Graphs
From Novice to Graphanista

Dom Davis
@idomdavis
“Visualise and control your IT”
“Doing bad things to innocent graphs”
Chart Appearances by Type

- Bar: 4
- Pie: 3
- Dounut: 1
- Line: 1
Chart Appearances by Type

Bar: 4
Pie: 3
Donut: 1
Line: 1
A directed graph or digraph is a graph in which edges have orientations. It is written as an ordered pair $G = (V, A)$ (sometimes $G = (V, E)$) with

- $V$ a set whose elements are called vertices, nodes, or points;
- $A$ a set of ordered pairs of vertices, called arrows, directed edges (sometimes simply edges with the corresponding set named $E$ instead of $A$), directed arcs, or directed lines.

An arrow $(x, y)$ is considered to be directed from $x$ to $y$; $y$ is called the head and $x$ is called the tail of the arrow; $y$ is said to be a direct successor of $x$ and $x$ is said to be a direct predecessor of $y$. If a path leads from $x$ to $y$, then $y$ is said to be a successor of $x$ and reachable from $x$, and $x$ is said to be a predecessor of $y$. The arrow $(y, x)$ is called the inverted arrow of $(x, y)$.

A directed graph $G$ is called symmetric if, for every arrow in $G$, the corresponding inverted arrow also belongs to $G$. A symmetric loopless directed graph $G = (V, A)$ is equivalent to a simple undirected graph $G' = (V, E)$, where the pairs of inverse arrows in $A$ correspond one-to-one with the edges in $E$; thus the number of edges in $G'$ is $|E| = |A|/2$, that is half the number of arrows in $G$. 
“A graph comprises of vertices and edges, where the edges may be directed or undirected.”

Dom Davis, ACCU Conference 2018
1. Draw two circles
2. Draw the rest of the owl
1 \text{ Follows } 2
Follows
Noun → Verb → Noun
Graphs are everywhere.
Speaker  \(\rightarrow\)  Gives  \(\rightarrow\)  Talk
subject - verb - object
subject - object - verb
verb - subject - object
verb - object - subject
object - verb - subject
object - subject - verb
subject - verb - object
verb(subject, object)
subject.getObject()
Subject Has Object
CYPHER
MATCH (s)-->(o) RETURN s, o
MATCH (s)<--(o) RETURN s, o
MATCH (s) -- (o) RETURN s, o
MATCH (s)-->(o) RETURN s, o
MATCH (s)-[r]->(o) RETURN s, r, o
We make ritual noise
We weave the fabric of dreams
We make cities of sound
We feel the rhythm of time

Covenant - Ritual Noise
CREATE

(:We)-[:MAKE]->(:"Ritual noise"),
(:We)-[:WEAVE]->(:"The fabric of dreams"),
(:We)-[:BUILD]->(:"Cities of sound"),
(:We)-[:FEEL]->(:"The rhythm of time"),
CREATE
(we:We)-[:MAKE]->(Ritual noise),
(we)-[:WEAVE]->(The fabric of dreams),
(we)-[:BUILD]->(Cities of sound),
(we)-[:FEEL]->(The rhythm of time)
$ \text{match } (n) \text{ return } n$
CREATE (we:Lyric {words: 'We'})
CREATE (we:Lyric {words: 'We'}),
(we)-[:MAKE]->(Lyric {words: 'ritual noise'}),
(we)-[:WEAVE]->(Lyric {words: 'the fabric of dreams'}),
(we)-[:BUILD]->(Lyric {words: 'cities of sound'}),
(we)-[:FEEL]->(Lyric {words: 'the rhythm of time'})
$\text{match } (n) \text{ return } n$

Displaying 5 nodes, 4 relationships.
We make ritual noise
We weave the fabric of dreams
We make cities of sound
We feel the rhythm of time

Covenant - Ritual Noise
MATCH (l1:Lyric), (l2:Lyric), (l3:Lyric), (l4:Lyric)
WHERE
  l1.words = 'ritual noise' AND
  l2.words = 'the fabric of dreams' AND
  l3.words = 'cities of sound' AND
  l4.words = 'the rhythm of time'
CREATE
  (:Start)-[:NEXT]->(l1)-[:NEXT]->(l2)-[:NEXT]->(l3)-[:NEXT]->(l4)
$\text{match } (n) \text{ return } n$
CREATE (we:Lyric {words: 'We'}),
(we)-[:MAKE {line: 1}]->(:Lyric {words: 'ritual noise'}),
(we)-[:WEAVE {line: 2}]->(:Lyric {words: 'the fabric of dreams'}),
(we)-[:BUILD {line: 3}]->(:Lyric {words: 'cities of sound'}),
(we)-[:FEEL {line: 4}]->(:Lyric {words: 'the rhythm of time'})
<table>
<thead>
<tr>
<th>s.words</th>
<th>toLower(type(r))</th>
<th>o.words</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;make&quot;</td>
<td>&quot;ritual noise&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;weave&quot;</td>
<td>&quot;the fabric of dreams&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;build&quot;</td>
<td>&quot;cities of sound&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;feel&quot;</td>
<td>&quot;the rhythm of time&quot;</td>
</tr>
</tbody>
</table>
$ MATCH (s)-[r]->(o) RETURN s.words, toLower(type(r)), o.words ORDER BY r.line

