Intro to graph modeling

What are we going to do?

- Intro to the property graph model
- The airport dataset
- The modeling workflow
- Load CSV data

The labelled property graph

- Nodes
- Relationships
- Properties
- Labels

Property Graph Model Components

Nodes

- Represent the objects in the graph
- Can be *labeled*

Property Graph Model Components

Nodes

- Represent the objects in the graph
- Can be *labeled*

Relationships

- Relate nodes by *type* and *direction*

Property Graph Model Components

Nodes

- Represent the objects in the graph
- Can be *labeled*

Relationships

- Relate nodes by *type* and *direction*

Properties

- Name-value pairs that can go on nodes and relationships.

The Modeling Workflow

Models

The modeling workflow

The flights dataset

United States Department of Transportation

About DOT | Briefing Room | Our Activities

Office of the Assistant Secretary for Research and Technology

Bureau of Transportation Statistics

About OST-R | Press Room | Programs | OST-R Publications | Library | Contact Us

Search

About BTS | BTS Press Room | Data and Statistics | Publications | Subject Areas | External Links

OST-R > BTS

TranStats

Search this site:

Go

Advanced Search

Resources

- Database Directory
- Glossary
- Upcoming Releases
- Data Release History

Data Tools

- Table Profile
- Download

: On-Time Performance

Database Profile | Data Tables | Table Profile

Latest Available Data: July 2016

<<Prev Rows : 1 - 100 of 111 Next>>

Field Name	Description	
Summaries		
*OntimeArrivalPct	Percent of flights that arrive on time. For percent of on time arrivals at specific airports, click Analysis. **Note**: If you select Origin as a category, you get percent of flights that depart from those airports and arrive on time.	Analysis
*OntimeDeparturePct	Percent of flights that depart on time. For percent of on time departures at specific airports, click Analysis. **Note**: If you select Dest as a category, you get percent of flights that depart on time and arrive at those airports.	Analysis
Time Period		
Year	Year	
Quarter	Quarter (1-4)	Analysis
Month	Month	Analysis
DayofMonth	Day of Month	
DayOfWeek	Day of Week	Analysis
FlightDate	Flight Date (yyyymmdd)	
Airline		
UniqueCarrier	Unique Carrier Code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.	Analysis
AirlineID	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation.	Analysis
Carrier	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique. For analysis, use the Unique Carrier Code.	

What data do we have?

Origin		
OriginAirportID	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.	Analysis
OriginAirportSeqID	Origin Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.	
OriginCityMarketID	Origin Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.	Analysis
Origin	Origin Airport	Analysis
OriginCityName	Origin Airport, City Name	
OriginState	Origin Airport, State Code	Analysis
OriginStateFips	Origin Airport, State Fips	Analysis
OriginStateName	Origin Airport, State Name	
OriginWac	Origin Airport, World Area Code	Analysis

Destination		
DestAirportID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.	Analysis
DestAirportSeqID	Destination Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.	
DestCityMarketID	Destination Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.	Analysis
Dest	Destination Airport	Analysis
DestCityName	Destination Airport, City Name	
DestState	Destination Airport, State Code	Analysis
DestStateFips	Destination Airport, State Fips	Analysis
DestStateName	Destination Airport, State Name	
DestWac	Destination Airport, World Area Code	Analysis

Time Period		
Year	Year	
Quarter	Quarter (1-4)	Analysis
Month	Month	Analysis
DayofMonth	Day of Month	
DayOfWeek	Day of Week	Analysis
FlightDate	Flight Date (yyyymmdd)	

Find the most popular airports

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Derive questions

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Is Airport A connected to Airport B?

Identity entities

Is **Airport** A connected to **Airport** B?

```
airport
```

Identify relationships between entities

Is Airport A **connected to** Airport B?

```
airport CONNECTED_TO airport
```

It’s a graph!

Is Airport A **connected to** Airport B?

```
(airport)-[:CONNECTED_TO]->(airport)
```

It’s a graph!

neo4j

Is Airport A **connected to** Airport B?

```
(airport)-[:CONNECTED_TO]->(airport)
```

Labels

Labels are for categorizing nodes.

