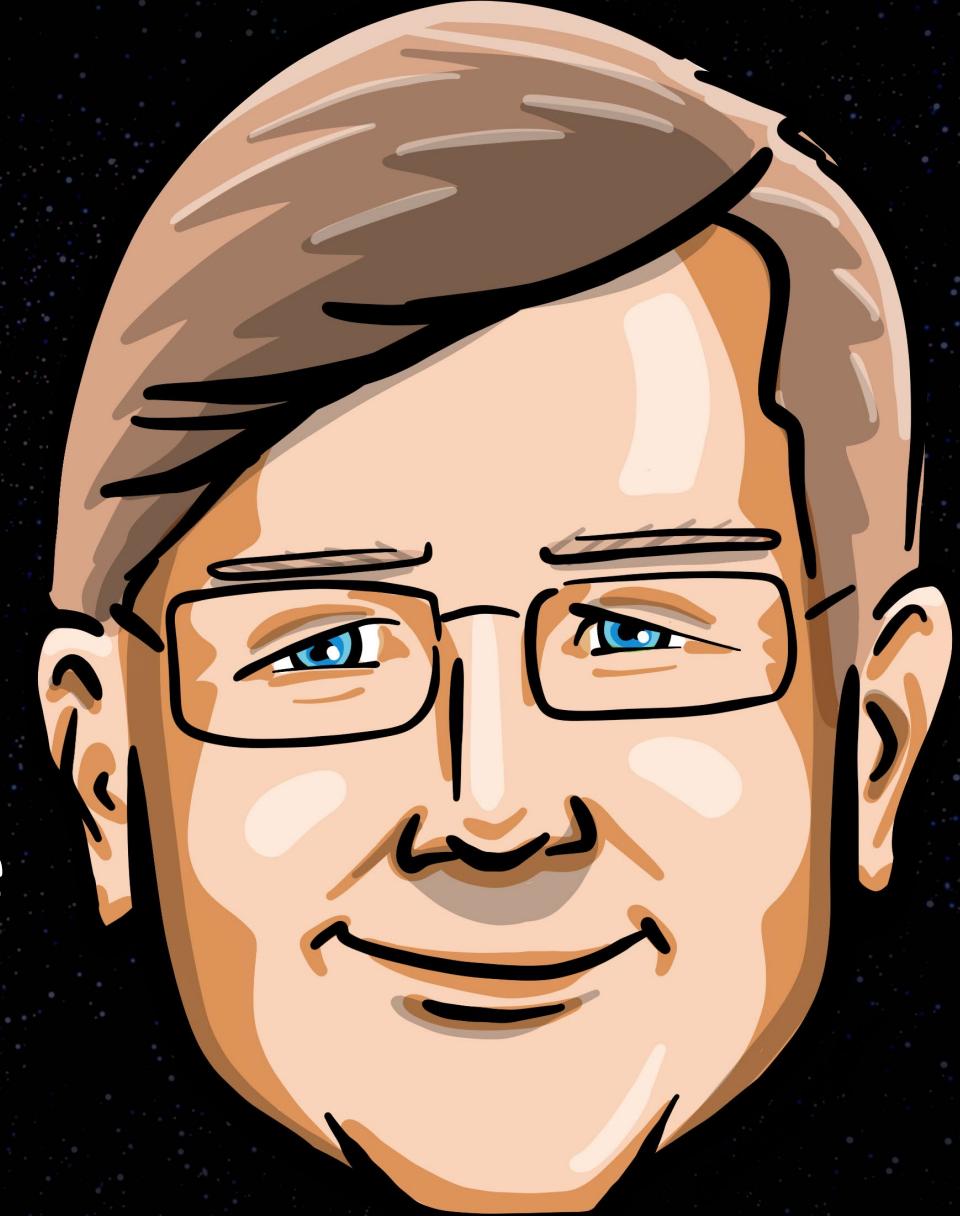




**GRAPHING  
YOUR WAY  
—THROUGH—  
THE COSMOS**

# Who is Chad Green

- ✉️ chadgreen@chadgreen.com
- .twitch TaleLearnCode
- 🌐 TaleLearnCode.com
- 🐦 ChadGreen & TaleLearnCode
- linkedin ChadwickEGreen





# What are Graph Databases

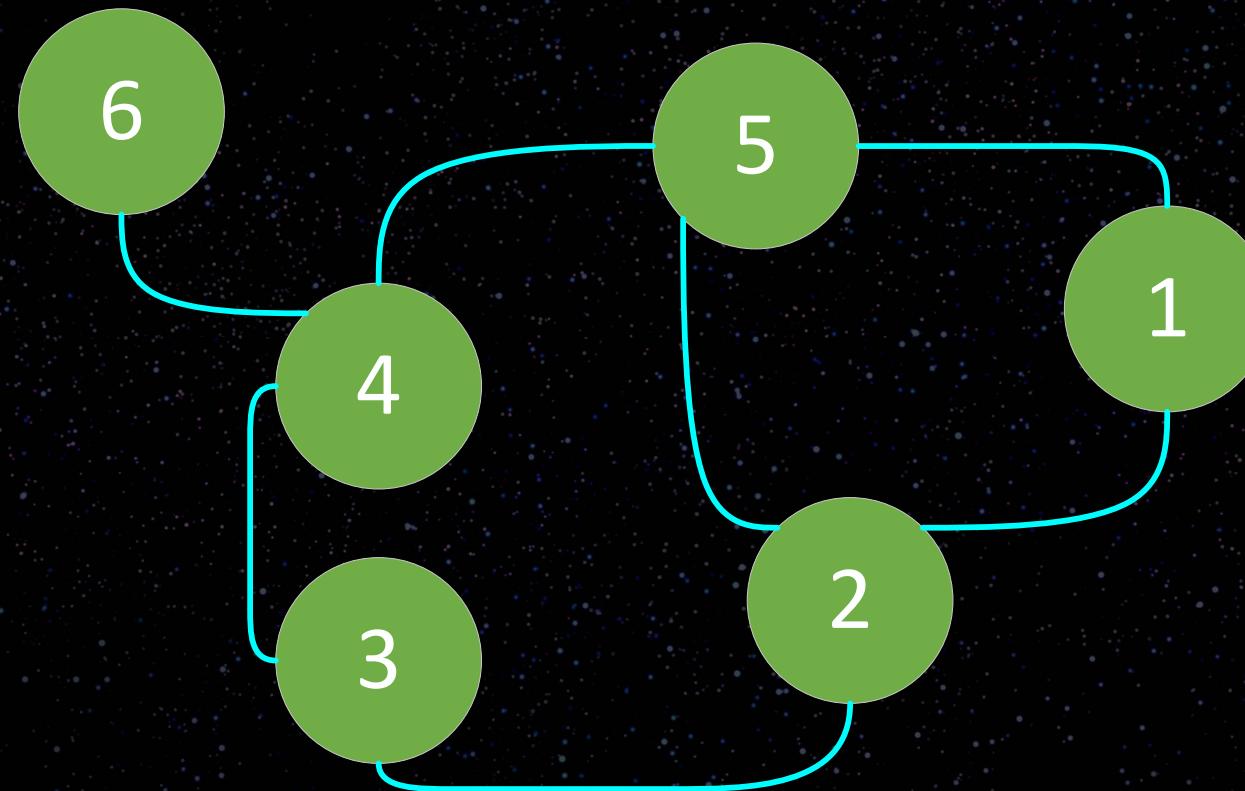
Graphing Your Way Through the Cosmos

# What is a Graph

Structure amounting to set of objects in which some pairs of the objects are in some sense related



# What is a Graph





# What is a Graph

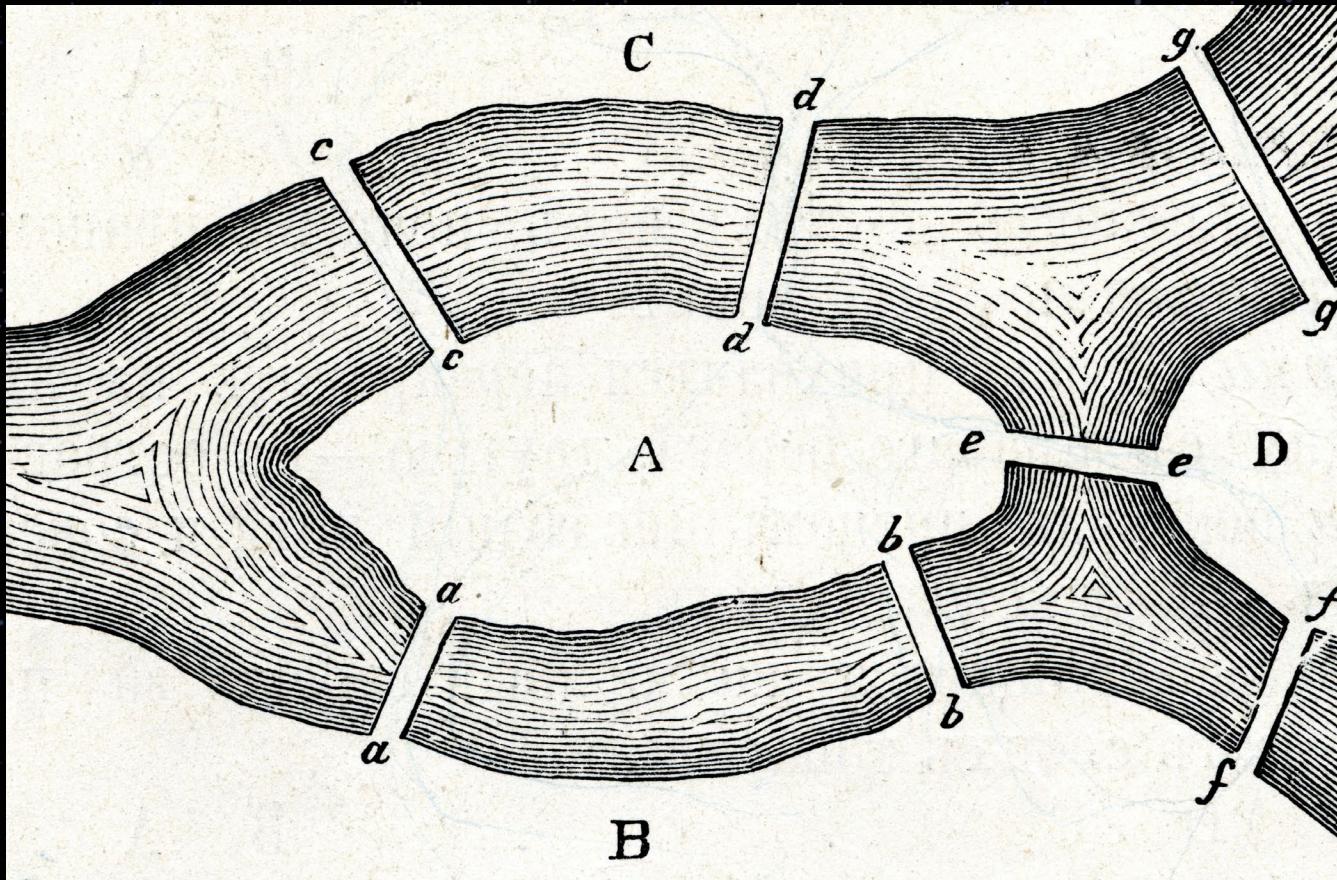
GIVE



# History Graph Theory



# History Graph Theory





# History Graph Theory

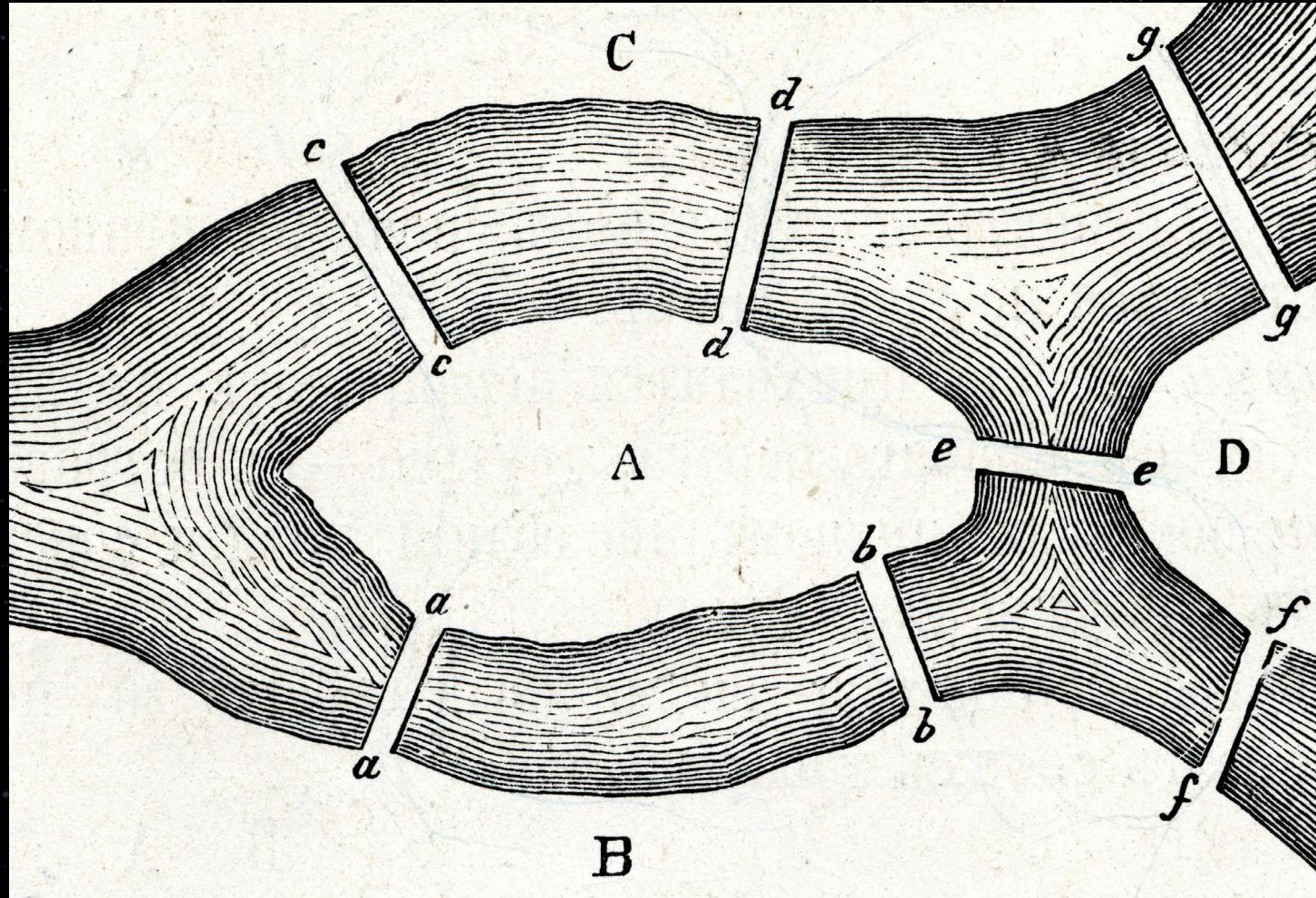


SCHWEIZERISCHE NATIONALBANK  
+  
BANCA NAZIONALE SVIZRA



# Leonard Euler

# History Graph Theory

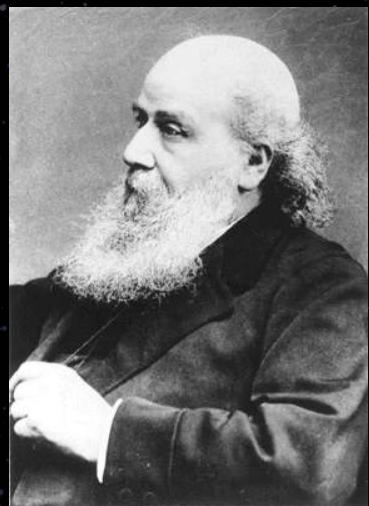
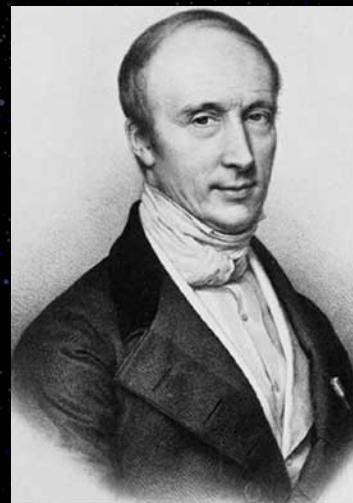


