Getting Gremlins to Improve Your Data

Chad Green
Beer City Code
May 31, 2019

THANKS TO ALL OUR SPONSORS!

BizStream™

THE C2 GROUP

<aptera>

KL&A
KUNZ, LEIGH AND ASSOCIATES

KFORCE®

SpinDance

OST | OpenDigital

ATOMIC OBJECT

TEKsystems®

Progress®

MAGIC WRIGHTER INC.

SALESPAD™

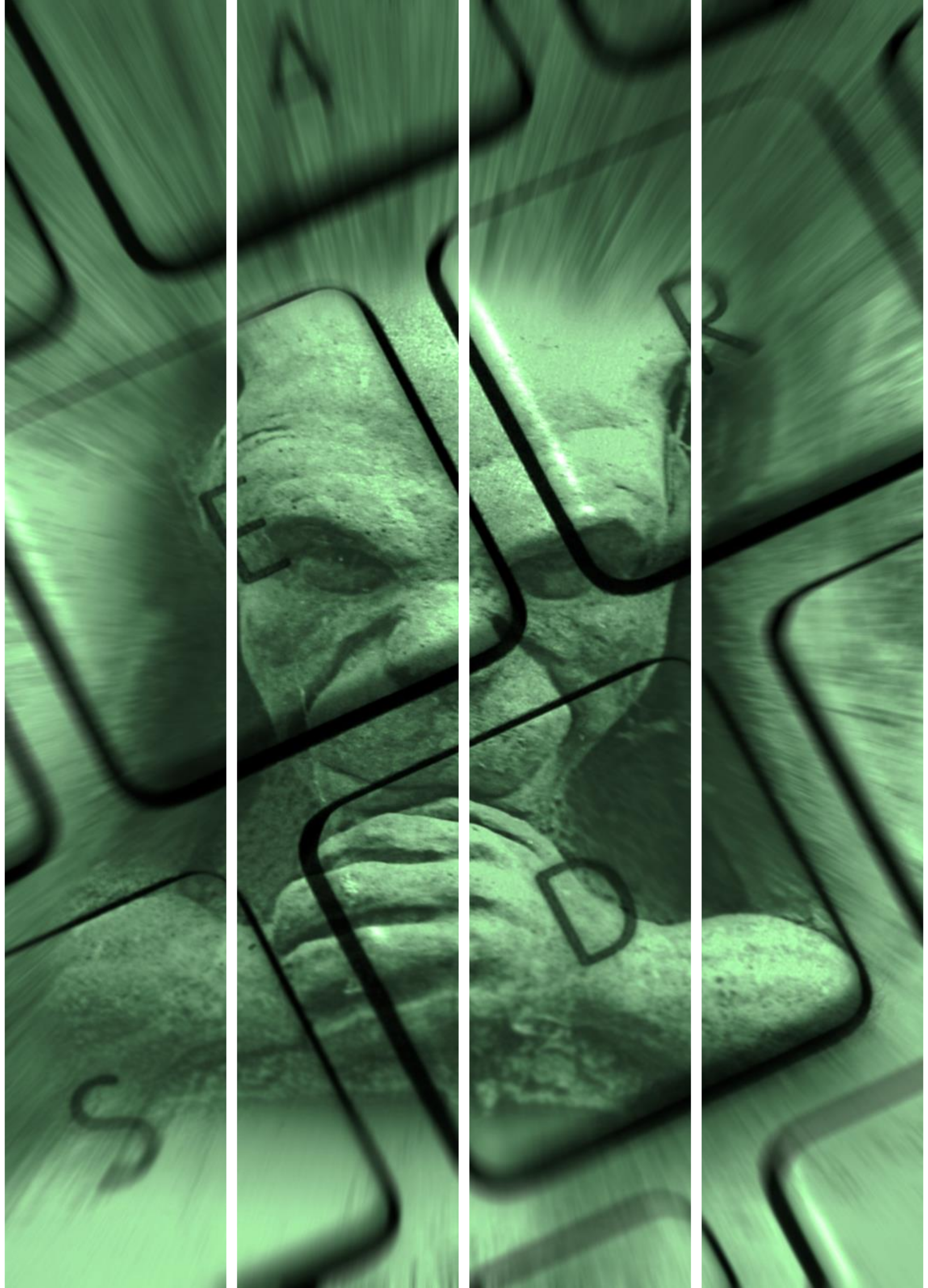
Microsoft

{M} MICHIGAN SOFTWARE LABS

Collective Idea

Introductions

Getting Gremlins to Improve Your Data



Workshop Overview

Getting Gremlins to Improve Your Data

1 **Introductions**
Learn what we will be covering today.

2 **What are Graph Databases**
Introduction to graph databases

3 **Introduction to Gremlin**
Find out what Gremlin is

4 **Introduction to Cosmos DB**
What is Azure Cosmos DB.

5 **Gremlin in Cosmos DB**
How has Cosmos DB implemented Gremlin

6 **Hands-On Lab**
Build a website that allows you to search for flights between two points

7 **Graph Partitioning**
Getting the full power of Cosmos in your graph databases

8 **Wrap Up**
Wrap up everything that was discussed over the day

Who is Chad Green

Director of Software Development
ScholarRx

chadgreen@chadgreen.com

chadgreen.com

ChadGreen

ChadwickEGreen

Workshop Logistics

Getting Gremlins to Improve Your Data

- Coffee/Restroom Breaks
- Lunch
- Wi-Fi

Workshop Prerequisites

Getting Gremlins to Improve Your Data

• Microsoft Account	https://account.microsoft.com
• Visual Studio 2019	https://visualstudio.com/downloads
• Azure Cosmos DB Emulator	https://aka.ms/cosmosdb-emulator
• Gremlin Console	https://tinkerpop.apache.org

What are Graph Databases

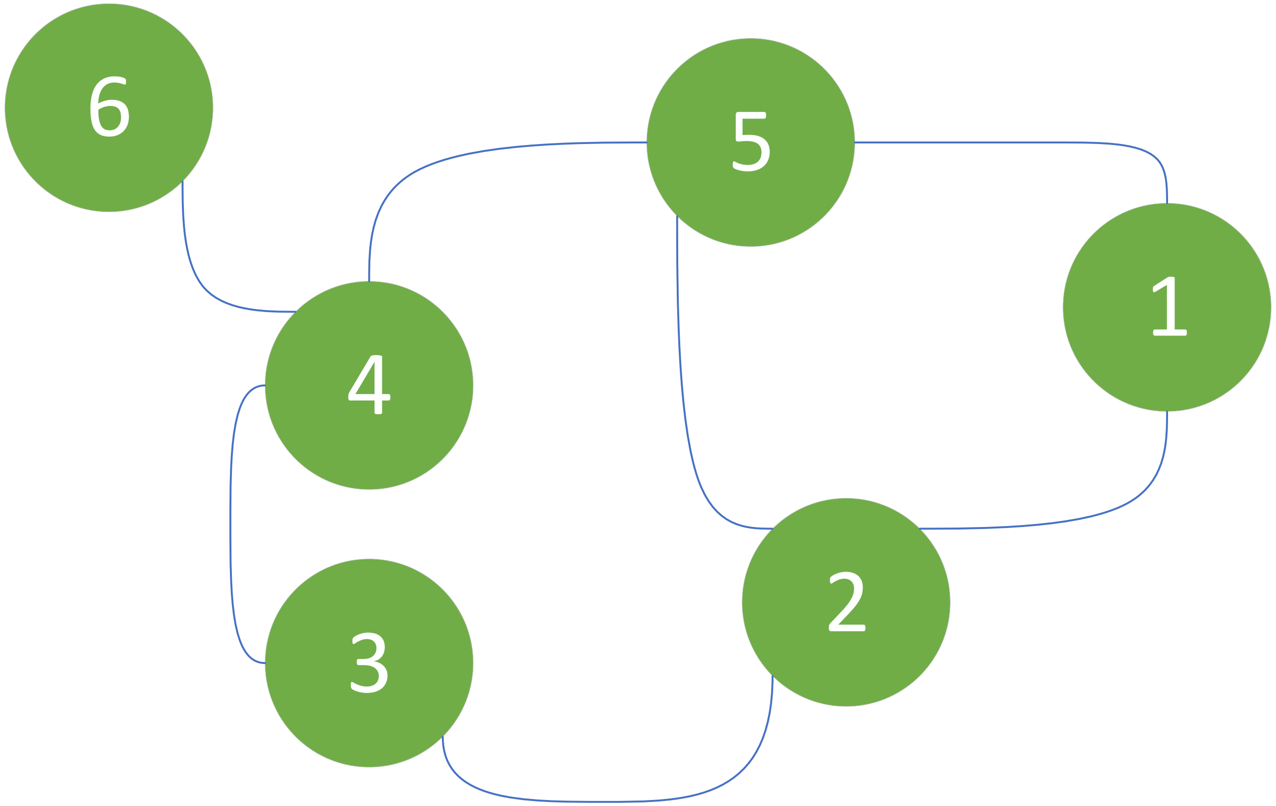
Getting Gremlins to Improve Your Data

What is a Graph

- Discrete mathematics
- Structure amounting to a set of objects in which some pairs of the objects are in some sense related
- Objects correspond to mathematical abstractions called vertices and each of the related pairs of vertices is called an edge
- Graph Theory is the study of graphs