Properties

Properties are for defining attributes of nodes or relationships.

We need an airport code to uniquely identify each airport.

Modeling different connections

Busy airports will have multiple connections between them. We can model this in different ways:

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Sample data

code: LAS
Airport
CONNECTED_TO
airline: WN
flightNumber: 500
departure: 1445
arrival: 1710
date: 2008-1-3
Airport
code: LAX

Prepare your Neo4j graph.db directory

Copy folders

- `import`
- `plugins`

from **USB Stick**

to the **`default.graphdb`** folder

(or **`$NEO4J_HOME`**)

Make sure you've got Neo4j running

1. Start the server.
2. It should be running on: http://localhost:7474
3. Log-in with default credentials
   - user: **neo4j** password: **neo4j**
4. Choose a **new** password

We're good to go!

Find the most popular airports

Open your browser to http://localhost:7474 and execute the following command:

```
:play http://guides.neo4j.com/modeling_airports
```

Play the guides in your browser until you see...

The MERGE command

MERGE

The MERGE command is a combination of `MATCH` and `CREATE`. If the pattern already exists then it will return it; if not it will create it.

We can MERGE nodes...

```
MERGE (las:Airport {code: "LAS"})
RETURN las
```

...or relationships...

```
MATCH (las:Airport {code: "LAS"})
MATCH (lax:Airport {code: "LAX"})
MERGE (las)-[:CONNECTED_TO]->(lax)
```

...or both

```
MERGE (las:Airport {code: "LAS"})
MERGE (lax:Airport {code: "LAX"})
MERGE (las)-[:CONNECTED_TO]->(lax)
```

```
MERGE (las:Airport {code: "LAS"})-[:CONNECTED_TO]->(lax:Airport
{code: "LAX"})
```

Continue playing the guide

Continue the guide in your browser

The modeling workflow

The modeling workflow

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Is **Airport** A connected to **Airport** B?

```
|---------+------+------------|
|  Origin | Dest | FlightNum  |
|---------+------+------------|
|  IAD    | TPA  | 335        |
|  IAD    | TPA  | 3231       |
|  IND    | BWI  | 448        |
|  IND    | BWI  | 3920       |
|---------+------+------------|
```

```
(airport)-[:CONNECTED_TO]->(airport)
```

```
MATCH (origin:Airport {code: "LAX"})
      -[flight:CONNECTED_TO]->
      (destination:Airport {code: "LAS"})
RETURN origin, destination, flight
LIMIT 10
```

```
CREATE (lax:Airport {code: "LAX"})
CREATE (las:Airport {code: "LAS"})
CREATE (las)-[connection:CONNECTED_TO {
    airline: "WN",
    flightNumber: "82",
    date: "2008-1-3",
    departure: 1715,
    arrival: 1820}]->(lax)
```

Modeling airports

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Nodes

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Labels

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Relationships

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Properties

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones

Loading CSV data

Load CSV

A clause in the Cypher query language that lets us iterate over CSV files and create graph structures based on the data contained in each row.

Load CSV

```
LOAD CSV      // load csv data
WITH HEADERS // optionally use first header row as keys in "row" map
FROM "url"    // file:// URL relative to $NEO4J_HOME/import or http://
AS row        // return each row of the CSV as list of strings or map
// ... rest of the Cypher statement ...
```

Load CSV

```
[USING PERIODIC COMMIT] // optionally batch transactions
LOAD CSV      // load csv data
WITH HEADERS // optionally use first header row as keys in "row" map
FROM "url"    // file:// URL relative to $NEO4J_HOME/import or http://
AS row        // return each row of the CSV as list of strings or map
[FIELDTERMINATOR ";"] // optionally alt. delimiter
// ... rest of the Cypher statement ...
```

Consider this CSV file

Origin	Dest	FlightNum
IAD	TPA	335
IAD	TPA	3231
IND	BWI	448
IND	BWI	3920

Create nodes

```
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row

CREATE (:Airport {code: row.Origin})

CREATE (:Airport {code: row.Dest})
```

