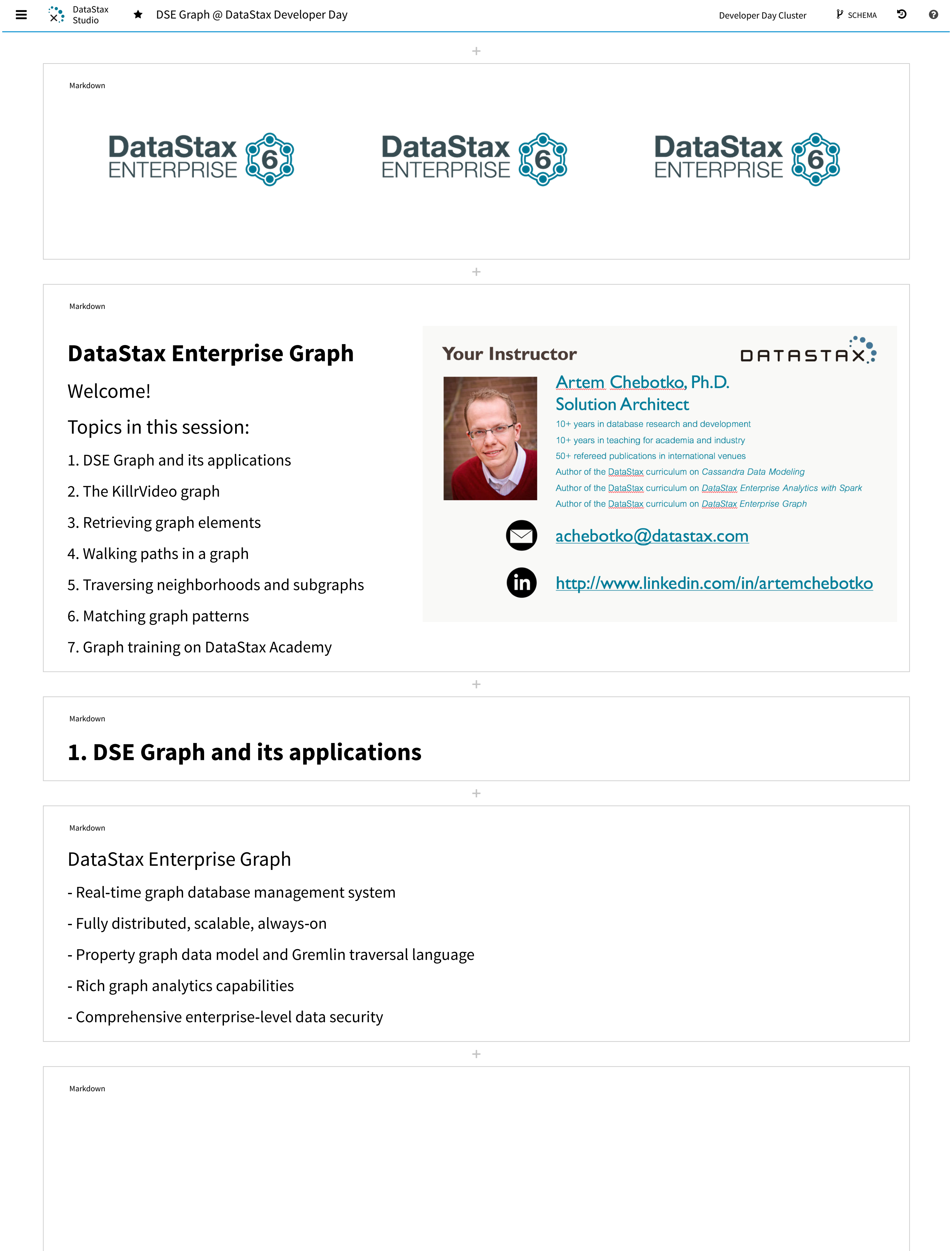


DataStax ENTERPRISE 6 DataStax ENTERPRISE 6 DataStax ENTERPRISE 6

DataStax Enterprise Graph

Welcome!