Solutio problematis ad  
geometriam situs  
pertinentis

The solution of a problem  
relating to the geometry  
of position



# History Graph Theory



# Application of Graph Theory

Linguistics

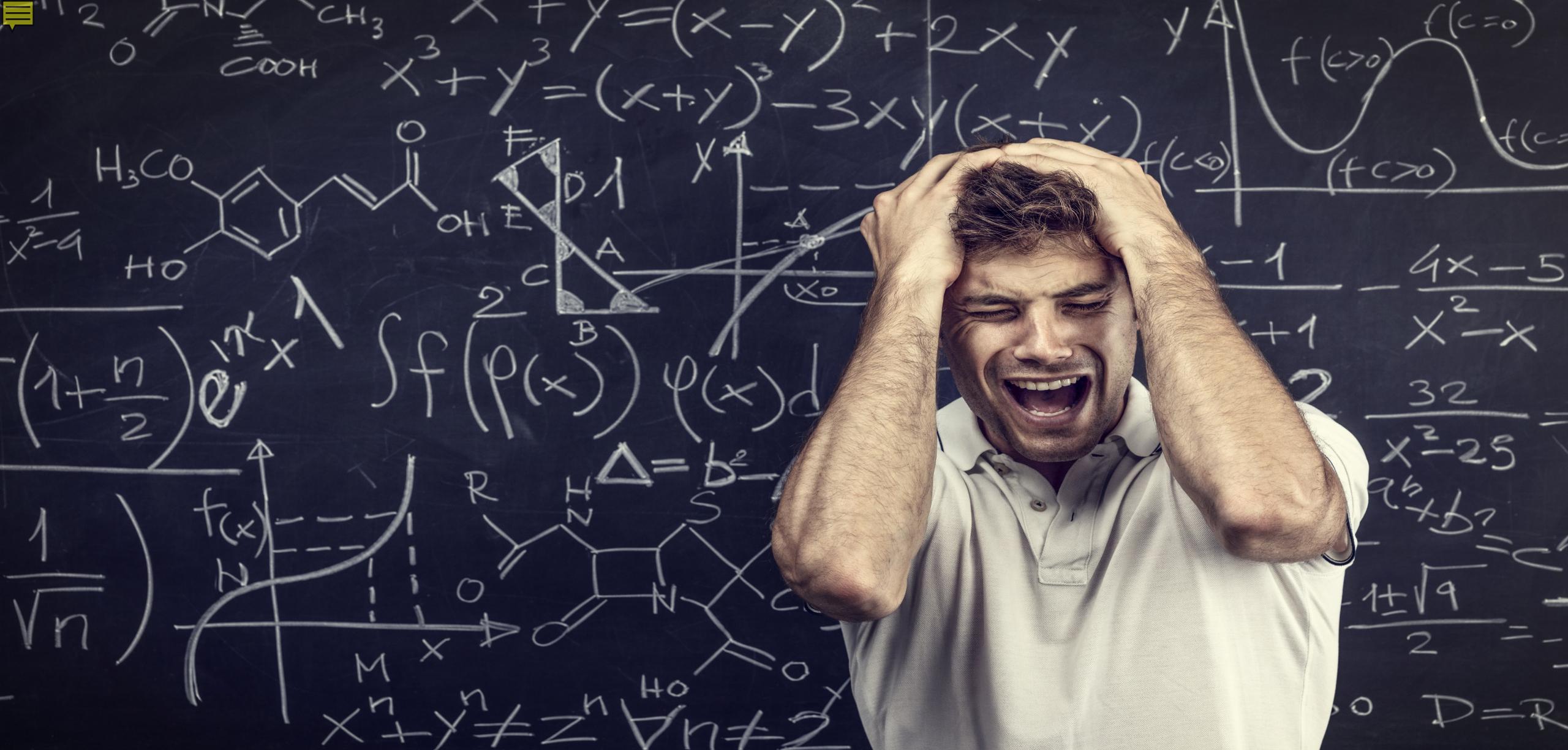
Physics and  
Chemistry

Social Sciences

Biology

Computer  
Science



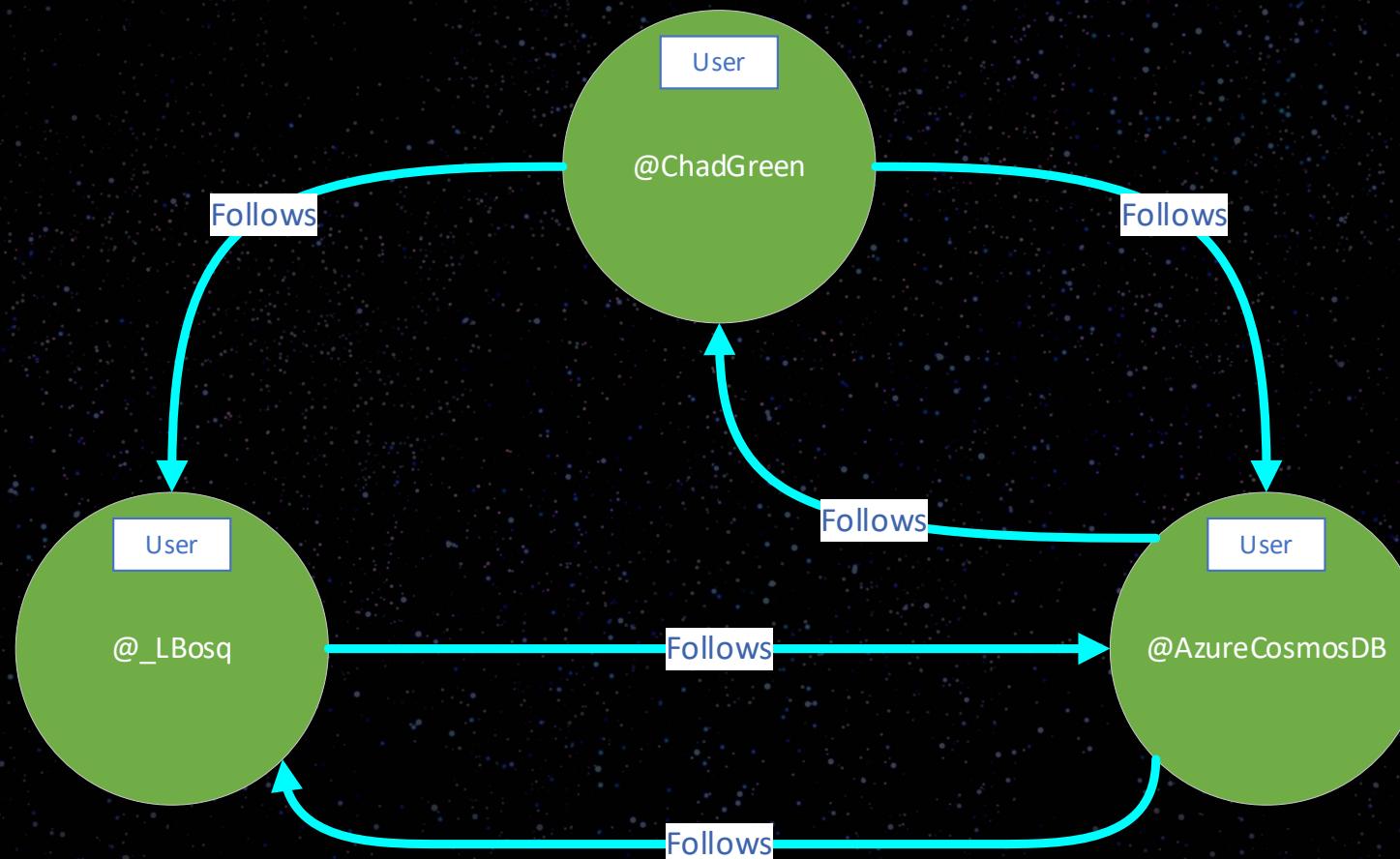


# What is a Graph

Allows us to model all  
kinds of scenarios



# What is a Graph



# What is a Graph Database

Database that uses graph structures  
to represent and store data

# What is a Graph Database

Represents data as it exists in the real world as they are naturally connected

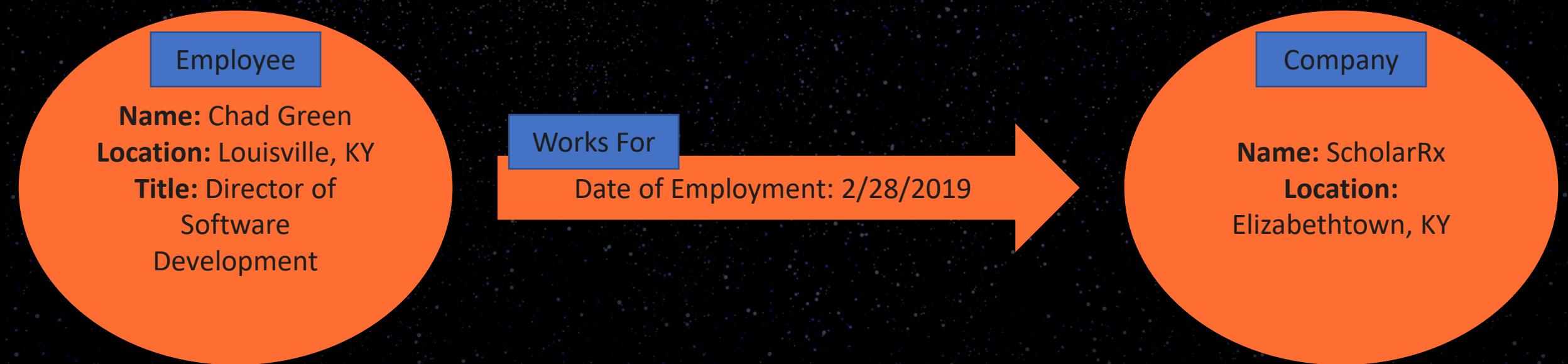


# Property Graph Model

Contains **nodes** (vertices) and **relationships** (edges)

Nodes and relationships contain **properties**

Relationships are **named** and **directed** with a **start** and **end** node



# The Power of Graph Databases

Performance

Flexibility

Agility



# Common Graph Use Cases

Internet of Things

Asset Management

Recommendations

Fraud Detection

Data Integration

Identity and Access Management

Social Networks

Communication Networks

Genomics

Epidemiology

Sematic Web

Search





# Graph vs Relational

Graphing Your Way Through the Cosmos

# Graph vs Relational

## Relational

Tables

Schema with nullables

Relations with foreign keys

Related data fetched with joins

## Graph

Vertices (Nodes)

No schema

Relation is first class citizen

Related data fetched with a pattern

# Human Resource Data

Graph Databases vs Relational Databases



# Human Resource Data

Graph **vs** Relational

EmployeeID	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering

Employees	
💡	EmployeeID
	EmployeeName
	EmployeeGroup

# Human Resource Data

Graph **vs** Relational

-- Create the Employee Table

```
CREATE TABLE Employees
(
    EmployeeID      INT          IDENTITY(1,1),
    EmployeeName    VARCHAR(64),
    EmployeeGroup   VARCHAR(32),
    CONSTRAINT pkEmployees PRIMARY KEY CLUSTERED (EmployeeId)
)
GO
```

-- Populate the Employee Table

```
INSERT INTO Employees (EmployeeName, EmployeeGroup)
VALUES ('Willis B. Hawkins', 'Sales'),
       ('Neil S. Vega',      'Sales'),
       ('Ada C. Lavigne',     'Engineering');
```

GO

# Human Resource Data

Graph **vs** Relational

```
// Create group nodes
g.addV('group').property('id', 'Sales')
g.addV('group').property('id', 'Engineering')

// Create employee nodes
g.addV('employee').property('id', 'Willis B. Hawkins')
g.addV('employee').property('id', 'Neil S. Vega')
g.addV('employee').property('id', 'Ada C. Lavigne')

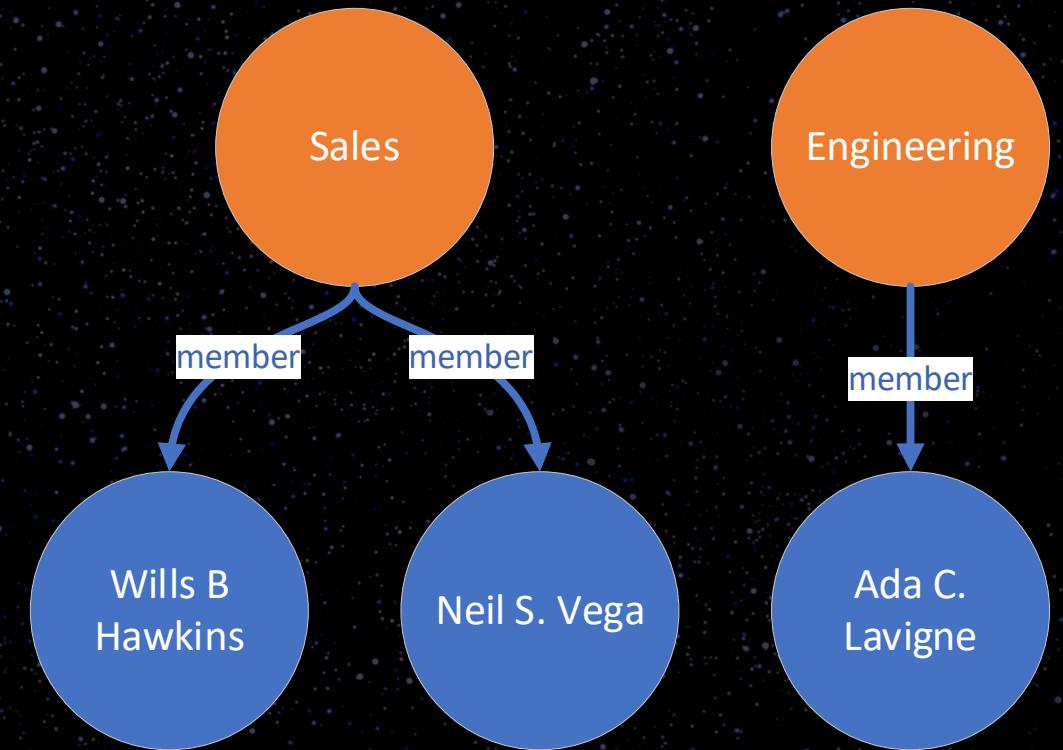
// Create relationships between groups and employees
g.V('Sales').addE('member').to(g.V('Willis B. Hawkins'))
g.V('Sales').addE('member').to(g.V('Neil S. Vega'))
g.V('Engineering').addE('member').to(g.V('Ada C. Lavigne'))
```

# Human Resource Data

Graph **vs** Relational

EmployeeId	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering

3 rows, 3 columns



8 documents (vertices and edges)

# Human Resource Data

Graph **vs** Relational

EmployeeId	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering



`SELECT * FROM Employees;`      `g.V().hasLabel('employee')`



# Employees can now belong to multiple groups

Graph **vs** Relational

-- Create the Groups table

```
CREATE TABLE Groups
```

```
(
```

```
    GroupId   INT      IDENTITY(1,1),
```

```
    GroupName VARCHAR(64),
```

```
    CONSTRAINT pkcGroups PRIMARY KEY CLUSTERED (GroupId)
```

```
)
```

# Employees can now belong to multiple groups

Graph **vs** Relational

```
-- Create the Employee_Group join table
CREATE TABLE Employee_Group
(
    GroupId INT,
    EmployeeId INT,
    CONSTRAINT pkcEmployeeGroup PRIMARY KEY CLUSTERED (GroupId, EmployeeId),
    CONSTRAINT fkEmployeeGroup_Groups FOREIGN KEY (GroupId) REFERENCES Groups(GroupId),
    CONSTRAINT fkEmployeeGroup_Employees FOREIGN KEY (EmployeeId) REFERENCES Employees(EmployeeId)
)
```

# Employees can now belong to multiple groups

Graph **vs** Relational

```
-- Populate the Employee_Group table from Employees and Groups
INSERT INTO Employee_Group (GroupId, EmployeeId)
SELECT Groups.GroupId,
       Employees.EmployeeId
  FROM Employees,
       Groups
 WHERE Groups.GroupName = Employees.EmployeeGroup
```

# Employees can now belong to multiple groups

Graph **vs** Relational

```
-- Drop the Employees.EmployeeGroup column that is no longer valid  
ALTER TABLE Employees DROP COLUMN EmployeeGroup
```

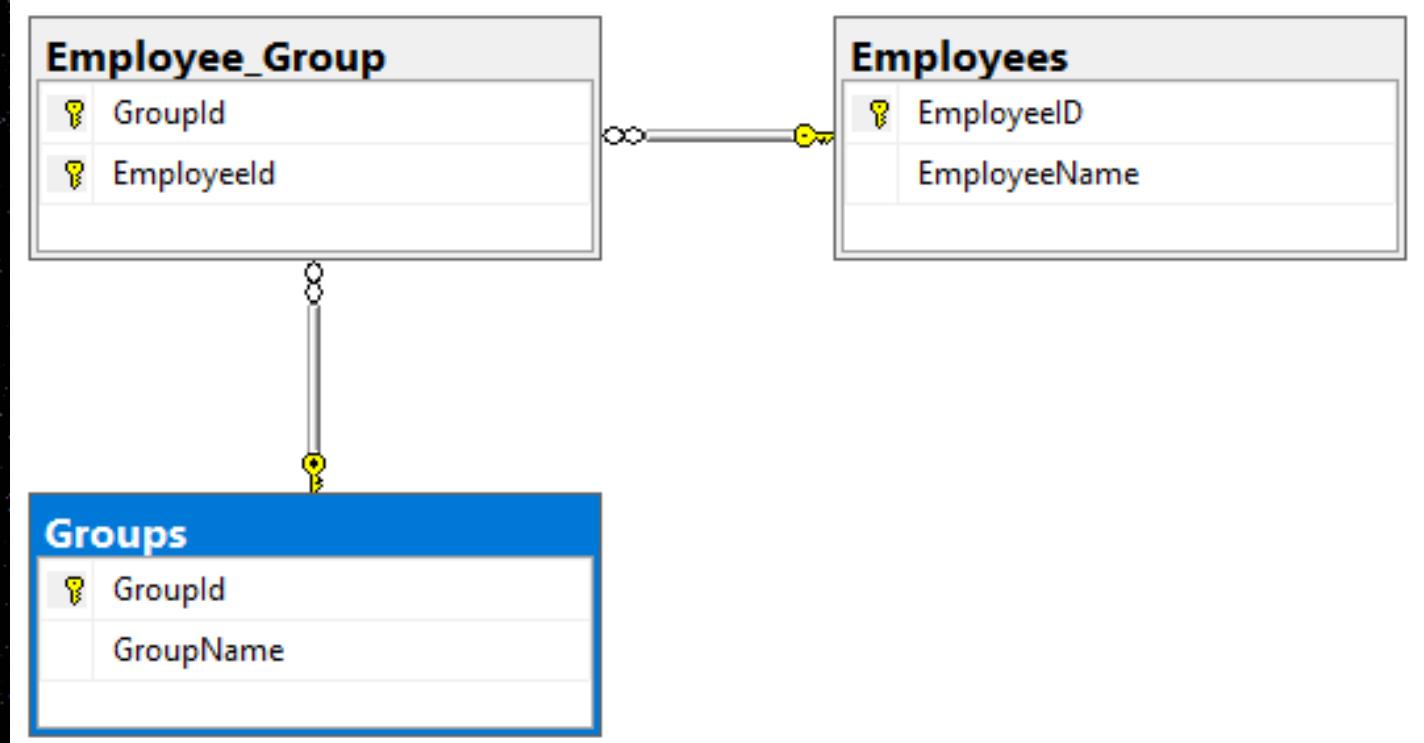
# Employees can now belong to multiple groups

Graph **vs** Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales

GroupId	EmployeeId
1	3
2	1
2	2



# Employees can now belong to multiple groups

Graph **vs** Relational

```
// Add link to existing node
g.V('Sales').addE('member').to(g.V('Ada C. Lavigne'))
```