What is a Graph

- Depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges

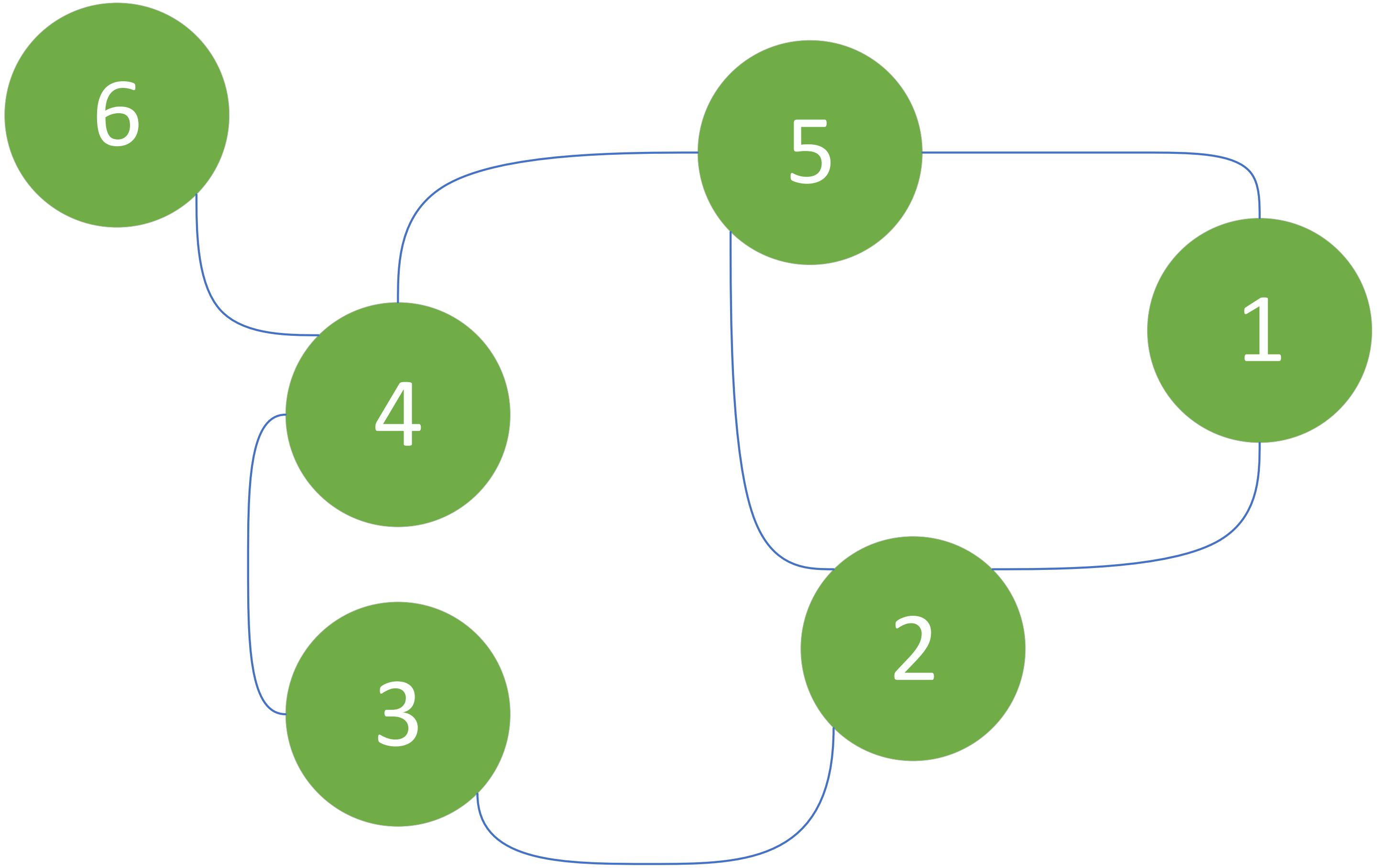


What is a Graph

$$G = (V, E)$$

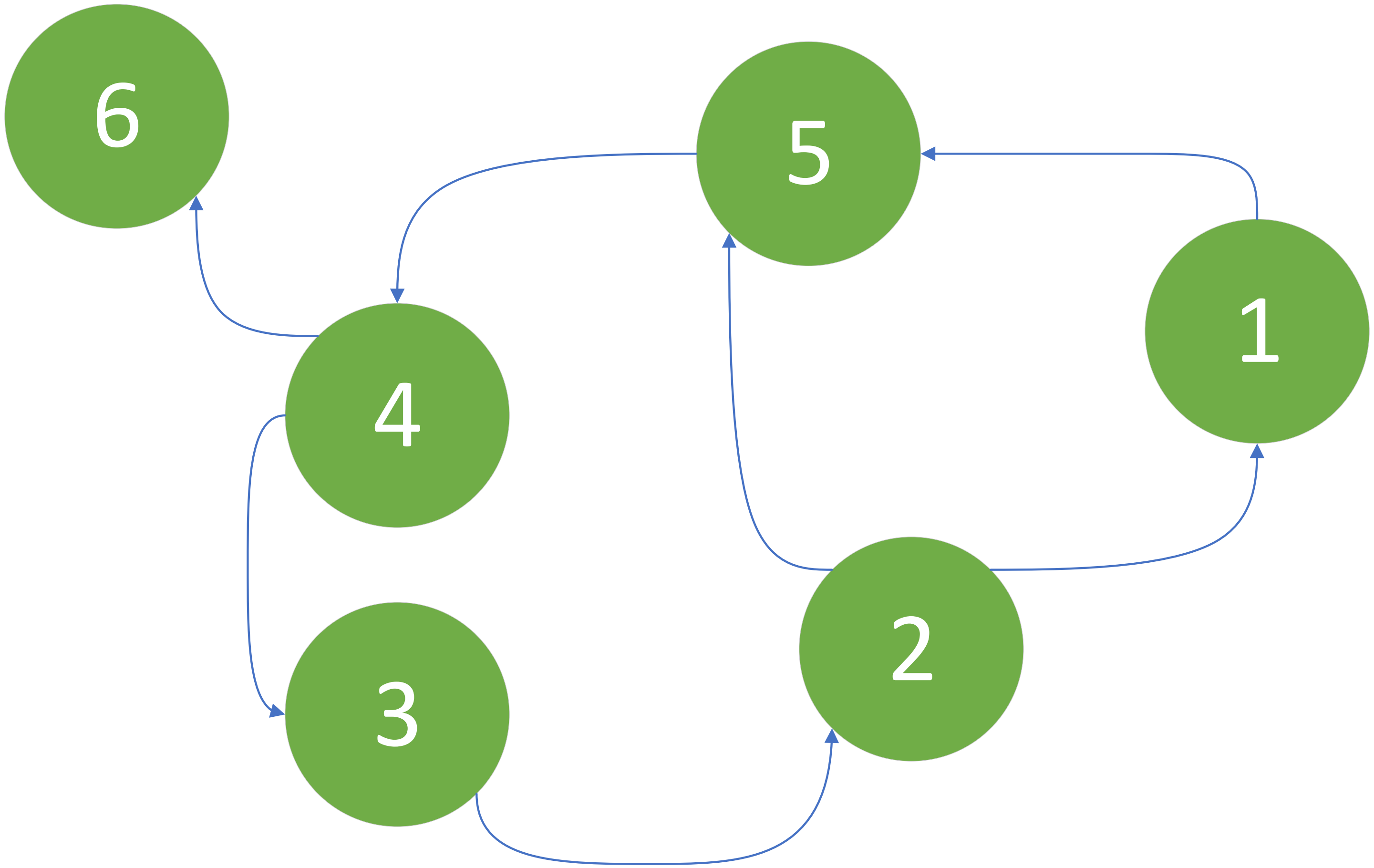
What is a Graph

Undirected



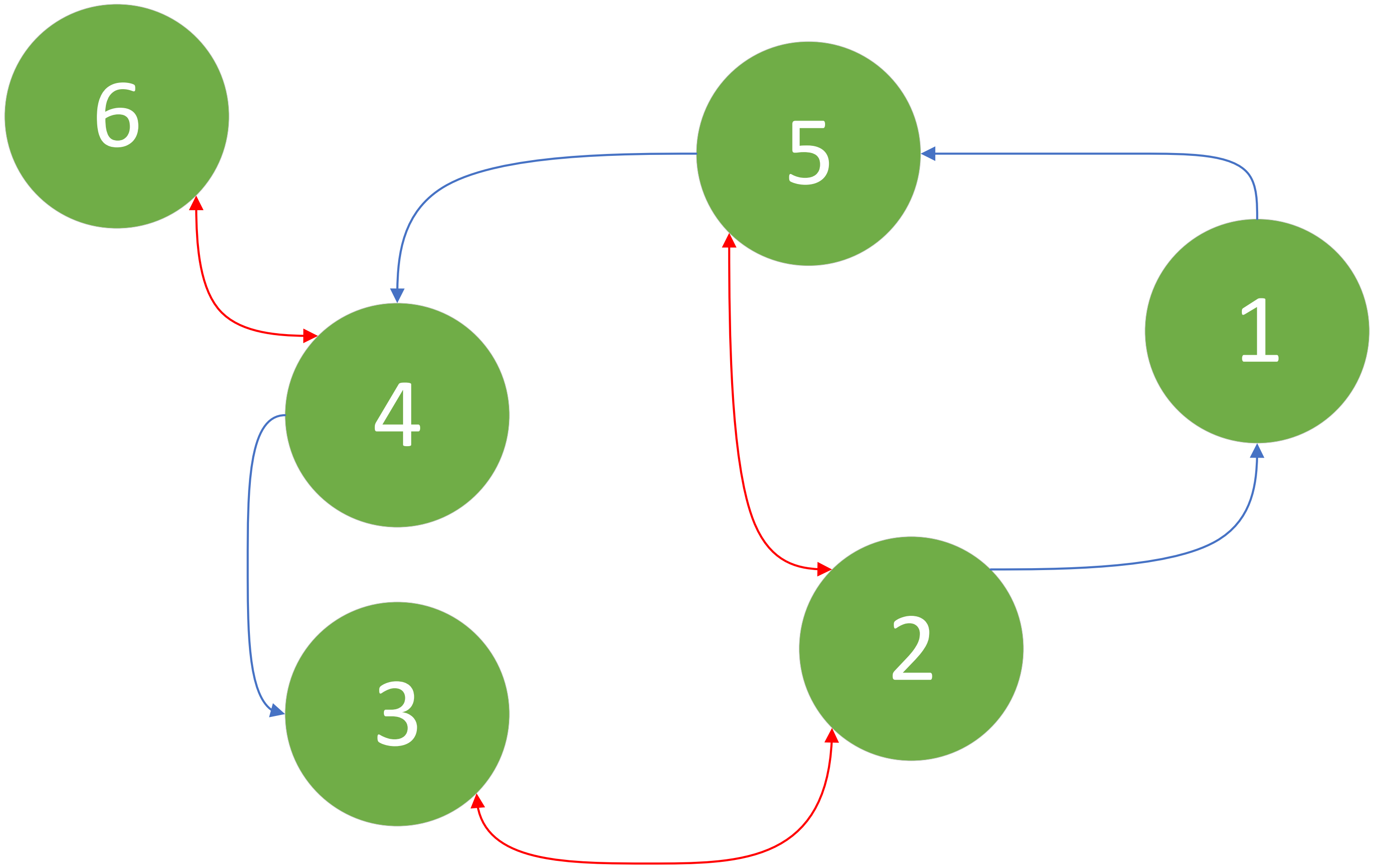
What is a Graph

Directed



What is a Graph

Multidirectional



What is a Graph Theory

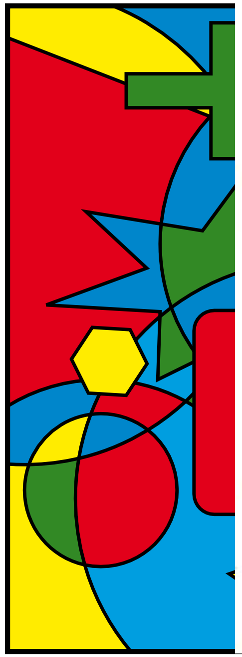
- Study of graphs

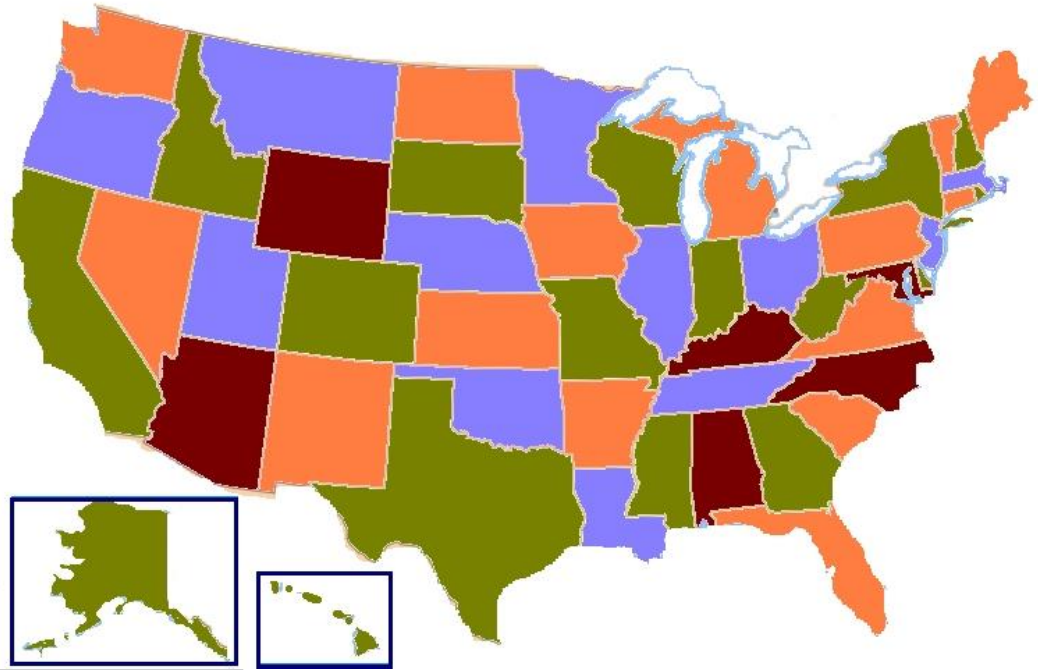
History of Graph Theory

- Alexandre-Théophile Vandermonde publishes paper on the knight problem
- Augustin-Louis Caunchy & Simon Antoine Jean L’Huilier used Euler’s formula to begin topology
- Term “graph” introduced in 1878 by James Joseph Sylvester
- First textbox on graph theory written by Dénes Kőnig in 1936
- In 1969, Frank Harary publishes the “definitive textbook on the subject"

History of Graph Theory

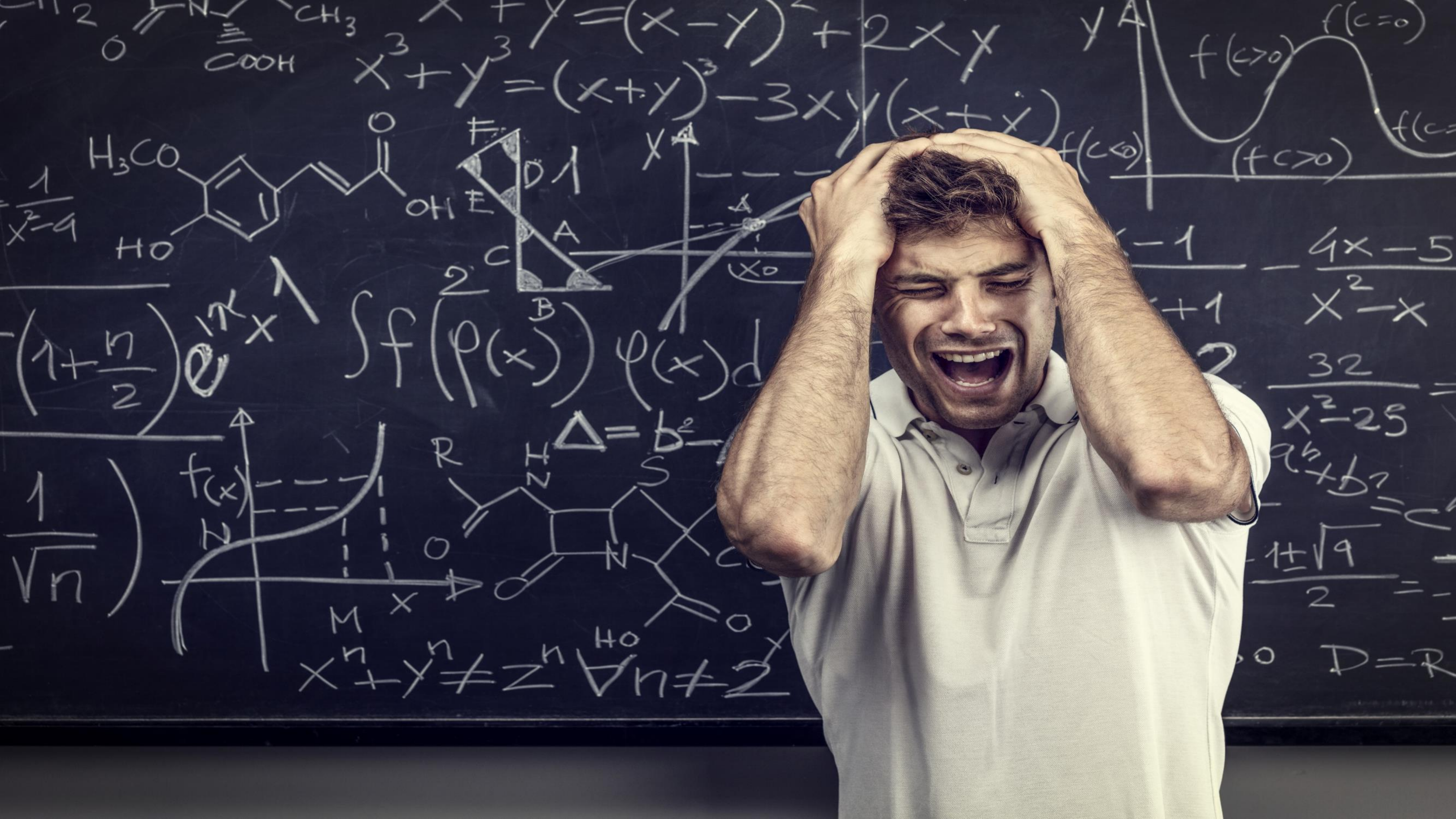
- Four color problem posed by Francis Guthrie in 1852; Heinrich Heesch published method for solving in 1969 using computers; computer-aided proof produced in 1976 by Kenneth Appel and Wolfgang Haken





Applications of Graph Theory

- Linguistics
- Physics and Chemistry
- Social Sciences
- Biology
- Computer Science

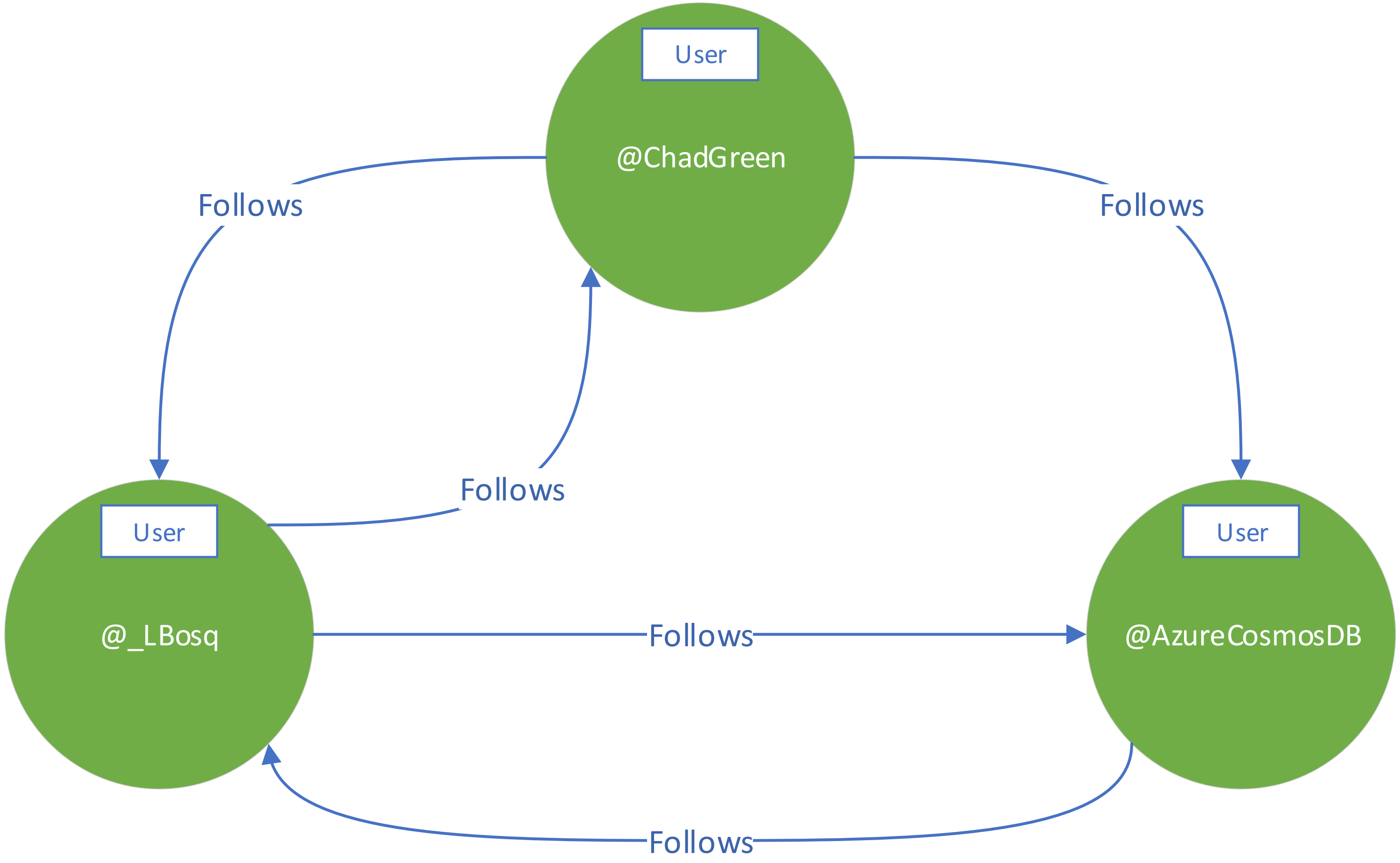


COOH
x3+y3=(x+y)3−3xy(x+y)
H3CO
OH
HO
Δ=b2
f(x)
x
M
HO
xn+yn≠zn ∀n≠2

What is a Graph

- Collection of *vertices* and *edges*
- Represent entities as vertices and the ways in which those entities relate to the world as relationships
- Allow us to model all kinds of scenarios

What is a Graph



What is a Graph Database

A graph database is a database that uses graph structures to represent and store data.

