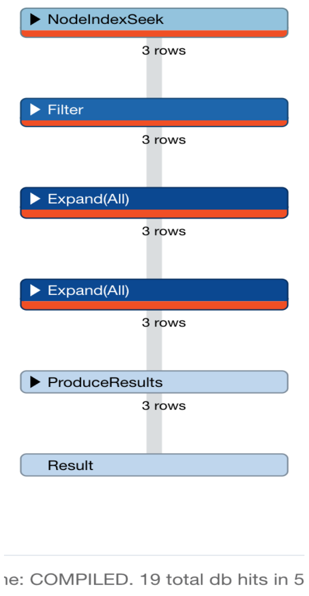
Modify and execute this query to use the new model and compare your results for this query profile after the refactoring

Exercise 5.6: Profile our second query that finds all flights for airline 'WN' with flight number '1016'. (Solution)

Modify and execute this query to use the new model and compare your results for this query profile after the refactoring

```
PROFILE
MATCH (origin)<-[:ORIGINATES_FROM]-(flight:Flight)-
      [:LANDS_IN]->(destination)
WHERE flight.airline = 'WN' AND
      flight.number = '1016' RETURN origin, destination, flight
```

The result returned should be:



ıe: COMPILED. 19 total db hits in 5

The original query had 2146 db hits. The revised query with the refactored model is better with 19 db hits.

Exercise 5.7: Delete the `CONNECTED_TO` relationship from the graph. (Instructions)

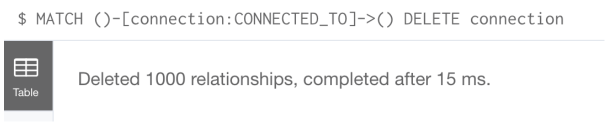
Write and execute the Cypher statement to remove the `CONNECTED_TO` relationships from the graph.

Exercise 5.7: Delete the `CONNECTED_TO` relationship from the graph. (Solution)

Write and execute the Cypher statement to remove the `CONNECTED_TO` relationships from the graph.

```
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection
```

The result returned should be:



```
$ MATCH ()-[connection:CONNECTED_TO]->() DELETE connection
```

Deleted 1000 relationships, completed after 15 ms.

Exercise 5: Creating the `Flight` node from the `CONNECTED_TO` relationships (Summary)

In this exercise, you refactored the nodes and relationships in the graph for a model and implementation that performs better. You added indexes and constraints and created `Flight` nodes from the `CONNECTED_TO` relationships. You also added the `ORIGINATES_FROM` and `LANDS_IN` relationships between `Flight` nodes and `Airport` nodes. You profiled queries to confirm that the refactoring improved query performance. Finally, you deleted the `CONNECTED_TO` relationship.

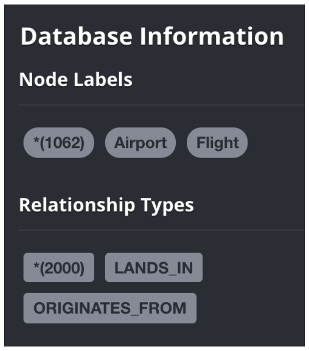
Continue to Exercise 6

Exercise 6

Exercise 6: Creating the `AirportDay` node from the `Airport` and `Flight` nodes (Preparations)

Before you begin this exercise, make sure that you have performed the refactoring from Exercise 5.

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



If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise.

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime);
CREATE INDEX ON :Airport(code);
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE;
CREATE INDEX ON :Flight(number);
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight { flightId: connection.airline +  connection.flightNumber +
       '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination);
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

`AirportDay` **`Airport`** **`Flight`**

Exercise 6: Creating the node from the and nodes (Overview)

Although we have refactored our graph to make two queries perform better, we have yet another query that our model needs to support. The new query is one where a user is trying to find flights from one airport to another on a particular day. In this exercise, you will first add more data to the graph and then profile a new query to see that the query is not performing well. Then you will refactor the graph to implement a new model and profile again.

Here are the tasks you will perform:

- **Exercise 6.1**: Add more data to the graph.
- **Exercise 6.2**: Profile the new query that finds all flights from airport to another on a given day.
- **Exercise 6.3**: Add a unique constraint on the `airportDayId` property of the `AirportDay` node.
- **Exercise 6.4**: Refactor the graph to create `AirportDay` nodes from `Flight` nodes and `Airport` nodes.
- **Exercise 6.5**: Profile our first query that finds all flights that land in Las Vegas.
- **Exercise 6.6**: Profile our second query that finds all flights for airline 'WN' with flight number '1016'.
- **Exercise 6.7**: Profile the new query that finds all flights from airport to another on a given day.

Go to the next page to start this exercise.

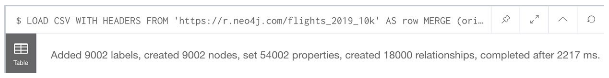
Exercise 6.1: Add more data to the graph. (Instructions)

Execute this Cypher statement to add 10k flights to the graph.

```
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_10k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (newFlight)-[:ORIGINATES_FROM]->(origin)
MERGE (newFlight)-[:LANDS_IN]->(destination)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

The result returned should be:



```
$ LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_10k' AS row MERGE (ori…
Added 9002 labels, created 9002 nodes, set 54002 properties, created 18000 relationships, completed after 2217 ms.
```

Exercise 6.2: Profile the new query that finds all flights from airport to another on a given day. (Instructions)

Suppose you are looking for all flights that go from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3).

Using the current model in the graph, write and execute the query to find those flights.

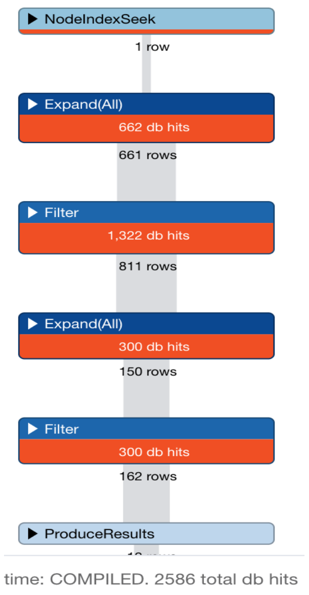
Exercise 6.2: Profile the new query that finds all flights from airport to another on a given day. (Solution)

Suppose you are looking for all flights that go from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3).

Using the current model in the graph, write and execute the query to find those flights.

```
PROFILE MATCH (origin:Airport {code: 'LAS'})
    <-[:ORIGINATES_FROM]-(flight:Flight)-[:LANDS_IN]->
    (destination:Airport {code: 'MDW'})
WHERE flight.date = '2019-1-3'
RETURN origin, destination, flight
```

The result returned should be:



```
NodeIndexSeek
1 row
Expand(All)
662 db hits
661 rows
Filter
1,322 db hits
811 rows
Expand(All)
300 db hits
150 rows
Filter
300 db hits
162 rows
ProduceResults
time: COMPILED. 2586 total db hits
```

Here we see 2586 db hits. We have increased the number of flights in the database tenfold, but we might be able to do better. We will give it a try and create the `AirportDay` node where each `Airport` will be connected by a `HAS_DAY` relationship to an `AirportDay` node for each day.

Exercise 6.3: Add a unique constraint on the `AirportDay.airportDayId` property. (Instructions)

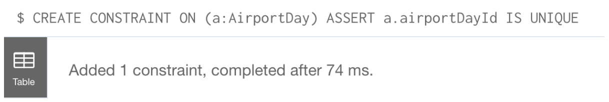
Write and execute the Cypher statement to add a unique constraint to the `AirportDay` nodes based upon the `airportDayId` property.

Exercise 6.3: Add a unique constraint on the `AirportDay.airportDayId` property. (Solution)

Write and execute the Cypher statement to add a unique constraint to the `AirportDay` nodes based upon the `airportDayId` property.