# Employees can now belong to multiple groups

Graph vs Relational

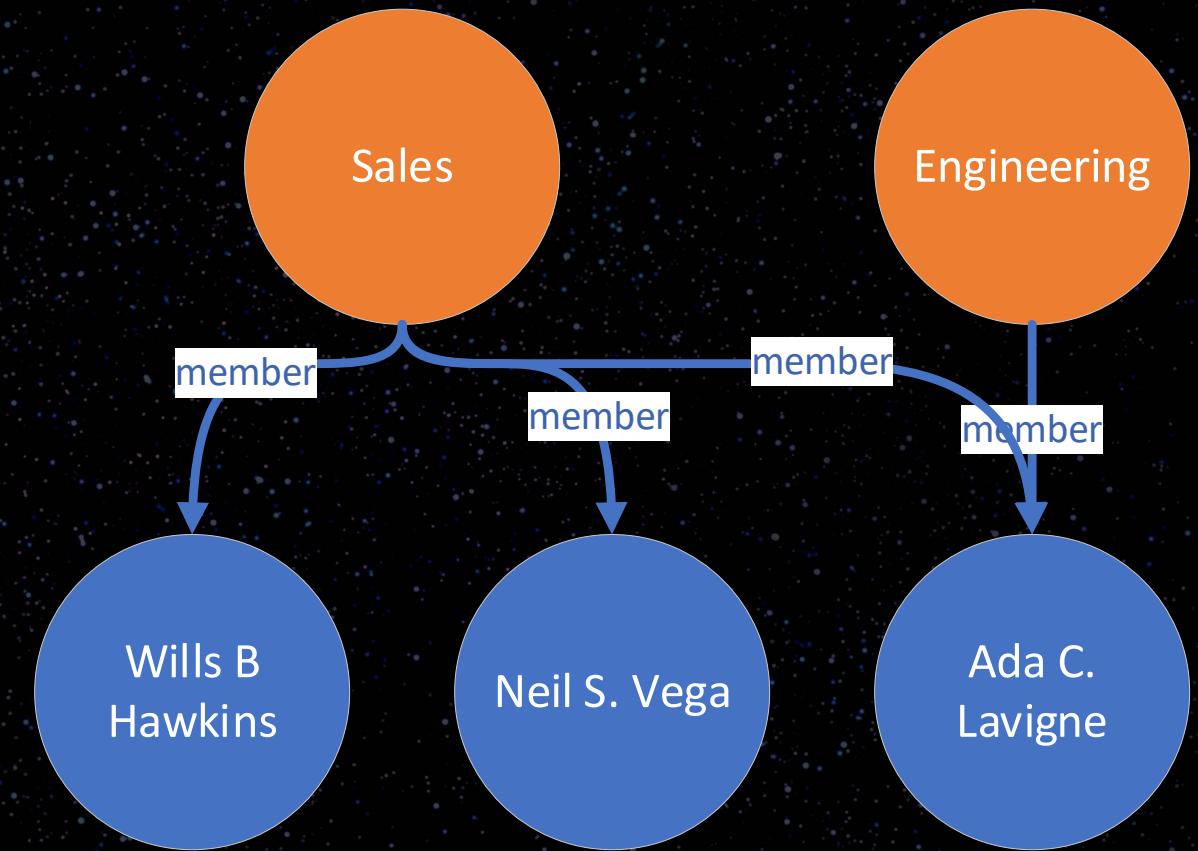
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales

GroupId	EmployeeId
1	3
2	1
2	2

Added 2 tables; 6 rows; 4 new columns  
Removed a column

+1 document



# Employees can now belong to multiple groups

Graph **vs** Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

```
SELECT Employees.EmployeeId,  
       Employees.EmployeeName  
  FROM Employees  
INNER JOIN Employee_Group  
         ON Employee_Group.EmployeeId = Employees.EmployeeId  
INNER JOIN Groups  
         ON Groups.GroupId = Employee_Group.GroupId  
 WHERE Groups.GroupName = 'Sales'
```

```
g.V('Sales').outE('member').inV()
```



# Corporate Merger

Graph Databases vs Relational Databases



# Nested Groups

Graph **vs** Relational

-- Create the new Product Group

```
INSERT INTO Groups (GroupName) VALUES ('Product Group')
```

# Nested Groups

Graph **vs** Relational

```
-- Associate everyone to the new Product Group
INSERT INTO Employee_Group (GroupId, EmployeeId)
SELECT Groups.GroupId,
       Employees.EmployeeId
  FROM Groups,
       Employees
 WHERE Groups.GroupName = 'Product Group'
```

# Nested Groups

Graph **vs** Relational

```
-- Create the Group/Group union table
CREATE TABLE Group_Group
(
    ParentGroupId INT,
    ChildGroupId INT,
    CONSTRAINT pkcGroup_Group PRIMARY KEY CLUSTERED (ParentGroupId, ParentGroupId),
    CONSTRAINT fkGroupGroup_Groups_Parent FOREIGN KEY (ParentGroupId) REFERENCES Groups(groupId),
    CONSTRAINT fkGroupGroup_Groups_Child FOREIGN KEY (ChildGroupId) REFERENCES Groups(groupId)
)
```

# Nested Groups

Graph **vs** Relational

```
-- Relate the child groups to the parent group
INSERT INTO Group_Group (ParentGroupId, ChildGroupId)
SELECT (SELECT GroupId FROM Groups WHERE GroupName = 'Product Group'),
       Groups.GroupId
  FROM Groups
 WHERE Groups.GroupName <> 'Product Group'
```

# Nested Groups

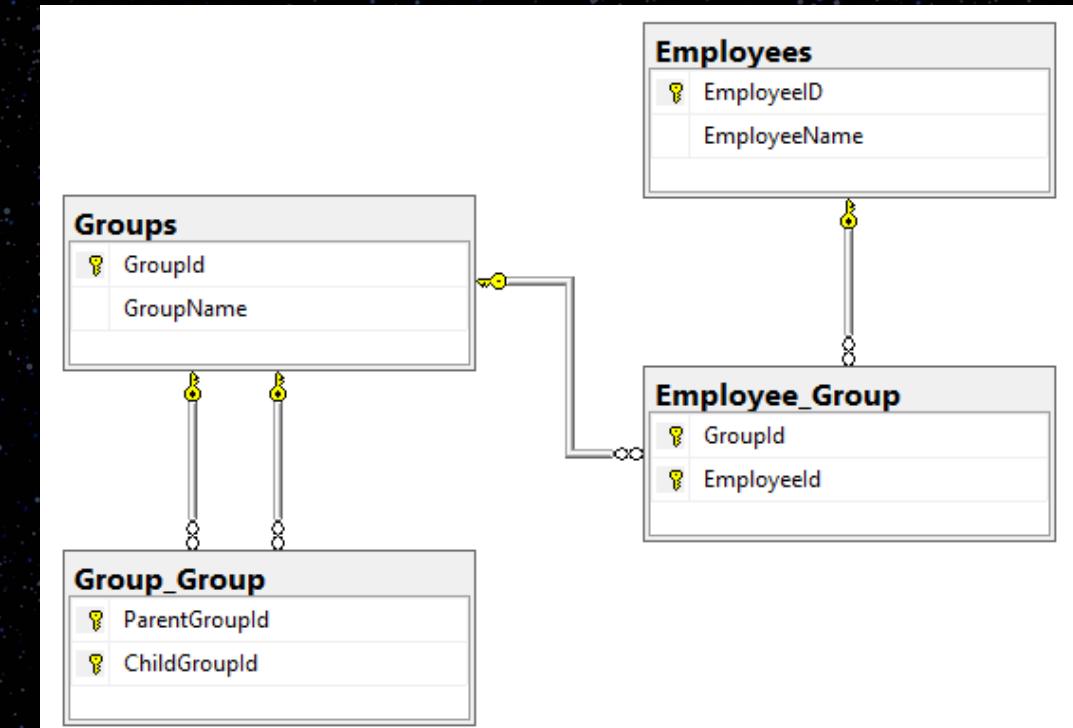
Graph **vs** Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3



# Nested Groups

Graph **vs** Relational

```
// Add supergroup node
g.addV('group').property('id', 'Product Group')

// Link to adjacent nodes
g.V('Product Group').addE('contains_subgroup').to(g.V('Engineering'))
g.V('Product Group').addE('contains_subgroup').to(g.V('Sales'))
```

# Nested Groups

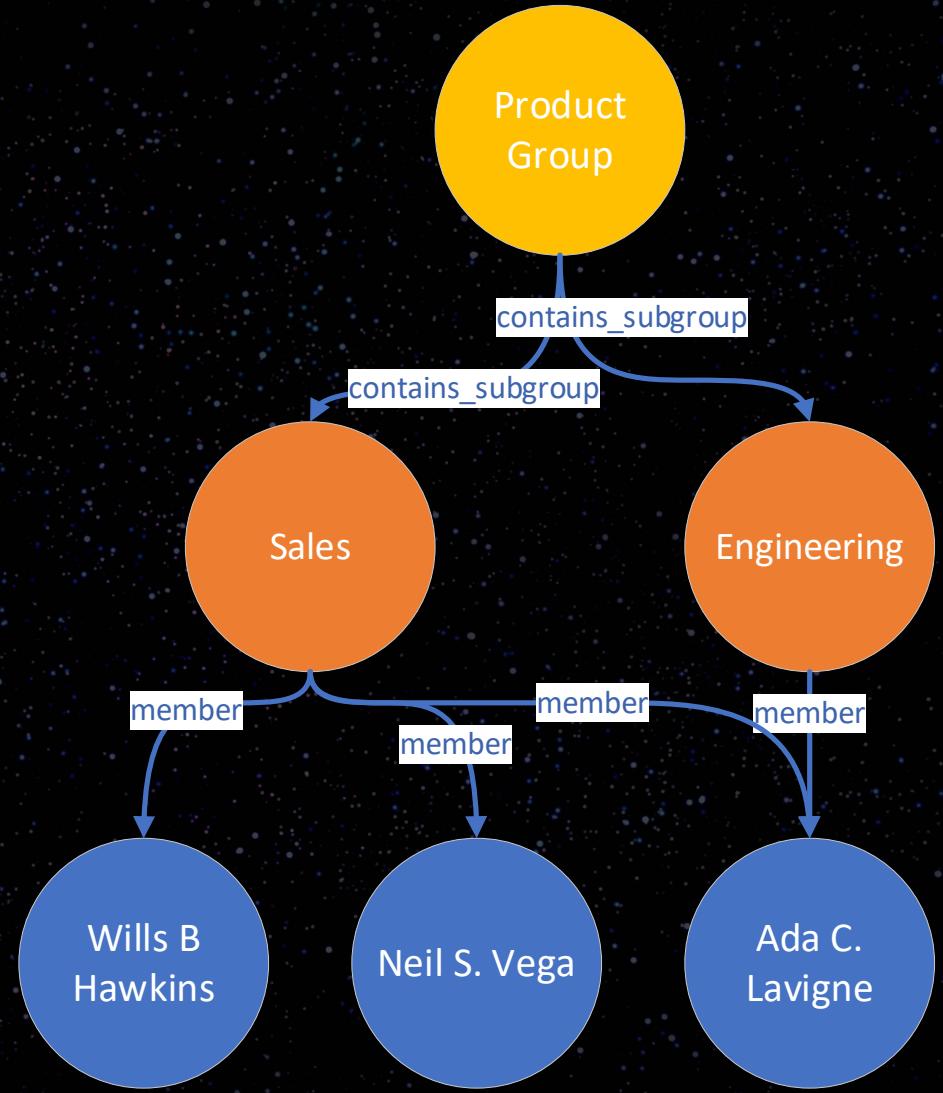
Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3



Added 1 table; 6 rows; 2 new columns

+3 documents

# Nested Groups

Graph **vs** Relational

GroupId	GroupName
1	Engineering
2	Sales



```
SELECT Groups.GroupId,  
       Groups.GroupName  
  FROM Groups  
INNER JOIN Group_Group  
    ON Group_Group.ChildGroupId = Groups.GroupId  
 WHERE Group_Group.ParentGroupId =  
      (SELECT GroupId  
        FROM Groups  
       WHERE GroupName = 'Product Group')
```

```
g.V('Product Group')  
.outE('contains_subgroup')  
.inV()
```

# Management

Graph Databases vs Relational Databases



# Additional Hierarchies

Graph **vs** Relational

```
-- Create the Employee/Employee join table
CREATE TABLE Employee_Employee
(
    ParentEmployeeId INT,
    ChildEmployeeId INT,
    CONSTRAINT pkcEmployeeEmployee PRIMARY KEY CLUSTERED (ParentEmployeeId, ChildEmployeeId),
    CONSTRAINT fkEmployeeEmployee_Employee_Parent FOREIGN KEY (ParentEmployeeId) REFERENCES Employees(EmployeeId),
    CONSTRAINT fkEmployeeEmployee_Employee_Child FOREIGN KEY (ChildEmployeeId) REFERENCES Employees(EmployeeId)
)
```

# Additional Hierarchies

Graph **vs** Relational

```
-- Make Ada the boss
INSERT INTO Employee_Employee (ParentEmployeeId, ChildEmployeeId)
SELECT (SELECT EmployeeId FROM Employees WHERE EmployeeName = 'Ada C. Lavigne'),
       EmployeeId
  FROM Employees
 WHERE EmployeeId IN (SELECT EmployeeId
                         FROM Employee_Group
                        WHERE Employee_Group.GroupId = (SELECT GroupId
                                                       FROM Groups
                                                      WHERE GroupName = 'Sales'))
```

# Additional Hierarchies

Graph **vs** Relational

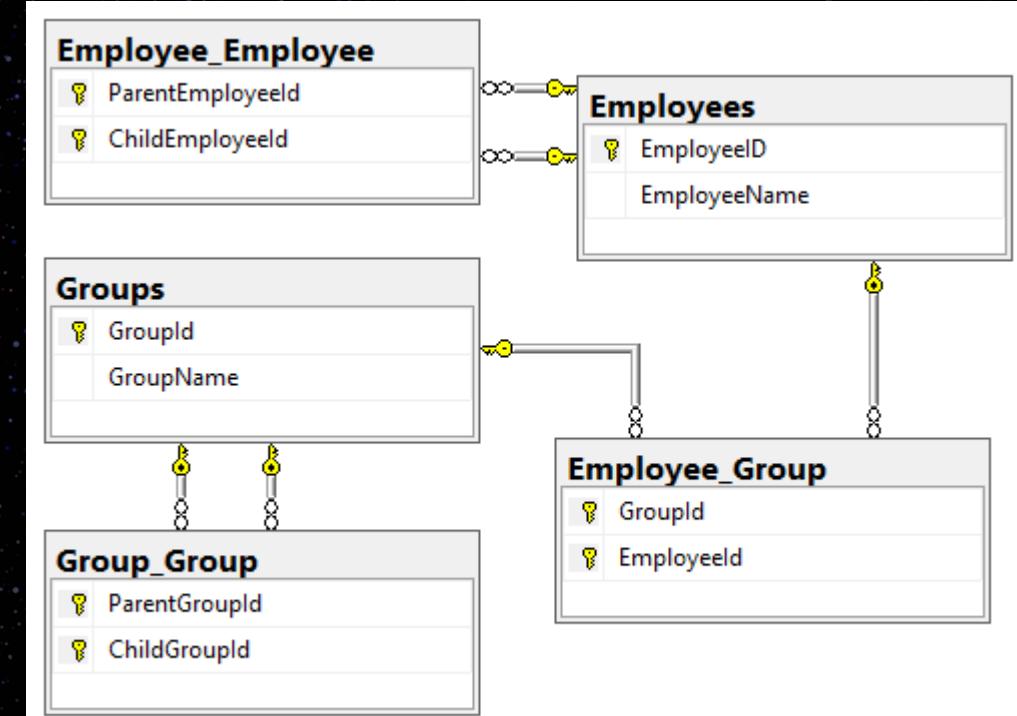
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3

ParentEmployeeId	ChildEmployeeId
3	1
3	2
3	3



# Additional Hierarchies

Graph **vs** Relational

```
// Add relationships
g.V('Ada C. Lavigne').addE('has_report').to(g.V('Willis B. Hawkins'))
g.V('Ada C. Lavigne').addE('has_report').to(g.V('Neil S. Vega'))
```

# Additional Hierarchies

Graph **vs** Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

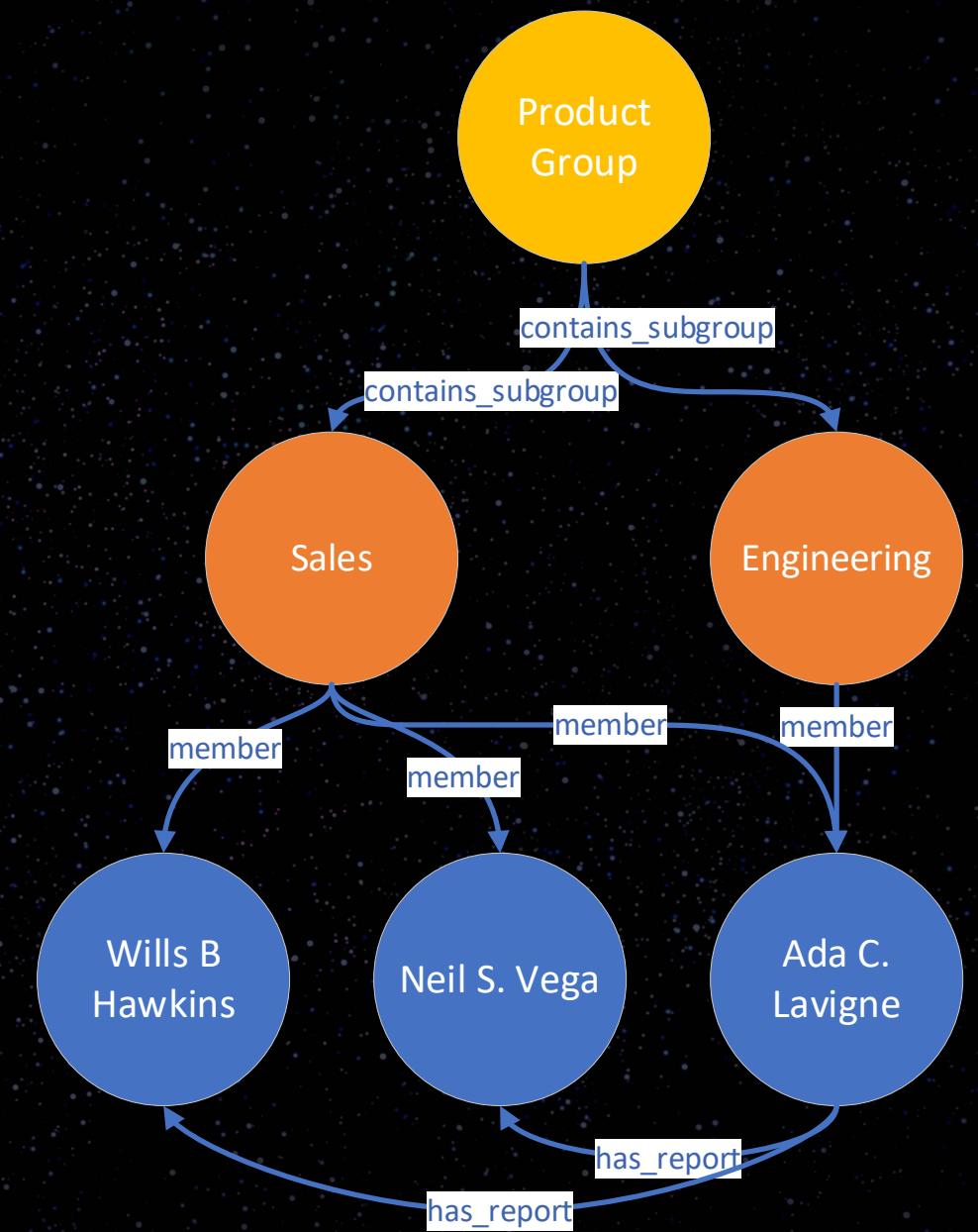
GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3

ParentEmployeeId	ChildEmployeeId
3	1
3	2
3	3



Added 1 table; 2 rows; 2 new columns

+2 documents

# Additional Hierarchies

Graph **vs** Relational

EmployeeName

Ada C. Lavigne

Ada C. Lavigne

```
SELECT DISTINCT EmployeeName
  FROM Employees
 INNER JOIN Employee_Group
    ON Employee_Group_EmployeeId
     = Employees.EmployeeId
 INNER JOIN Employee_Employee
    ON Employee_Employee.ParentEmployeeId
     = Employees.EmployeeId
 WHERE Employee_Group.GroupId =
 (SELECT GroupId
   FROM Groups
  WHERE GroupName = 'Engineering')
```

```
g.V('Engineering')
.outE('member')
.inV()
.outE('has_report')
.values('id')
```