What is a Graph Database

- Represents data as it exists in the real world that are naturally connected
- Does not try to change them in any way to define them as entities
- Graphs are composed of *vertices* and *edges*
- Vertices represent specific objects
- Edge is a relation between vertices
- Both vertices and edges can have any number of properties

The Power of Graph Databases

- Performance
  - Graph database performance tends to remain relatively constant, even as the dataset grows
- Flexibility
  - Graph data model better accommodates changing business needs
- Agility
  - Equip us to perform frictionless development and graceful system maintenance
  - Governance is typically applied in a programmatic fashion

The Power of Graph Databases

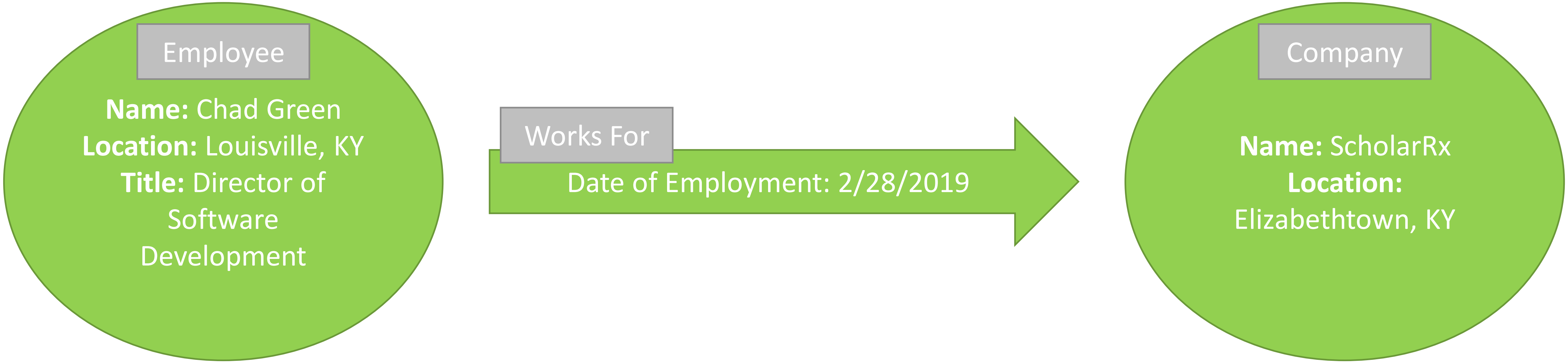
- Easily extendable and expandable
- Friendly to the human brain
- Whiteboard compatible

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



Graph Databases vs Relational Databases

Relational	Graph
Tables	Vertices (Nodes)
Schema with nullables	No schema
Relations with foreign keys	Relation is first class citizen
Related data fetched with joins	Related data fetched with a pattern

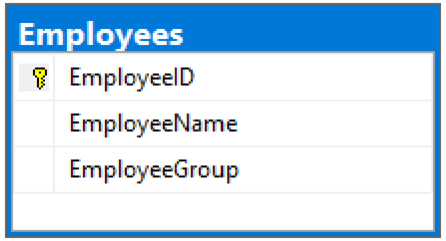
Human Resource Data

Graph Databases vs Relational Databases

Human Resource Data

Graph Databases vs Relational Databases

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



Human Resource Data

Graph Databases vs Relational Databases

```sql
-- Create the Employee Table
CREATE TABLE Employees
(
    EmployeeID    INT          IDENTITY(1,1),
    EmployeeName  VARCHAR(64),
    EmployeeGroup VARCHAR(32),
    CONSTRAINT pkcEmployees 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 Databases vs Relational Databases

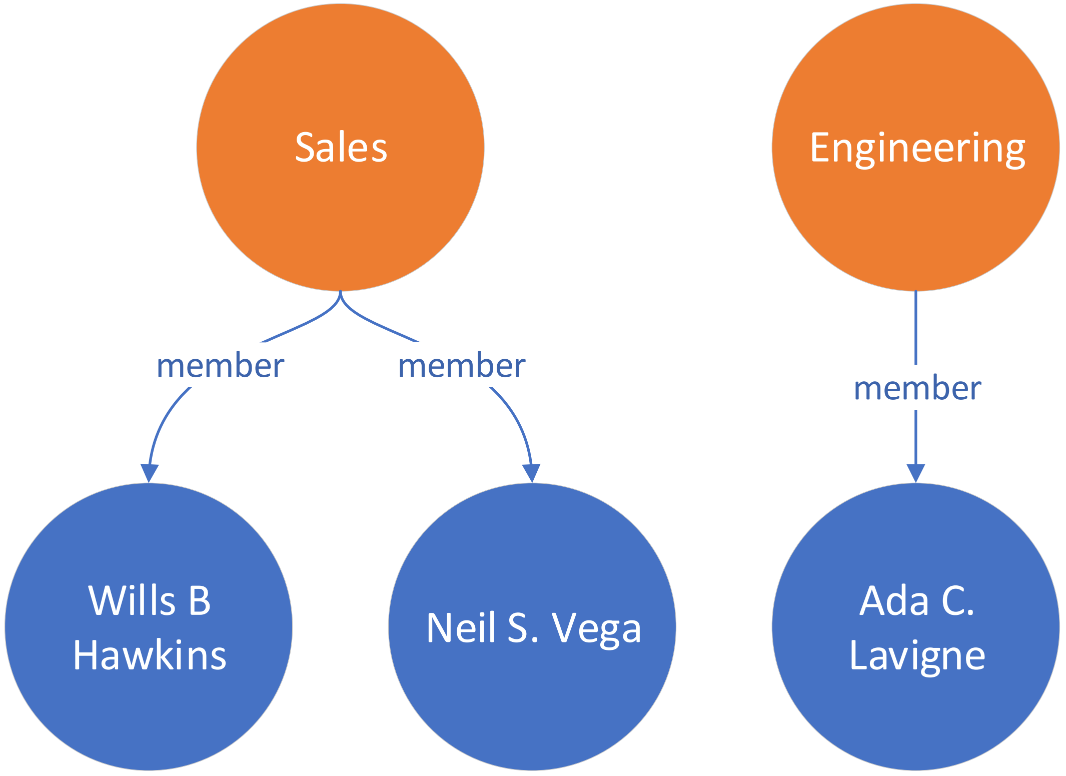
```
// Create group nodes
g.addV('group').property('id', 'Sales’)
g.addV('group').property('id', 'Engineering’)

// 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. Lavignee'))
```

Human Resource Data

Graph Databases vs Relational Databases

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

3 rows, 3 columns

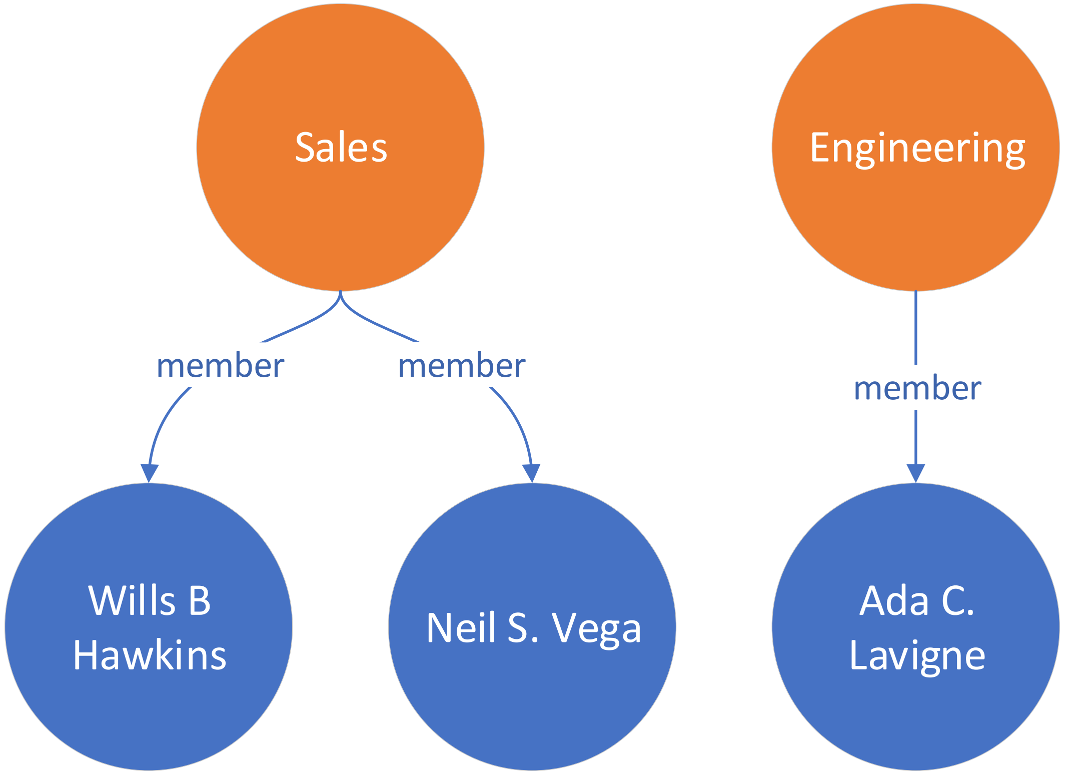


8 documents (vertices and edges)

Human Resource Data

Graph Databases vs Relational Databases

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')
```

Company Reorganization

Employees can now belong to multiple groups

Graph Databases vs Relational Databases

```
-- 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 Databases vs Relational Databases

```sql
-- 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 Databases vs Relational Databases

```sql
-- 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 Databases vs Relational Databases

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

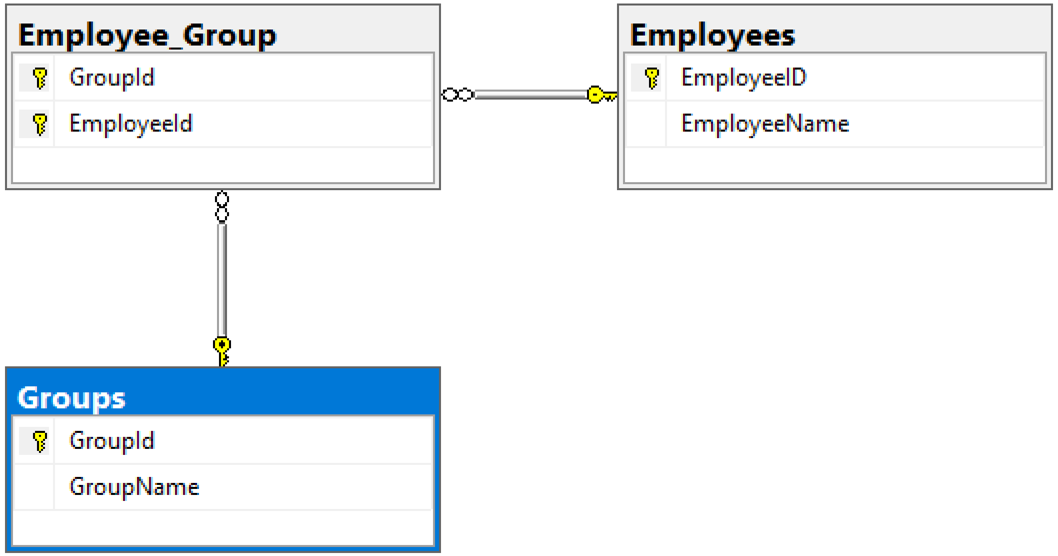
Employees can now belong to multiple groups

Graph Databases vs Relational Databases

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 Databases vs Relational Databases

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

Employees can now belong to multiple groups

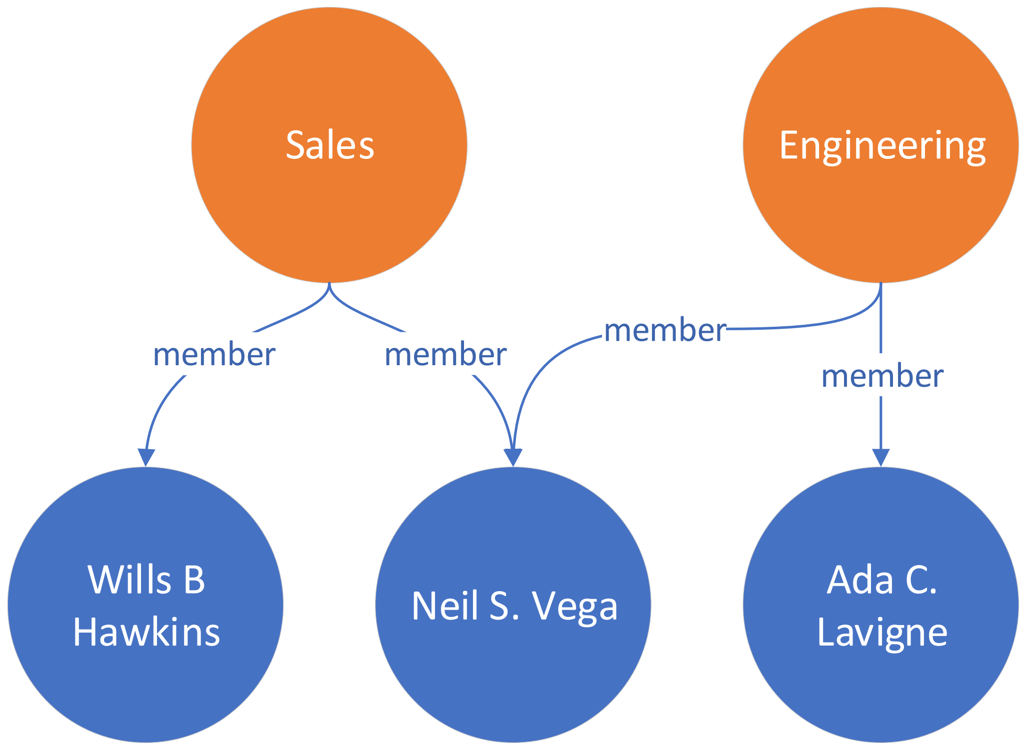
Graph Databases vs Relational Databases

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



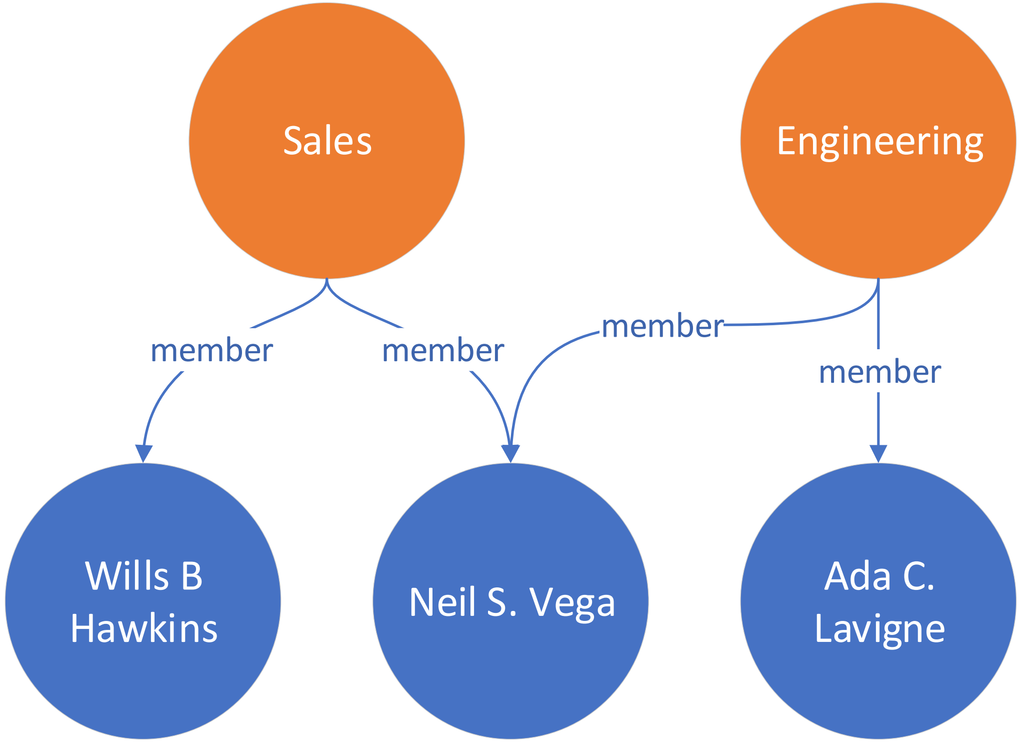
+1 document

Employees can now belong to multiple groups

Graph Databases vs Relational Databases

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').out('member')
```