```
CREATE CONSTRAINT ON (a:AirportDay)
ASSERT a.airportDayId IS UNIQUE
```

The result returned should be:



```
$ CREATE CONSTRAINT ON (a:AirportDay) ASSERT a.airportDayId IS UNIQUE
```

Added 1 constraint, completed after 74 ms.

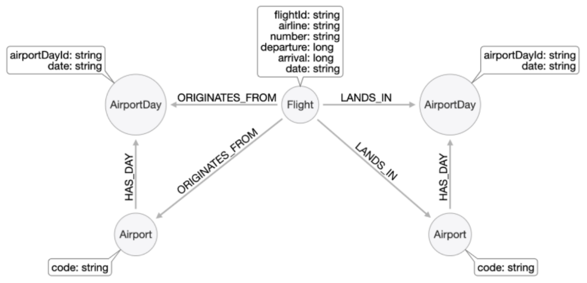
Exercise 6.4: Refactor the graph to create `AirportDay` nodes from `Flight` nodes and `Airport` `nodes`. (Instructions)

`AirportDay` nodes will be created from existing `Flight` nodes and their relationships with `Airports`. Each `Airport` node will have the `HAS_DAY` relationship with an `AirportDay` node which represents a particular day.

The properties for the `AirportDay` nodes will be set as follows:

- `airportDayId`: origin.code + '' + *flight.date or destination.code* '' + flight.date , depending on whether the relationship is an `ORIGINATES_FROM` or `LANDS_IN` relationship.
- `date`: flight.date

Here is the model you will refactor to:



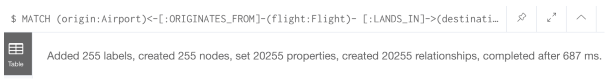
Write and execute the Cypher statement to go through all `Flights` and how they are connected to `Airports` to create the `AirportDay` nodes.

Exercise 6.4: Refactor the graph to create `AirportDay` nodes from `Flight` nodes and `Airport` nodes. (Solution)

Write and execute the Cypher statement to go through all `Flights` and how they are connected to `Airports` to create the `AirportDay` nodes.

```
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
       [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
      {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
       {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)
```

The result returned should be:



```
$ MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)- [:LANDS_IN]->(destinati…
```

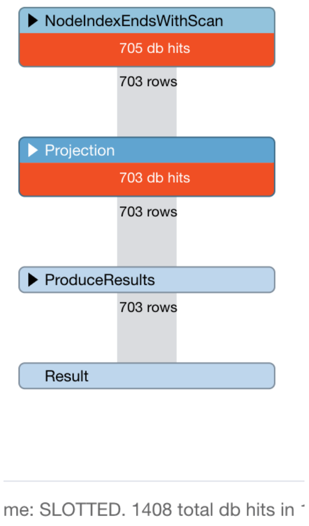
Added 255 labels, created 255 nodes, set 20255 properties, created 20255 relationships, completed after 687 ms.

Exercise 6.5: Profile our first query that finds all flights that land in Las Vegas. (Instructions)

Execute this query again with the refactored graph:

```
PROFILE
MATCH (f:Flight)
WHERE f.flightId ENDS WITH 'LAS'
RETURN f.flightId
```

The result returned should be:



me: SLOTTED. 1408 total db hits in

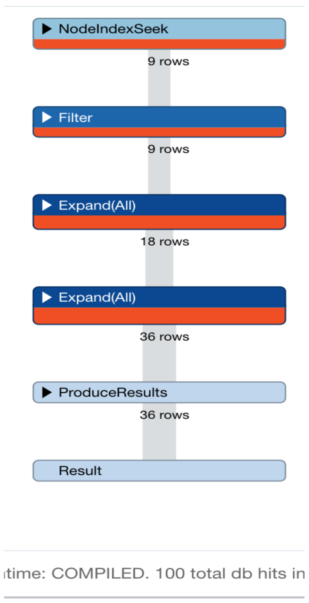
This same query with the previous model had 98 db hits. This query now has 1408 db hits, but we have increased the number of flights by a factor of 10 so this result is expected.

Exercise 6.6: Profile our second query that finds all flights for airline 'WN' with flight number '1016'. (Instructions)

Execute this query again with the refactored graph:

```
PROFILE
MATCH (origin)<-[:ORIGINATES_FROM]-(flight:Flight)-
      [:LANDS_IN]->(destination)
WHERE flight.airline = 'WN' AND
      flight.number = '1016' RETURN origin, destination, flight
```

The result returned should be:



This query with the previous model had 19 db hits. This query now has 100 db hits, but we have increased the number of flights by a factor of 10 so this result is expected.

Exercise 6.7: Profile the new query that finds all flights from airport to another on a given day. (Instructions)

Our original query to find all flights that fly from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3) was:

```
PROFILE MATCH (origin:Airport {code: 'LAS'})
    <-[:ORIGINATES_FROM]-(flight:Flight)-[:LANDS_IN]->
    (destination:Airport {code: 'MDW'})
WHERE flight.date = '2019-1-3'
RETURN origin, destination, flight
```

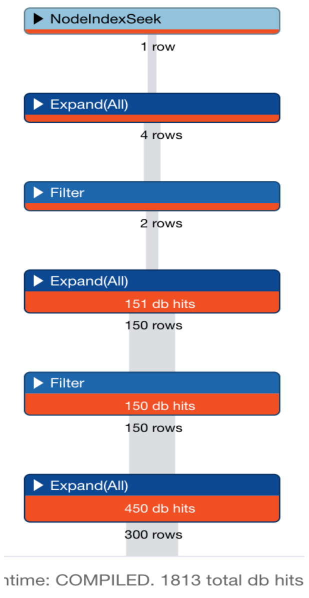
Use the refactored model to rewrite this query and execute it against the graph.

Exercise 6.7: Profile the new query that finds all flights from airport to another on a given day. (Solution)

Use the refactored model to rewrite this query and execute it against the graph.

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(:AirportDay
      {date: '2019-1-3'})<-[:ORIGINATES_FROM]-(flight:Flight),
      (flight)-[:LANDS_IN]->(:AirportDay
      {date: '2019-1-3'})<-[:HAS_DAY]-(destination:Airport {code: 'MDW'})
RETURN origin, destination, flight
```

The result returned should be:



The previous query before the refactoring yielded 2586 db hits. After the refactoring and rewriting the query, we see 1813 db hits which is an improvement.

Exercise 6: Creating the `AirportDay` node from the `Airport` and `Flight` nodes (Summary)

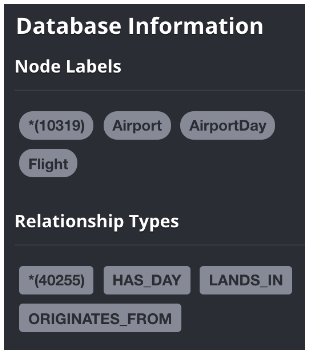
In this exercise, you refactored the nodes and relationships in the graph for a model and implementation that performs better. You created the `AirportDay` node from `Flight` and `Airport` nodes so that a date-specfic query will perform better. Then, you profiled queries to confirm that they perform better with the refactored graph.