Origin	Dest	FlightNum
IAD	TPA	335
IAD	TPA	3231
IND	BWI	448
IND	BWI	3920

Create relationships

```
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row

CREATE (origin:Airport {code: row.Origin})

CREATE (destination:Airport {code: row.Dest})

CREATE (origin)-[:CONNECTED_TO {flightNumber: row.flightNum}]->(destination)
```

Origin	Dest	FlightNum
IAD	TPA	335
IAD	TPA	3231
IND	BWI	448
IND	BWI	3920

Find existing nodes and relationships

```
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row

MATCH (origin:Airport {code: row.Origin})

MATCH (destination:Airport {code: row.Dest})

MATCH (origin)-[:CONNECTED_TO {flightNumber: row.flightNum}]->(destination)

...
```

Origin	Dest	FlightNum
IAD	TPA	335
IAD	TPA	3231
IND	BWI	448
IND	BWI	3920

Update existing nodes and relationships

```
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row

MATCH (origin:Airport {code: row.Origin})

MATCH (destination:Airport {code: row.Dest})

MATCH (origin)-[c:CONNECTED_TO {flightNumber:row.flightNum}]->(destination)

SET c.airline = row.UniqueCarrier
```

Origin	Dest	FlightNum	UniqueCarrier
IAD	TPA	335	WN
IAD	TPA	3231	WN
IND	BWI	448	WN
IND	BWI	3920	WN

Idempotently create nodes and relationships

```
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row

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

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

MERGE (origin)-[:CONNECTED_TO {flightNumber: row.flightNum]->(destination)
```

Origin	Dest	FlightNum
IAD	TPA	335
IAD	TPA	3231
IND	BWI	448
IND	BWI	3920

Let’s give it a try

Continue the guide in your browser

End of Module
Intro to Graph Modeling

Questions ?

Missing domain concept

What are we going to do?

- Profile queries
- Introduce a missing concept in the domain
- Constraints and indexes
- Write our first refactoring query

Start playing the next guide….

...if you aren't playing it already

Flight as a first class citizen

:play http://guides.neo4j.com/modeling_airports/02_flight.html

Profiling queries

How do I profile a query?

By prefixing the query with:

`EXPLAIN`

shows the execution plan without actually executing it or returning any results.

How do I profile a query?

By prefixing the query with:

`EXPLAIN`

shows the execution plan without actually executing it or returning any results.

`PROFILE`

executes the statement and returns the results along with profiling information.

How do I profile a query?

```
EXPLAIN
MATCH (origin:Airport)-[c:CONNECTED_TO]
     ->(destination:Airport)
WHERE c.code = "LAS"
RETURN origin, destination, c
LIMIT 10
```

How do I profile a query?

```
PROFILE
MATCH (origin:Airport)-[c:CONNECTED_TO]
     ->(destination:Airport)
WHERE destination.code = "LAS"
RETURN origin, destination, c
LIMIT 10
```

Anatomy of an execution plan

```
PROFILE
MATCH (origin:Airport)-[c:CONNECTED_TO]
      ->(destination:Airport)
WHERE destination.code = "LAS"
RETURN origin, destination, c
LIMIT 10
```

Operations

Rows come in

↓

Do some work

↓

Rows go out

What’s our goal?

neo4j

At a high level, the goal is simple: **get the number of db hits down.**

What is a database hit?

neo4j

“an abstract unit of storage engine work.”

Let’s profile some queries

Continue playing the guide in your browser

Introducing a missing domain concept

Introducing a missing domain concept

Sometimes we design a model and after working with it for a bit we realise that we've missed an important domain concept.

Introducing a missing domain concept

Sometimes we design a model and after working with it for a bit we realise that we've missed an important domain concept.