Corporate Merger

Nested Groups

Graph Databases vs Relational Databases

```sql
-- Create the new Product Group
INSERT INTO Groups (GroupName) VALUES ('Product Group')
```

Nested Groups

Graph Databases vs Relational Databases

```
-- 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 Databases vs Relational Databases

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

Nested Groups

Graph Databases vs Relational Databases

```sql
-- 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

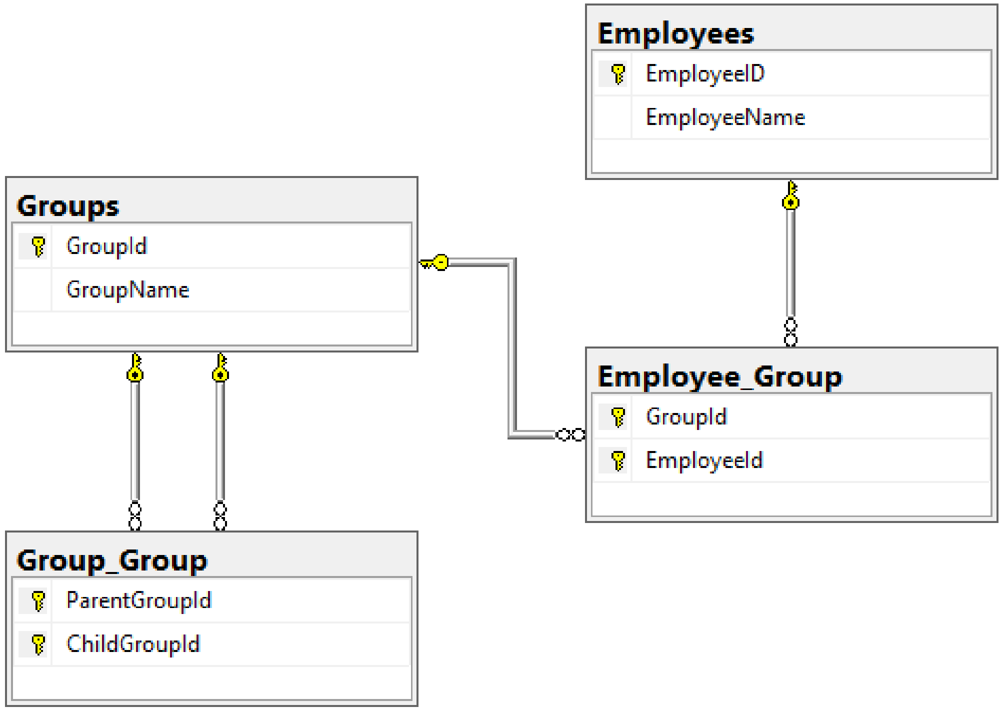
Graph Databases vs Relational Databases

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 Databases vs Relational Databases

```
// 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

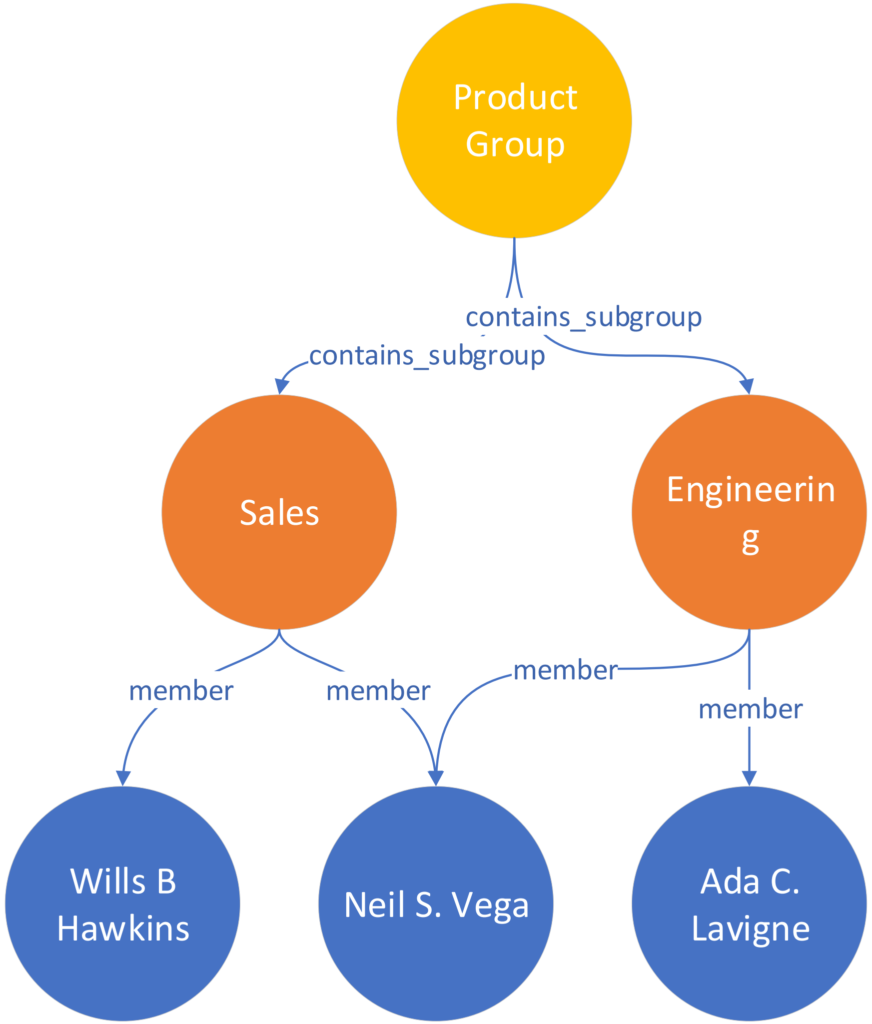
Graph Databases vs Relational Databases

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



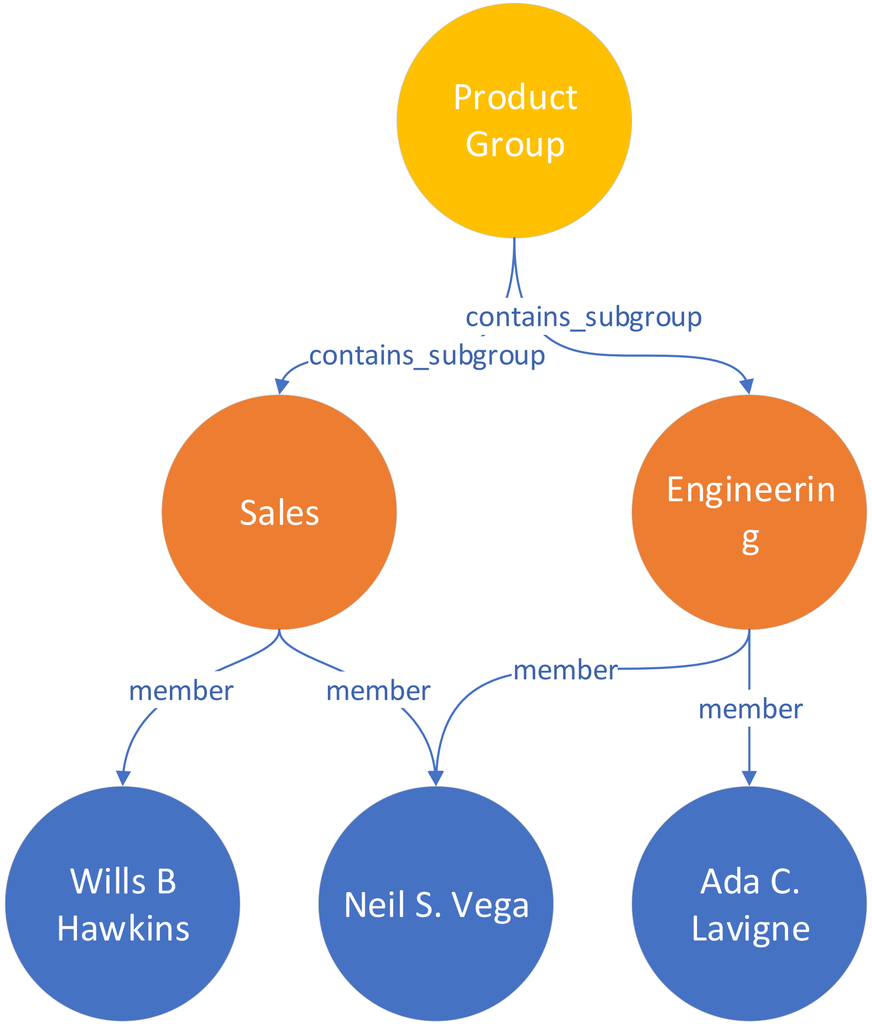
Added 1 table; 6 rows; 2 new columns

+3 documents

Nested Groups

Graph Databases vs Relational Databases

GroupId	GroupName
1	Engineering
2	Sales




```sql
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').out('contains_subgroup')
```

Management

Additional Hierarchies

Graph Databases vs Relational Databases

```
-- 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 Databases vs Relational Databases

```sql
-- 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 Databases vs Relational Databases

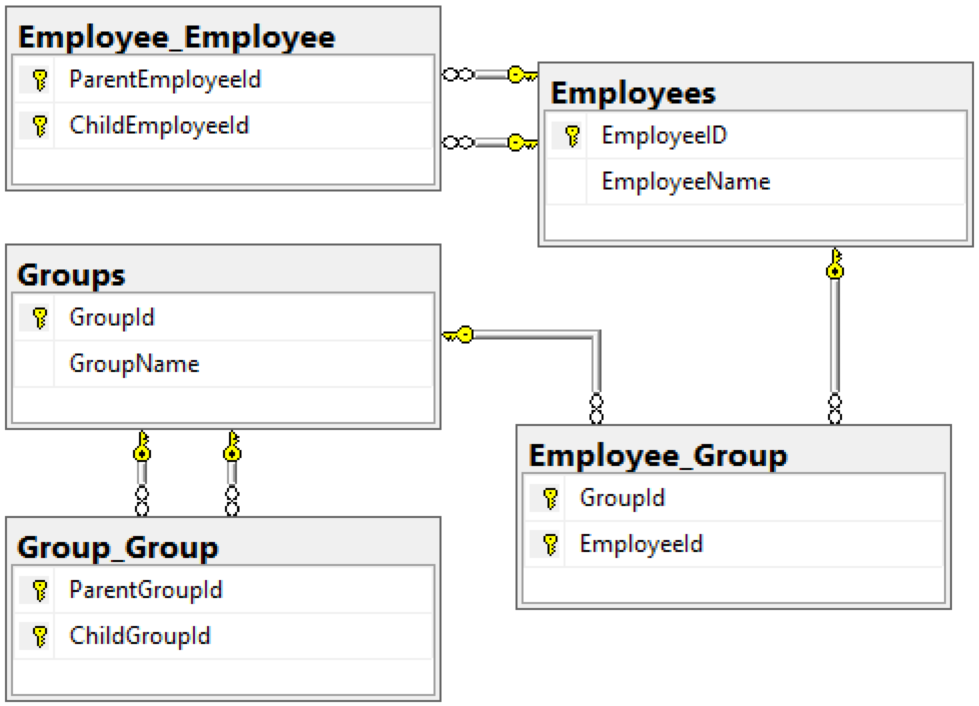
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

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 Databases vs Relational Databases

```
// 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 Databases vs Relational Databases

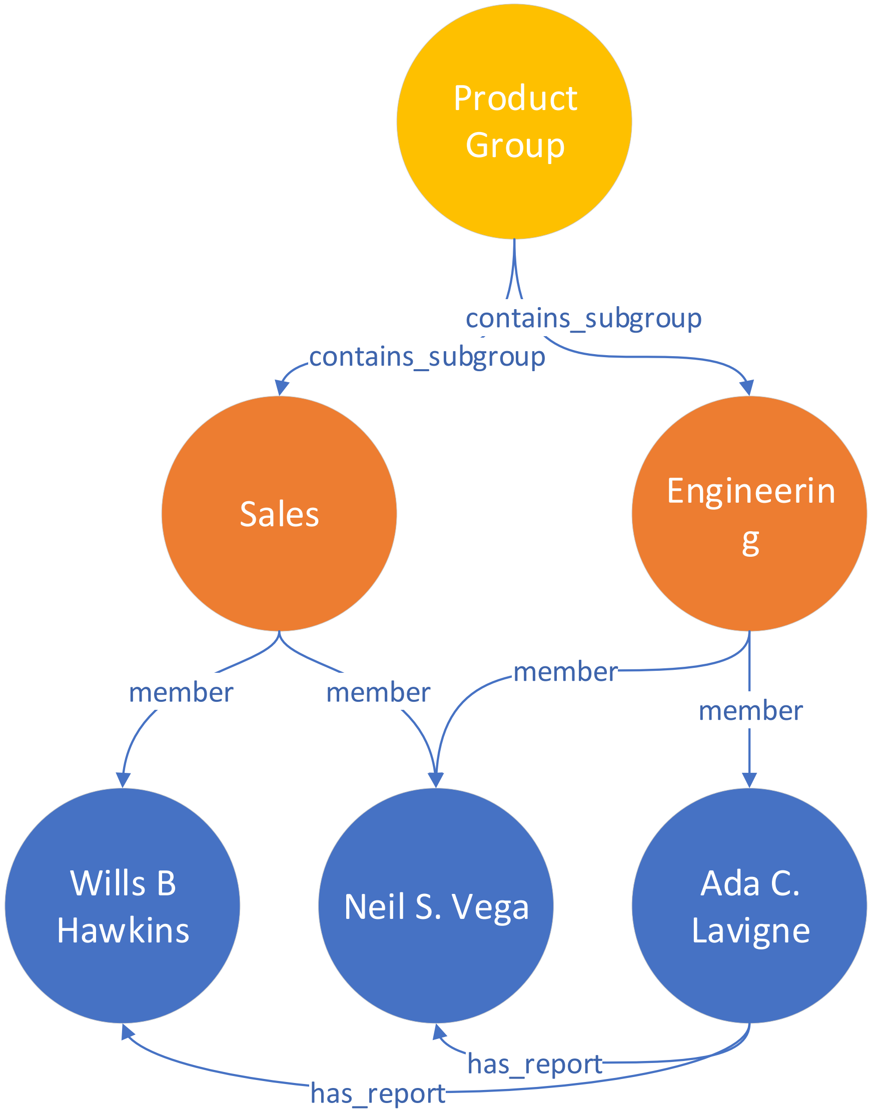
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