Continue to Exercise 7

Exercise 7: Creating specific relationships (Preparations)

Before you begin this exercise, make sure that you have performed the refactoring from Exercise 6.

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


Database Information

Node Labels

*(10319) Airport AirportDay Flight

Relationship Types

*(40255) HAS_DAY LANDS_IN ORIGINATES_FROM

If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise.

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime);
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE;
CREATE INDEX ON :Flight(number);
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight { flightId: connection.airline +  connection.flightNumber +
     '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination);
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection;
CREATE INDEX ON :Airport(code);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_10k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
```

```
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (newFlight)-[:ORIGINATES_FROM]->(origin)
MERGE (newFlight)-[:LANDS_IN]->(destination);
CREATE CONSTRAINT ON (a:AirportDay)
ASSERT a.airportDayId IS UNIQUE;
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
       [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
      {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
       {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

Exercise 7: Creating specific relationships (Overview)

Neo4j is optimized for searching by unique relationship types and in this case the date of a flight provides that uniqueness.

You will refactor the model to change the `HAS_DAY` relationship to be the date of the flight instead, for example `ON_2019-1-3`.

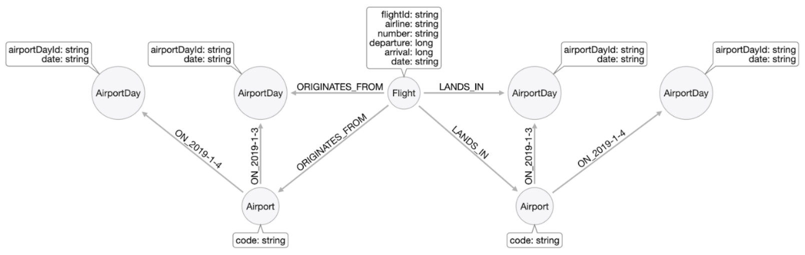
Here are the tasks you will perform:

- **Exercise 7.1**: Refactor the graph to change all `HAS_DAY` relationships to specific day relationships.
- **Exercise 7.2**: Rewrite and profile the previous query that finds all flights from an airport to another on a given day.

Go to the next page to start this exercise.

Exercise 7.1: Refactor the graph to change all `HAS_DAY` relationships to specific day relationships. (Instructions)

Rather than having the `HAS_DAY` relationship from `Airport` nodes to `AirportDay` nodes, we want a specific relationship that will be `ON_yyyy-dd-mm`. The revised model is:



Write and execute the Cypher statement to refactor the model.

Hint: Use apoc.create.relationship().

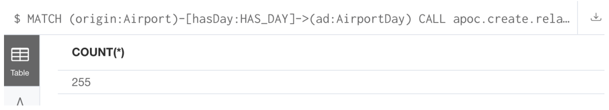
Exercise 7.1: Refactor the graph to change all `HAS_DAY` relationships to specific day relationships. (Solution)

Write and execute the Cypher statement to refactor the model.

Hint: Use apoc.create.relationship().

```
MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
CALL apoc.create.relationship(startNode(hasDay),
                              'ON_' + ad.date,
                              {},
                              endNode(hasDay) ) YIELD rel
RETURN COUNT(*)
```

The result returned should be:



```
$ MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay) CALL apoc.create.rela…
COUNT(*)
255
```

Exercise 7.2: Rewrite and profile the previous query that finds all flights from an airport to another on a given day. (Instructions)

Previously, we used this query to look for all flights that go from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3).

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(:AirportDay
      {date: '2019-1-3'})<-[:ORIGINATES_FROM]-(flight:Flight),
      (flight)-[:LANDS_IN]->(:AirportDay
      {date: '2019-1-3'})<-[:HAS_DAY]-(destination:Airport {code: 'MDW'})
RETURN origin, destination, flight
```

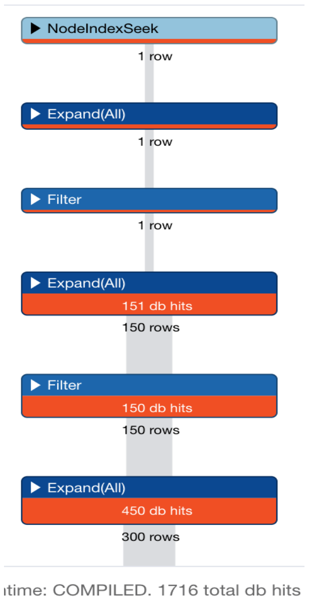
Modify this query to use the refactored graph and execute the query.

Exercise 7.2: Rewrite and profile the previous query that finds all flights from airport to another on a given day. (Solution)

Modify this query to use the refactored graph and execute the query.

```
PROFILE
MATCH (origin:Airport {code: 'LAS'})-
[:`ON_2019-1-3`]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight),
(flight)-[:LANDS_IN]->(destinationDay),
(destinationDay:AirportDay)<-[:`ON_2019-1-3`]-
(destination:Airport {code: 'MDW'})
RETURN flight.date, flight.number, flight.airline,
flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

The result returned should be:



The previous query before this refactoring yielded 1813 db hits. After this refactoring and rewriting the query, we see 1716 db hits, a slight improvement. If the graph had more data, we would probably see a difference in the performance after the last refactoring.

Exercise 7: Creating specific relationships (Taking it further)

Execute these previous two queries to see if the refactoring to specific relationships changed their performance:

Previously, this query showed 100 db hits:

```
PROFILE
MATCH (origin)<-[:ORIGINATES_FROM]-(flight:Flight)-
      [:LANDS_IN]->(destination)
WHERE flight.airline = 'WN' AND
      flight.number = '1016' RETURN origin, destination, flight
```

Previously, this query showed 1408 db hits:

```
PROFILE
MATCH (f:Flight)
WHERE f.flightId ENDS WITH 'LAS'
RETURN f.flightId
```

Exercise 7: Creating specific relationships (Summary)

In this exercise, you refactored the `HAS_DAY` relationship between the `Airport` and the `AirportDay` to be a specific date relationship. For the existing graph with 10K flight nodes, this change did not make a significant difference. If the graph had more data, we would probably see an improvement.

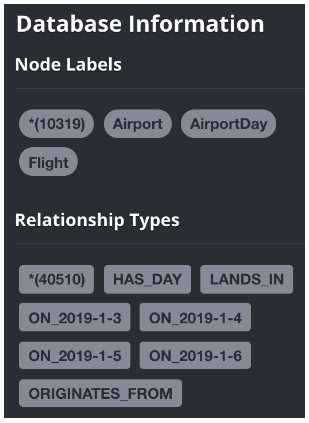
In the next exercise, you will load even more data in to the graph.

Continue to Exercise 8

Exercise 8: Refactoring large graphs (Preparations)

Before you begin this exercise, make sure that you have performed the refactoring from Exercise 7.