# Challenges of Relational Databases

Schema Management

Table Alterations

Costly Writes Against Multiple Tables

Multiple JOIN Operations

Complex Read Queries



# What is Gremlin

Graphing Your Way Through the Cosmos



# What is TinkerPop

**Open source, vendor-agnostic, graph computing framework**

# What is TinkerPop

Apache2 license

Vendor-agnostic, graph computing framework



# What is TinkerPop

Apache2 license

Model domain as  
graph and  
analyze using  
Gremlin

TinkerPop-  
enabled systems  
integrate with  
one another

Open source, vendor-agnostic, graph computing framework





# What is TinkerPop

Gremlin



# What is TinkerPop

Gremlin

Gremlin Console



# What is TinkerPop

Gremlin

Gremlin Console

Gremlin Server



# What is TinkerPop

Gremlin

Gremlin Console

Gremlin Server

TinkerGraph



# What is TinkerPop

Gremlin

Programming Interfaces

Gremlin Console

Gremlin Server

TinkerGraph



# What is TinkerPop

Gremlin

Programming Interfaces

Gremlin Console

Documentation

Gremlin Server

TinkerGraph

# What is TinkerPop

Gremlin

Programming Interfaces

Gremlin Console

Documentation

Gremlin Server

Useful Recipes

TinkerGraph





# What is Gremlin

- Graph traversal language and virtual machine
- Supports OLTP and OLAP
- Supports imperative and declarative querying
- Supports user-defined domain specified languages





# What is Cosmos DB

Graphing Your Way Through the Cosmos



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Turnkey global  
distribution





# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Turnkey global  
distribution





# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Elastic scale out of  
storage & throughput

Turnkey global  
distribution



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Elastic scale out of  
storage & throughput

Turnkey global  
distribution



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Turnkey global distribution

Elastic scale out  
of storage & throughput



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Turnkey global distribution

Elastic scale out  
of storage & throughput



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Five well-defined consistency models

Turnkey global distribution

Elastic scale out  
of storage & throughput

Guaranteed low latency  
at the 99<sup>th</sup> percentile



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Comprehensive SLAs

Turnkey global distribution

Elastic scale out of storage & throughput

Guaranteed low latency at the 99<sup>th</sup> percentile

Five well-defined consistency models



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Comprehensive SLAs

Turnkey global distribution

Elastic scale out  
of storage & throughput

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Five well-defined  
consistency models



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Battle Tested



XBOX LIVE



Office 365



Turnkey global distribution

Elastic scale out  
of storage & throughput

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Five well-defined  
consistency models

Comprehensive  
SLAs

# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Battle Tested



Liberty  
Mutual.  
INSURANCE



Rolls-Royce

ExxonMobil



Turnkey global  
distribution

Elastic scale out  
of storage & throughput

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Five well-defined  
consistency models

Comprehensive  
SLAs



# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Ubiquitous Regional Presence





# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

## Secure by default and enterprise ready

Turnkey global distribution

Elastic scale out  
of storage & throughput

Guaranteed low latency  
at the 99<sup>th</sup> percentile

Five well-defined  
consistency models

Comprehensive  
SLAs

# Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service



Table API



cassandra

Core  
(SQL)  
API



MongoDB



Key-value



Column-family



Document



Graph





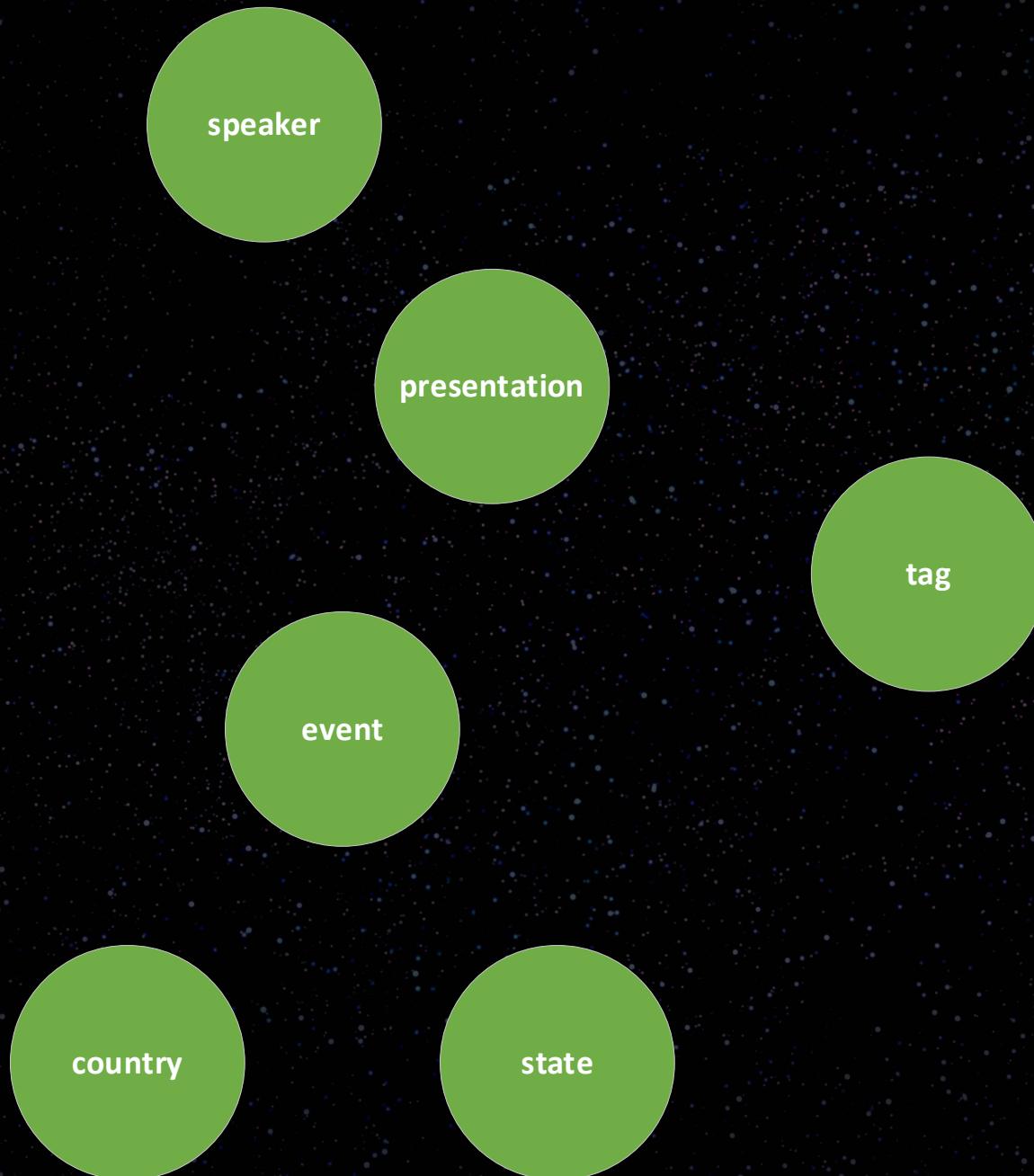
# Exploring Graph Traversals

Graphing Your Way Through the Cosmos

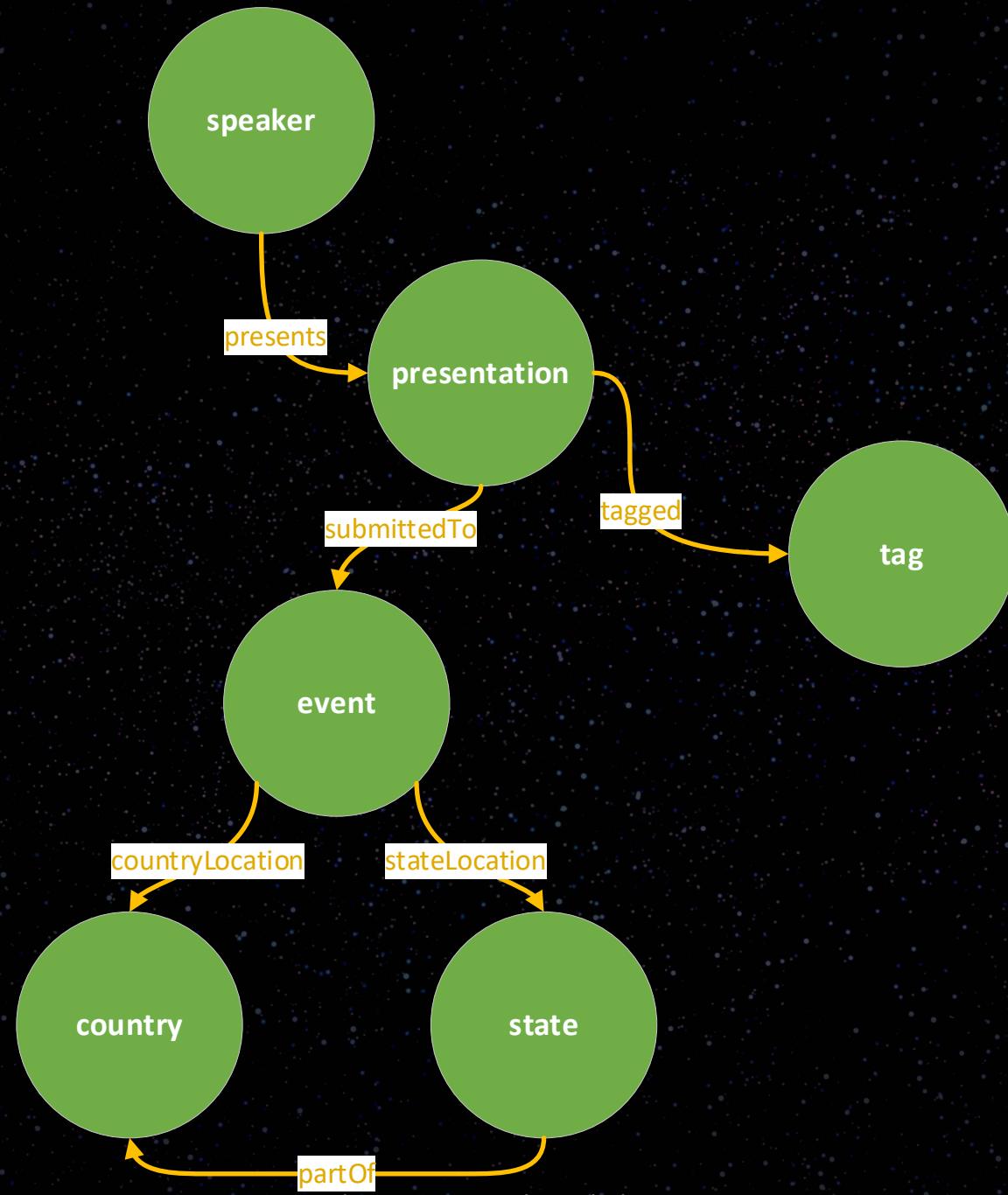
# Requirements for Speaking Engagement Management

- Where has a presentation been submitted?
- What presentations are tagged with a particular tag?
- Where have presentations tagged with a particular tag been accepted?
- What events has a speaker been accepted at?
- Where were the events a speaker has been accepted to?

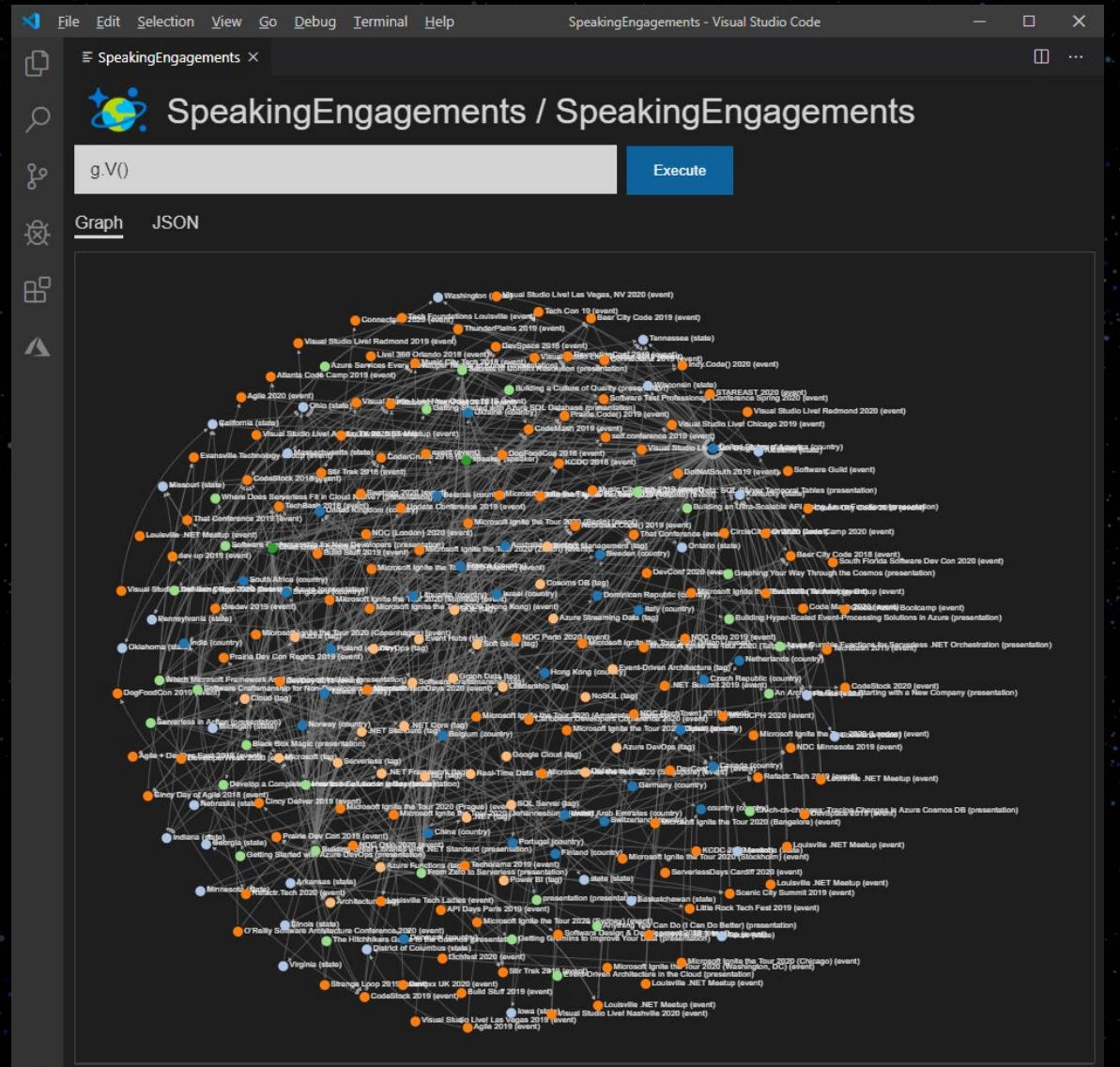
# “Schema”



# “Schema”



# View the whole graph



g.V()

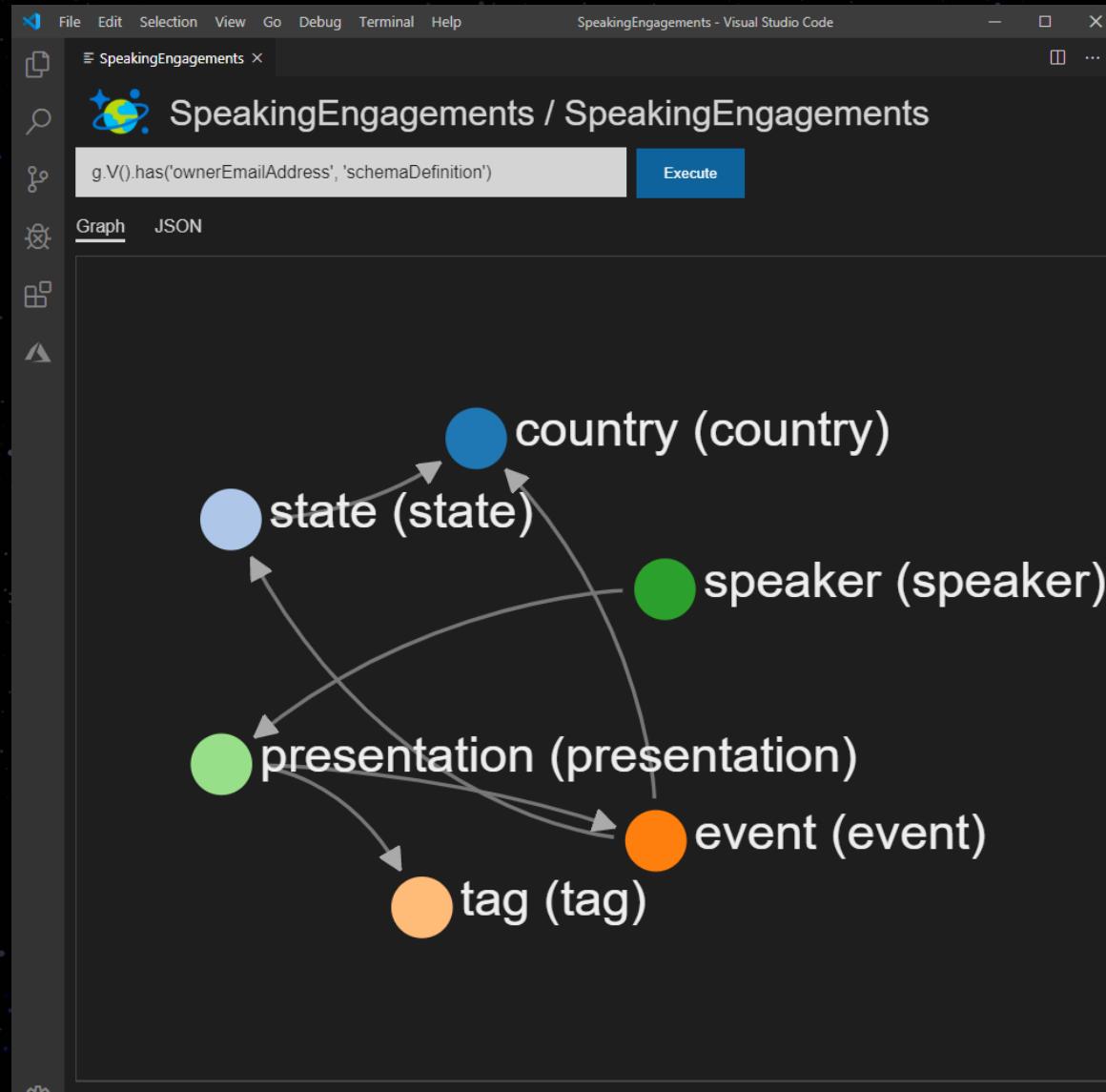
# View all of the edges

A screenshot of a Visual Studio Code terminal window titled "SpeakingEngagements - Visual Studio Code". The terminal shows a Gremlin query "g.E()" entered in the input field, with a blue "Execute" button to its right. The output is displayed in JSON format under the "JSON" tab. The JSON response is a list of edge documents, each containing fields like id, label, type, inVLabel, outVLabel, inV, outV, and properties. The properties field includes a "description" key for each edge type. The terminal interface includes standard VS Code icons for file operations and a gear icon for settings.

```
[{"id": "49739ebc-fd37-47bf-a379-80c2d2089968", "label": "partOf", "type": "edge", "inVLabel": "country", "outVLabel": "state", "inV": "a5cbc5f-1ff8-42cc-9192-78b9654ac882", "outV": "77e425ab-alb1-49c6-9313-6504d140c7cd", "properties": { "description": "The country the state is a part of" }}, {"id": "b35d003e-3b1d-4639-a648-2343d6aacaf1", "label": "countryLocation", "type": "edge", "inVLabel": "country", "outVLabel": "event", "inV": "a5cbc5f-1ff8-42cc-9192-78b9654ac882", "outV": "75738d8e-276c-46e5-a960-da27397ad563", "properties": { "description": "The country where the event is located" }}, {"id": "fbc2cb11-df01-42c5-ab6a-d35717574990", "label": "stateLocation", "type": "edge", "inVLabel": "state", "outVLabel": "event", "inV": "77e425ab-alb1-49c6-9313-6504d140c7cd", "outV": "75738d8e-276c-46e5-a960-da27397ad563", "properties": { "description": "The country where the event is located" }}, {"id": "c8471b48-89f0-47ba-a8a9-d8a65df9eb88", "label": "tagged", "type": "edge", "inVLabel": "tag", "outVLabel": "presentation", "inV": "83b89345-d2a3-4136-9a48-0892d1857fa9", "outV": "7d082568-426f-496b-b1a5-4cf0d9fe2f6a", "properties": { "description": "Associates a tag with a presentation" }}]
```