Topics in this session:

1. DSE Graph and its applications
2. The KillrVideo graph
3. Retrieving graph elements
4. Walking paths in a graph
5. Traversing neighborhoods and subgraphs
6. Matching graph patterns
7. Graph training on DataStax Academy

Your Instructor DATASTAX

Artem Chebotko, Ph.D.
Solution Architect

10+ years in database research and development

10+ years in teaching for academia and industry

50+ refereed publications in international venues

Author of the DataStax curriculum on *Cassandra Data Modeling*

Author of the DataStax curriculum on *DataStax Enterprise Analytics with Spark*

Author of the DataStax curriculum on *DataStax Enterprise Graph*

✉ achebotko@datastax.com

in http://www.linkedin.com/in/artemchebotko

1. DSE Graph and its applications

DataStax Enterprise Graph

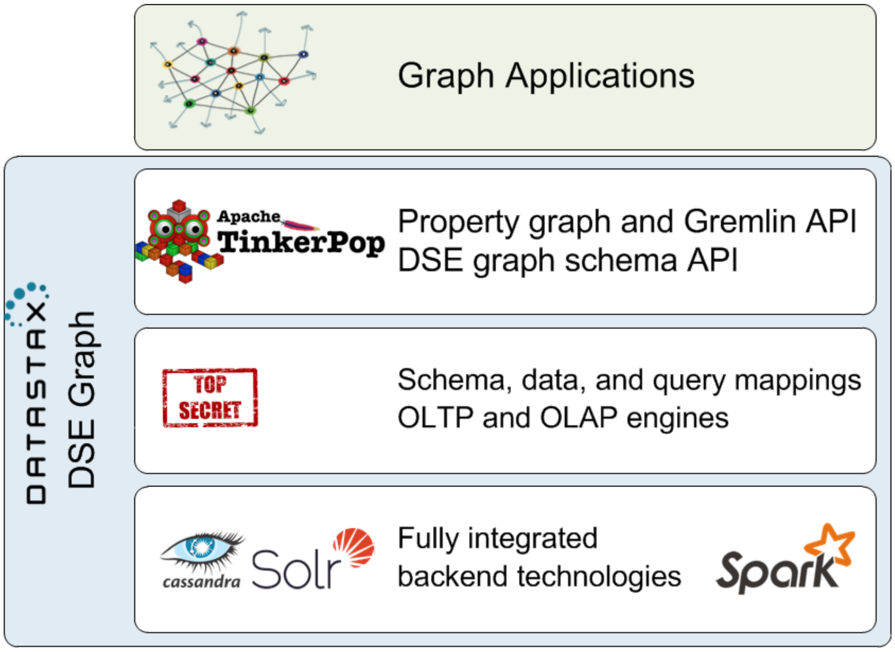
- Real-time graph database management system

- Fully distributed, scalable, always-on

- Property graph data model and Gremlin traversal language

- Rich graph analytics capabilities

- Comprehensive enterprise-level data security



Graph Applications

DSE Graph

Property graph and Gremlin API
DSE graph schema API

Schema, data, and query mappings
OLTP and OLAP engines

Fully integrated
backend technologies

Markdown

Identifying graph problems for DSE Graph

- Problem domain is naturally represented as a network, web, or graph

- Problem focus is on connections, links, relationships, and dependencies

- Solution has real-time requirements

- Solution must be efficient, scalable, and fault-tolerant

Markdown

Common traversal patterns and their sample applications

- Looking for paths

- social networks - connection between two people
- road networks - the shortest route between two locations

- Exploring neighborhoods

- customer 360-degree view
- social networks - friend of a friend
- sensor networks - area affected by a wildfire

- Matching complex graph patterns

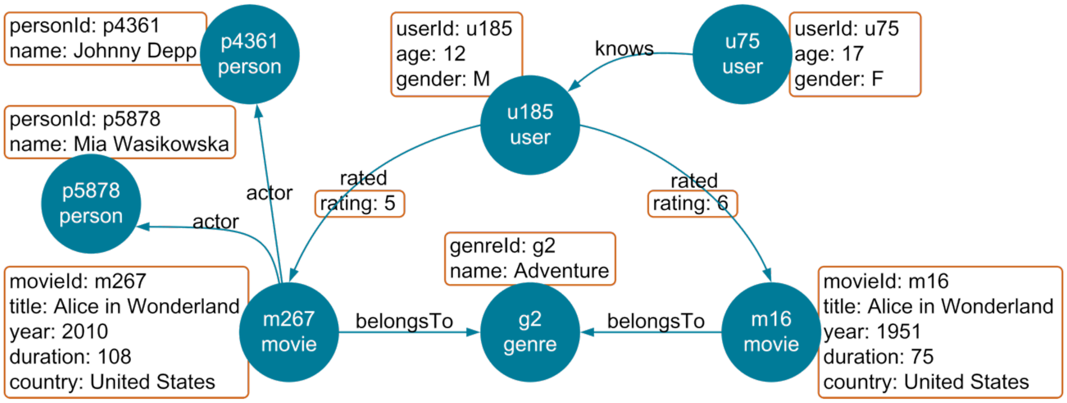
- recommendation engines - similar items
- entity resolution - similar items
- fraud detection - abnormal patterns

Markdown

2. The KillrVideo graph

Property graph – vertex-labeled, edge-labeled, directed, binary, attributed multi-graph

- Vertices
- Edges
- Properties



Let's do these steps together!