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


Database Information

Node Labels

*(10319) Airport AirportDay Flight

Relationship Types

*(40510) HAS_DAY LANDS_IN ON_2019-1-3 ON_2019-1-4 ON_2019-1-5 ON_2019-1-6 ORIGINATES_FROM

If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise.

```
MATCH (n) DETACH DELETE n;
CALL apoc.schema.assert({},{},true);
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime);
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE;
CREATE INDEX ON :Flight(number);
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight { flightId: connection.airline +  connection.flightNumber +
       '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination);
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection;
CREATE INDEX ON :Airport(code);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_10k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (newFlight)-[:ORIGINATES_FROM]->(origin)
MERGE (newFlight)-[:LANDS_IN]->(destination);
CREATE CONSTRAINT ON (a:AirportDay)
ASSERT a.airportDayId IS UNIQUE;
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
       [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
      {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
       {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay);
MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
CALL apoc.create.relationship(startNode(hasDay),
                              'ON_' + ad.date,
                              {},
                              endNode(hasDay) ) YIELD rel
RETURN COUNT(*)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

Exercise 8: Refactoring large graphs (Overview)

As your graph gets larger and you profile the queries that are important to your use cases, you will find that the refactoring is not possible without the ability to batch the refactoring into chunks of processing. The APOC library has a number of procedures that are useful for refactoring a graph. The procedure that you use to process batches of work on a graph is `apoc.periodic.commit()`. In this exercise, you will load 100K flights into the graph and use APOC to help you refactor the graph.

Here are the tasks you will perform:

- **Exercise 8.1**: Load more airports into the graph.
- **Exercise 8.2**: Load 100K flights into the graph.
- **Exercise 8.3**: Prepare the `Flight` nodes for batch processing.
- **Exercise 8.4**: Refactor the graph to use `AirportDay` by batching commits.
- **Exercise 8.5**: Prepare the `AirportDay` nodes for batch processing.
- **Exercise 8.6**: Refactor the graph to use specific relationships by batching commits.
- **Exercise 8.7**: Profile the query that finds flights on a specific day using the general relationship.
- **Exercise 8.8**: Profile the query that finds flights on a specific day using the specific relationship.

Go to the next page to start this exercise.

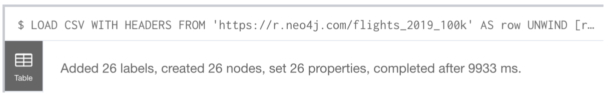
Exercise 8.1: Load more airports into the graph. (Instructions)

Execute this Cypher statement to load the Airport nodes from the CSV file:

```
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_100k'  AS row
UNWIND [row.Origin, row.Dest] AS airport
WITH DISTINCT airport
MERGE (:Airport {code: airport})
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

The result returned should be:



```
$ LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_100k' AS row UNWIND [r…
Added 26 labels, created 26 nodes, set 26 properties, completed after 9933 ms.
```

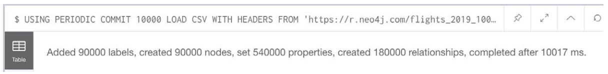
Exercise 8.2: Load 100k flights into the graph. (Instructions)

Execute this Cypher statement to load the Flight nodes from the CSV file:

```
USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_100k' AS row
MATCH (origin:Airport {code: row.Origin})
MATCH (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

The result returned should be:



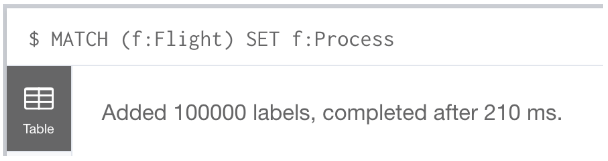
```
$ USING PERIODIC COMMIT 10000 LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_100…
Added 90000 labels, created 90000 nodes, set 540000 properties, created 180000 relationships, completed after 10017 ms.
```

Exercise 8.3: Prepare the `Flight` nodes for batch processing. (Instructions)

Execute this Cypher code to add the `Process` label to all `Flight` nodes that will be processed for the batch refactoring:

```
MATCH (f:Flight)
SET f:Process
```

The result returned should be:



```
$ MATCH (f:Flight) SET f:Process
Added 100000 labels, completed after 210 ms.
```

Exercise 8.4: Refactor the graph to use `AirportDay` by batching commits. (Instructions)

Here is the code that you used previously to refactor the graph to use Airport day:

```
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
       [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
      {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
       {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)
```

Modify this Cypher code to batch process the `Flight` nodes in batches of 500 and execute the code.

Hint: `Use apoc.periodic.commit().`

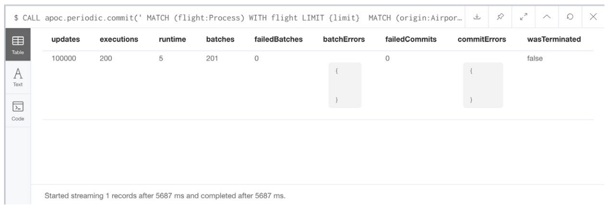
Exercise 8.4: Refactor the graph to use `AirportDay` by batching commits. (Solution)

Modify this Cypher code to batch process the `Flight` nodes in batches of 500 and execute the query.

Hint: `Use apoc.periodic.commit().`

```
CALL apoc.periodic.commit('
MATCH (flight:Process)
WITH flight LIMIT {limit}

MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight)-[:LANDS_IN]->(destination:Airport)

MERGE (originAirportDay:AirportDay {airportDayId: origin.code + "_" + flight.date})
ON CREATE SET originAirportDay.date = flight.date

MERGE (destinationAirportDay:AirportDay {airportDayId: destination.code + "_" + flight.date})
ON CREATE SET destinationAirportDay.date = flight.date

MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (originAirportDay)<-[:ORIGINATES_FROM]-(flight)
MERGE (flight)-[:LANDS_IN]-(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)

REMOVE flight:Process
RETURN COUNT(*)

',{limit:500}
)
```

The result returned should be:



```
$ CALL apoc.periodic.commit(' MATCH (flight:Process) WITH flight LIMIT {limit}  MATCH (origin:Airpor…
```

updates	executions	runtime	batches	failedBatches	batchErrors	failedCommits	commitErrors	wasTerminated
100000	200	5	201	0	{ }	0	{ }	false

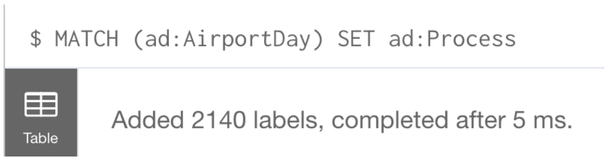
Started streaming 1 records after 5687 ms and completed after 5687 ms.

Exercise 8.5: Prepare the `AirportDay` nodes for batch processing. (Instructions)

Execute this Cypher code to add the `Process` label to all `AirportDay` nodes that will be processed for the batch refactoring:

```
MATCH (ad:AirportDay)
SET ad:Process
```

The result returned should be:


```
$ MATCH (ad:AirportDay) SET ad:Process
```

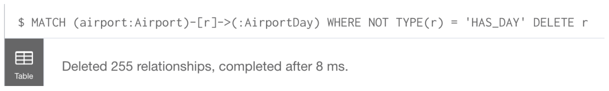
```
Added 2140 labels, completed after 5 ms.
```

Exercise 8.6: Refactor the graph to use specific by batching commits. (Instructions)

To make the refactoring simpler, first delete all existing specific relationships by executing this Cypher statement:

```
MATCH (airport:Airport)-[r]->(:AirportDay)
WHERE NOT TYPE(r) = 'HAS_DAY'
DELETE r
```

The result returned should be:


```
$ MATCH (airport:Airport)-[r]->(:AirportDay) WHERE NOT TYPE(r) = 'HAS_DAY' DELETE r
```

```
Deleted 255 relationships, completed after 8 ms.
```

Here is the code that you used previously to refactor the graph to use specific relationships:

```
MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
CALL apoc.create.relationship(startNode(hasDay),
                              'ON_' + ad.date,
                              {},
                              endNode(hasDay) ) YIELD rel
RETURN COUNT(*)
```

Modify this Cypher code to batch process the `AirportDay` nodes in batches of 500 and execute the code.

Hint: Use `apoc.periodic.commit()`.

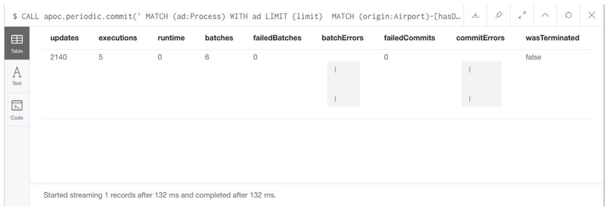
Exercise 8.6: Refactor the graph to use specific by batching commits. (Solution)

Modify this Cypher code to batch process the `AirportDay` nodes in batches of 500 and execute the code.

Hint: Use `apoc.periodic.commit()`.