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



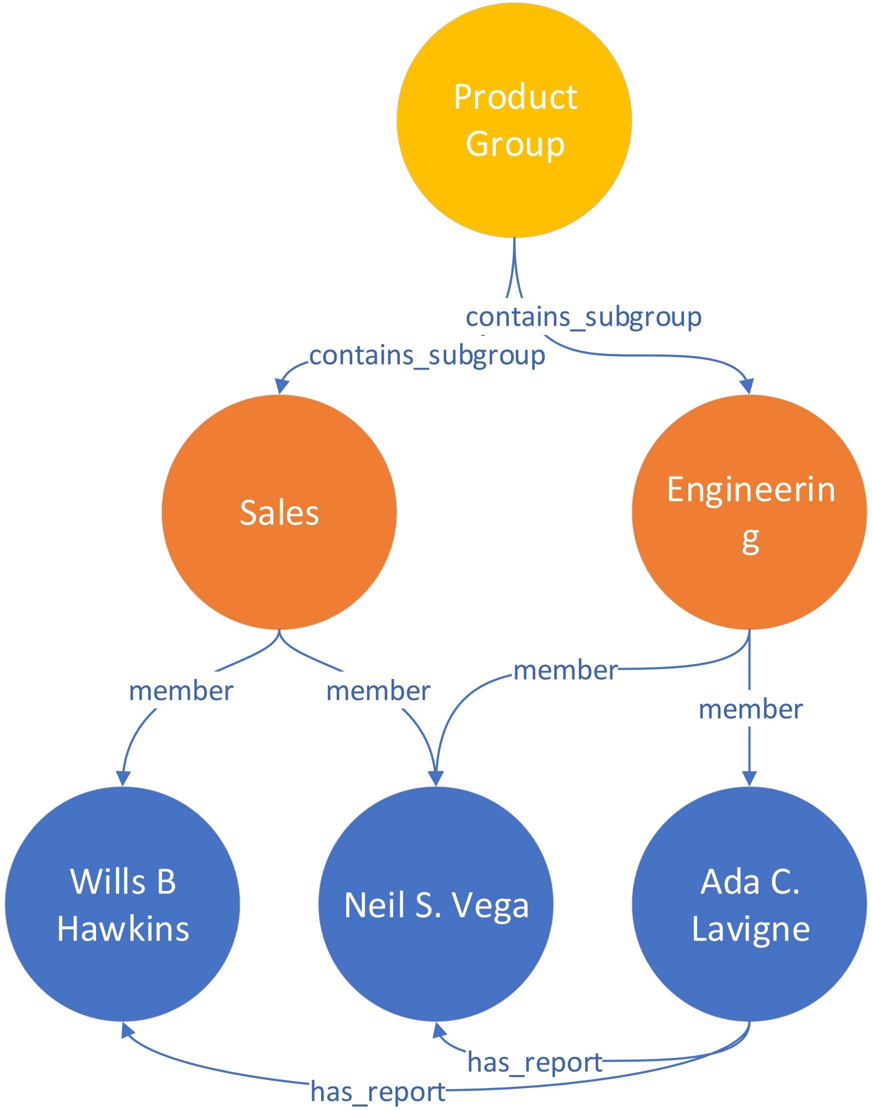
Added 1 table; 2 rows; 2 new columns

+2 documents

Additional Hierarchies

Graph Databases vs Relational Databases

EmployeeName
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 Groups.GroupId
                                   FROM Groups
                                  WHERE Groups.GroupName = 'Engineering')
```

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

Challenges of Relational Databases

Graph Databases vs Relational Databases

- Schema management
- Table alterations
- Costly writes against multiple tables
- Multiple JOIN operations
- Complex read queries

Common Graph Use Cases

- Internet of Things
- Customer 360
- Asset management
- Recommendations
- Fraud detection
- Data Integration
- Identity and access management
- Social networks
- Communication networks
- Genomics
- Epidemiology
- Semantic Web
- Search

Summary

- Graph is a structure amounting to a set of objects in which some pairs of objects are in some sense related
- Graphs are normally depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges
- Graph theory originated from solving the Seven Bridges of Königsberg problem
- A graph database is a database that uses graph structures to represent and store data
- Represents data as it exists in the real world that are naturally connected; does not try to change them in any way to define them as entities
- Graph databases provide Performance, Flexibility, and Agility

Introduction to TinkerPop & Gremlin

Getting Gremlins to Improve Your Data

What is TinkerPop

- Open source, vendor-agnostic, graph computing framework
- Apace2 license
- Allows users to model their domain as graph and analyze using Gremlin
- TinkerPop-enabled systems integrate with one another

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server
- TinkerGraph
- Programming Interfaces
- Documentation
- Useful Recipes

What is Gremlin

- Graph traversal language and virtual machine
- Works for both OLTP-based graph databases as well as OLAP-based graph processors
- Supports imperative and declarative querying
- Supports user-defined domain specific languages
- Supports single- and multi-machine execution modes
- Supports hybrid depth-and-breadth-first evaluation

What is Gremlin

- October 20, 2009 – TinkerPop project is born
- December 25, 2009 – v0.1 is released
- May 21, 2011 – v1.0 is released
- May 24, 2012 – v2.0 is released
- July 9, 2015 – v3.0 is released
- January 16, 2015 – TinkerPop becomes an Apache Incubator project
- March 18, 2019 – TinkerPop 3.4.1

What is Gremlin

TinkerPop 3.4.1 Changelog

- Gremlin.NET driver: fixed removal of closed connections and added round-robin scheduling
- Added GraphBinary serializer for TraversalMetrics
- Added registration for SparqStrategy for GraphSON
- Fixed up SparqStrategy so that it could be used property with RemoteStrategy
- Fixed ByteBuffer serialization for GraphBinary
- Fixed Path.ToString() in gremlin-javascript which was referencing an invalid object
- Fiex potential for an infinite loop in connection creation for gremlin-dotnet
- Added fallback resolver to TypeSerializerRegistry for GraphBinary
- Added easier to understand expecptions for connection problems in the Gremlin.NET driver
- Support configuring the type registry builder for GraphBinary
- Bump to Groovy 2.5.6
- Release working buffers in case of failure for GraphBinary
- GraphBinary: Use the same ByteBuff instance to write during serialization. Changed signature of write methods in type serializers.
- Remove unused parameter in GraphBinary's RsponseMesageSerializer.
- Changed SparqTraversalSource so as to enable Gremlin steps to be used to process results from the sparq() step.
- GraphBinary: Cache expression to obtain the method in PSerializer.

What is the Gremlin Console

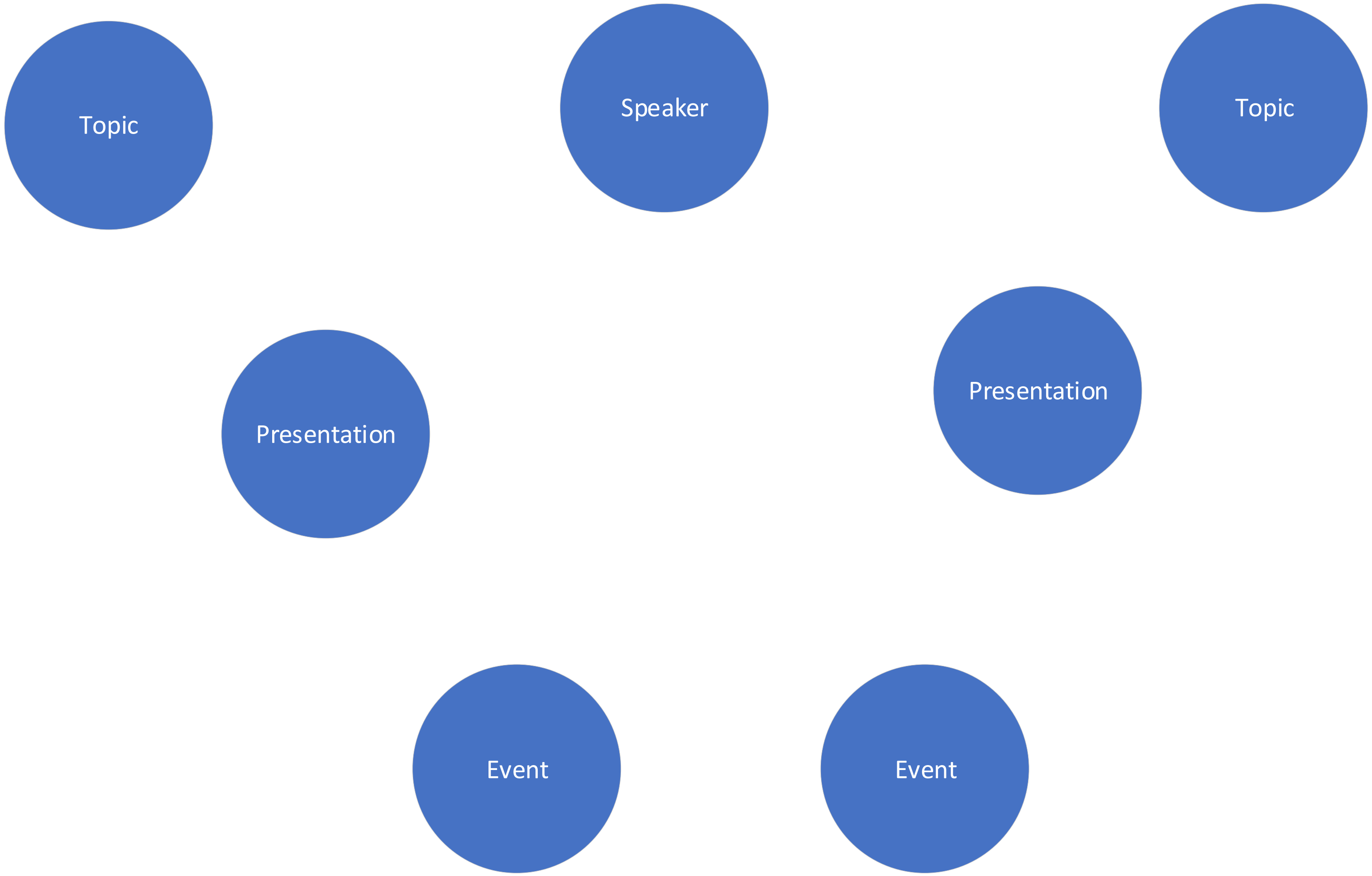
- Interactive terminal or REPL to traverse graphs and interact with the data they contain
- “Most common” method for performing ad-hoc analysis
- Other tools
  - Azure Portal
  - Visual Studio Code

What is the Gremlin Console

```
Command Prompt - bin\gremlin.bat

D:\gremlin-console>bin\gremlin.bat
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/D:/gremlin-console/lib/groovy-2.5.6
-indy.jar) to constructor java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)
WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.vmplugin.v7.Java7$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

Modeling Data as Property Graphs

Vertices



Modeling Data as Property Graphs

Vertices

g.addV(‘topic’).property(‘name’, ‘Database’)

g.addV(‘topic’).property(‘name’, ‘DevOps’)

g.addV(‘speaker’).property(‘firstName’, ‘Chad’).property(‘lastName’, ‘Green’)

g.addV(‘presentation’).property(‘name’, ‘Getting Started with Azure DevOps’)

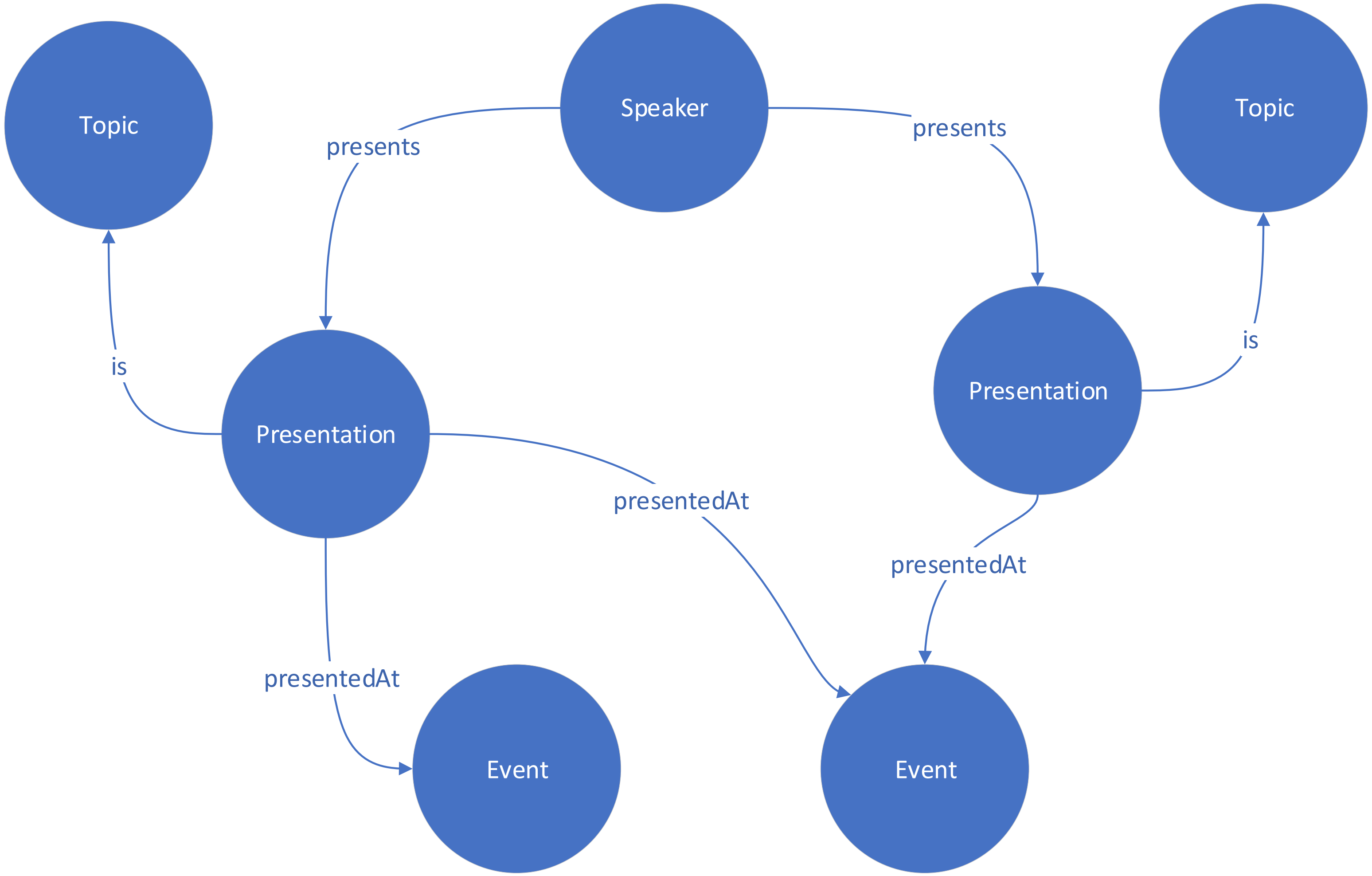
g.addV(‘presentation’).property(‘name’, ‘Getting Started with Azure SQL Database’)

g.addV(‘event’).property(‘name’, ‘DotNetSouth’)

g.addV(‘event’).property(‘name’, ‘KCDC’)

Modeling Data as Property Graphs

Edges



Modeling Data as Property Graphs

Edges

```
g.V().hasLabel('presentation').has('name', 'Getting Started with Azure DevOps')
.addE('presentedAt').to(                                        )

g.V().hasLabel('event').has('name', 'DotNetSouth')

   g.V().hasLabel('presentation').has('name', 'Getting Started with Azure DevOps')
   .addE('presentedAt').to(g.V().hasLabel('event).has('name', 'DotNetSouth')
```

Modeling Data as Property Graphs

Edges

g.V().hasLabel(‘presentation’).has(‘name’, ‘Getting Started with Azure DevOps’)

.addE(‘presentedAt’).to(g.V().hasLabel(‘event).has(‘name’, ‘DotNetSouth’))

g.V().hasLabel(‘speaker’).has(‘firstName’, ‘Chad’).has(‘lastName’, ‘Green’)

.addE(‘presents’).to(g.V().hasLabel(‘presentation’).has(‘name’, ‘Getting Started with Azure DevOps’))

Modeling Data as Property Graphs

Edges

g.V().hasLabel('presentation').has('name', 'Getting Started with Azure DevOps')
.addE('presentedAt').to(g.V().hasLabel('event).has('name', 'DotNetSouth'))

---

g.V().hasLabel('speaker').has('firstName', 'Chad').has('lastName', 'Green')
.addE('presents').to(g.V().hasLabel('presentation').has('name', 'Getting Started with Azure DevOps))

---

g.V().hasLabel('speaker').has('firstName', 'Chad').has('lastName', 'Green')
.addE('presents').to(g.V().hasLabel('presentation').has('name', 'Getting Started with Azure SQL Database'))

---

g.V().hasLabel('presentation').has('name', 'Getting Started with Azure SQL Database')
.addE('is').to(g.V().hasLabel('topic').has('name', 'Database'))

---

g.V().hasLabel('presentation').has('name', 'Getting Started with Azure DevOps')
.addE('is').to(g.V().hasLabel('topic').has('name', 'DevOps'))

Modeling Data as Property Graphs

Edges

g.V().hasLabel(‘speaker’).has(‘firstName’, ‘Chad’).has(‘lastName’, ‘Green’)
.addE(‘presents’).to(g.V().hasLabel(‘presentation’).has(‘name’, ‘Getting Started with Azure SQL Database’))