In our case we're missing nodes to represent **Flights**

Introducing a missing domain concept

Flight as a first class citizen

Flight as a first class citizen

```
MATCH  (o:Airport)-[:CONNECTED_TO]->(d:Airport)
MERGE...
```

Refactoring: Derive node from relationship

This is the first of the refactoring patterns that we'll encounter today.

We have this pattern:

```
(origin)-[:CONNECTED_TO]->(destination)
```

And we'll create a new node using the properties of the `CONNECTED_TO` relationship:

```
(origin)<-[:ORIGIN]-(flight)-[:DESTINATION]->(destination)
```

Constraints and indexes

Unique Constraints

We create unique constraints to:

- ensure uniqueness
- allow fast lookup of nodes which match label-property pairs.

Unique Constraints

We create unique constraints to:

- ensure uniqueness
- allow fast lookup of nodes which match label-property pairs.

```
CREATE CONSTRAINT ON (label:Label)
ASSERT label.property IS UNIQUE
```

Indexes

We create indexes to:

- allow fast lookup of nodes which match label-property pairs.

Indexes

We create indexes to:

- allow fast lookup of nodes which match label-property pairs.

```
CREATE INDEX ON :Label(property)
```

What gets indexed?

The following are index backed:

- Equality
- `STARTS WITH`
- `CONTAINS`
- `ENDS WITH`
- Range searches
- (Non-) existence checks

How are indexes used in neo4j?

Indexes are **only** used to find the starting point for queries.

Relational

Use index scans to look up rows in tables and join them with rows from other tables

Graph

Use indexes to find the starting points for a query.

Let’s get on with the refactoring

Continue playing the guide in your browser

End of Module
Missing Concept

Questions?

Relationship Bundling

Relationship Bundling

So far we've been writing queries from the perspective of an air travel enthusiast.

In this section we're going to evolve the model based on the needs of a frequent traveller who wants to find flights on a specific date.

What are we going to do?

- Write a query to book a flight on a specific day
- Profile our flight booking query
- Optimise our model to deal with this query

Find a flight to book

As a frequent traveller

I want to find flights from <origin> to <destination> on <date>

So that I can book my business flight

Start playing the next guide….

…if you aren’t playing it already

Flight booking

:play http://guides.neo4j.com/modeling_airports/03_flight_booking.html

Bundling flight relationships

Bundling flight relationships by airport day

Searching for flights by day is a bit problematic. We have to scan every relationship between origin and destination airports and analyse properties on each flight to work out which ones we're interested in.

Bundling flight relationships by airport day

Searching for flights by day is a bit problematic. We have to scan every relationship between origin and destination airports and analyse properties on each flight to work out which ones we're interested in.

We can reduce this problem by bundling flights together by **AirportDay**

Our current model

Introducing AirportDay

Refactoring to Airport Day

Bundling flight relationships by airport day

Refactoring: Derive node from property

This refactoring creates a new node using the property of an existing node.

We currently have this pattern:

```
(origin)<-[:ORIGIN]-(flight)-[:DESTINATION]->(destination)
```

And we'll create a new node using the `date` property of the `flight` node:

```
(origin)-[:HAS_DAY]->(originAirportDay)<-[:ORIGIN]-(flight),
(destination)-[:HAS_DAY]->(destAirportDay)<-[:DESTINATION]-(flight)
```

Let’s get on with the refactoring

Over to you!

End of Module
Bundling Relationships

Questions?

Modeling Guidelines

What are we going to do?

- Recap of the Property Graph Model
- Modeling choices
- Refactorings

Nodes, relationships, labels, properties

Our model

date: string
Airport Day
date: string
Airport Day
HAS_DAY
ORIGIN
DESTINATION
HAS_DAY
Airport
Flight
Airport
code: string
airline: string
number: string
departure: long
arrival: long
date: string
code: string

Nodes are for things

Labels for grouping

Relationships for structure

Properties for attributes

Modeling choices

Properties vs. Relationships

Properties vs Relationships

Properties vs Relationships

We only need to pull out a node if we're going to query through it, otherwise a property will suffice.