```
CALL apoc.periodic.commit('
  MATCH (ad:Process)
  WITH ad LIMIT {limit}

  MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
  CALL apoc.create.relationship(startNode(hasDay), "ON_" + ad.date, {}, endNode(hasDay) ) YIELD
rel

  REMOVE ad:Process
  RETURN COUNT(*)
',{limit:500})
```

The result returned should be:



```
$ CALL apoc.periodic.commit(' MATCH (ad:Process) WITH ad LIMIT {limit}  MATCH (origin:Airport)-[hasD…
```

updates	executions	runtime	batches	failedBatches	batchErrors	failedCommits	commitErrors	wasTerminated
2140	5	0	6	0	{ }	0	{ }	false

Started streaming 1 records after 132 ms and completed after 132 ms.

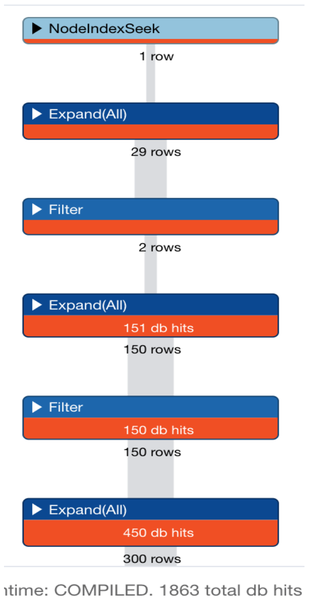
Exercise 8.7: Profile the query that finds flights on a specific day using the general relationship. (Instructions)

Previously, we used this query to look for all flights that go from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3).

Execute this Cypher query:

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(:AirportDay
      {date: '2019-1-3'})<-[:ORIGINATES_FROM]-(flight:Flight),
      (flight)-[:LANDS_IN]->(:AirportDay
      {date: '2019-1-3'})<-[:HAS_DAY]-(destination:Airport {code: 'MDW'})
RETURN origin, destination, flight
```

The result returned should be:



Previously on the graph with 10k flights, this query had 1813 db hits. After loading 100k Flights, we see 1863 db hits.

Exercise 8.8: Profile the query that finds flights on a specific day using the specific relationship. (Instructions)

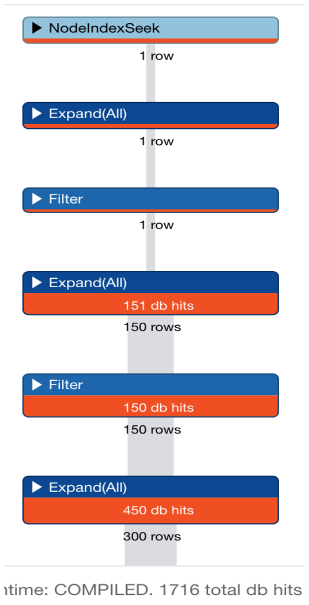
Previously, we used this query to look for all flights that go from Las Vegas (LAS) to Chicago (MDW) on January 3, 2019 (2019-1-3).

Execute this Cypher query:

```
PROFILE
MATCH (origin:Airport {code: 'LAS'})-
```

```
 [:`ON_2019-1-3`]->(originDay:AirportDay),
 (originDay)<-[:ORIGINATES_FROM]-(flight:Flight),
 (flight)-[:LANDS_IN]->(destinationDay),
 (destinationDay:AirportDay)<-[:`ON_2019-1-3`]-
 (destination:Airport {code: 'MDW'})
RETURN flight.date, flight.number, flight.airline,
flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

The result returned should be:



Previously on the graph with 10k flights, this query had 1716 db hits. After loading 100k Flights, we see 1716 db hits again, probably due to the fact that we did not add any more flights on this date.

Exercise 8: Refactoring large graphs (Summary)

In this exercise, you loaded the graph with 100k flights. Then you used `apoc.periodic.commit()` to refactor the graph in batches of 500 where you added the `AirportDay` nodes and you added specific relationships. You reran the query that looked for flights on a specific day and found that using the specific relationship was faster than using the general relationship.

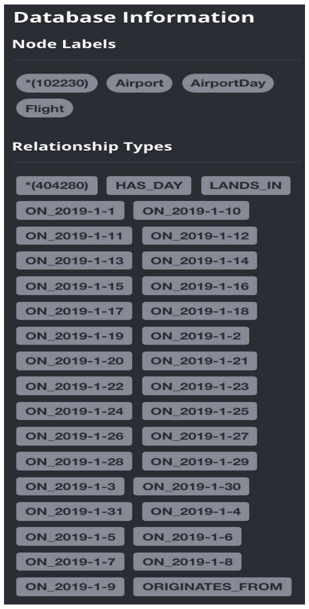
Whenever you change the model and refactor the graph, you must profile your queries to ensure that the changes you have made to the graph will be beneficial.

Continue to Exercise 9

Exercise 9: Maintaining models (Preparations)

Before you begin this exercise, make sure that you have performed the loading and refactoring from Exercise 8.

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



Database Information

Node Labels

*(102230) Airport AirportDay Flight

Relationship Types

*(404280) HAS_DAY LANDS_IN ON_2019-1-1 ON_2019-1-10 ON_2019-1-11 ON_2019-1-12 ON_2019-1-13 ON_2019-1-14 ON_2019-1-15 ON_2019-1-16 ON_2019-1-17 ON_2019-1-18 ON_2019-1-19 ON_2019-1-2 ON_2019-1-20 ON_2019-1-21 ON_2019-1-22 ON_2019-1-23 ON_2019-1-24 ON_2019-1-25 ON_2019-1-26 ON_2019-1-27 ON_2019-1-28 ON_2019-1-29 ON_2019-1-3 ON_2019-1-30 ON_2019-1-31 ON_2019-1-4 ON_2019-1-5 ON_2019-1-6 ON_2019-1-7 ON_2019-1-8 ON_2019-1-9 ORIGINATES_FROM

If your database does not have the same nodes and relationships, you can execute these Cypher statements to reset your graph to what it should be before you start this exercise.