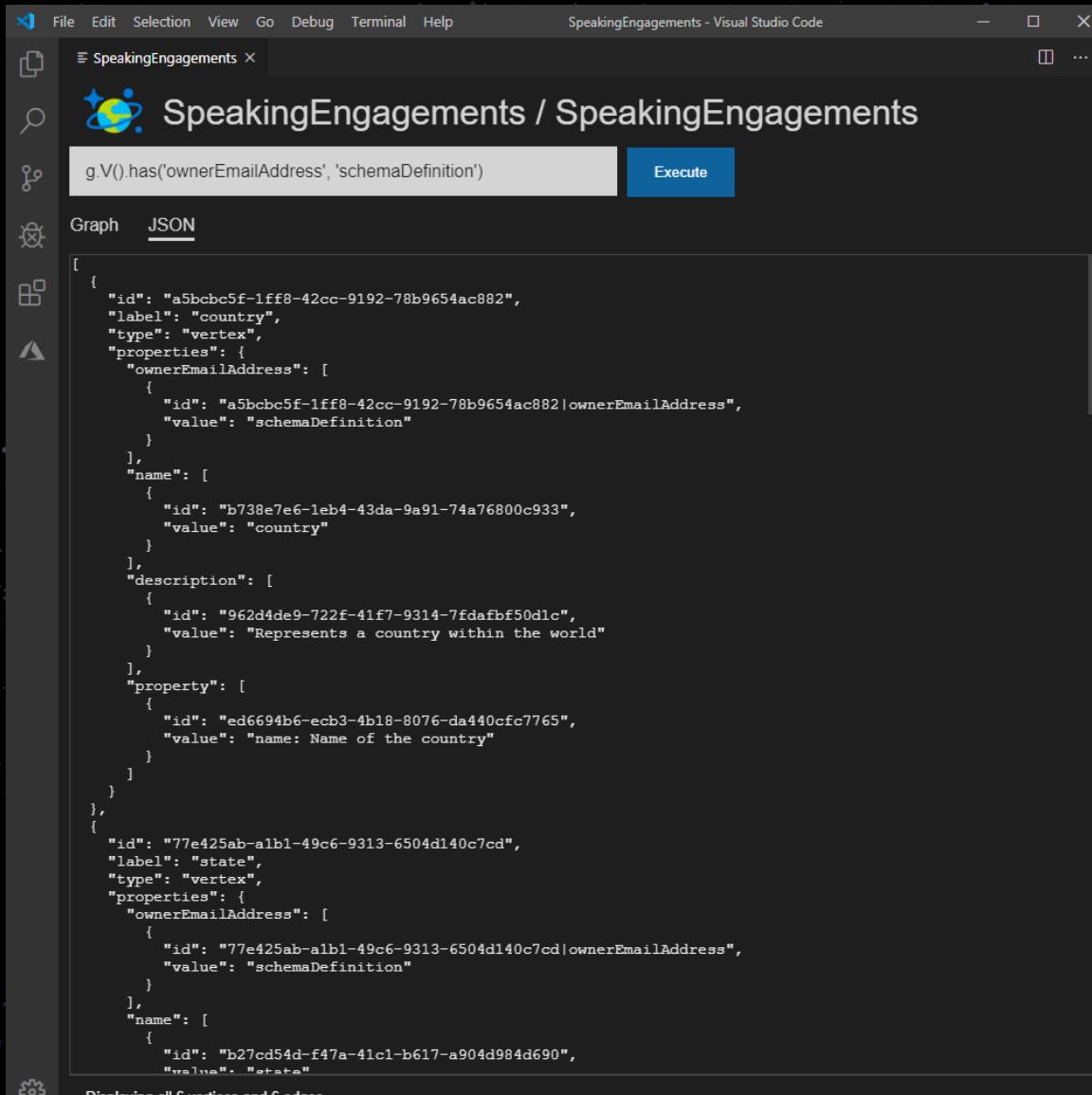
g.E()

# View the schema definition



```
g.V()  
.has('ownerEmailAddress',  
'schemaDefinition')
```

# View the schema definition



A screenshot of Visual Studio Code showing a Gremlin query in the terminal. The terminal title is "SpeakingEngagements - Visual Studio Code". The query is:

```
g.V().has('ownerEmailAddress', 'schemaDefinition')
```

The results are displayed in JSON format:

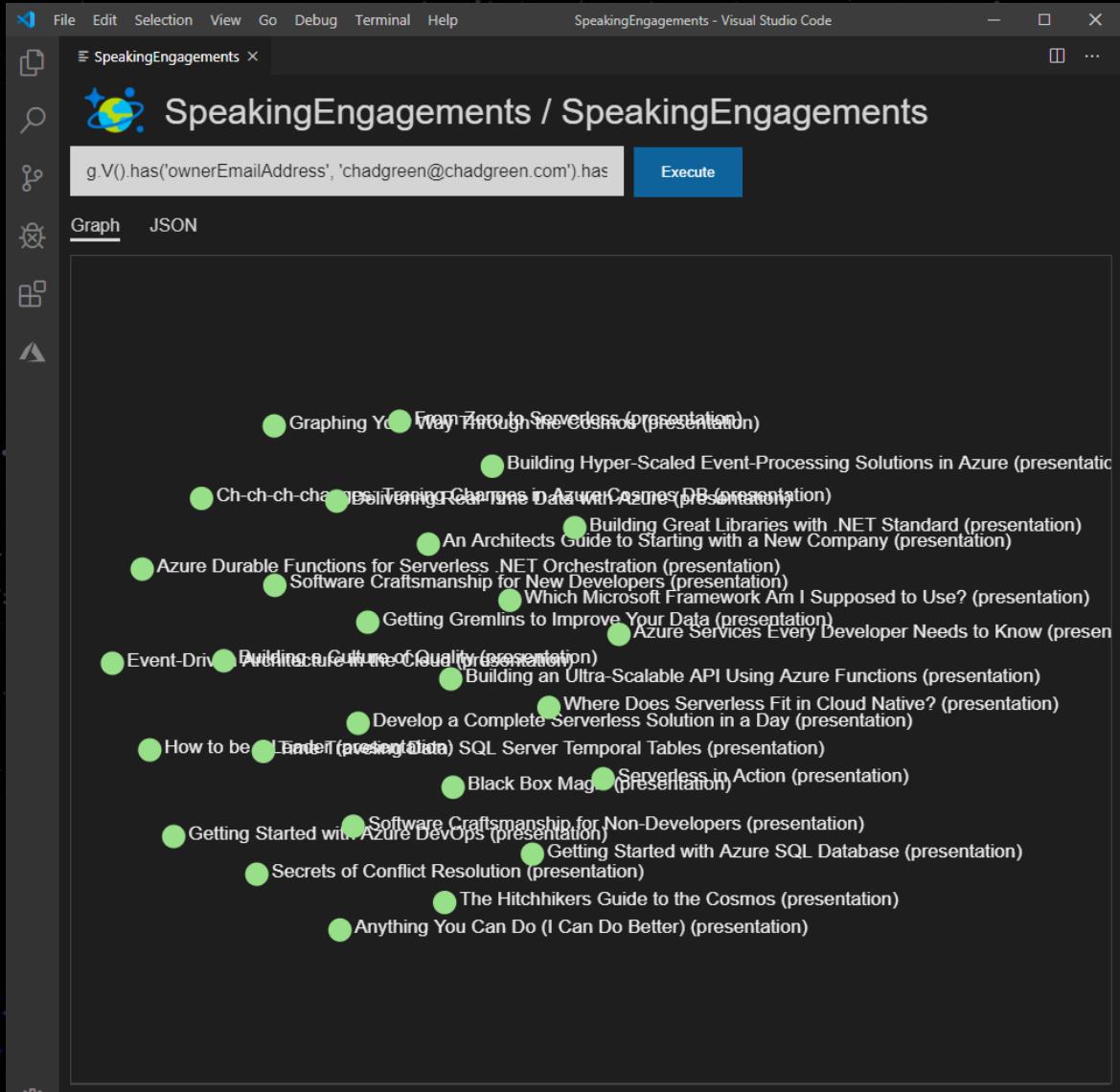
```
[{"id": "a5bcbe5f-1ff8-42cc-9192-78b9654ac882", "label": "country", "type": "vertex", "properties": {"ownerEmailAddress": [{"id": "a5bcbe5f-1ff8-42cc-9192-78b9654ac882|ownerEmailAddress", "value": "schemaDefinition"}]}, "name": [{"id": "b738e7e6-1eb4-43da-9a91-74a76800c933", "value": "country"}], "description": [{"id": "962d4de9-722f-41f7-9314-7fdafbf50d1c", "value": "Represents a country within the world"}], "property": [{"id": "ed6694b6-ecb3-4b18-8076-da440cf7765", "value": "name: Name of the country"}]}, {"id": "77e425ab-a1b1-49c6-9313-6504d140c7cd", "label": "state", "type": "vertex", "properties": {"ownerEmailAddress": [{"id": "77e425ab-a1b1-49c6-9313-6504d140c7cd|ownerEmailAddress", "value": "schemaDefinition"}]}, "name": [{"id": "b27cd54d-f47a-41c1-b617-a904d984d690", "value": "state"}]}
```

At the bottom of the terminal, it says "Displaying all 6 vertices and 6 edges".

g.V()  
.has('ownerEmailAddress',  
'schemaDefinition')



# What presentations are in my repertoire?



A screenshot of Visual Studio Code showing a terminal window titled "SpeakingEngagements - Visual Studio Code". The terminal displays a Gremlin query:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').hasLabel('presentation')
```

The "Execute" button is highlighted in blue. Below the terminal, there is a graph visualization showing a network of 27 vertices (green circles) connected by edges. Each vertex has a label indicating the title of a presentation and its type (e.g., "presentation").

Some visible presentation titles include:

- Graphing Your Way Through the Cosmos (presentation)
- Building Hyper-Scaled Event-Processing Solutions in Azure (presentation)
- Ch-ch-ch-changes! Training Changes in Azure Cosmos DB (presentation)
- An Architects Guide to Starting with a New Company (presentation)
- Azure Durable Functions for Serverless .NET Orchestration (presentation)
- Software Craftsmanship for New Developers (presentation)
- Getting Gremlins to Improve Your Data (presentation)
- Azure Services Every Developer Needs to Know (presentation)
- Event-Driven Building a Culture of Quality (presentation)
- Building an Ultra-Scalable API Using Azure Functions (presentation)
- Where Does Serverless Fit in Cloud Native? (presentation)
- Develop a Complete Serverless Solution in a Day (presentation)
- How to be a Leader (presentation)
- SQL Server Temporal Tables (presentation)
- Black Box Magician (presentation)
- Software Craftsmanship for Non-Developers (presentation)
- Getting Started with Azure DevOps (presentation)
- Getting Started with Azure SQL Database (presentation)
- Secrets of Conflict Resolution (presentation)
- The Hitchhiker's Guide to the Cosmos (presentation)
- Anything You Can Do (I Can Do Better) (presentation)

At the bottom of the terminal, it says "Displaying all 27 vertices and 0 edges".

```
g.V()  
.has('ownerEmailAddress',  
'chadgreen@chadgreen.com')  
.hasLabel('presentation')
```



T 0 0 0 Azure: chadgreen@chadgreen.com

ay Through the Cosmos



# What presentations are in my repertoire?

The screenshot shows the Visual Studio Code interface with the Neo4j extension open. The title bar reads "SpeakingEngagements - Visual Studio Code". The main area displays a graph visualization of presentations. A query in the editor pane is:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').hasLabel('presentation')
```

The graph shows 27 green circular nodes, each representing a presentation. The nodes are arranged in a hierarchical structure. Some node labels are partially visible or obscured by other nodes. The bottom status bar indicates "Displaying all 27 vertices and 0 edges".

g.V()  
.has('ownerEmailAddress',  
'chadgreen@chadgreen.com')  
.hasLabel('presentation')