But if we pull out every single property then we end up with an RDF model and lose the benefit of the property graph

Relationship Granularity

Relationship Granularity

Symmetric Relationships

Symmetric relationships

Bidirectional Relationships

Bidirectional relationships

No need to have the relationship in both directions

Use single relationship and ignore direction in queries

```
MATCH (:Person {name:'Eric'})-[:MARRIED_TO]-(p2)
RETURN p2
```

General vs. Specific Relationships

General Relationships

General: Find flights on a specific date

```
MATCH (origin:Airport {code: "LAS"})-[:HAS_DAY]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight),
      (flight)-[:DESTINATION]->(destinationDay),
      (destinationDay:AirportDay)<-[:HAS_DAY]-(destination:Airport {code: "MDW"})

WHERE originDay.date = "2008-1-3" AND destinationDay.date = "2008-1-3"

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure
```

Specific Relationships

Specific: Find flights on a specific date

```
MATCH (origin:Airport {code: "LAS"})-[:`2008-1-3`]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight),
      (flight)-[:DESTINATION]->(destinationDay),
      (destinationDay:AirportDay)<-[:`2008-1-3`]-(destination:Airport {code: "MDW"})

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure;
```

General: Find flights by year and month

```
MATCH (origin:Airport {code: "LAS"})-[:HAS_DAY]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight)

WHERE originDay.date STARTS WITH "2008-1"

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure
```

Specific: Find flights by year and month

```
MATCH (origin:Airport {code: "LAS"})
       -[:`2008-1-3`|:`2008-1-4`|:`2008-1-5`|:`2008-1-6`]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight)

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure
```

Best of both worlds?

Refactorings

Derive node from relationship

```
(origin)-[:CONNECTED_TO]->(destination)
```

```
(origin)<-[:ORIGIN]-(flight)-[:DESTINATION]->(destination)
```

Derive node from property

```
(origin)<-[:ORIGIN]-(flight)-[:DESTINATION]->(destination)
```

```
(origin)-[:HAS_DAY]->(originAirportDay)<-[:ORIGIN]-(flight),
(destination)-[:HAS_DAY]->(destAirportDay)<-[:DESTINATION]-(flight)
```

Derive relationship from property

```
(airport)-[:HAS_DAY]->(airportDay {date: "2008-1-3"})
```

```
(airport)-[:`2008-1-3`]->(airportDay {date: "2008-1-3"})
```

Evolving the Model

Revealing nodes

Revealing nodes?

End of Module
Modeling Guidelines

Questions ?

Specific Relationship Types

What are we going to do?

- Granularity of relationship types
- Refactor our model to use more specific relationship types

Our current model

So what’s the problem?

Neo4j is optimized for searching by relationship types. As we add more data, the number of HAS_DAY relationships that we have to traverse increases.

So what’s the problem?

Neo4j is optimized for searching by relationship types. As we add more data, the number of HAS_DAY relationships that we have to traverse increases.

If we have 10 years worth of data we have to traverse 3,650 relationships from the `Airport` to find the `AirportDay` that we’re interested in.

Refactoring: Derive relationship from property

Refactoring: Derive relationship from property

We have this pattern:

```
(airport)-[:HAS_DAY]->(airportDay {date: "2008-1-3"})
```

And we'll create a new relationship using the `date` property on `airportDay`:

```
(airport)-[:`2008-1-3`]->(airportDay {date: "2008-1-3"})
```

Start playing the next guide….

neo4j

...if you aren’t playing it already

Specific Relationship Types

:play http://guides.neo4j.com/modeling_airports/04_specific_relationship_types.html

End of Module
Specific Relationship Types

Questions ?

Refactoring Large Graphs

What are we going to do?

- Why do we need to batch?
- The batch refactoring workflow
- Automate batch refactorings with the apoc library

Why do we need to batch?

Transaction State

Cypher keeps **all transaction state in memory** while running a query, which is fine most of the time.

When refactoring the graph, however, this state can get **very large** and may result in an **OutOfMemory** exception.