```
CALL apoc.schema.assert({},{},true);
MATCH (:Flight)-[rel:ORIGINATES_FROM]-(:Airport)  DELETE rel;
MATCH (:Flight)-[rel:LANDS_IN]-(:Airport) DELETE rel;
MATCH (:Flight)-[rel:ORIGINATES_FROM]-(:AirportDay)  DELETE rel;
MATCH (:Flight)-[rel:LANDS_IN]-(:AirportDay) DELETE rel;
MATCH (n) DETACH DELETE n;
CREATE
  (`0` :Airport {code:'LAS'}) ,
  (`1` :Airport {code:'LAX'}) ,
  (`2` :Airport {code:'ABQ'}) ,
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'82',date:'2019-1-
3',departure:'1715',arrival:'1820'}]->(`1`),
  (`0`)-[:`CONNECTED_TO` {airline:'WN',flightNumber:'500',date:'2019-1-
3',departure:'1445',arrival:'1710'}]->(`2`);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_1k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (origin)-[connection:CONNECTED_TO {
  airline: row.UniqueCarrier,
  flightNumber: row.FlightNum,
  date: toInteger(row.Year) + '-' + toInteger(row.Month) + '-' + toInteger(row.DayofMonth)}]->
(destination)
ON CREATE SET connection.departure = toInteger(row.CRSDepTime), connection.arrival =
toInteger(row.CRSArrTime);
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE;
CREATE INDEX ON :Flight(number);
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight { flightId: connection.airline +  connection.flightNumber +
       '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination);
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection;
CREATE INDEX ON :Airport(code);
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_10k' AS row
MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (newFlight)-[:ORIGINATES_FROM]->(origin)
MERGE (newFlight)-[:LANDS_IN]->(destination);
CREATE CONSTRAINT ON (a:AirportDay)
ASSERT a.airportDayId IS UNIQUE;
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
      [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
     {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
      {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay);
MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
CALL apoc.create.relationship(startNode(hasDay),
                              'ON_' + ad.date,
                              {},
                              endNode(hasDay) ) YIELD rel
RETURN COUNT(*);

LOAD CSV WITH HEADERS FROM "https://r.neo4j.com/flights_2019_100k" AS row
UNWIND [row.Origin, row.Dest] AS airport
WITH DISTINCT airport
MERGE (:Airport {code: airport});

USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM "https://r.neo4j.com/flights_2019_100k" AS row
MATCH (origin:Airport {code: row.Origin})
MATCH (destination:Airport {code: row.Dest})
MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + "_" + row.Year + "-" +
row.Month + "-" + row.DayofMonth + "_" + row.Origin + "_" + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + "-" + toInteger(row.Month) + "-" +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination);

MATCH (f:Flight)
SET f:Process;

call apoc.periodic.commit('
MATCH (flight:Process)
WITH flight LIMIT {limit}

MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight)-[:LANDS_IN]->(destination:Airport)

MERGE (originAirportDay:AirportDay {airportDayId: origin.code + "_" + flight.date})
ON CREATE SET originAirportDay.date = flight.date

MERGE (destinationAirportDay:AirportDay {airportDayId: destination.code + "_" + flight.date})
ON CREATE SET destinationAirportDay.date = flight.date

MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (originAirportDay)<-[:ORIGINATES_FROM]-(flight)
MERGE (flight)-[:LANDS_IN]-(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)
REMOVE flight:Process
RETURN COUNT(*)

',{limit:500}
);

MATCH (ad:AirportDay)
SET ad:Process;

MATCH (airport:Airport)-[r]->(:AirportDay)
WHERE NOT TYPE(r) = "HAS_DAY"
DELETE r;

call apoc.periodic.commit('
  MATCH (ad:Process)
  WITH ad LIMIT {limit}

  MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
  CALL apoc.create.relationship(startNode(hasDay), "ON_" + ad.date, {}, endNode(hasDay) ) YIELD
rel

  REMOVE ad:Process
  RETURN COUNT(*)
',{limit:500})
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

Exercise 9: Maintaining models (Overview)

In a real application, whether in development or in production, it is likely that at some point in time, the requirements for the application change. As a developer, you must be prepared to refactor (possibly large) graphs and perhaps archive data in the graph that is no longer needed. In this exercise, you will perform a few tasks related to maintaining a graph.

Here are the tasks you will perform:

- **Exercise 9.1**: Profile a query to find flights in January, 2019 using general relationships.
- **Exercise 9.2**: Profile a query to find flights in January, 2019 using specific relationships.
- **Exercise 9.3**: Add flight data.
- **Exercise 9.4**: Query new flight data.
- **Exercise 9.5**: Archive flight data.

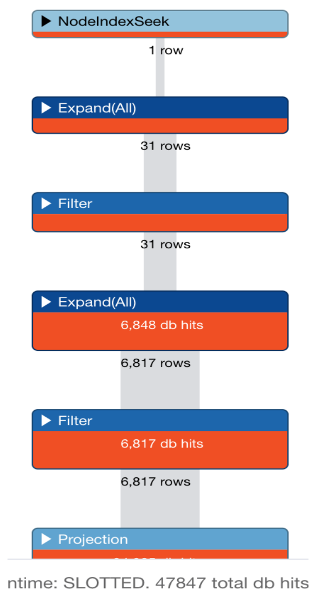
Go to the next page to start this exercise.

Exercise 9.1: Profile a query to find flights in January, 2019 using general relationships. (Instructions)

Execute this Cypher code to find flights in January, 2019 using general relationships:

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight)
WHERE originDay.date STARTS WITH '2019-1'
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

The result returned should be:



ntime: SLOTTED. 47847 total db hits

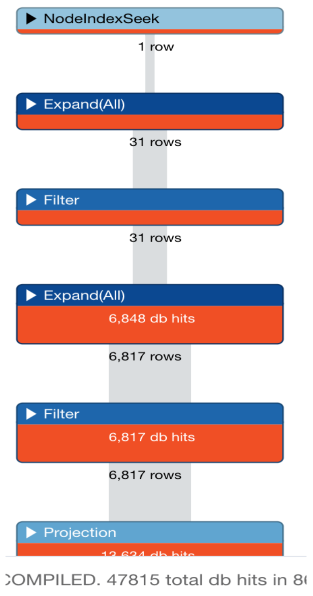
The query produces 47847 db hits.

Exercise 9.2: Profile a query to find flights in January, 2019 using specific relationships. (Instructions)

Execute this Cypher code to find flights in January, 2019 using specific relationships:

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:`ON_2019-1-1`|
                :`ON_2019-1-2`|
                :`ON_2019-1-3`|
                :`ON_2019-1-4`|
                :`ON_2019-1-5`|
                :`ON_2019-1-6`|
                :`ON_2019-1-7`|
                :`ON_2019-1-8`|
                :`ON_2019-1-9`|
                :`ON_2019-1-10`|
                :`ON_2019-1-11`|
                :`ON_2019-1-12`|
                :`ON_2019-1-13`|
                :`ON_2019-1-14`|
                :`ON_2019-1-15`|
                :`ON_2019-1-16`|
                :`ON_2019-1-17`|
                :`ON_2019-1-18`|
                :`ON_2019-1-19`|
                :`ON_2019-1-20`|
                :`ON_2019-1-21`|
                :`ON_2019-1-22`|
                :`ON_2019-1-23`|
                :`ON_2019-1-24`|
                :`ON_2019-1-25`|
                :`ON_2019-1-26`|
                :`ON_2019-1-27`|
                :`ON_2019-1-28`|
                :`ON_2019-1-29`|
                :`ON_2019-1-30`|
                :`ON_2019-1-31`
                ]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight)
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

The result returned should be:



The query produces 47815 db hits. A small savings. In this case what Cypher code would you prefer to write? The general relationship is much easier to query so in this case, it pays to have both types of relationships in the model.

