E6893 Big Data Analytics Lecture 6:

Linked Big Data Graph Database and Analytics

Ching-Yung Lin, Ph.D.
Adjunct Professor, Dept. of Electrical Engineering and Computer Science

October 11, 2018
Spark GraphX

- The Property Graph
  - Example Property Graph
- Graph Operators
  - Summary List of Operators
  - Property Operators
  - Structural Operators
  - Join Operators
  - Neighborhood Aggregation
    - Aggregate Messages (aggregateMessages)
    - Map Reduce Triplets Transition Guide (Legacy)
    - Computing Degree Information
    - Collecting Neighbors
  - Caching and Uncaching
- Pregel API
- Graph Builders
- Vertex and Edge RDDs
  - VertexRDDs
  - EdgeRDDs
- Optimized Representation
- Graph Algorithms
  - PageRank
  - Connected Components
  - Triangle Counting
Graph Analytics
Graph Definitions and Concepts

- A graph:
  \[ G = (V, E) \]
  
  \( V = \text{Vertices or Nodes} \)
  
  \( E = \text{Edges or Links} \)

- The number of vertices: “Order”
  \[ N_v = |V| \]
  
  \[ N_e = |E| \]
Property Graph

Vertex Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Property (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(rxin, student)</td>
</tr>
<tr>
<td>7</td>
<td>(jgonzal, postdoc)</td>
</tr>
<tr>
<td>5</td>
<td>(franklin, professor)</td>
</tr>
<tr>
<td>2</td>
<td>(istoica, professor)</td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>SrcId</th>
<th>DstId</th>
<th>Property (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>Collaborator</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Advisor</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Colleague</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>PI</td>
</tr>
</tbody>
</table>
GraphX Graph Operations

// Information about the Graph ======
val numEdges: Long
val numVertices: Long
val inDegrees: VertexRDD[Int]
val outDegrees: VertexRDD[Int]
val degrees: VertexRDD[Int]

In-degree = 8

// Views of the graph as collections ===
val vertices: VertexRDD[VD]
val edges: EdgeRDD[ED]
val triplets: RDD[EdgeTriplet[VD, ED]]

Out-degree = 8

Vertices: A

Edges: A → B

Triplets: A → B
Degree Distribution Example: Power-Law Network


\[ p_k = e^{-m} \cdot \frac{m^k}{k!} \]

\[ p_k = C \cdot k^{-\tau} e^{-k/\kappa} \]

Newman, Strogatz and Watts, 2001
Another example of complex network: Small-World Network

Six Degree Separation:
adding long range link, a regular graph can be transformed into a small-world network, in which the average number of degrees between two nodes become small.

from Watts and Strogatz, 1998

C: Clustering Coefficient, L: path length,
(C(0), L(0)): (C, L) as in a regular graph
(C(p), L(p)): (C,L) in a Small-world graph with randomness p.
Some examples of Degree Distribution

(a) scientist collaboration: biologists (circle) physicists (square), (b) collaboration of movie actors, (d) network of directors of Fortune 1000 companies
Network size is positively correlated with performance.

- Each person in your email address book at work is associated with $948 dollars in annual revenue.

$\text{1 direct contact in a person’s network}$

$$\text{$74.07$ increase in monthly revenues or$948$ annual revenues}$$

\text{Std error =(26.38)***}

\text{Significant at } p < 0.01
Basic graph algorithms in GraphX

```scala
// Basic graph algorithms
-----------------------------
def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
def connectedComponents(): Graph[VertexId, ED]
def triangleCount(): Graph[Int, ED]
def stronglyConnectedComponents(numIter: Int): Graph[VertexId, ED]
```
Centrality

“There is certainly no unanimity on exactly what centrality is or its conceptual foundations, and there is little agreement on the procedure of its measurement.” – Freeman 1979.

Degree (centrality)
Closeness (centrality)
Betweenness (centrality)
Eigenvector (centrality)
Closeness: A vertex is ‘close’ to the other vertices

\[ c_{CI}(v) = \frac{1}{\sum_{u \in V} \text{dist}(v,u)} \]

where \( \text{dist}(v,u) \) is the geodesic distance between vertices \( v \) and \( u \).
Betweenness ==> Bridges

Example: Healthcare experts in the world

Connections between different divisions

Example: Healthcare experts in the U.S.