---

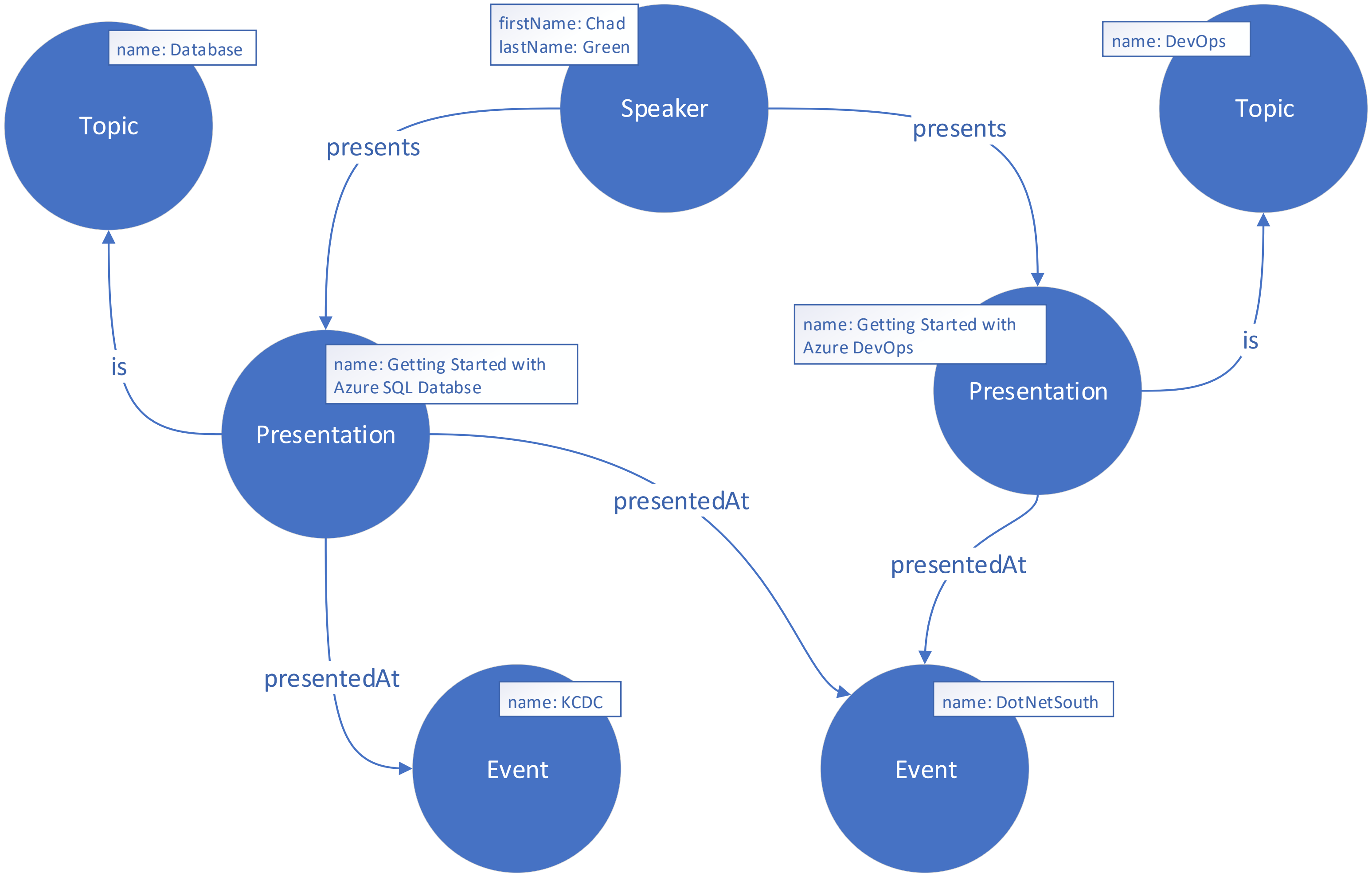
g.V().hasLabel(‘presentation’).has(‘name’, ‘Getting Started with Azure SQL Database’)
.addE(‘presentedAt’).to(g.V().hasLabel(‘event’).has(‘name’, ‘DotNetSouth’))

---

g.V().hasLabel(‘presentation’).has(‘name’, ‘Getting Started with Azure SQL Database’)
.addE(‘presentedAt’).to(g.V().hasLabel(‘event’).has(‘name’, ‘KCDC’))

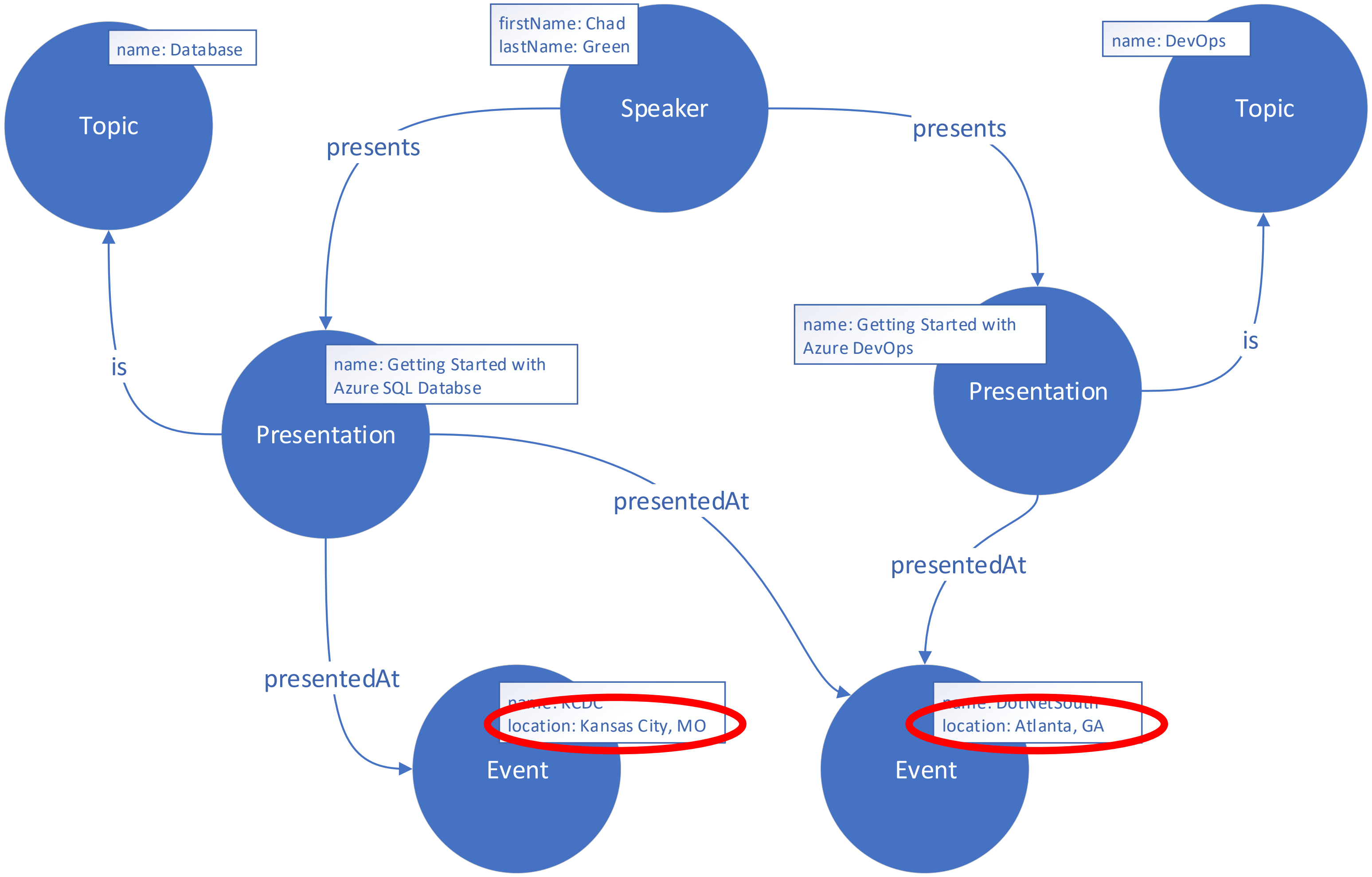
Modeling Data as Property Graphs

Vertices & Edges



Modeling Data as Property Graphs

Updating a Vertex



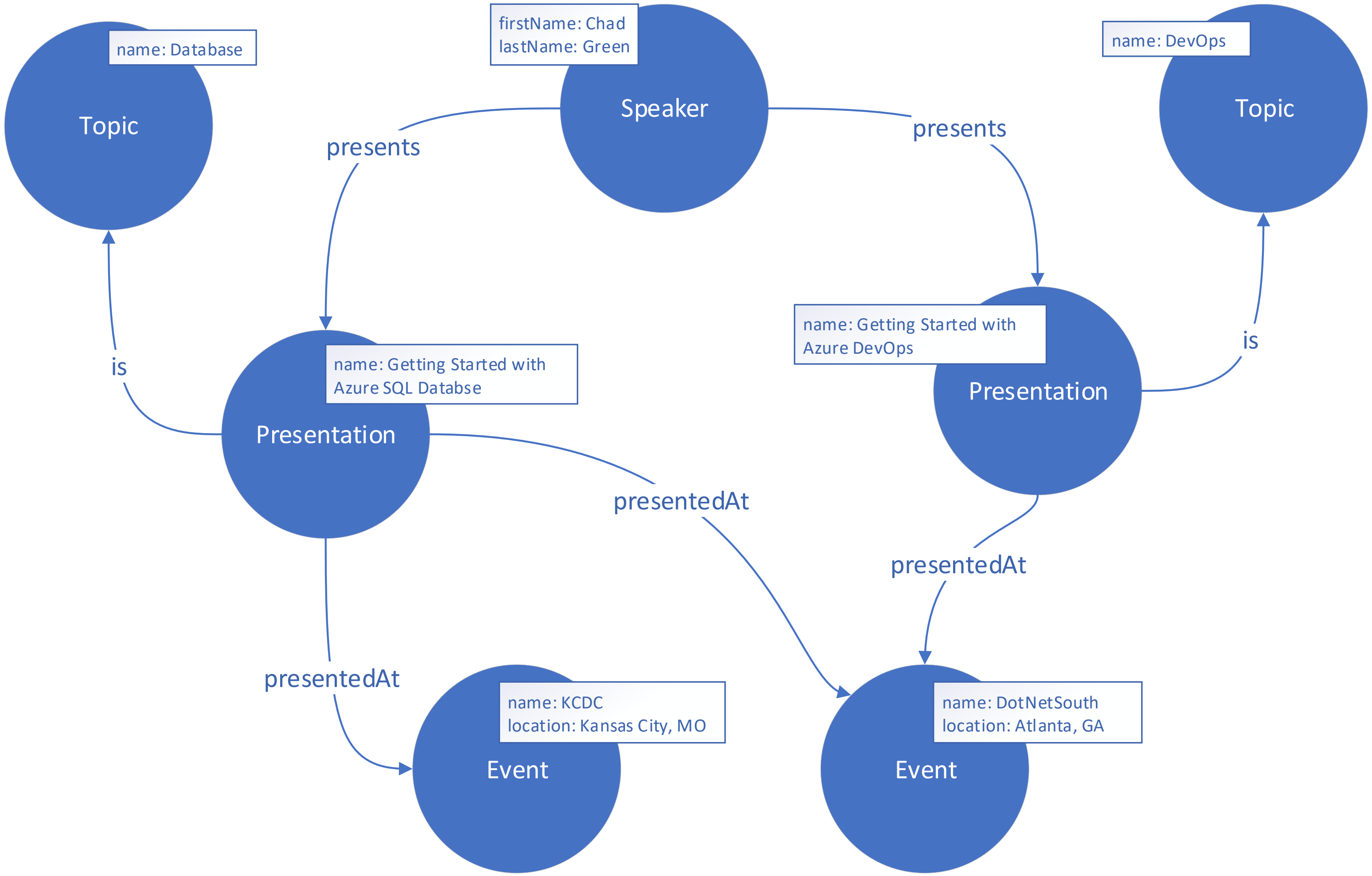
Modeling Data as Property Graphs

Updating a Vertex

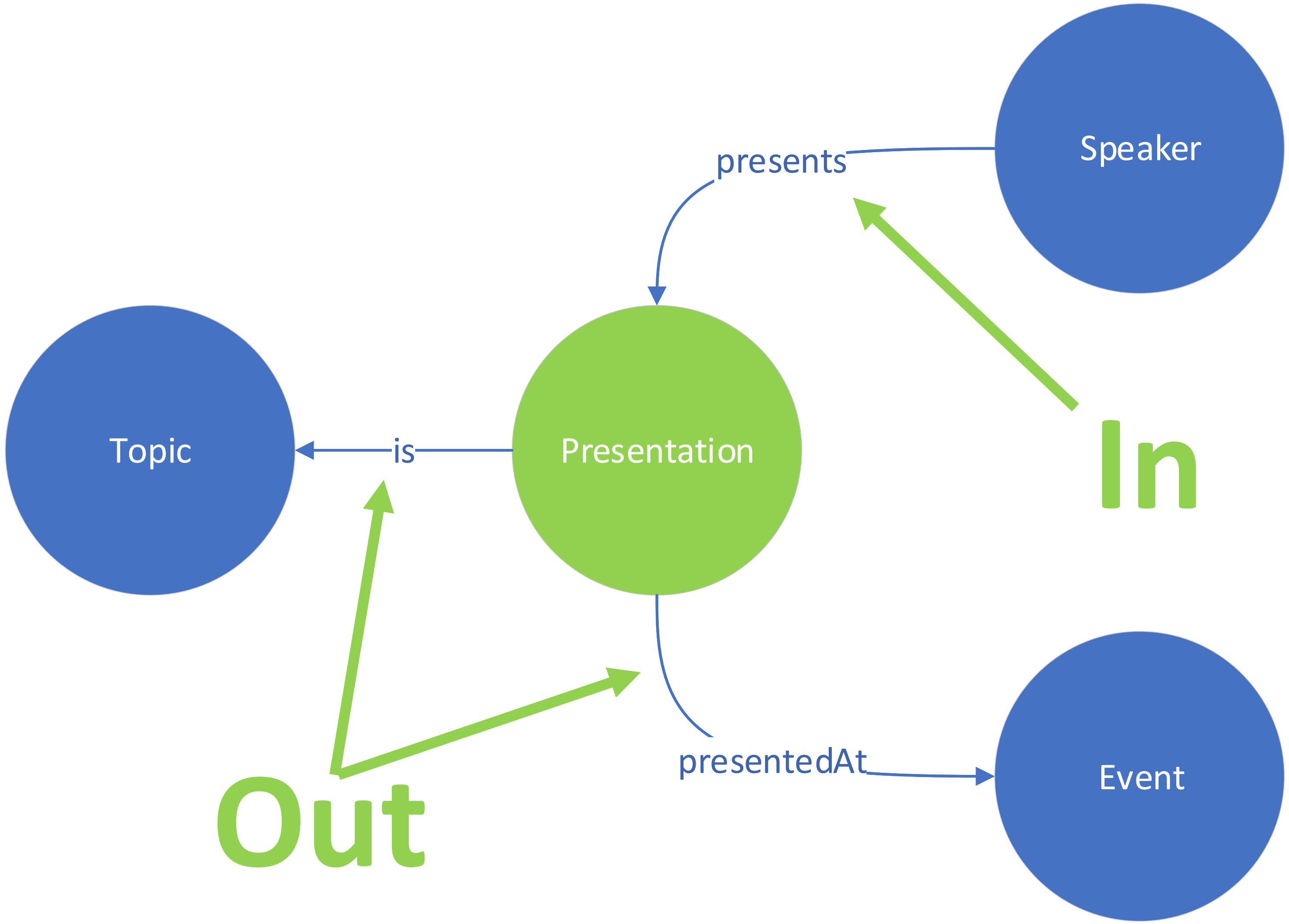
g.V().hasLabel('event').has('name', 'DotNetSouth').property('location', 'Atlanta, GA')

g.V().hasLabel('event').has('name', 'KCDC').property('location', 'Kansas City, MO')