Exercise 9.3: Add flight data. (Instructions)

You want to add February, 2019 flight data to the graph.

Using the code you have previously seen and executed, write and execute the Cypher code to load the February, 2019 flight data making sure that the AirportDay and specific and general relationships are added.

The file containing the data is **'https://r.neo4j.com/flights_2019_february'**, or copied to your import folder.

Exercise 9.3: Add flight data. (Solution)

You want to add February, 2019 flight data to the graph.

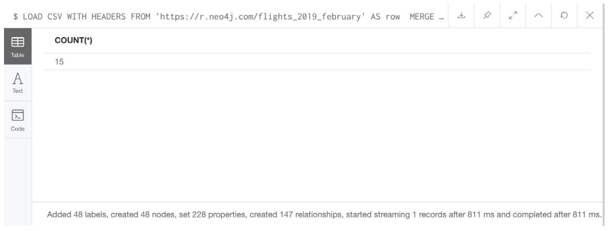
Using the code you have previously seen and executed, write and execute the Cypher code to load the February, 2019 flight data making sure that the AirportDay and specific and general relationships are added.

The file containing the data is **'https://r.neo4j.com/flights_2019_february'**, or copied to your import folder.

```
LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_february' AS row

MERGE (origin:Airport {code: row.Origin})
MERGE (destination:Airport {code: row.Dest})

MERGE (newFlight:Flight { flightId: row.UniqueCarrier + row.FlightNum + '_' + row.Year + '-' +
row.Month + '-' + row.DayofMonth + '_' + row.Origin + '_' + row.Dest }   )
ON CREATE SET newFlight.date = toInteger(row.Year) + '-' + toInteger(row.Month) + '-' +
toInteger(row.DayofMonth),
              newFlight.airline = row.UniqueCarrier,
              newFlight.number = row.FlightNum,
              newFlight.departure = toInteger(row.CRSDepTime),
              newFlight.arrival = toInteger(row.CRSArrTime)
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination)
MERGE (originAirportDay:AirportDay {airportDayId: origin.code + '_' + newFlight.date})
ON CREATE SET originAirportDay.date = newFlight.date

MERGE (destinationAirportDay:AirportDay {airportDayId: destination.code + '_' + newFlight.date})
ON CREATE SET destinationAirportDay.date = newFlight.date

MERGE (originAirportDay)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]-(destinationAirportDay)

MERGE (origin)-[hasDayOrigin:HAS_DAY]->(originAirportDay)
MERGE (destination)-[hasDayDestination:HAS_DAY]->(destinationAirportDay)

WITH *
UNWIND [[hasDayOrigin, originAirportDay], [hasDayDestination, destinationAirportDay]] AS pair

WITH DISTINCT pair
WITH apoc.convert.toRelationship(pair[0]) AS hasDay, pair[1] AS day

CALL apoc.create.relationship(startNode(hasDay), "ON_" + day.date, {}, endNode(hasDay) )
YIELD rel
RETURN COUNT(*)
```

Note: You must use the "file:///" syntax for loading the data from the import folder.

The result returned should be:



```
$ LOAD CSV WITH HEADERS FROM 'https://r.neo4j.com/flights_2019_february' AS row  MERGE …
```

COUNT(*)
15

Added 48 labels, created 48 nodes, set 228 properties, created 147 relationships, started streaming 1 records after 811 ms and completed after 811 ms.

Exercise 9.4: Query new flight data. (Instructions)

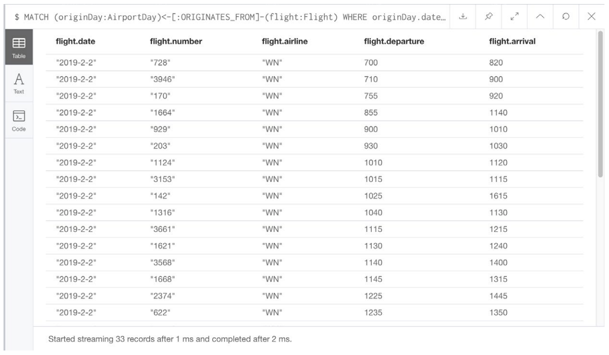
Write and execute a query to find all flights in February, 2019

Exercise 9.4: Query new flight data. (Solution)

Write and execute a query to find all flights in February, 2019

```
MATCH (originDay:AirportDay)<-[:ORIGINATES_FROM]-(flight:Flight)
WHERE originDay.date STARTS WITH '2019-2'
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

The result returned should be:



```
$ MATCH (originDay:AirportDay)<-[:ORIGINATES_FROM]-(flight:Flight) WHERE originDay.date…
```

flight.date	flight.number	flight.airline	flight.departure	flight.arrival
"2019-2-2"	"728"	"WN"	700	820
"2019-2-2"	"3946"	"WN"	710	900
"2019-2-2"	"170"	"WN"	755	920
"2019-2-2"	"1664"	"WN"	855	1140
"2019-2-2"	"929"	"WN"	900	1010
"2019-2-2"	"203"	"WN"	930	1030
"2019-2-2"	"1124"	"WN"	1010	1120
"2019-2-2"	"3153"	"WN"	1015	1115
"2019-2-2"	"142"	"WN"	1025	1615
"2019-2-2"	"1316"	"WN"	1040	1130
"2019-2-2"	"3661"	"WN"	1115	1215
"2019-2-2"	"1621"	"WN"	1130	1240
"2019-2-2"	"3568"	"WN"	1140	1400
"2019-2-2"	"1668"	"WN"	1145	1315
"2019-2-2"	"2374"	"WN"	1225	1445
"2019-2-2"	"622"	"WN"	1235	1350

Started streaming 33 records after 1 ms and completed after 2 ms.

Exercise 9.5: Archive flight data. (Instructions)

At some point during your application lifecycle, you will want to archive data, especially if the graph is growing large and older data is not required by users.

Note: Before you export to a file using APOC, you must add this setting to your **neo4j.conf** file for the database and restart the database: `apoc.export.file.enabled=true`

Write and execute Cypher code to:

1. Export flights with the date of '2019-1-1'. 2. Delete these flights from the graph.

Note: If you are using a Neo4j Sandbox, you will not be able to write the data to a file in the Sandbox so you can skip this part of the exercise.

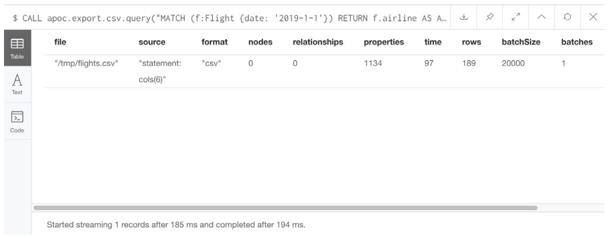
Exercise 9.5: Archive flight data. (Solution)

Write and execute Cypher code to:

1. Export flights with the date of '2019-1-1'.