Key social bridges
Betweenness

Betweenness measures are aimed at summarizing the extent to which a vertex is located ‘between’ other pairs of vertices.

Freeman’s definition:

\[
C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma(s, t \mid v)}{\sigma(s, t)}
\]

Calculation of all betweenness centralities requires calculating the lengths of shortest paths among all pairs of vertices. Computing the summation in the above definition for each vertex.
Eigenvector Centrality

Try to capture the ‘status’, ‘prestige’, or ‘rank’. More central the neighbors of a vertex are, the more central the vertex itself is.

\[ c_{Ei}(v) = \alpha \sum_{\{u,v\} \in E} c_{Ei}(u) \]

The vector \( \mathbf{c}_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T \) is the solution of the eigenvalue problem:

\[ A \cdot \mathbf{c}_{Ei} = \alpha^{-1} \mathbf{c}_{Ei} \]
PageRank Algorithm (Simplified)
Economic Issues – Network Topology and Worker Productivity

- Topological point of views
  - What type of network structure is beneficial?

**Cohesive Network**
- Trust
- Absorptive capacity
- Precision, Reliability

**Structurally Diverse Network**
- Brokering position
- Access to many pools of diverse, novel information

What type of network structure is most beneficial in an electronic network for consultants?
- Importance of Direct Contacts?
- Importance of Indirect Contacts?
- Constrained vs. unconstrained?
Network Topology Measures

Direct Contacts

Size(7) = 4
Size(12) = 3
+ No information distortion
- High maintenance cost

Network size $\rightarrow$ strong work performance (?)

Indirect Contacts

Btw(7) = 33
Btw(12) = 6
3steps(7) = 11
3steps(12) = 8
+ Access diverse information
- Information distortion

Btw-centrality $\rightarrow$ Strong work performance (?)
3-step Reach $\rightarrow$ Strong work performance (?)

Structural Diversity

Div(7) = 0.53
Div(12) = 0.16
+ Transfer complex knowledge
- Access diverse knowledge

Diversity $\rightarrow$ Strong work performance (?)
Enterprise becomes more successful utilizing Social Network Analysis

- We and MIT studied 2,038 IBM Global Business Consultants for 2 years, it was found that:
  - After a consultant started using SmallBlue, his social network/capital obviously grew and his monthly billable revenue for IBM increased by $584.15 (i.e., $7,010 per year)

- Joint analysis of social capital and economic capital:
  - Adding a person in personal network (i.e., someone with frequent communications), increases $948 yearly revenue for IBM. *(selected by BusinessWeek Magazine as the Top Story of the Week, April 8, 2009)*
  - 1% increase in social network diversity is associated with $239.5 in monthly revenue (i.e., $2,874 revenue increase per year).
  - 1% increase in social network diversity is associated with an increase of 11.8% in job retention (i.e., surviving layoff).

SmallBlue / Atlas was featured in 120+ news articles, including 4 times by BusinessWeek (Jan and May 2008, April and June 2009)
Observations from Personal Social Networks vs. Revenue

- Structural Diverse networks with abundance of structural holes are associated with higher performance.
  - *Having diverse friends helps.*

- Betweenness is negatively correlated.
  - *Being a bridge between a lot of people is not helpful.*

- Network reach are highly corrected.
  - *The number of people reachable in 3 steps is positively correlated with higher performance.*

- Having too many strong links — the same set of people one communicates frequently is negatively correlated with performance.
  - *Perhaps frequent communication to the same person may imply redundant information exchange.*
    - Future textual analysis can be done to confirm this.
Project Team Composition—Managers

The number of managers in a project exhibit an inverted-U shaped curve.