<table>
<thead>
<tr>
<th>s.words</th>
<th>toLower(type(r))</th>
<th>o.words</th>
</tr>
</thead>
<tbody>
<tr>
<td>'We'</td>
<td>'make'</td>
<td>&quot;ritual noise&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'weave'</td>
<td>&quot;the fabric of dreams&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'build'</td>
<td>&quot;cities of sound&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'feel'</td>
<td>&quot;the rhythm of time&quot;</td>
</tr>
</tbody>
</table>
MATCH (s)-[r]->(o)
RETURN s.words, toLower(type(r)), o.words
ORDER BY r.line
$ MATCH (s)-[r]->(o) RETURN s.words, toLower(type(r)), o.words ORDER BY r.line

<table>
<thead>
<tr>
<th>s.words</th>
<th>toLower(type(r))</th>
<th>o.words</th>
</tr>
</thead>
<tbody>
<tr>
<td>We</td>
<td>make</td>
<td>ritual noise</td>
</tr>
<tr>
<td>We</td>
<td>weave</td>
<td>the fabric of dreams</td>
</tr>
<tr>
<td>We</td>
<td>build</td>
<td>cities of sound</td>
</tr>
<tr>
<td>We</td>
<td>feel</td>
<td>the rhythm of time</td>
</tr>
</tbody>
</table>
MATCH (s {words: "We"})-[r1]->(o)
OPTIONAL MATCH (o)-[r2 {line: r1.line}]->(n1)
OPTIONAL MATCH (n1)-[r3 {line: r1.line}]->(n2)
RETURN r1.line, s.words, toLower(type(r1)), o.words, toLower(type(r2)), n1.words, toLower(type(r3)), n2.words
ORDER BY r1.line
(graphs)-[:ARE]->(everywhere)
$ MATCH (s)-[r]->(o) RETURN s.words, toLower(type(r)), o.words ORDER BY r.line

<table>
<thead>
<tr>
<th>s.words</th>
<th>toLower(type(r))</th>
<th>o.words</th>
</tr>
</thead>
<tbody>
<tr>
<td>'We'</td>
<td>'make'</td>
<td>&quot;ritual noise&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'weave'</td>
<td>&quot;the fabric of dreams&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'build'</td>
<td>&quot;cities of sound&quot;</td>
</tr>
<tr>
<td>'We'</td>
<td>'feel'</td>
<td>&quot;the rhythm of time&quot;</td>
</tr>
</tbody>
</table>
{
    "name": "react-app",
    "version": "0.1.0",
    "private": true,
    "dependencies": {
        "react": "^16.3.0",
        "react-dom": "^16.3.0",
        "react-scripts": "1.1.2"
    },
    "scripts": {
        "start": "react-scripts start",
        "build": "react-scripts build",
        "test": "react-scripts test --env=jsdom",
        "eject": "react-scripts eject"
    }
}
“Typing ability is inversely proportional to the number of people watching.”

Dom’s first Law
MODELLING
“A person has a position at a company.”
Person has Position at Company
Dom is CTO

Claire is director

SiT is director

TM is CTO

SR is director
(:Person {name: "Dom Davis"})
(:Person)-[:HAS_NAME]->(:`Dom Davis`)
(:Person {name: "Dom Davis"})
 -[:HAS_ROLE {type: "Primary"}]->(:Role {title: "CTO"})
 -[:INCOMPANY]-->(:Company {name: "Tech Marionette"})
-[:HAS_ROLE {type: "Primary"}]->
(Person {name: "Dom Davis"})
-[:HAS_PRIMARY_ROLE]-(Role {title: "CTO"})
-[:IN_COMPANY]-(Company {name: "Tech Marionette"})
(r:Role {title: "CTO"}),
(:Person {name: "Dom Davis"})-[:HAS_ROLE]->(r)
    -[:IN_COMPANY]->(:Company {name: "Tech Marionette"}),
(r)-[:TYPE]->(:Primary)
Drive the model from the language of the domain
Stuff
Stuff

Other Stuff
Stuff Linked To Other Stuff
(:Stuff {
    property1: "some value",
    // : : :
    propertyN: "some other value"
})
(:Concept {
  properties: ["A", "B", "C"]
})
Stuff has properties
(:Stuff)-[:HAS]->(p:Property)
SET p.Name = "A", p.Value = "foo"
(:Stuff)-[:ALIAS]->(:Property)
(s)-[:ALIAS {name: "Dom"}]-(o),
(s)-[:ALIAS {name: "Dominic"}]-(o),
(s)-[:ALIAS {name: "@idomdavis"}]-(o)
(g:Graph)-[:DESCRIBED_BY]->(g)
NO SQL