```
CALL apoc.export.csv.query("MATCH (f:Flight {date: '2019-1-1'})
RETURN  f.airline AS Airline, f.arrival AS Arrival, f.date AS Date, f.departure AS Departure,
f.flightId AS `Flight ID`, f.number AS FlightNumber",'/tmp/flights.csv', {})
```

The result returned should be:

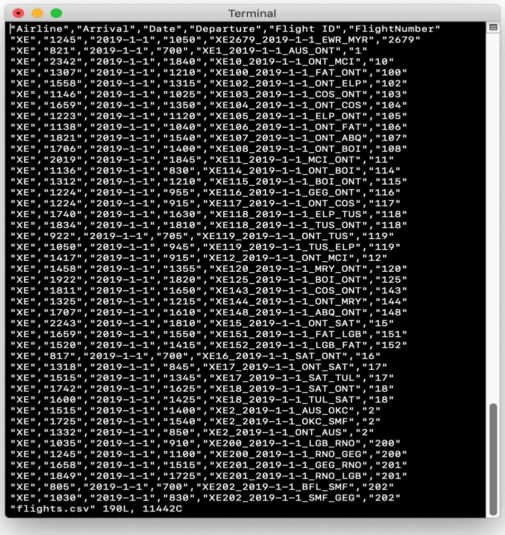


```
$ CALL apoc.export.csv.query("MATCH (f:Flight {date: '2019-1-1'}) RETURN f.airline AS A…
```

file	source	format	nodes	relationships	properties	time	rows	batchSize	batches
"/tmp/flights.csv"	"statement: cols(6)"	"csv"	0	0	1134	97	189	20000	1

Started streaming 1 records after 185 ms and completed after 194 ms.

And here is what the flights.csv file looks like:

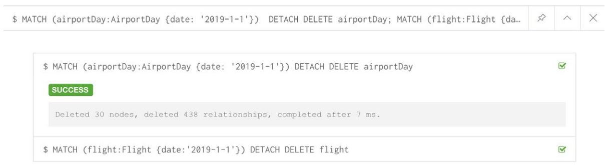


```
"Airline","Arrival","Date","Departure","Flight ID","FlightNumber"
"XE","1245","2019-1-1","1050","XE2679_2019-1-1_EWR_MYR","2679"
"XE","821","2019-1-1","700","XE1_2019-1-1_AUS_ONT","1"
"XE","2342","2019-1-1","1840","XE10_2019-1-1_ONT_MCI","10"
"XE","1307","2019-1-1","1210","XE100_2019-1-1_FAT_ONT","100"
"XE","1558","2019-1-1","1315","XE102_2019-1-1_ONT_ELP","102"
"XE","1146","2019-1-1","1025","XE103_2019-1-1_COS_ONT","103"
"XE","1659","2019-1-1","1350","XE104_2019-1-1_ONT_COS","104"
"XE","1223","2019-1-1","1120","XE105_2019-1-1_ELP_ONT","105"
"XE","1138","2019-1-1","1040","XE106_2019-1-1_ONT_FAT","106"
"XE","1821","2019-1-1","1540","XE107_2019-1-1_ONT_ABQ","107"
"XE","1706","2019-1-1","1400","XE108_2019-1-1_ONT_BOI","108"
"XE","2019","2019-1-1","1845","XE11_2019-1-1_MCI_ONT","11"
"XE","1136","2019-1-1","830","XE114_2019-1-1_ONT_BOI","114"
"XE","1312","2019-1-1","1210","XE115_2019-1-1_BOI_ONT","115"
"XE","1224","2019-1-1","955","XE116_2019-1-1_GEG_ONT","116"
"XE","1224","2019-1-1","915","XE117_2019-1-1_ONT_COS","117"
"XE","1740","2019-1-1","1630","XE118_2019-1-1_ELP_TUS","118"
"XE","1834","2019-1-1","1810","XE118_2019-1-1_TUS_ONT","118"
"XE","922","2019-1-1","705","XE119_2019-1-1_ONT_TUS","119"
"XE","1050","2019-1-1","945","XE119_2019-1-1_TUS_ELP","119"
"XE","1417","2019-1-1","915","XE12_2019-1-1_ONT_MCI","12"
"XE","1458","2019-1-1","1355","XE120_2019-1-1_MRY_ONT","120"
"XE","1922","2019-1-1","1820","XE125_2019-1-1_BOI_ONT","125"
"XE","1811","2019-1-1","1650","XE143_2019-1-1_COS_ONT","143"
"XE","1325","2019-1-1","1215","XE144_2019-1-1_ONT_MRY","144"
"XE","1707","2019-1-1","1610","XE148_2019-1-1_ABQ_ONT","148"
"XE","2243","2019-1-1","1810","XE15_2019-1-1_ONT_SAT","15"
"XE","1659","2019-1-1","1550","XE151_2019-1-1_FAT_LGB","151"
"XE","1520","2019-1-1","1415","XE152_2019-1-1_LGB_FAT","152"
"XE","817","2019-1-1","700","XE16_2019-1-1_SAT_ONT","16"
"XE","1318","2019-1-1","845","XE17_2019-1-1_ONT_SAT","17"
"XE","1515","2019-1-1","1345","XE17_2019-1-1_SAT_TUL","17"
"XE","1742","2019-1-1","1625","XE18_2019-1-1_SAT_ONT","18"
"XE","1600","2019-1-1","1425","XE18_2019-1-1_TUL_SAT","18"
"XE","1515","2019-1-1","1400","XE2_2019-1-1_AUS_OKC","2"
"XE","1725","2019-1-1","1540","XE2_2019-1-1_OKC_SMF","2"
"XE","1332","2019-1-1","850","XE2_2019-1-1_ONT_AUS","2"
"XE","1035","2019-1-1","910","XE200_2019-1-1_LGB_RNO","200"
"XE","1245","2019-1-1","1100","XE200_2019-1-1_RNO_GEG","200"
"XE","1658","2019-1-1","1515","XE201_2019-1-1_GEG_RNO","201"
"XE","1849","2019-1-1","1725","XE201_2019-1-1_RNO_LGB","201"
"XE","805","2019-1-1","700","XE202_2019-1-1_BFL_SMF","202"
"XE","1030","2019-1-1","830","XE202_2019-1-1_SMF_GEG","202"
"flights.csv" 190L, 11442C
```

2. Delete these flights from the graph

```
MATCH (airportDay:AirportDay {date: '2019-1-1'})
DETACH DELETE airportDay;
MATCH (flight:Flight {date:'2019-1-1'})
DETACH DELETE flight
```

The result returned should be:



```
$ MATCH (airportDay:AirportDay {date: '2019-1-1'})  DETACH DELETE airportDay; MATCH (flight:Flight {da…

$ MATCH (airportDay:AirportDay {date: '2019-1-1'}) DETACH DELETE airportDay
SUCCESS
Deleted 30 nodes, deleted 438 relationships, completed after 7 ms.

$ MATCH (flight:Flight {date:'2019-1-1'}) DETACH DELETE flight
```

Exercise 9: Maintaining models (Taking it further)

If you have time, you can evolve the model further.

Here are a couple of files that contain additional data (you may have also copied them to the import folder):

https://r.neo4j.com/Modeling_Airports

https://r.neo4j.com/Modeling_Aircraft

You can:

- Add additional airport data from **Modeling_Airports** and integrate it into the model.
- Add additional properties to the graph such as flight delays and cancellations from the existing CSV files.
- Add **Modeling_Aircraft** data to the graph.

Whatever data you choose to work with, try and think of some questions that you want to answer before you import the data and refactor the model.

Exercise 9: Maintaining models (Summary)

In this exercise, you performed more queries that contrast the use of general and specific relationships. You also added flight data to the graph for a new day as well as archived and deleted flight data from the graph.

Congratulations! You have completed the exercises for this course.