Gremlin Traversal



Gremlin Traversal

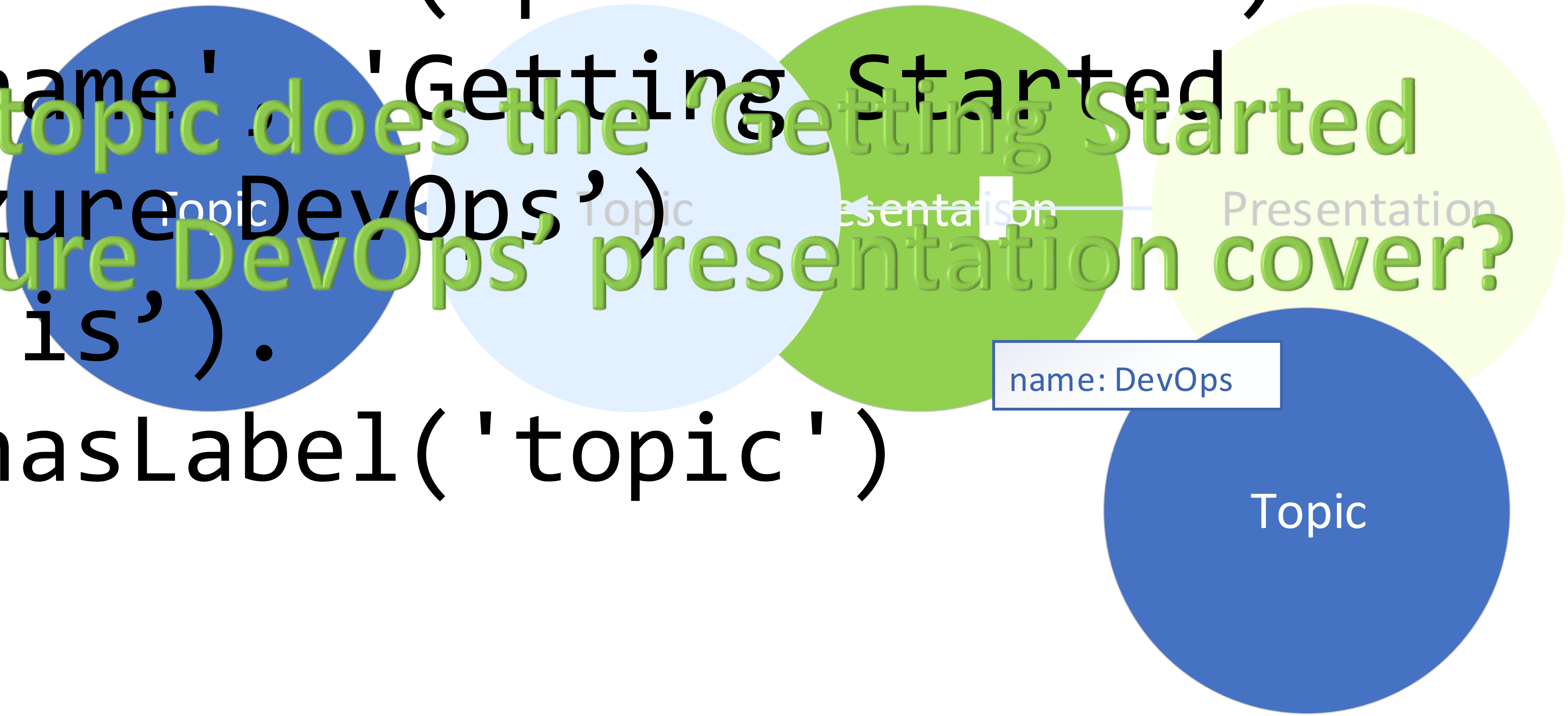


Gremlin Traversal

What topic does the ‘Getting Started with Azure DevOps’ presentation cover?

```
g.V().hasLabel('presentation’)
.has('name', 'Getting Started
with Azure DevOps’)
.outE('is’).
inV().hasLabel('topic')
```

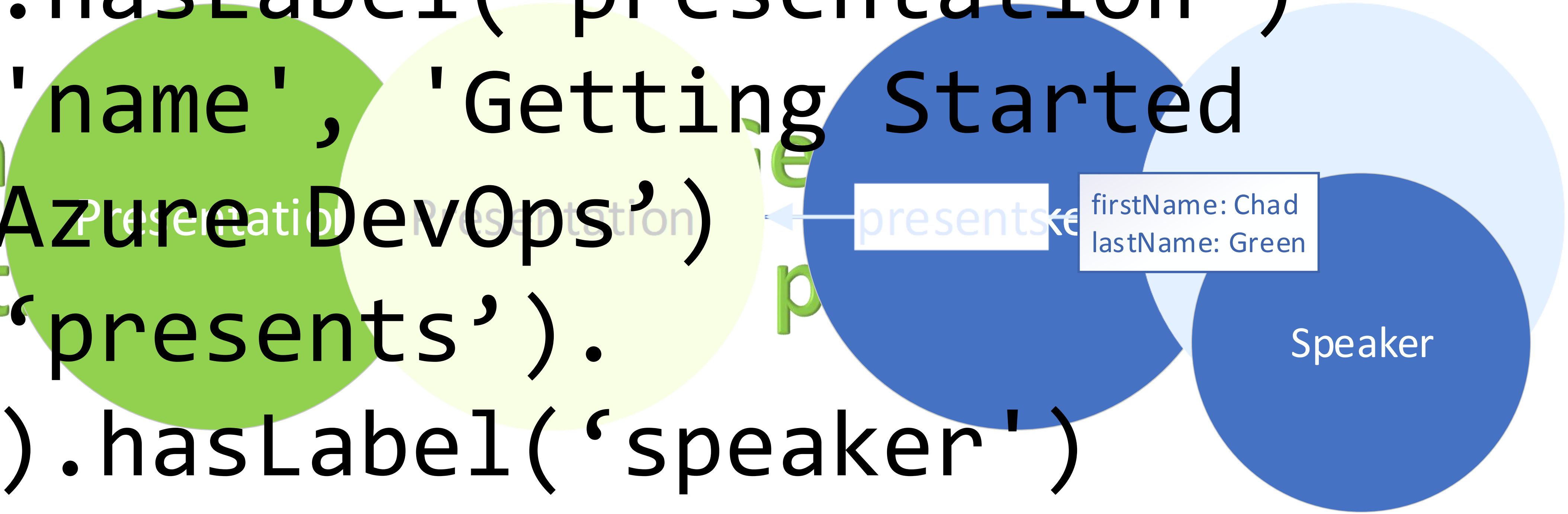
What topic does the ‘Getting Started with Azure DevOps’ presentation cover?



Gremlin Traversal

Who presents the ‘Getting Started with Azure DevOps’ presentation?

```
g.V().hasLabel('presentation’)
.has('name', 'Getting Started
with Azure DevOps’)
.inE(‘presents’).
outV().hasLabel(‘speaker')
```



Gremlin Traversal

```
g.V().hasLabel('speaker’)
.has('firstName’,'Chad’)
.has('lastName','Green’)
.outE('presents’)
.InV().hasLabel('presentation')
.outE('presentedAt')
.InV().hasLabel('event')
```

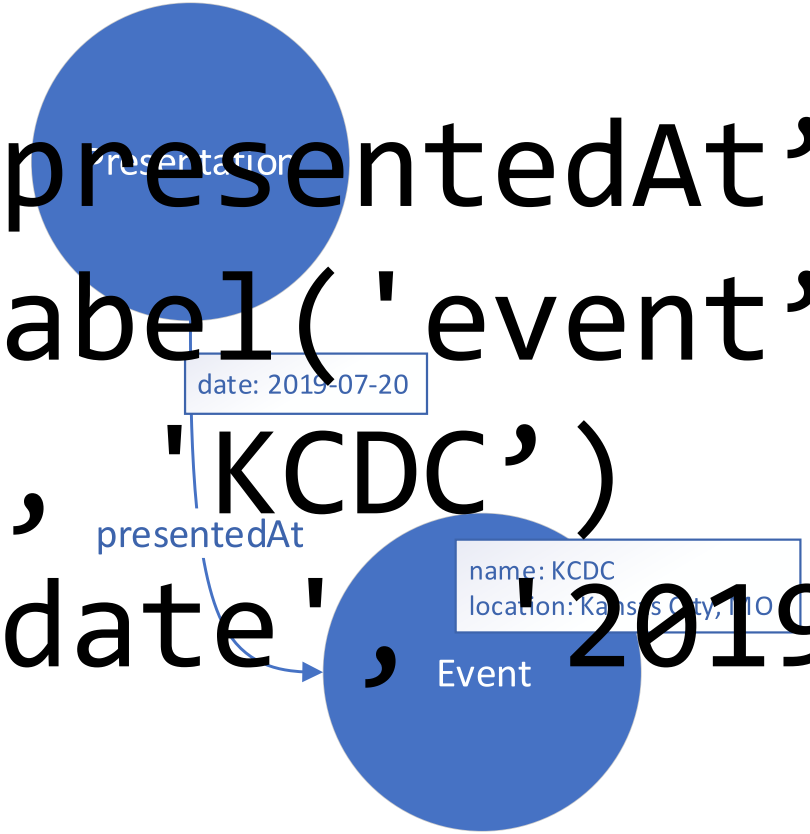

```
g.V().hasLabel('speaker’)
.has('firstName','Chad’)
.has('lastName','Green’)
.outE('presents')
.InV().hasLabel('presentation’)
```


What event's has Chad presented at?

Additional Gremlin Commands to Know

Edges Properties

```
g.E()
.hasLabel('presentedAt’)
.inV().hasLabel('event’)
.has('name', 'KCDC’)
.property('date', '2019-07-20')
```



Additional Gremlin Commands to Know

Dropping a Vertex

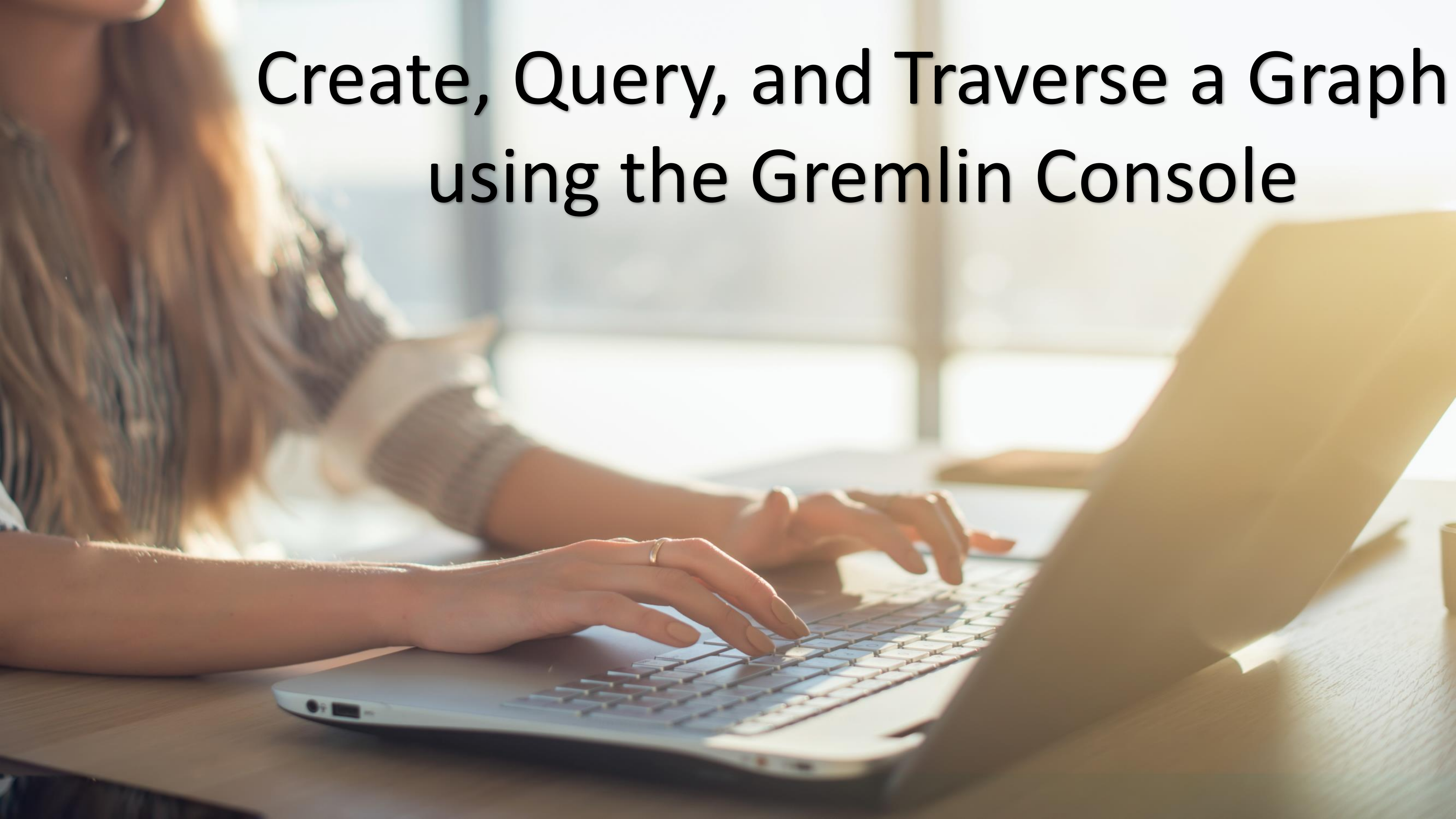
```
g.V()
.hasLabel('topic’)
.has('name', 'Database’)
.drop()
```

Additional Gremlin Commands to Know

Clearing the Graph

```
g.E().drop()

g.V().drop()
```

Create, Query, and Traverse a Graph using the Gremlin Console



Installing the Gremlin Console

- Download at https://tinkerpop.apache.org/downloads.html
- Unzip the package to somewhere on your computer

Do not start the Gremlin Console yet!

Introducing the Azure Cosmos Local Emulator

- Provides a local environment that emulates the Azure Cosmos DB service
- Can develop using the SQL, Cassandra, MongoDB, **Gremlin**, and Table APIs
- Data Explorer is only SQL API

Installing the Azure Cosmos Local Emulator

- Download at https://aka.ms/cosmosdb-emulator
- Run the downloaded azure-cosmosdb-emulator MSI

You must have administrative privileges on the computer.

Do not start the Local Emulator yet!