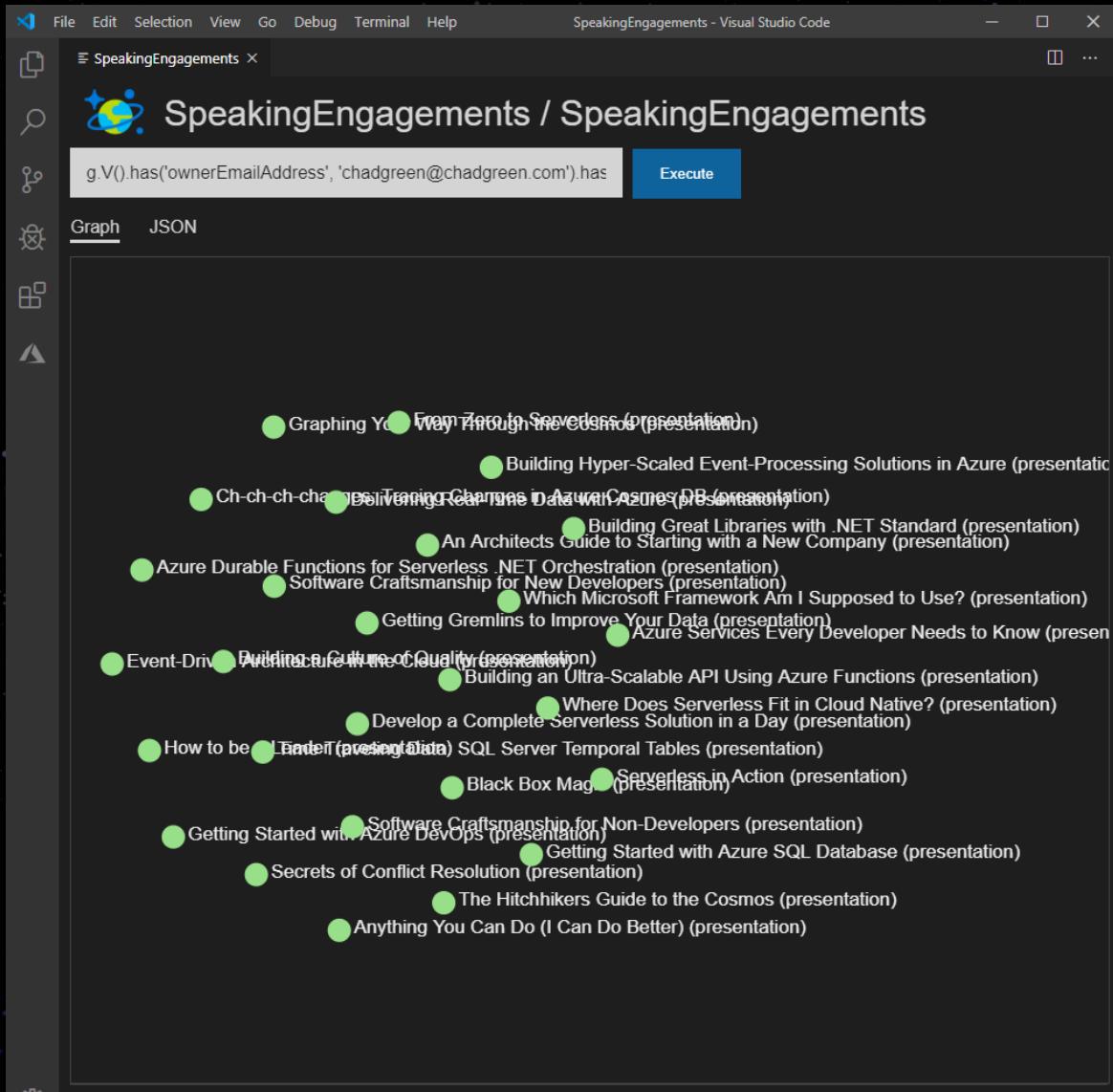


0 0 0 Azure: chadgreen@chadgreen.com

ay Through the Cosmos



# What presentations are in my repertoire?



The screenshot shows the Visual Studio Code interface with the Neo4j extension open. The title bar says "SpeakingEngagements - Visual Studio Code". The main area displays a graph visualization of presentations. A query in the editor pane is:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').hasLabel('presentation')
```

The graph shows 27 green circular nodes, each representing a presentation. Some node labels are partially visible or obscured by other nodes. The bottom status bar says "Displaying all 27 vertices and 0 edges".

g.V()  
.has('ownerEmailAddress',  
'chadgreen@chadgreen.com')  
.hasLabel('presentation')

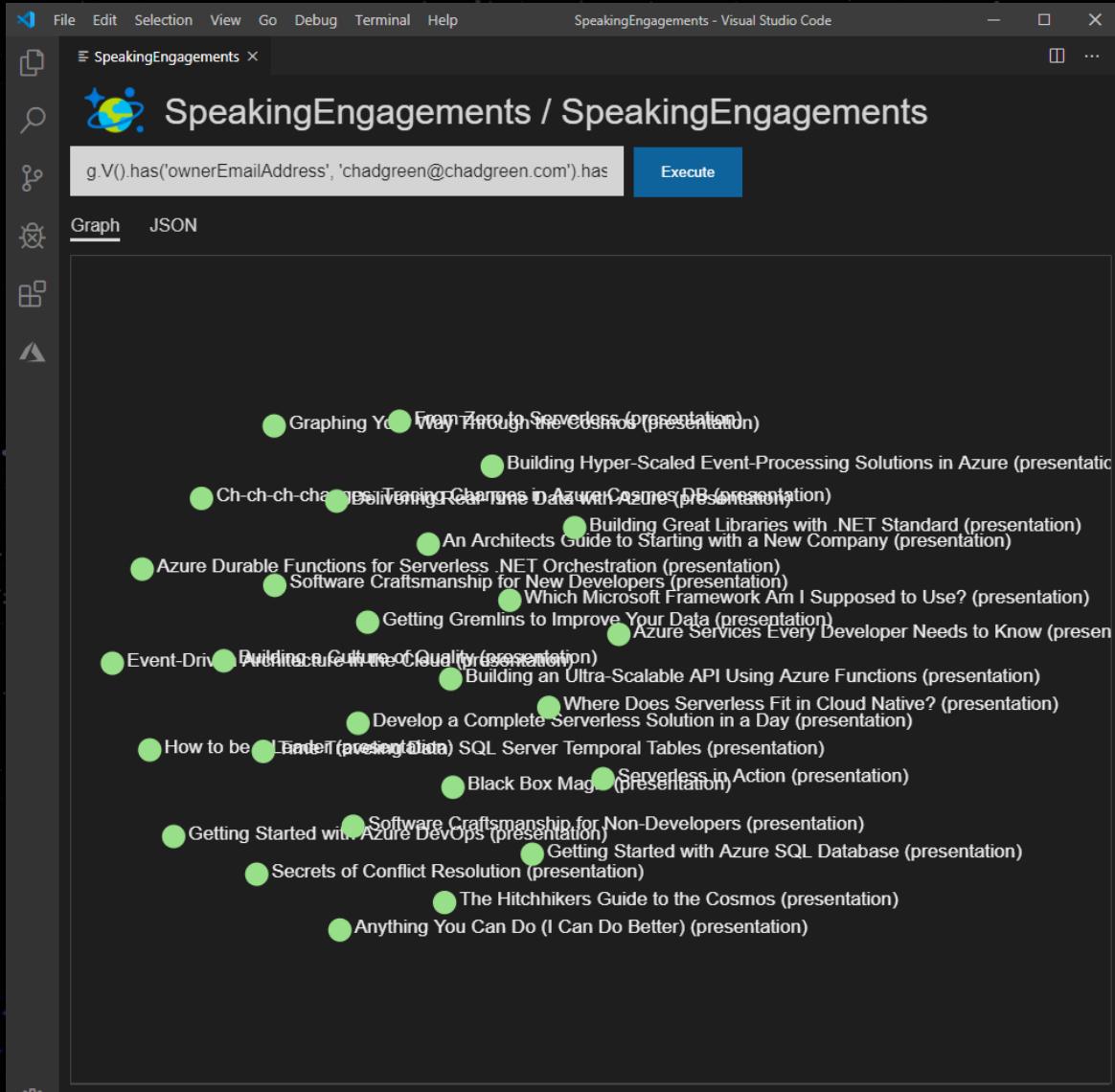


0 0 0 Azure: chadgreen@chadgreen.com

ay Through the Cosmos



# What presentations are in my repertoire?



A screenshot of Visual Studio Code showing a Gremlin query in the "SpeakingEngagements" workspace. The query is:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').hasLabel('presentation')
```

The results show 27 vertices, each represented by a green circle and a presentation title:

- Graphing Your Way Through the Cosmos (presentation)
- Building Hyper-Scaled Event-Processing Solutions in Azure (presentation)
- Ch-ch-ch-changes! Training Changes in Azure Cosmos DB (presentation)
- Building Great Libraries with .NET Standard (presentation)
- An Architects Guide to Starting with a New Company (presentation)
- Azure Durable Functions for Serverless .NET Orchestration (presentation)
- Software Craftsmanship for New Developers (presentation)
- Which Microsoft Framework Am I Supposed to Use? (presentation)
- Getting Gremlins to Improve Your Data (presentation)
- Azure Services Every Developer Needs to Know (presentation)
- Event-Driven Building a Culture of Quality (presentation)
- Building an Ultra-Scalable API Using Azure Functions (presentation)
- Where Does Serverless Fit in Cloud Native? (presentation)
- Develop a Complete Serverless Solution in a Day (presentation)
- How to be a Leader (presentation)
- SQL Server Temporal Tables (presentation)
- Black Box Magician (presentation)
- Serverless in Action (presentation)
- Getting Started with Azure DevOps (presentation)
- Software Craftsmanship for Non-Developers (presentation)
- Getting Started with Azure SQL Database (presentation)
- Secrets of Conflict Resolution (presentation)
- The Hitchhiker's Guide to the Cosmos (presentation)
- Anything You Can Do (I Can Do Better) (presentation)

At the bottom left, there are icons for GitHub, Twitter, LinkedIn, and a gear. The status bar at the bottom says "Displaying all 27 vertices and 0 edges".

g.V()  
.hasLabel('presentation')



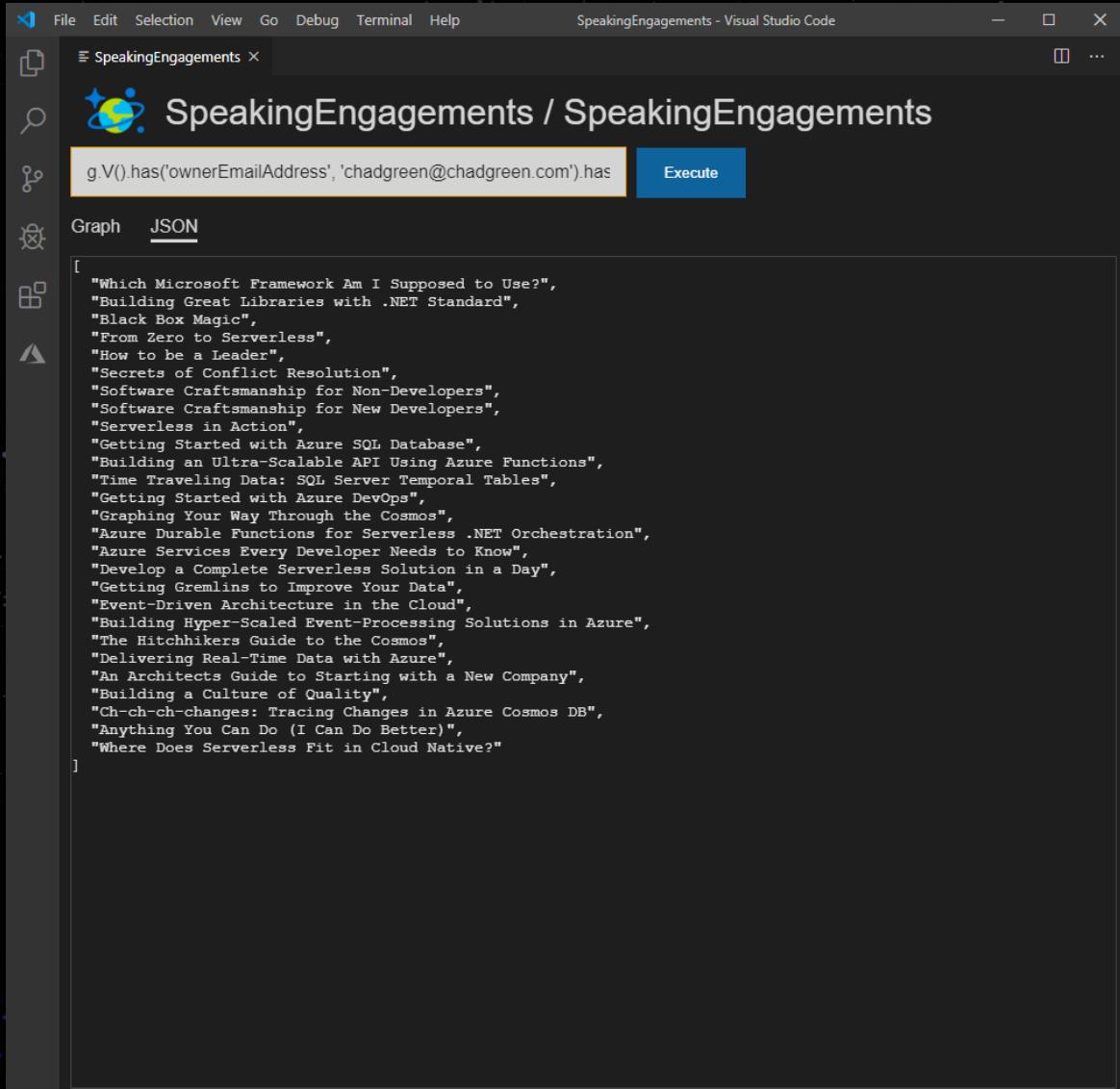
0 0 0 Azure: chadgreen@chadgreen.com



ay Through the Cosmos



# What presentations are in my repertoire?



A screenshot of Visual Studio Code showing a Gremlin query in the SpeakingEngagements workspace. The query is:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').has
```

The results are displayed in JSON format:

```
[  
    "Which Microsoft Framework Am I Supposed to Use?",  
    "Building Great Libraries with .NET Standard",  
    "Black Box Magic",  
    "From Zero to Serverless",  
    "How to be a Leader",  
    "Secrets of Conflict Resolution",  
    "Software Craftsmanship for Non-Developers",  
    "Software Craftsmanship for New Developers",  
    "Serverless in Action",  
    "Getting Started with Azure SQL Database",  
    "Building an Ultra-Scalable API Using Azure Functions",  
    "Time Traveling Data: SQL Server Temporal Tables",  
    "Getting Started with Azure DevOps",  
    "Graphing Your Way Through the Cosmos",  
    "Azure Durable Functions for Serverless .NET Orchestration",  
    "Azure Services Every Developer Needs to Know",  
    "Develop a Complete Serverless Solution in a Day",  
    "Getting Gremlins to Improve Your Data",  
    "Event-Driven Architecture in the Cloud",  
    "Building Hyper-Scaled Event-Processing Solutions in Azure",  
    "The Hitchhiker's Guide to the Cosmos",  
    "Delivering Real-Time Data with Azure",  
    "An Architects Guide to Starting with a New Company",  
    "Building a Culture of Quality",  
    "Ch-ch-ch-changes: Tracing Changes in Azure Cosmos DB",  
    "Anything You Can Do (I Can Do Better)",  
    "Where Does Serverless Fit in Cloud Native?"  
]
```

```
g.V()  
.hasLabel('presentation')  
.values('name')
```

# What are the events that I have submitted to?

The screenshot shows a Neo4j browser window titled "SpeakingEngagements - Visual Studio Code". The query entered is:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').hasLabel('event').has('year', '2020')
```

The results are displayed as a graph with orange circular nodes representing events. The nodes are labeled with their names and descriptions, such as "DevConf 2020 (event)", "Microsoft Ignite the Tour 2020 (Dubai) (event)", and "NDC {London} 2020 (event)". There are approximately 49 nodes shown.

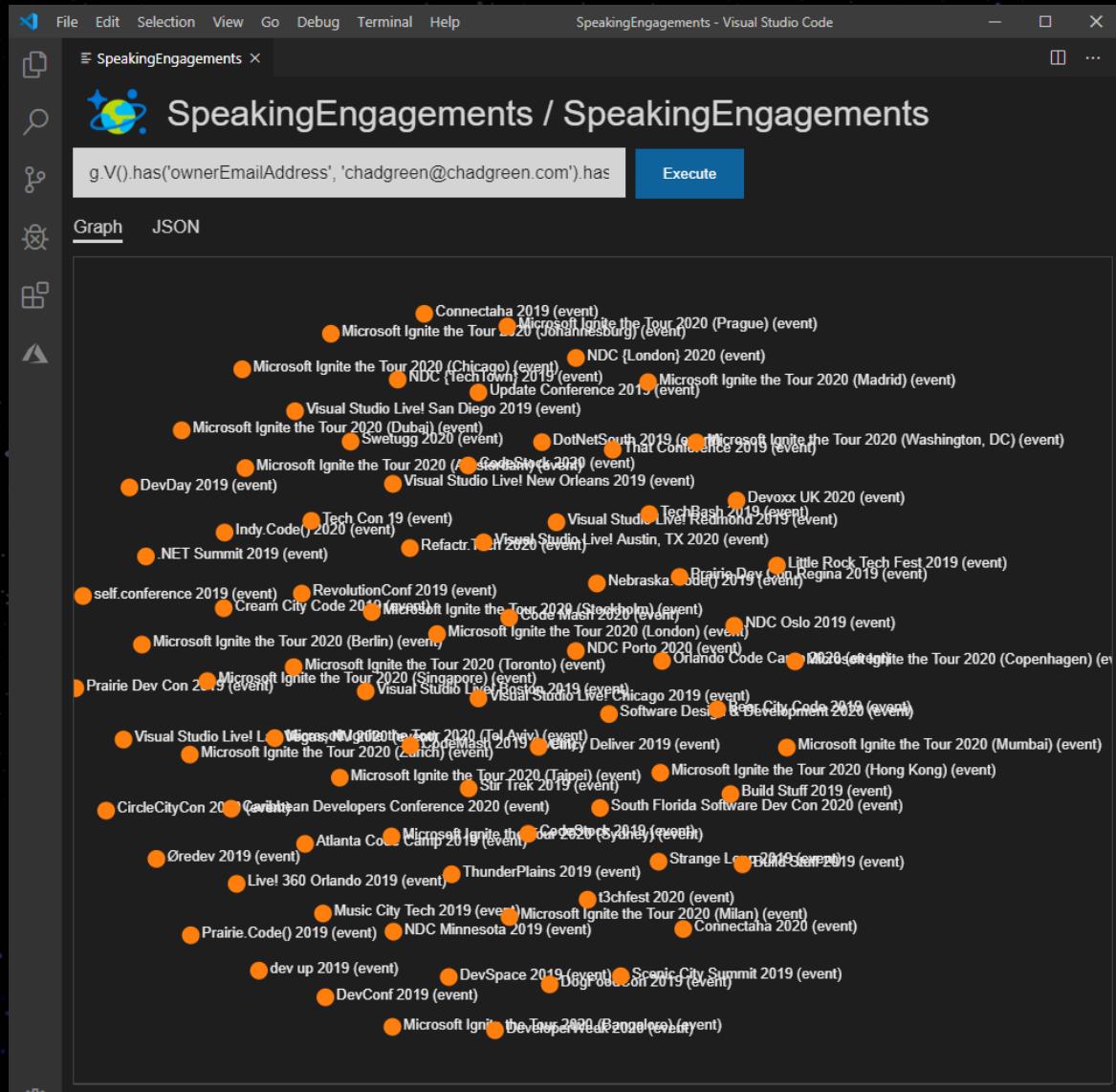
Graph JSON

```
[{"id": 1, "label": "DevConf 2020 (event)"}, {"id": 2, "label": "Microsoft Ignite the Tour 2020 (Milan) (event)"}, {"id": 3, "label": "NDC Porto 2020 (event)"}, {"id": 4, "label": "Techfest 2020 (event)"}, {"id": 5, "label": "Agile 2020 (event)"}, {"id": 6, "label": "NDC {London} 2020 (event)"}, {"id": 7, "label": "Indy.Code() 2020 (event)"}, {"id": 8, "label": "CircleCityCon 2020 (event)"}, {"id": 9, "label": "Microsoft Ignite the Tour 2020 (London) (event)"}, {"id": 10, "label": "Visual Studio Live! Redmond 2020 (event)"}, {"id": 11, "label": "Visual Studio Live! Na\u00f3o 2020 (event)"}, {"id": 12, "label": "Software Professionals Conference 2020 (event)"}, {"id": 13, "label": "Refactr.Tech 2020 (event)"}, {"id": 14, "label": "Microsoft Ignite the Tour 2020 (Paris) (event)"}, {"id": 15, "label": "Microsoft Ignite the Tour 2020 (Mumbai) (event)"}, {"id": 16, "label": "Microsoft Ignite the Tour 2020 (Berlin) (event)"}, {"id": 17, "label": "NDC Oslo 2020 (event)"}, {"id": 18, "label": "Microsoft Ignite the Tour 2020 (Madrid) (event)"}, {"id": 19, "label": "Microsoft Ignite the Tour 2020 (Amsterdam) (event)"}, {"id": 20, "label": "Microsoft Ignite the Tour 2020 (Copenhagen) (event)"}, {"id": 21, "label": "Microsoft Ignite the Tour 2020 (Singapore) (event)"}, {"id": 22, "label": "Caribbean Developers Conference 2020 (event)"}, {"id": 23, "label": "South Florida Software Dev Con 2020 (event)"}, {"id": 24, "label": "Microsoft Ignite the Tour 2020 (Sydney) (event)"}, {"id": 25, "label": "Microsoft Ignite the Tour 2020 (Stockholm) (event)"}, {"id": 26, "label": "Microsoft Ignite the Tour 2020 (Tel Aviv) (event)"}, {"id": 27, "label": "Connectaha 2020 (event)"}, {"id": 28, "label": "Visual Studio Live! San Diego 2020 (event)"}, {"id": 29, "label": "Microsoft Ignite the Tour 2020 (Madrid) (event)"}, {"id": 30, "label": "Microsoft Ignite the Tour 2020 (Tel Aviv) (event)"}, {"id": 31, "label": "MicroConf 2020 (event)"}, {"id": 32, "label": "DeveloperWeek 2020 (event)"}, {"id": 33, "label": "O'Reilly Software Architecture Conference 2020 (event)"}, {"id": 34, "label": "Microsoft Ignite the Tour 2020 (Hong Kong) (event)"}, {"id": 35, "label": "Devoxx UK 2020 (event)"}, {"id": 36, "label": "Visual Studio Live! Austin, TX 2020 (event)"}, {"id": 37, "label": "CodeStock 2020 (event)"}, {"id": 38, "label": "Visual Studio Live! Las Vegas, NV 2020 (event)"}, {"id": 39, "label": "Swetugg 2020 (event)"}, {"id": 40, "label": "MicroCPH 2020 (event)"}, {"id": 41, "label": "Microsoft Ignite the Tour 2020 (Chicago) (event)"}, {"id": 42, "label": "Microsoft Ignite the Tour 2020 (Washington, DC) (event)"}, {"id": 43, "label": "Microsoft Ignite the Tour 2020 (Taipei) (event)"}]
```

Displaying all 49 vertices and 0 edges

g.V()  
.hasLabel('event')  
.has('year', '2020')

# Where has *Graphing Your Way Through the Cosmos* been submitted to?



A screenshot of a Neo4j browser interface titled "SpeakingEngagements - Visual Studio Code". The query entered is `g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').has`. The results show a network of 81 vertices, each representing a speaking engagement. The nodes are orange circles with labels like "Connectaha 2019 (event)", "Microsoft Ignite the Tour 2020 (Prague) (event)", and "Microsoft Ignite the Tour 2020 (Milan) (event)". The graph is displayed in a grid-like structure.