The batch refactoring workflow

The batch refactoring workflow

- tag all the nodes we need to process with a temporary label e.g. `Process`
- iterate over a subset of nodes flagged with that label (using `LIMIT`) and execute the refactoring
- remove the tag from the node
- return a count of how many rows were processed
- once the count reaches 0 then we've finished.

The batch refactoring workflow

```
MATCH (itemToProcess:Process)
WITH itemToProcess
LIMIT 1000
REMOVE itemToProcess:Process
WITH itemToProcess
// do the refactoring
RETURN count(*)
```

Start playing the next guide….

neo4j

…if you aren’t playing it already

Refactoring large graphs

:play http://guides.neo4j.com/modeling_airports/05_refactoring_large_graphs.html

End of Module
Refactoring Large Graphs

Questions ?

Multiple Models

What are we going to do?

- General vs specific relationships
- The read/write tradeoff

So what's the problem?

Different models perform better for different queries but worse for others

Optimising for reads may mean we pay a write and maintenance penalty

General Relationships

Specific Relationships

Find flights on a specific date: Specific wins

```
MATCH (origin:Airport {code: "LAS"})-[:`2008-1-3`]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight),
      (flight)-[:DESTINATION]->(destinationDay),
      (destinationDay:AirportDay)<-[:`2008-1-3`]-(destination:Airport {code: "MDW"})

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure;
```

Find flights by year and month: General wins

```
MATCH (origin:Airport {code: "LAS"})-[:HAS_DAY]->(originDay:AirportDay),
      (originDay)<-[:ORIGIN]-(flight:Flight)

WHERE originDay.date STARTS WITH "2008-1"

RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival

ORDER BY flight.date, flight.departure
```

Best of both worlds?

Read/Write trade off

Read/Write trade off

By keeping both models we optimise for both the read use cases but pay a write penalty.

Adding a new day

If we want to add a new day we have to add two relationships instead of one.

```
MATCH (airport {code:  "LAX"})
MERGE (airportDay:AirportDay {id: "LAX_2008-2-1"})
ON CREATE SET airportDay.date = "2008-2-1"
CREATE (airport)-[:HAS_DAY]->(airportDay)
CREATE (airport)-[:`2008-2-1`]->(airportDay)
```

Removing a day

If we want to remove a day we have to delete two relationships instead of one.

```
MATCH (airport {code:  "LAX"})
MATCH (airportDay:AirportDay {id: "LAX_2008-1-1"})
MATCH (airport)-[general:HAS_DAY]->(airportDay)
MATCH (airport)-[specific:`2008-1-1`]->(airportDay)
DELETE general, specific
```

Start playing the next guide….

neo4j

…if you aren’t playing it already

Multiple Models

:play http://guides.neo4j.com/modeling_airports/06_multiple_models.html

Do we need multiple models?

Questions ?

Our current model

Shrinking the model

At the start of the day the first thing we did was create `Airport` nodes but we actually don’t need them anymore!

Flight as a first class citizen

Let’s get on with the refactoring

Continue playing the guide in your browser

Multiple Models

Your turn to evolve the model

What are we going to do?

- Add to the model
  - Airports
  - Aircrafts
  - Cancellations
  - Any other data you can think of

Form groups with the people near you

Create groups of 4 or 5 people with those sitting near you. It'll be more fun to work on this next bit together.

Once you've done that open the next guide:

Your Turn

:play http://guides.neo4j.com/modeling_airports/07_your_turn.html

Wrapping up

What have we done today?

- Modeling workflow
  - Nodes for things
  - Labels for grouping
  - Relationships for structure
  - Build the model iteratively

What have we done today?

- Modeling guidelines
  - Nodes, labels, relationships, properties
  - Properties vs Relationships
  - Relationship Granularity
  - Symmetric and Bidirectional Relationships
  - General vs Specific Relationships

What have we done today?

- Refactorings
  - Derive node from relationship
  - Derive node from property
  - Derive relationship from property

What have we done today?

But above all:

Design your model to answer the questions asked of the data!

End of Module
Your turn to evolve the model

Questions ?