Starting the Azure Cosmos Local Emulator

- Open an administrator command prompt
- Start the emulator
  - "C:\Program Files\Azure Cosmos DB Emulator\Microsoft.Azure.Cosmos.Emulator.exe“ /EnableGremlinEndpoint

Starting the Gremlin Console

- Open a regular command prompt
- Navigate to the folder where you unzipped the Gremlin Console
- Run the following commands (and wait for everyone to catch up)

  copy /y conf\remote.yaml conf\remote-localcompute.yaml

  Notepad.exe conf\remote-local.compute.yaml

Starting the Gremlin Console

```
hosts: [localhost]
port: 8901
username: /dbs/bcc/colls/coll
password: C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==
connectionPool: {
enableSsl: false}
serializer: { className: org.apache.tinkerpop.gremlin.driver.ser.GraphSONMessageSerializerV1d0,
config: { serializeResultToString: true  }}
```

Starting the Gremlin Console

- bin\gremlin.bat   or   bin/gremlin.sh
- :remote connect tinkerpop.server conf/remote-local.yaml
- :remote console

Create Vertices and Edges

Input

```
g.addV('person’)
.property('firstName’, 'Thomas’)
.property('lastName', 'Andersen’)
.property('age', 44)
.property('userid', 1)
```

Output

==>[id:796cdccc-2acd-4e58-a324-91d6f6f5ed6d,label:person,type:vertex,properties:[firstName:[[id:f02a749f-b67c-4016-850e-910242d68953,value:Thomas]],lastName:[[id:f5fa3126-8818-4fda-88b0-9bb55145ce5c,value:Andersen]],age:[[id:f6390f9c-e563-433e-acbf-25627628016e,value:44]],userid:[[id:796cdccc-2acd-4e58-a324-91d6f6f5ed6d|userid,value:1]]]]

Create Vertices and Edges

Input

```
g.addV('person’)
.property('firstName’, ‘Mary Kay’)
.property('lastName', 'Andersen’)
.property('age’, 39)
.property('userid’, 2)
```

Create Vertices and Edges

Input

```
g.addV('person’)
.property('firstName’, ‘Robin’)
.property('lastName', ‘Wakefield’)
.property('userid’, 3)
```

Create Vertices and Edges

Input

```
g.addV('person’)
.property('firstName’, ‘Ben’)
.property('lastName’, ‘Miller’)
.property('userid’, 4)
```

Create Vertices and Edges

Input

```
g.addV('person’)
.property('firstName’, ‘Jack’)
.property('lastName’, ‘Connor’)
.property('userid’, 5)
```

Create Vertices and Edges

Input

```
g.V()
.hasLabel('person')
.has('firstName', 'Thomas')
.addE('knowns')
.to(
g.V()
.hasLabel('person')
.has('firstName', 'Mary Kay'))
```

Create Vertices and Edges

Input

```
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Thomas’)
.addE(‘knows’)
.to(
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Robin’))
```

Create Vertices and Edges

Input

```
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Robin’)
.addE(‘knowns’)
.to(
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Ben’)
```

Update a Vertex

Input

```
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Thomas’)
.property(‘age’, 45)
```

Query for Vertices

Input

```
g.V()
.hasLabel(‘person’)
.has(‘age’, gt(40))
.values(‘firstName’)
```

Traverse the Graph

Input

```
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Thomas’)
.outE(‘knows’)
.inV().hasLabel(‘person’)
```

Drop a Vertex

Input

```
g.V()
.hasLabel(‘person’)
.has(‘firstName’, ‘Jack’)
.drop()
```

Count vertices in the graph

Input

```
g.V().count()
```

Output

4

Property Projection

Input

```
g.V().hasLabel('person').values('firstName')
```

Output

```
==>Thomas
==>Mary Kay
==>Robin
==>Ben
```

Clear the Graph

Input

```
g.E().drop()

g.V().drop()
```

Exiting the Gremlin Console

:exit

Introduction to Cosmos DB

Getting Gremlins to Improve Your Data

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

Azure Cosmos DB

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

Azure Cosmos DB

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

Elastic scale out
of storage & throughput

Guaranteed low latency
at the 99th percentile

Turnkey global
distribution

Comprehensive
SLAs

Azure Cosmos DB

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

Consistent Prefix / Bounded Staleness / Eventual / Session / Strong

Five well-defined consistency models

Elastic scale out of storage & throughput

Guaranteed low latency at the 99th percentile

Turnkey global distribution

Comprehensive SLAs

Azure Cosmos DB

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

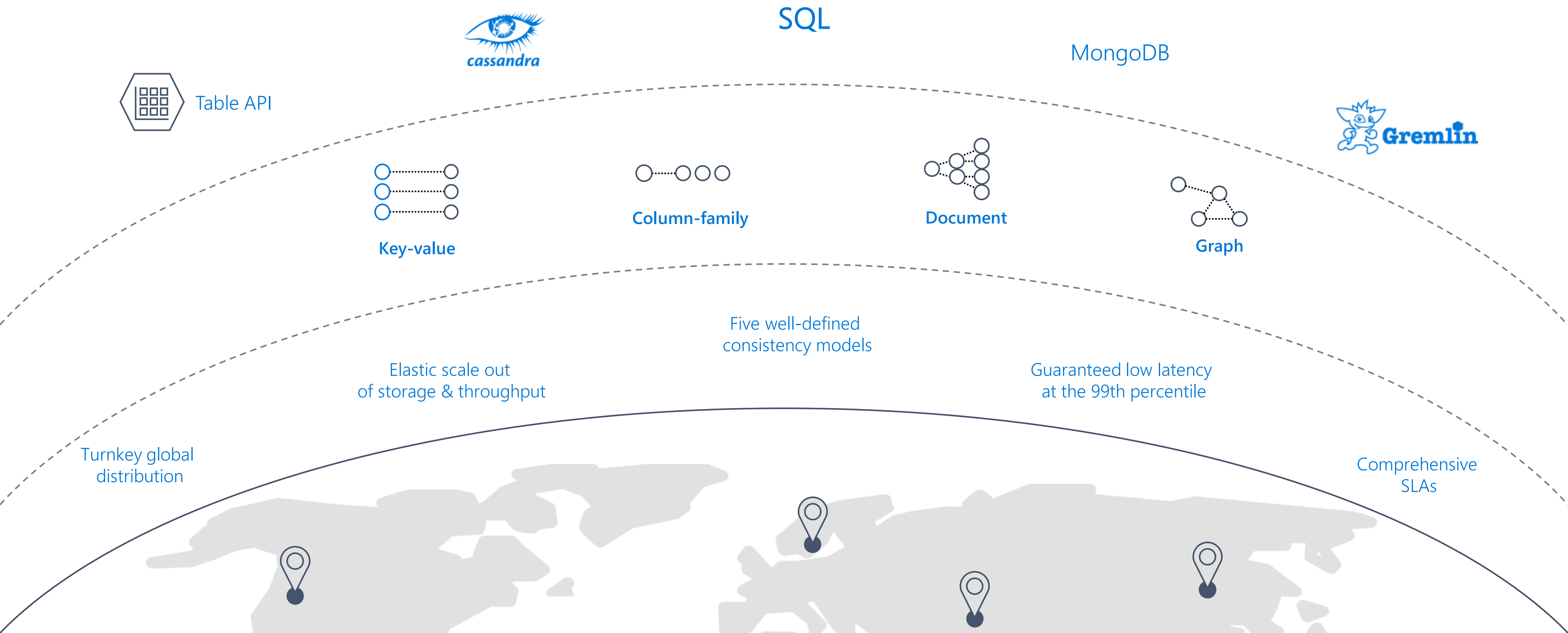
Turnkey global distribution

Regions where Azure Cosmos DB is available

nprehensive SLAs

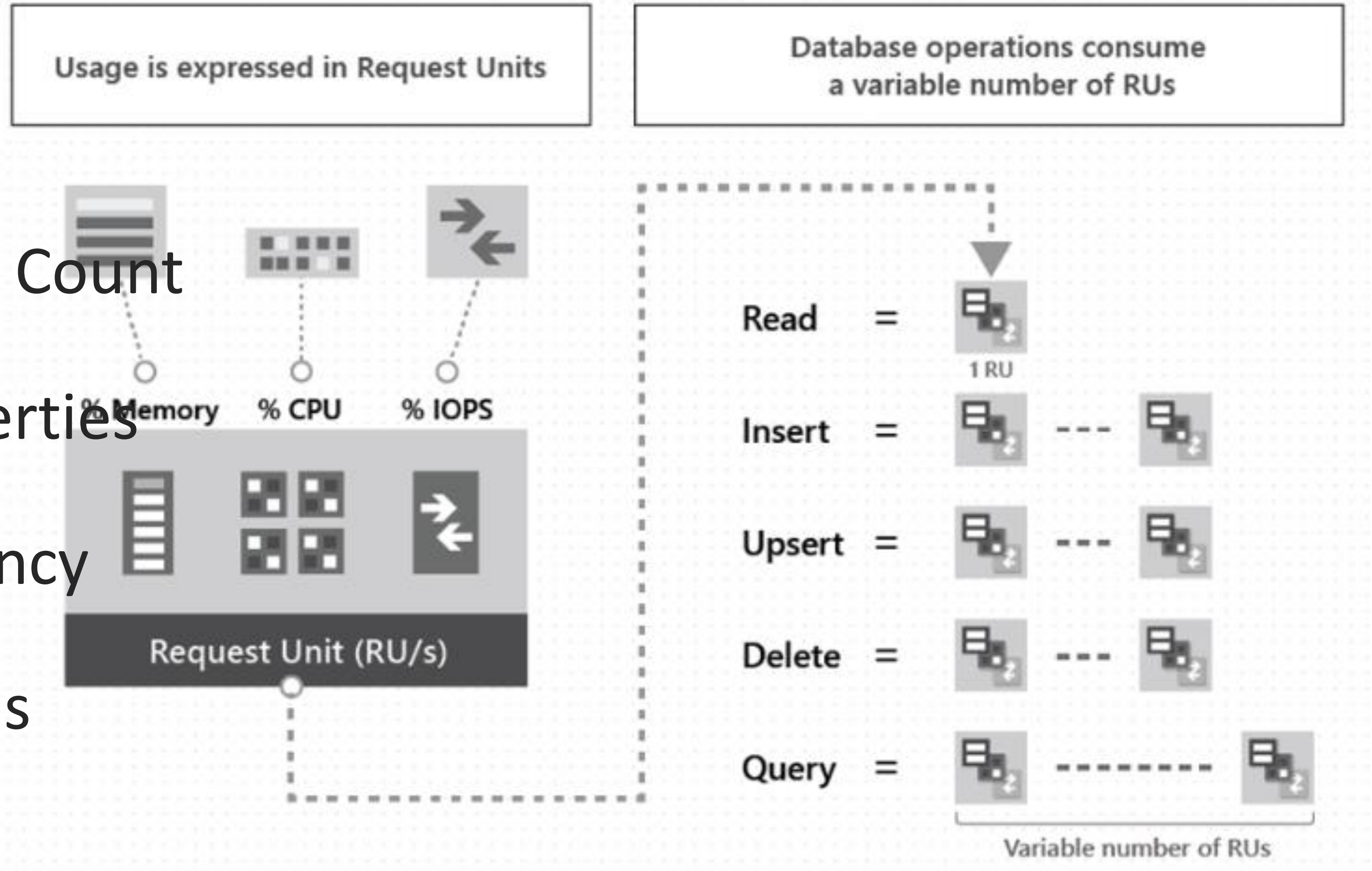
Azure Cosmos DB

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



Azure Cosmos DB Request Units

- Item Size
- Item Indexing
- Item Property Count
- Indexed Properties
- Data Consistency
- Query Patterns
- Script Usage



Azure Cosmos DB Pricing

Unit	Price
Provisioned Throughput (multiple region writes) per 100 RU/s	$0.016/hour
Provisioned Throughput (single region writes) per 100 RU/s	$0.008/hour
SSD Storage (per GB)	$0.25 GB/month

Starts at approximately $23.61/month

Save 15-65% with Reserved Pricing

Gremlin in Cosmos DB

Getting Gremlins to Improve Your Data

Azure Cosmos DB Graph Customers

Gremlin Features

- Graph Features
    - Provides Persistence and Concurrent Access
    - Designed to support Transactions

Gremlin Features

- Variable Features
  - Boolean
  - Integer
  - Byte
  - Double
  - Float
  - Integer
  - Long
  - String

Gremlin Features

- Vertex Features
  - AddVertices
  - RemoveVertices
  - MultiProperties
  - MetaProperties
  - AddProperty
  - RemoveProperty
  - StringIds
  - UserSuppliedIds

Gremlin Features

- Vertex Property Features
  - AddProperty; RemoveProperty
  - StringIds; UserSuppliedIds
  - BooleanValues, ByteValues, DoubleValues, FloatValues, IntegerValues, LongValues, StringValues

Gremlin Features

- Edge Features
  - AddEdges, RemoveEdges
  - StringIds, UserSuppliedIds
  - AddProperty, RemoveProperty

Gremlin Features

- Edge Property Features
  - Properties
  - BooleanValues, ByteValues, DoubleValues, FloatValues, IntegerValues, LongValues, StringValues

Gremlin Wire Format: GraphSON

- id
  - ID for the vertex; must be unique; automatically supplied if not provided
- label
  - Label of the vertex; used to describe the entity type
- type
  - Used to distinguish vertices from non-graph documents
- properties
  - Bag of user-defined properties associated with vertex; each property can have multiple values
- _partition
  - Partition key of the vertex
- outE
  - List of out edges from a vertex

Gremlin Wire Format: GraphSON

```
{
 "id": "a7111ba7-0ea1-43c9-b6b2-efc5e3aea4c0",
 "label": "person",
 "type": "vertex",
 "outE": {
    "knows": [
      { "id": "3ee53a60-c561-4c5e-9a9f-9c7924bc9aef",
        "inV": "04779300-1c8e-489d-9493-50fd1325a658“
      },
      { "id": "21984248-ee9e-43a8-a7f6-30642bc14609",
        "inV": "a8e3e741-2ef7-4c01-b7c8-199f8e43e3bc“
      }
    ]
 },
 "properties": {
     "firstName": [ { "value": "Thomas“ } ],
     "lastName":  [ { "value": "Andersen“ } ],
     "age":       [ { "value": 45 } ]
  }
}
```

Gremlin Wire Format: GraphSON

- id
  - ID for the edge; must be unique
- label
  - Label of the edge; optional; used to describe relationship type
- inV
  - List of in vertices for an edge
- properties
  - Bag of user-defined properties associated with the edge

Gremlin Steps

- addE
  - Adds an edge between two vertices
- addV
  - Add a vertex to the graph
- and
  - Ensures that all the traversals return a value
- as
  - A step modulator to assign a variable to the output of a step
- by
  - A step modulator used with group and order
- coalesce
  - Returns the first traversal that returns a result

Gremlin Steps

- constant
  - Returns a constant value. Used with coalesce.
- count
  - Returns the count from the traversal
- dedup
  - Returns the values with duplicates removed
- drop
  - Drops the values (vertex/edge)
- executionProfile
  - Creates a description of all operations generated by the executed Gremlin step
- fold
  - Acts as a barrier that computes the aggregate of results

Gremlin Steps

- group
    - Groups the values based on the labels specified
- has
    - Used to filter properties, vertices, and edges. Supports hasLabel, hasId, hasNot, and has variants.
- inject
    - Inject values into a stream
- is
    - Used to perform a filter using a boolean expression
- limit
    - Used to limit the number of items in the traversal
- local
    - Local wraps a section of a traversal, similar to a subquery

Gremlin Steps

- not
  - Used to produce the negation of a filter
- optional
  - Returns the result of the specified traversal if it yields a result else it returns the calling element
- or
  - Ensures at least on of the traversals returns a value
- order
  - Returns results in the specified sort order
- path
  - Returns the full path of the traversal
- project
  - Provides the properties as a Map

Gremlin Steps

- tree
    - Aggregate paths from a vertex into a tree
- unfold
    - Unroll an iterator as a step
- union
    - Merge results from multiple traversals
- V
    - Includes the steps necessary for traversals between vertices and edges V, E, out, in, both, outE, inV, bothE, outV, inV, bothV, and otherV
- where
    - Used to filter results from the traversal. Supports eq, neq, lt, lte, gt, gte, and between operators


R
T
Y
U
I
Redo
Undrln
Italics
D
E
M
O
K
Find
C
V
B
N
M
Copy
Paste
Bold


Create a free Azure
Cosmos DB Account

https://azure.microsoft.com/en-us/try/cosmosdb/

Build a .NET Core Application using the Gremlin API

Cosmos DB Flight Search

Find Flights

From | To | Submit

Graph Partitioning

Getting Gremlins to Improve Your Data

Using a Partitioned Graph

- Requirements for partitioned graph
    - Partition is required
    - Both vertices and edges are stored as JSON documents
    - Vertices require a partition key
    - Edges stored with their source vertex
    - Graph queries need to specify a partition key

Using a Partitioned Graph

- Graph queries need to specify a partition
  - /id and /label not supported
  - Select vertex by ID, then the partition key
    - g.V(‘vertex_id’).has(‘partitionKey’, ‘partitionKey_value’)
  - Select a vertex by specify tuple including partition key value and ID
    - g.V(‘partitionKey_value’, ‘vertex_id’])
  - Specify array of tuples of partition key values and IDs
    - g.V(['partitionKey_value0', 'verted_id0'], ['partitionKey_value1', 'vertex_id1'], ...)
  - Selecting set of vertices and specifying a list of partition key values
    - g.V('vertex_id0', 'vertex_id1', 'vertex_id2', ...).has('partitionKey', within('partitionKey_value0', 'partitionKey_value01', 'partitionKey_value02', ...)

Best Practice for using Partitioned Graphs

- Always specify partition key value when creating vertex
- Use outgoing direction when querying edges whenever it is possible
- Choose partition key that evenly distributes data across partitions
- Optimize queries to obtain data within boundaries of a partition

Wrap Up

Getting Gremlins to Improve Your Data

Thank You

chadgreen@chadgreen.com

chadgreen.com

ChadGreen

ChadwickEGreen