g.V()  
.hasLabel('presentation')  
.has('name', 'Graphing Your  
Way Through the Cosmos')  
.outE('submittedTo')  
.inV()



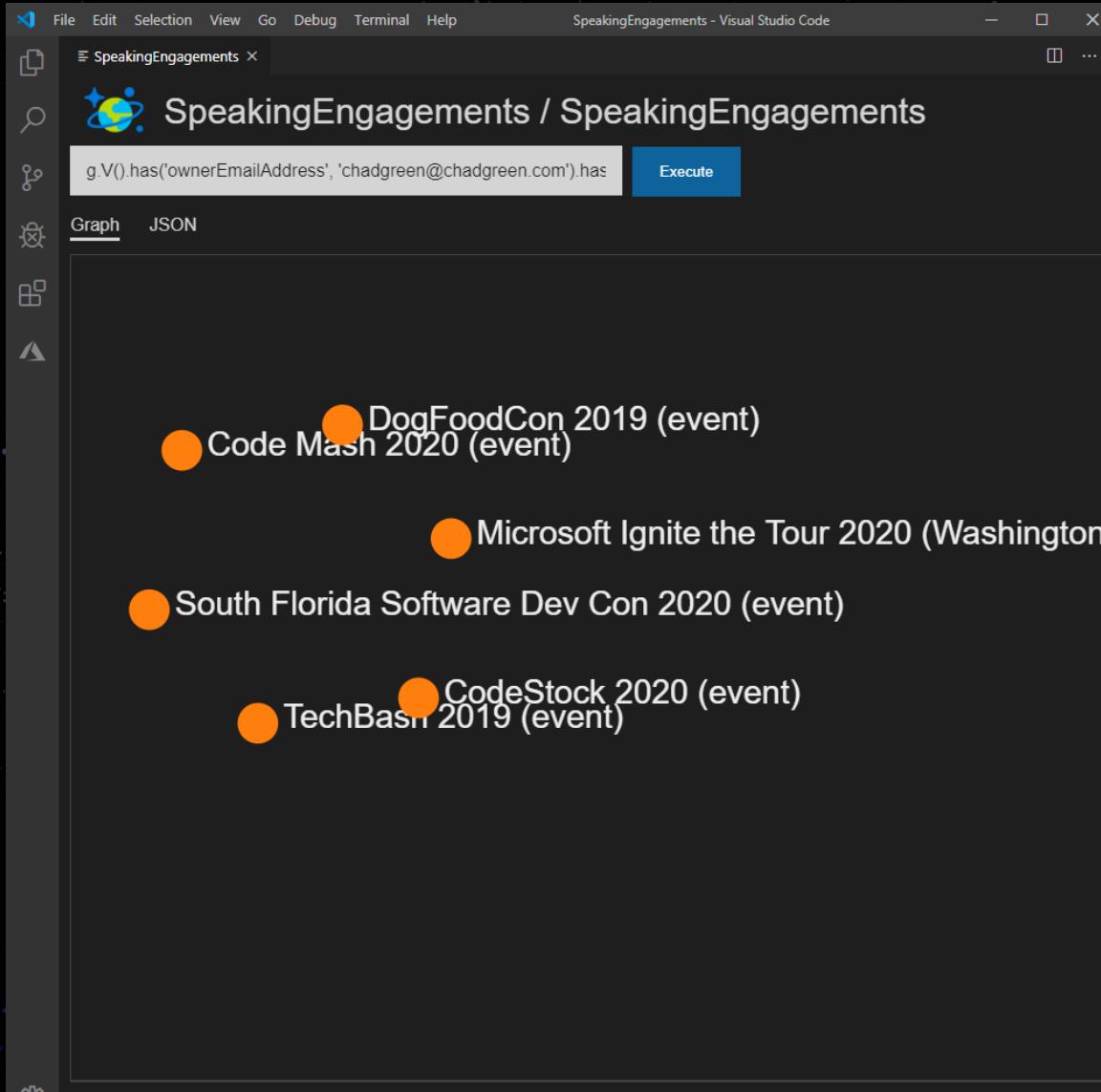
Displaying all 81 vertices and 0 edges



CHADGREEN

Graphing Your Way Through the Cosmos

# Where has *Graphing Your Way Through the Cosmos* been scheduled?



A screenshot of the Visual Studio Code interface, specifically the SpeakingEngagements extension. The title bar says "SpeakingEngagements - Visual Studio Code". The main area shows a graph visualization with several orange circular nodes representing events. The nodes are labeled with their names and descriptions: "DogFoodCon 2019 (event)", "Code Mash 2020 (event)", "Microsoft Ignite the Tour 2020 (Washington)", "South Florida Software Dev Con 2020 (event)", "TechBash 2019 (event)", and "CodeStock 2020 (event)". Below the graph, a code editor window displays the following Cypher query:

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').has
```

The "Execute" button is highlighted in blue.

```
g.V()  
.hasLabel('presentation')  
.has('name', 'Graphing Your  
Way Through the Cosmos')  
.outE('submittedTo')  
.has('status', 'Confirmed')  
.inV()
```



Displaying all 6 vertices and 0 edges

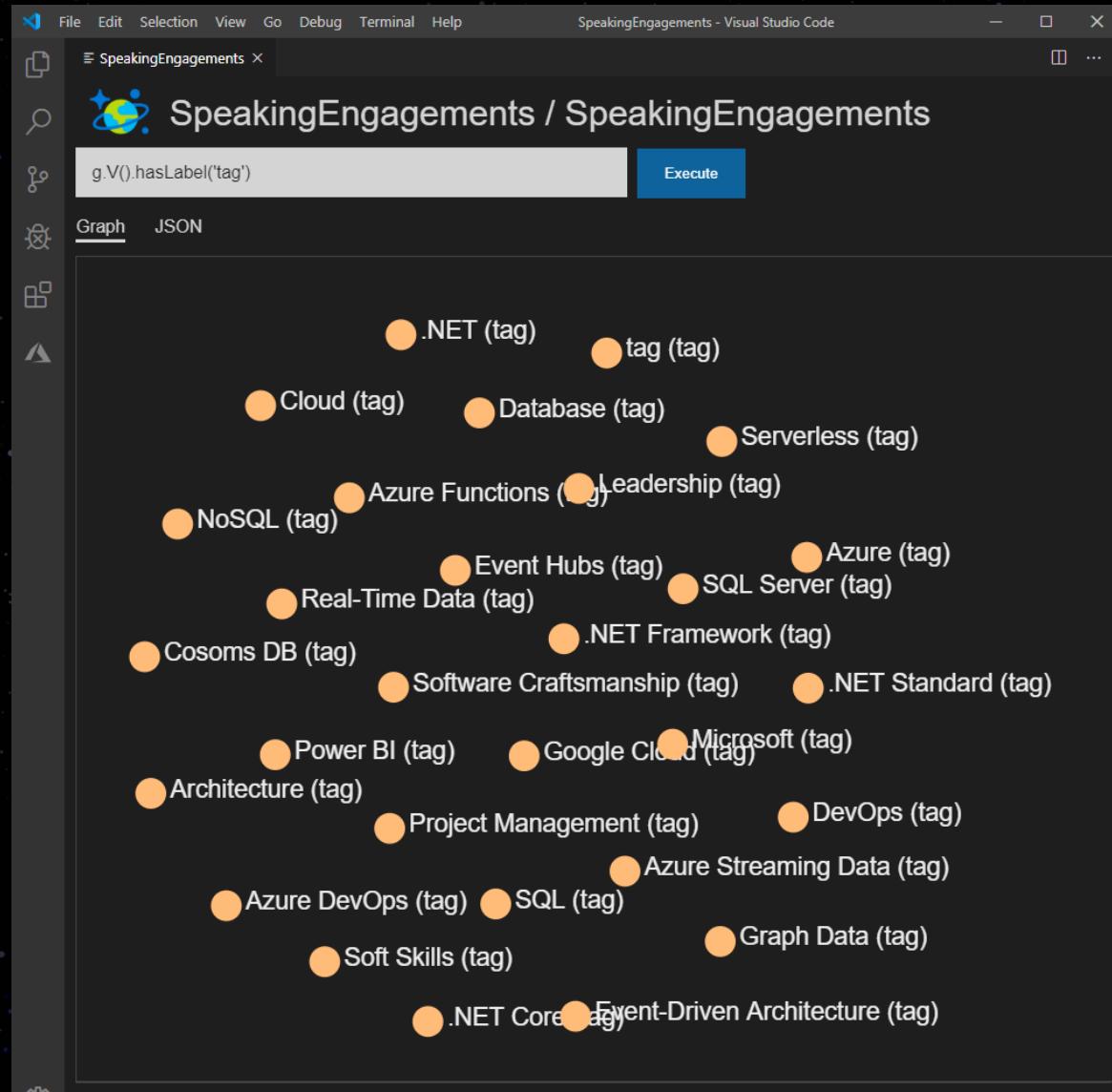


Azure: chadgreen@chadgreen.com

ay Through the Cosmos



# View all of the tags



The screenshot shows the Neo4j browser integrated into Visual Studio Code. The title bar reads "SpeakingEngagements - Visual Studio Code". The main interface displays a graph of 29 tags, each represented by an orange circle. The tags are:

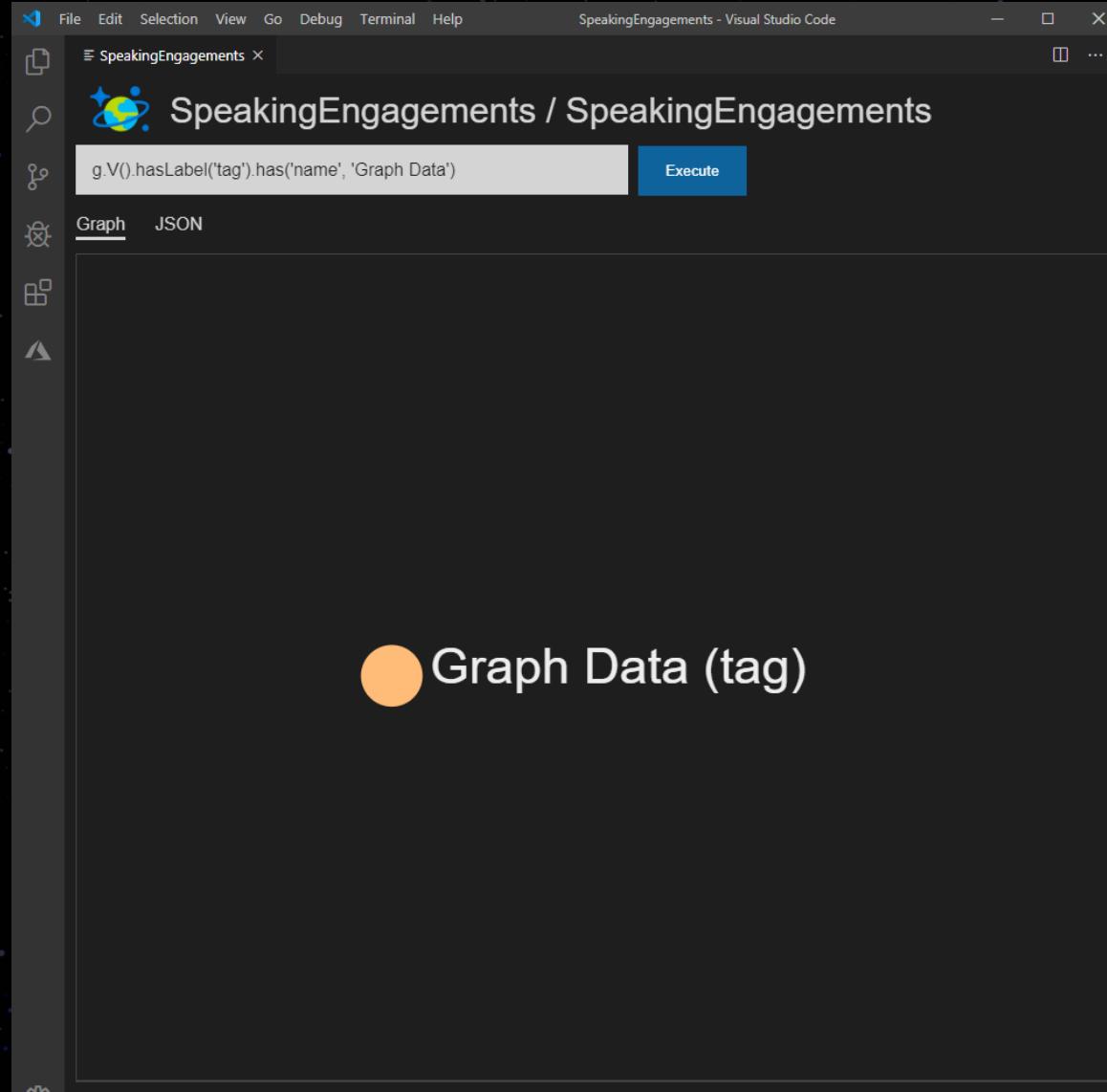
- .NET (tag)
- tag (tag)
- Cloud (tag)
- Database (tag)
- Serverless (tag)
- NoSQL (tag)
- Azure Functions (tag)
- Leadership (tag)
- Real-Time Data (tag)
- Event Hubs (tag)
- Azure (tag)
- SQL Server (tag)
- Cosmos DB (tag)
- .NET Framework (tag)
- Software Craftsmanship (tag)
- .NET Standard (tag)
- Power BI (tag)
- Google Cloud (tag)
- Microsoft (tag)
- Architecture (tag)
- Project Management (tag)
- DevOps (tag)
- Azure DevOps (tag)
- SQL (tag)
- Soft Skills (tag)
- Graph Data (tag)
- .NET Core (tag)
- Event-Driven Architecture (tag)

At the bottom left, it says "Displaying all 29 vertices and 0 edges".

g.V()  
.hasLabel('tag')



# Focus on the *Graph Data* tag



A screenshot of the Visual Studio Code interface, specifically the Neo4j extension. The title bar says "SpeakingEngagements - Visual Studio Code". The main area shows a query in the Neo4j Cypher language:

```
g.V().hasLabel('tag').has('name', 'Graph Data')
```

Below the query is a "Execute" button. To the right of the button are two tabs: "Graph" (which is selected) and "JSON". The results pane below the tabs is empty, displaying the message "Displaying all 1 vertices and 0 edges".

g.V()  
.hasLabel('tag')  
.has('name', 'Graph Data')



# What presentations are tagged with *Graph Data*?



A screenshot of Visual Studio Code showing a Gremlin query in the terminal. The terminal title is "SpeakingEngagements - Visual Studio Code". The query is:

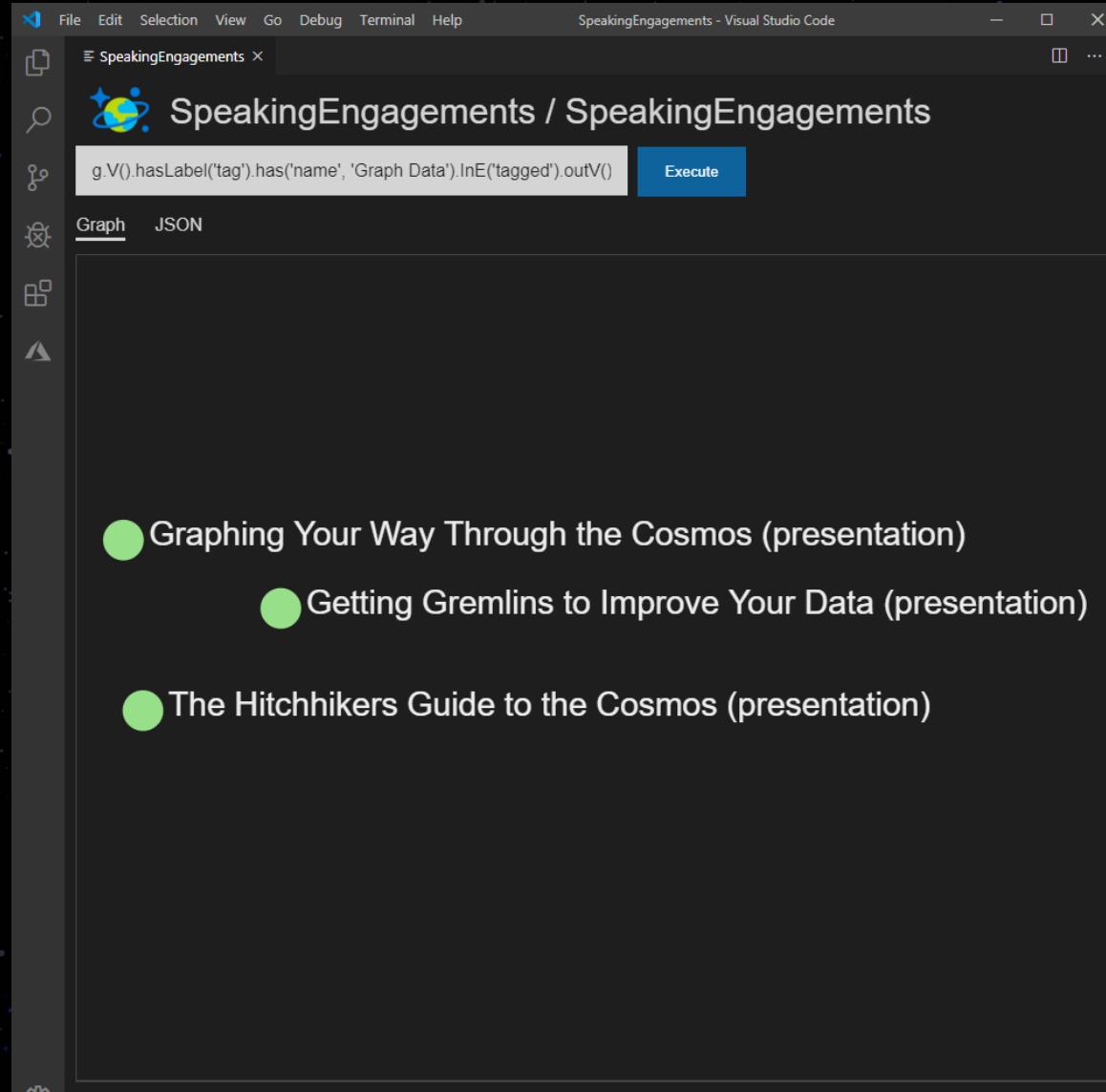
```
g.V().hasLabel('tag').has('name', 'Graph Data').inE('tagged')
```

The results are displayed in JSON format:

```
[{"id": "f4d6f410-6bfe-42b1-b3bb-a8035c6c49ec", "label": "tagged", "type": "edge", "inVLabel": "tag", "outVLabel": "presentation", "inV": "59624390-d2bc-4555-84cf-62185e2263eb", "outV": "60e69af2-0b2e-43ab-8b79-76482ae10b7b"}, {"id": "3fe31c1a-f83e-424f-84a5-2a7dc57a2e83", "label": "tagged", "type": "edge", "inVLabel": "tag", "outVLabel": "presentation", "inV": "59624390-d2bc-4555-84cf-62185e2263eb", "outV": "8f58f257-fba1-4fe2-9dfe-5ecb2e7b6414"}, {"id": "9aldbc06-01c9-49f3-87cf-0ad356ed21d5", "label": "tagged", "type": "edge", "inVLabel": "tag", "outVLabel": "presentation", "inV": "59624390-d2bc-4555-84cf-62185e2263eb", "outV": "c01b8bad-669c-4075-9046-17fa6fc3840d"}]
```

g.V()  
.hasLabel('tag')  
.has('name', 'Graph Data')  
.inE('tagged')

# What presentations are tagged with *Graph Data*?



A screenshot of a Visual Studio Code window titled "SpeakingEngagements - Visual Studio Code". The window shows a Gremlin query in the editor:

```
g.V().hasLabel('tag').has('name', 'Graph Data').inE('tagged').outV()
```

The "Execute" button is highlighted. Below the editor, there are two tabs: "Graph" and "JSON". The "Graph" tab is selected, displaying a hierarchical tree structure of presentations:

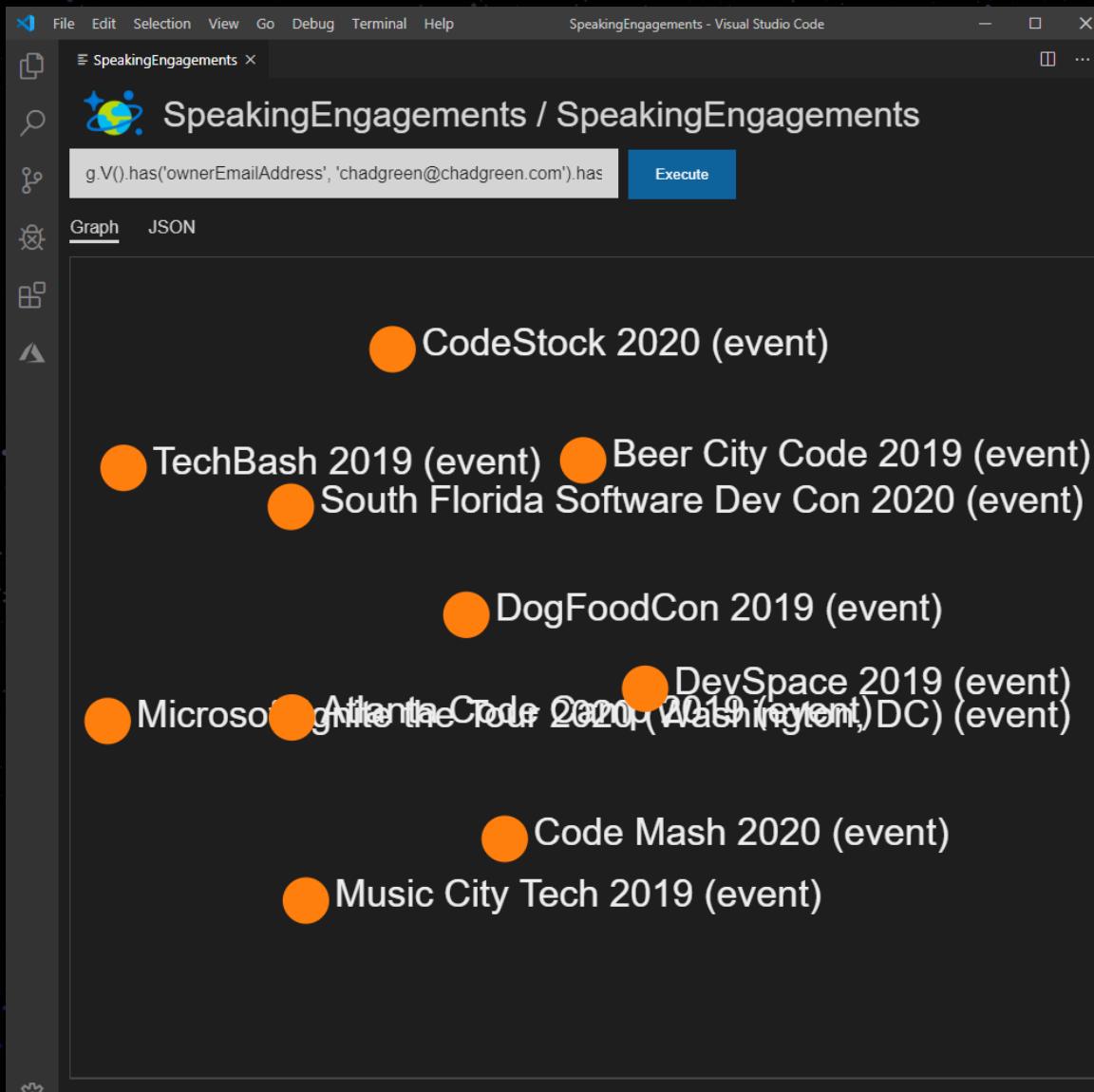
- Graphing Your Way Through the Cosmos (presentation)
  - Getting Gremlins to Improve Your Data (presentation)
- The Hitchhikers Guide to the Cosmos (presentation)

At the bottom of the code editor, it says "Displaying all 3 vertices and 0 edges".

g.V()  
.hasLabel('tag')  
.has('name', 'Graph Data')  
.inE('tagged')  
.outV()



# Where have presentations tagged *Graph Data* been scheduled?

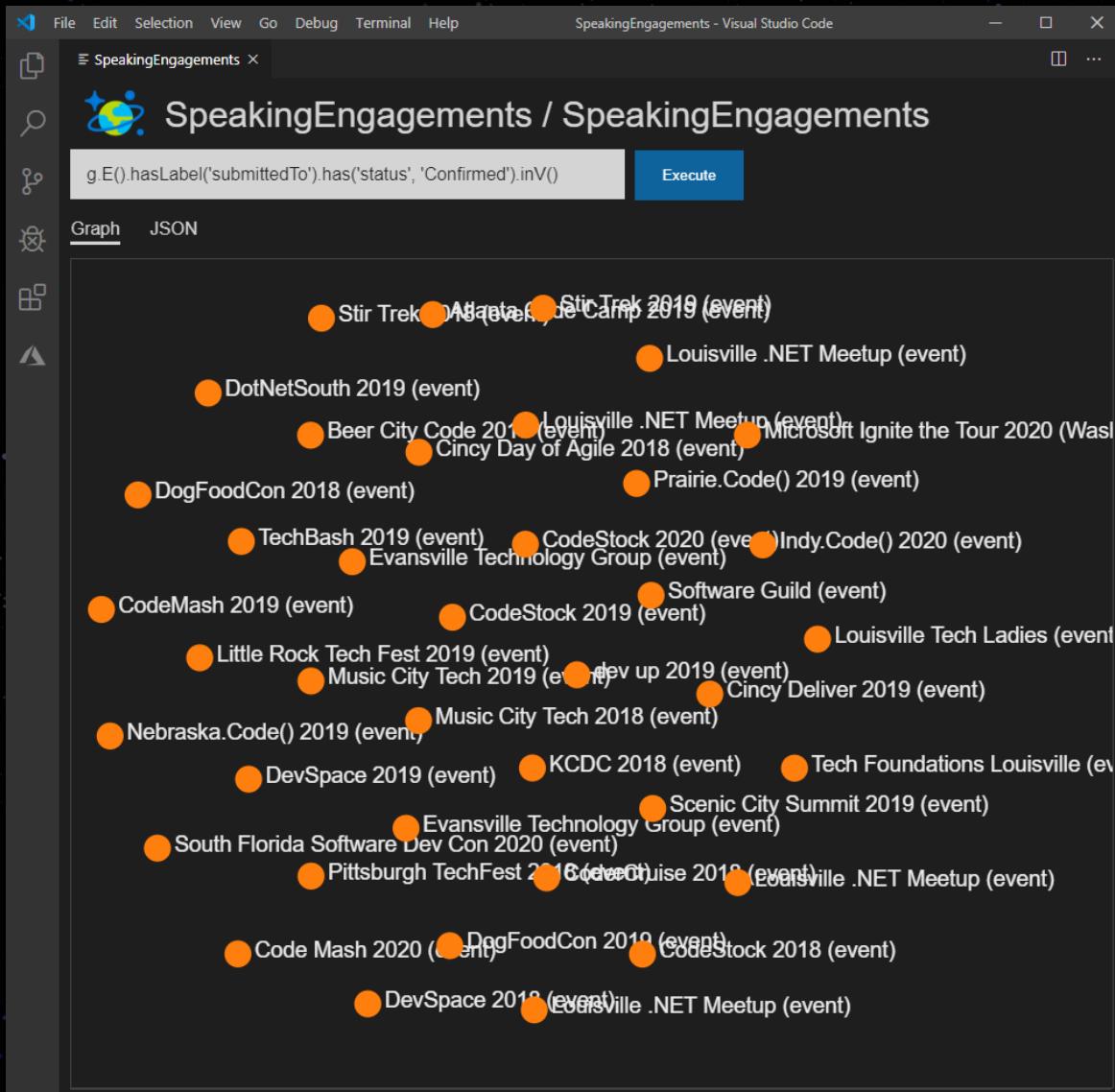


A screenshot of a Visual Studio Code window titled "SpeakingEngagements - Visual Studio Code". The workspace is named "SpeakingEngagements". In the center, there is a graph visualization showing several orange circular nodes representing events. The nodes are labeled with their names and descriptions, such as "CodeStock 2020 (event)", "TechBash 2019 (event)", "Beer City Code 2019 (event)", "South Florida Software Dev Con 2020 (event)", "DogFoodCon 2019 (event)", "DevSpace 2019 (event)", "Microsoft Atlanta Code 2020 (event)", "Atlanta Code 2019 (event)", "Washington DC (event)", "Code Mash 2020 (event)", and "Music City Tech 2019 (event)". Below the graph, the status bar displays "Displaying all 10 vertices and 0 edges". At the bottom left, there are icons for GitHub, Twitter, and LinkedIn, along with the text "Azure: chadgreen@chadgreen.com".

```
g.V().has('ownerEmailAddress', 'chadgreen@chadgreen.com').has
```

```
g.V()  
.hasLabel('tag')  
.has('name', 'Graph Data')  
.inE('tagged')  
.outV()  
.outE('submittedTo')  
.has('status', 'Confirmed')  
.inV()
```

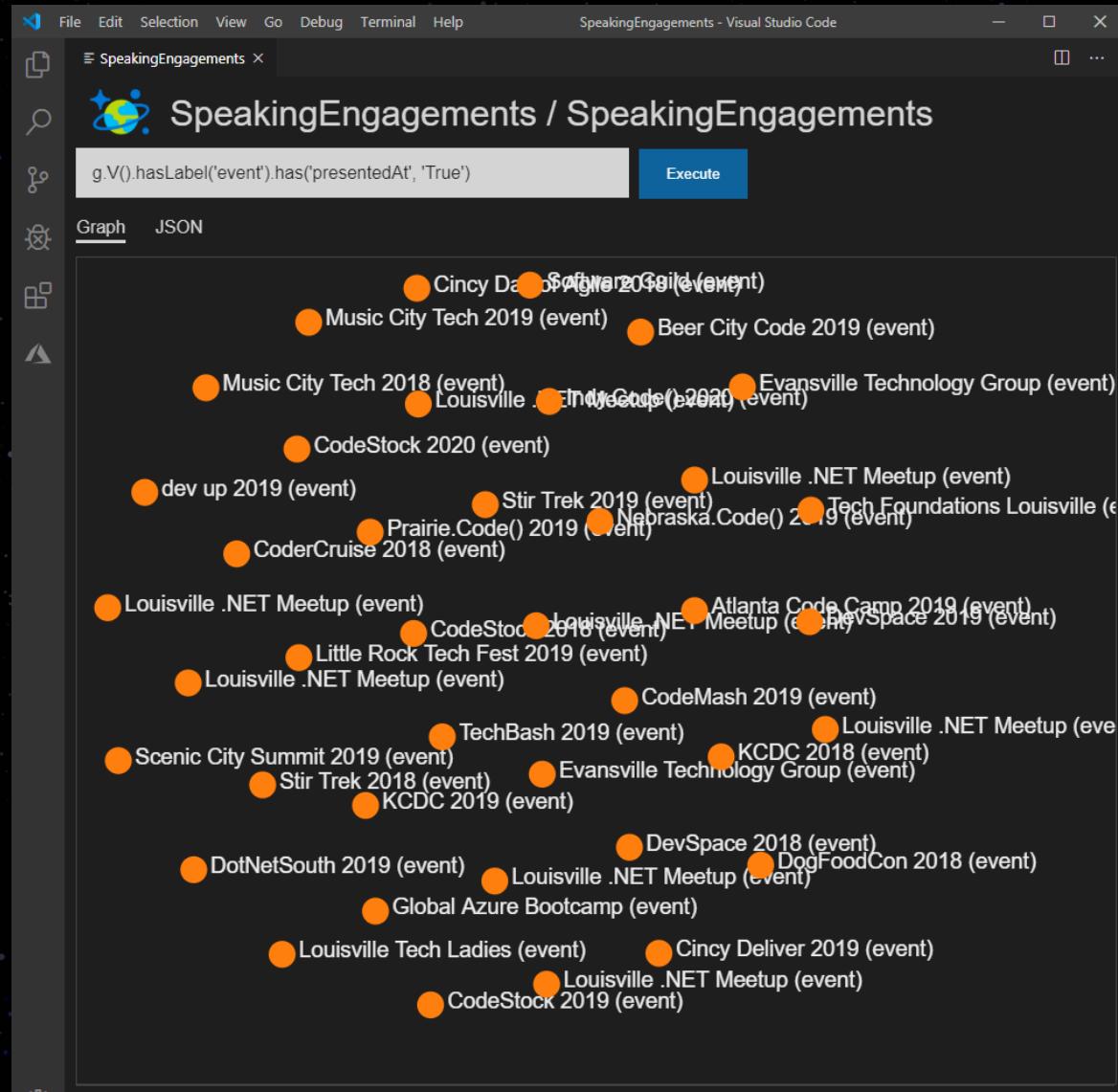
# What events have I been scheduled for?



```
g.E()  
.hasLabel('submittedTo')  
.has('status', 'Confirmed')  
.inV()
```



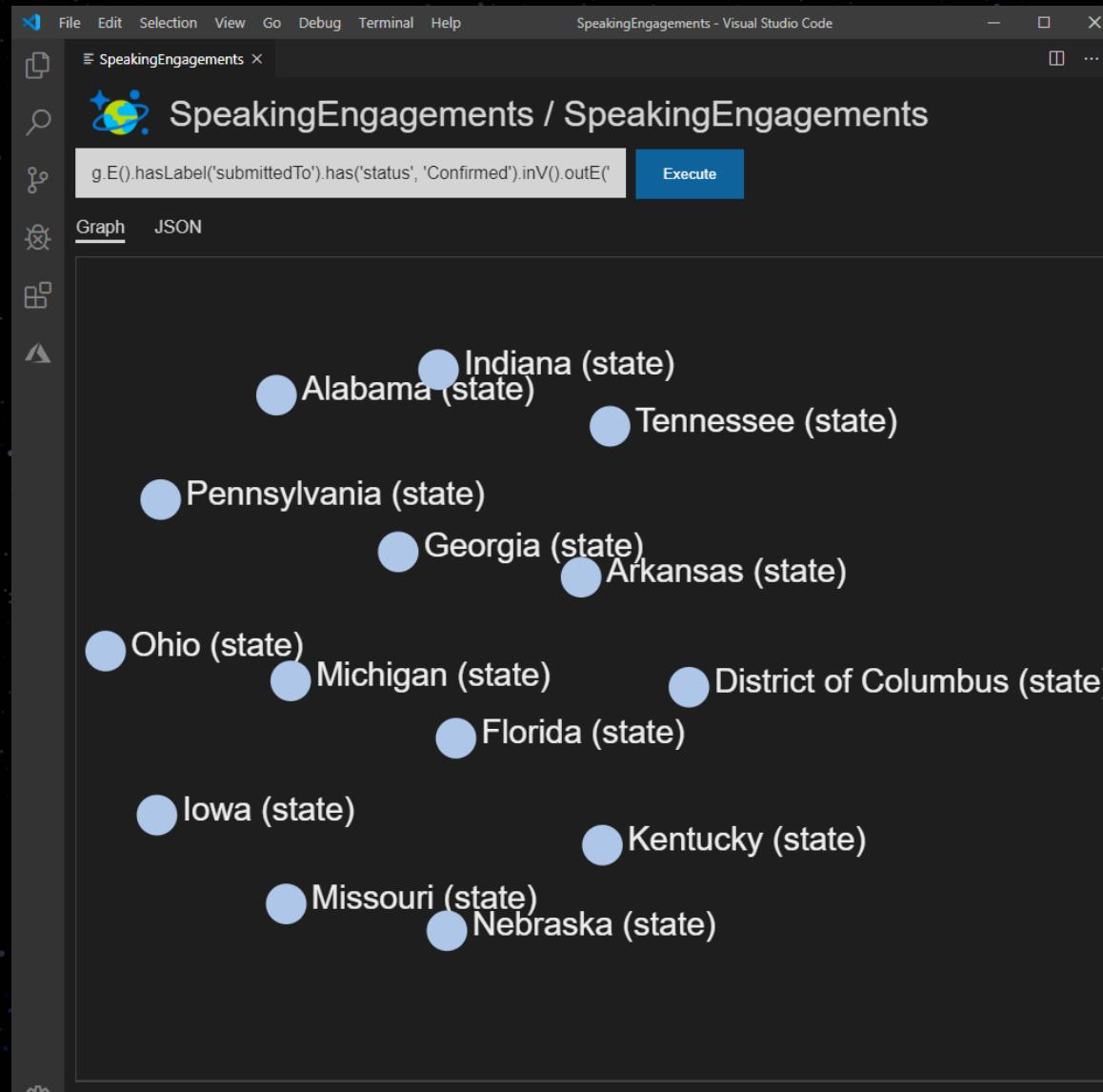
# What events have I been scheduled for?



g.E()  
.hasLabel('submittedTo')  
.has('status', 'Confirmed')  
.inV()  
  
g.V()  
.hasLabel('event')  
.has('presentedAt', 'True')



# What states have I been scheduled in?



g.E()  
.hasLabel('submittedTo')  
.has('status', 'Confirmed')  
.inV()  
.outE('stateLocation')  
.inV()





# Wrapping Up

- Graphs – set of objects in which pairs are in some sense related
- Graph Theory – Starts with the 7 bridges of Königsberg
- Graph databases – use graph structure to represent and store data
- Azure Cosmos DB – globally distributed, multi-model database service
- Graph vs Relational – lots of benefits that make graph database worth a look
- Graph Traversal – Navigating graph data using patterns



# Thank You!

Graphing Your Way Through the Cosmos

✉️ [chadgreen@chadgreen.com](mailto:chadgreen@chadgreen.com)

.twitch [TaleLearnCode](#)

🌐 [TaleLearnCode.com](#)

🐦 [ChadGreen & TaleLearnCode](#)

linkedin [ChadwickEGreen](#)