1. Having managers in a project is correlated with team performance initially.
2. Too many managers in a project is negatively associated with team performance.

\[ revenue = \alpha + \beta_1 \cdot mgr + \beta_2 \cdot mgr^2 + \gamma_1 \cdot otherfactor_1 + \ldots + \gamma_k \cdot otherfactor_k + \varepsilon \]
Culture Factor in CMC-based Communications

Collaborating Globally:

- preferences of CMC tools
- patterns of growing social network
- sentiments in conversations
Preferences of CMC Tools

IM vs. Email

Calendar Meet vs. IM

<table>
<thead>
<tr>
<th>Country</th>
<th>IM</th>
<th>IM&amp;EM</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Calendar Meet</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ratings:
- **only IM**
- **both IM&EM**
- **only EM**
- **only CA**
- **both CA&IM**
- **only IM**
Growing one’s Social Networks

![Graph showing the growth of social networks over time for different countries. The x-axis represents time (by month), and the y-axis represents the number of gained friends. The graph includes lines for the US (blue solid), DE (black dotted), CN (red solid), IN (green dotted), and JP (gray dotted).]
Sentiments in Conversation
Role Analysis

Role difference of normal behavior
Information Reuse Behavior (CHI ’11)

Percentage of slides with reused content

Number of slide pairs with exact vs. partial text reuse

Percentage of reused slides that were reused by the same author vs. by a different author

Percentage of downloaded material being reused
Anomaly Detection – algorithms and infrastructure

▪ **Thrust 1:** Anomaly Detection Algorithms
  -- New algorithms to detect abnormal humans (nodes) as well as abnormal contacts (edges) from social networks.
  -- Explore the structure feature and incorporate content (semantic) features.

▪ **Thrust 2:** Anomaly Usability
  -- Address the ‘lack-of-the ground-truth’ issue by
    (1) Interpretation friendly properties (e.g., non-negativity, sparseness, etc) into the current anomaly detection matrix factorization; and
    (2) providing some concise summarization to perform anomaly attribution.

▪ **Thrust 3:** Infrastructure Support
  -- General and scalable graph/network management system to process large
An analysis in a telecomm area of 6 million users in 2009.

In experiment
- Social Network Analysis is with recall of 89.97% and precision of 88.17% while comparison system is with 66.77% recall and 14.85% precision.
- SNA’s precision/recall area is 8 times larger.
Anomaly Detection – information flow-based approach

An illustrative example of an information spreading tree. This tree is of size 8, width 4, depth 3.

Probability ratio of email forwarding as a function of (a) hierarchical level difference and (b) organizational distance between initiators and spreaders. The information spreading exhibits some homophily effect.

Video demo: http://smallblue.research.ibm.com/demos/
Example 1: Internet Map
Nodes: ISPs; Edges: Connection
(33K Nodes, 290K edges)

Example 2: Social Network
Nodes: People; Edges: Friendship
(FaceBook has 500M+ Users)

Example 3: Web Graph
Nodes: Web Pages; Edges: Hyperlinks
(Yahoo Web: 1.4B nodes, 6.6B edges)

Multiple Scales, Multiple Disciplines
“Who are the most important actors?”

Three centralities
Degree: # of neighbors
Closeness: avg. shortest path length
Betweenness: # of times a node sits between shortest path

Application
Measuring the financial company value
Network attack monitoring

For 2 Billion Edges, standard closeness: 30,000 years
Graph Partitioning

Edge Cut

Vertex Cut
Distributed Graph Computation in GraphX

Property Graph

2D Vertex Cut Heuristic

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)
Example -- we proposed two new centralities (‘effective closeness’ and ‘LineRank’), and efficient large scale algorithms for billion-scale graphs.

Scalability Results
(Near-linear scalability)

For 2 Billion Edges,
- standard closeness: 30,000 years
- effective closeness: ~ 1 day!
1,000,000 times faster!

Kang, Tong, Sun, Lin, and Faloutsos, “GBase: A Scalable and general graph management system”, KDD 2011
RDF and SPARQL
WHAT DO RDF AND SPARQL BRING TO BIG DATA PROJECTS?