- Step 1. Create an empty graph
- Step 2. Create a graph schema
- Step 3. Import a graph dataset
- Step 4. Verify the result

+

Markdown

Step 2. Create a graph schema

+

Gremlin

```
// Set graph and schema configuration options
schema.config().option("graph.schema_mode").set("Production")
schema.config().option("graph.allow_scan").set(true)
schema.config().option("graph.tx_autostart").set(true)
schema.config().option("graph.tx_groups.default.read_consistency").set("LOCAL_ONE")
schema.config().option("graph.tx_groups.default.write_consistency").set("LOCAL_ONE")

// Define properties
schema.propertyKey("genreId").Text().create()
schema.propertyKey("personId").Text().create()
schema.propertyKey("userId").Text().create()
schema.propertyKey("movieId").Text().create()
schema.propertyKey("name").Text().create()
schema.propertyKey("age").Int().create()
schema.propertyKey("gender").Text().create()
schema.propertyKey("title").Text().create()
schema.propertyKey("year").Int().create()
schema.propertyKey("duration").Int().create()
schema.propertyKey("country").Text().create()
schema.propertyKey("production").Text().multiple().create()
schema.propertyKey("rating").Int().create()

// Define vertex labels
schema.vertexLabel("genre").partitionKey("genreId").properties("name").create()
schema.vertexLabel("person").partitionKey("personId").properties("name").create()
schema.vertexLabel("user").partitionKey("userId").properties("age","gender").create()
schema.vertexLabel("movie").partitionKey("movieId").properties("title","year","duration","country","production").create()

// Define edge labels
schema.edgeLabel("knows").connection("user","user").create()
schema.edgeLabel("rated").single().properties("rating").connection("user","movie").create()
schema.edgeLabel("belongsTo").single().connection("movie","genre").create()
schema.edgeLabel("actor").connection("movie","person").create()              // multiple() due to data
schema.edgeLabel("director").single().connection("movie","person").create()
schema.edgeLabel("composer").single().connection("movie","person").create()
schema.edgeLabel("screenwriter").connection("movie","person").create()       // multiple() due to data
schema.edgeLabel("cinematographer").single().connection("movie","person").create()

// Define vertex indexes
schema.vertexLabel("genre").index("genresByName").materialized().by("name").add()
schema.vertexLabel("person").index("personsByName").materialized().by("name").add()
schema.vertexLabel("user").index("usersByAge").secondary().by("age").add()
schema.vertexLabel("movie").index("moviesByTitle").materialized().by("title").add()
schema.vertexLabel("movie").index("search").search().by("title").asText().add()
schema.vertexLabel("movie").index("moviesByYear").secondary().by("year").add()

// Define edge indexes
schema.vertexLabel("user").index("toMoviesByRating").outE("rated").by("rating").add()
schema.vertexLabel("movie").index("toUsersByRating").inE("rated").by("rating").add()
```

+

Markdown

Step 3. Import a graph dataset

Gremlin

```
graph.io(IoCore.gryo()).readGraph("/var/lib/graph/KillrVideo.kryo")
```

Markdown

Step 4. Verify the result

Gremlin

```
g.V().has("movie","title","Alice in Wonderland").bothE().not(hasLabel("rated"))
//g.V().has("movie","title","Alice in Wonderland").bothE()
//g.V().count()
//g.E().count()
//g.V().groupCount().by(label)
//g.E().groupCount().by(label)
//g.V().groupCount().by(bothE().count())
```

Markdown

3. Retrieving graph elements

- Retrieving vertices and vertex properties
- Retrieving edges and edge properties

Markdown

Retrieving vertices and vertex properties

Markdown

Example 3.1: Find a movie with a known ID.

Gremlin

```
g.V().hasId('{~label=movie,movieId="m267"}')
//g.V('{~label=movie,movieId="m267"}').values("title","year")
//g.V().has("movie","movieId","m267").valueMap()
```

Markdown

Example 3.2: Find movies with a known property value.

Gremlin

```
g.V().has("movie","title","Alice in Wonderland")
//g.V().has("movie","year",gt(2010))
//g.V().has("movie","title",tokenRegex("Wonder.*")).values("title")
```

Markdown

Retrieving edges and edge properties

Markdown

Example 3.3: Find *rated*-edges for a *movie*-vertex with a known ID.

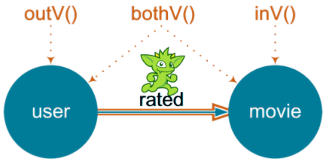
Gremlin

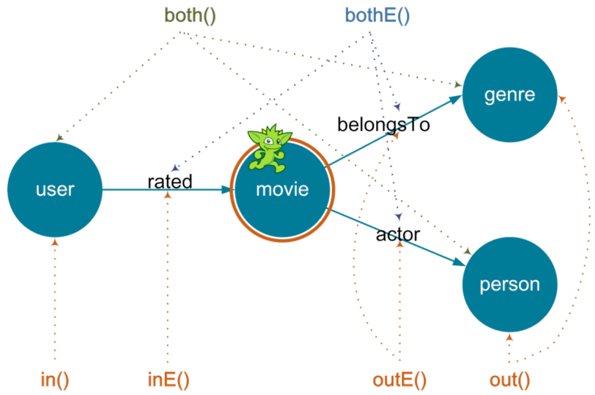
```
g.V().hasId('{~label=movie,movieId="m267"}').inE("rated")
//g.V().hasId('{~label=movie,movieId="m267"}').inE("rated").values("rating")
```

Markdown

4. Walking paths in a graph

- in(), out(), both()
- inE(), outE(), bothE()
- inV(), outV(), bothV(), otherV()
- select(), as(), by()
- repeat(), until(), times(), emit(), timeLimit()
- path(), simplePath()





Markdown

Simple navigation and projection

Markdown

Example 4.1: Find genres for Johnny Depp's movies.

Gremlin

```
g.V().has("person","name","Johnny Depp")
     .in("actor")
     .out("belongsTo")
     .dedup()
```

Markdown

Example 4.2: Find titles, years, and average ratings for Johnny Depp's movies.

Gremlin

```
g.V().has("person","name","Johnny Depp")
     .in("actor").as("t","y","r")
```

```
    .select("t","y","r")
    .by("title")
    .by("year")
    .by(inE("rated").values("rating").mean())
```

Markdown

Path existence and shortest path

Markdown

Example 4.3: Check if a path between actors Johnny Depp and Robert De Niro exists.

Gremlin

```
g.V().has("person","name","Johnny Depp")
    .repeat(both("actor").timeLimit(2000))
    .emit(has("person","name","Robert De Niro"))
    .limit(1)
```

Markdown

Example 4.4: Find the shortest path between actors Johnny Depp and Robert De Niro.

Gremlin

```
g.V().has("person","name","Johnny Depp")
    .repeat(both("actor").simplePath().timeLimit(2000))
    .until(has("person","name","Robert De Niro"))
    .path()
    .limit(1).unfold()
```

Markdown

5. Traversing neighborhoods and subgraphs

- subgraph(), cap(), iterate()

Markdown

Example 5.1: Extract an immediate neighborhood of the Johnny Depp vertex.

Gremlin

```
g.V().has("person","name","Johnny Depp")
    .bothE()
    .subgraph("johnnyGraph")
    .cap("johnnyGraph")
```

Gremlin

```
//t =
g.V().has("person","name","Johnny Depp")
     .bothE()
     .subgraph("johnnyGraph")
     .iterate()
//     .sideEffects.get("johnnyGraph").traversal()

//t.V().groupCount().by(label)
//t.E().groupCount().by(label)
```

Markdown

6. Matching graph patterns

Imperative and Declarative Traversals

Markdown

Example 6.1: Find directors who acted in their own Comedy movies (**imperative traversal**).

Gremlin

```
g.V().hasLabel("person").as("d")
     .in("director").as("m")
     .filter(out("belongsTo").has("name","Comedy"))
     .out("actor").as("a")
     .where("d",eq("a"))
     .select("d","m")
     .by("name")
     .by(valueMap("title","year"))
```

Markdown

Example 6.2: Find directors who acted in their own Comedy movies (**declarative traversal**).

Gremlin

```
g.V().match(
          __.as("m").hasLabel("movie"),
          __.as("m").out("director").as("p"),
          __.as("m").out("actor").as("p"),
          __.as("m").out("belongsTo").has("name","Comedy")
      )
     .select("p","m")
     .by("name")
     .by(valueMap("title","year"))
```

Markdown

7. Graph training on DataStax Academy

Gremlin

```
schema.drop()
schema.config().option("graph.schema_mode").set("Development")
```

Gremlin

```
def ds330 = graph.addVertex(label,"course","code","DS330","title","DataStax Enterprise 6 Graph","url","https://academy.datastax.com/resources/ds330-datastax-enterprise-6-
def m1 = graph.addVertex(label,"module","title","DSE Graph Overview"); m1.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Introducing DataStax Enterprise Graph").addEdge("partOf",m1)
         graph.addVertex(label,"unit","title","DSE Graph Architecture and Tools").addEdge("partOf",m1)
def m2 = graph.addVertex(label,"module","title","Managing Property Graphs"); m2.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Property Graph Data Model").addEdge("partOf",m2)
         graph.addVertex(label,"unit","title","Operations on Property Graphs").addEdge("partOf",m2)
def m3 = graph.addVertex(label,"module","title","Working With Graph Schemas"); m3.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Defining Graph Schemas").addEdge("partOf",m3)
         graph.addVertex(label,"unit","title","Understanding Graph Partitioning and Data Locality").addEdge("partOf",m3)
def m4 = graph.addVertex(label,"module","title","Indexing Graph Data"); m4.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Vertex Indexes").addEdge("partOf",m4)
         graph.addVertex(label,"unit","title","Property Indexes").addEdge("partOf",m4)
         graph.addVertex(label,"unit","title","Edge Indexes").addEdge("partOf",m4)
def m5 = graph.addVertex(label,"module","title","Inserting Graph Data"); m5.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Inserting Data Using Graph API").addEdge("partOf",m5)
         graph.addVertex(label,"unit","title","Exporting and Importing Graphs with Gremlin I/O").addEdge("partOf",m5)
         graph.addVertex(label,"unit","title","Loading Data Using DSE Graph Loader").addEdge("partOf",m5)
def m6 = graph.addVertex(label,"module","title","Graph Data Modeling"); m6.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Introduction to Graph Data Modeling").addEdge("partOf",m6)
         graph.addVertex(label,"unit","title","Data-Driven Approach to Graph Data Modeling").addEdge("partOf",m6)
         graph.addVertex(label,"unit","title","Graph Data Modeling Framework").addEdge("partOf",m6)
def m7 = graph.addVertex(label,"module","title","Traversing Graph Data"); m7.addEdge("partOf",ds330)
         graph.addVertex(label,"unit","title","Traversing Graph Data Preliminaries").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Gremlin Language Introduction").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Graph Traversal Steps").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Simple Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Branching Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Recursive Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Path Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Projecting Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Declarative Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Subgraph Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Statistical Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Ordering Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Mutating Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Domain-Specific Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Sack Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Recommender Traversals").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Traversal Behavior and Predictability Analysis").addEdge("partOf",m7)
         graph.addVertex(label,"unit","title","Understanding Bulking Optimization").addEdge("partOf",m7)
def ds332 = graph.addVertex(label,"course","code","DS332","title","DataStax Enterprise 6 Graph Analytics","url","https://academy.datastax.com/resources/ds332")
         graph.addVertex(label,"unit","title","Introduction to Graph Analytics").addEdge("partOf",ds332)
         graph.addVertex(label,"unit","title","Gremlin OLAP Traversals").addEdge("partOf",ds332)
         graph.addVertex(label,"unit","title","Introduction to DSE GraphFrames").addEdge("partOf",ds332)
         graph.addVertex(label,"unit","title","Using Gremlin with DSE GraphFrames").addEdge("partOf",ds332)
         graph.addVertex(label,"unit","title","Using Spark GraphFrame API with DSE GraphFrames").addEdge("partOf",ds332)
         graph.addVertex(label,"unit","title","Bulk Mutations with DSE GraphFrames").addEdge("partOf",ds332)
```

Gremlin

```
g.V().hasLabel("course")
```

Markdown

DS330: DataStax Enterprise 6 Graph (https://academy.datastax.com/resources/ds330-datastax-enterprise-6-graph)

DS332: DataStax Enterprise 6 Graph Analytics (https://academy.datastax.com/resources/ds332)

Markdown

Questions?

Thank You!