Bob DuCharme
TopQuadrant, Charlottesville, Virginia

Resource Description Format (RDF)

- A W3C standard since 1999
- Triples
- Example: A company has nine of part p1234 in stock, then a simplified triple representing this might be \{p1234 inStock 9\}.
- Instance Identifier, Property Name, Property Value.
- In a proper RDF version of this triple, the representation will be more formal. They require uniform resource identifiers (URIs).

```PREFIX fbd:<http://foobarco.net/data/>.
PREFIX fbv:<http://foobarco.net/vocab/>.

fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:supplier "Joe’s Part Company"."
An example complete description

@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.
fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe’s Part Company".
fbd:s9483 fbv:HomePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
Advantages of RDF

- Virtually any RDF software can parse the lines shown above as self-contained, working data file.
  - You can declare properties if you want.
  - The RDF Schema standard lets you declare classes and relationships between properties and classes.
  - The flexibility that the lack of dependence on schemas is the first key to RDF's value.

- Split trips into several lines that won't affect their collective meaning, which makes sharding of data collections easy.
  - Multiple datasets can be combined into a usable whole with simple concatenation.

- For the inventory dataset's property name URIs, sharing of vocabulary makes easy to aggregate.
The following SPARQL query asks for all property names and values associated with the fbd:s9483 resource:

```
PREFIX fbd: <http://foobarco.net/data/>
SELECT ?property ?value
WHERE { fbd:s9483 ?property ?value. }
```

The heart of any SPARQL query is the WHERE clause, which specifies the triples to pull out of the dataset. Various options for the rest of the query tell the SPARQL processor what to do with those triples, such as sorting, creating, or deleting triples. The above query’s WHERE clause has a single triple pattern, which resembles a triple but may have variables substituted for any or all of the triple’s three parts. The triple pattern above says that we’re interested in triples that have fbd:s9483 as the subject and—because variables function as wildcards—anything at all in the triple’s second and third parts.
The SPAQRL Query Result from the previous example

<table>
<thead>
<tr>
<th>property</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://foobarco.net/vocab/contactEmail">http://foobarco.net/vocab/contactEmail</a></td>
<td>&quot;<a href="mailto:gina.smith@joespartco.com">gina.smith@joespartco.com</a>&quot;</td>
</tr>
<tr>
<td><a href="http://foobarco.net/vocab/contactName">http://foobarco.net/vocab/contactName</a></td>
<td>&quot;Gina Smith&quot;</td>
</tr>
<tr>
<td><a href="http://foobarco.net/vocab/name">http://foobarco.net/vocab/name</a></td>
<td>&quot;Joe's Part Company&quot;</td>
</tr>
</tbody>
</table>
Another SPARQL Example

What is this query for?

```sparql
PREFIX fbd:<http://foobarco.net/data/>
PREFIX fbv:<http://foobarco.net/vocab/>

SELECT ?flangeContactEmail
WHERE
{
    ?part fbv:name "Blue reverse flange".
    ?supplier fbv:contactEmail ?flangeContactEmail.
}
```

Data

```
@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.
fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe’s Part Company".
fbd:s9483 fbv:homePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
```
Open Source Software – Apache Jena

A free and open source Java framework for building Semantic Web and Linked Data applications.

Get started now!  Download

RDF

RDF API

Interact with the core API to create and read Resource Description Framework (RDF) graphs. Sanitise your triples using popular formats such as RDF/XML or Turtle.

ARQ (SPARQL)

Query your RDF data using ARQ, a SPARQL 1.1 compliant engine. ARQ supports remote federated.

Triple store

TDB

Persist your data using TDB, a native high performance triple store. TDB supports the full range of Jena APIs.

Fuseki

Expose your triples as a SPARQL end-point accessible over HTTP. Fuseki provides REST-style interaction with your RDF data.

OWL

Ontology API

Work with models, RDFS and the Web Ontology Language (OWL) to add extra semantics to your RDF data.

Inference API

Reason over your data to expand and check the content of your triple store. Configure your own inference rules or
Property Graphs
A usual example

**Figure 2-1. Semantic relationships are hidden in a relational database**
Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking “who are Bob’s friends?” is easy, as shown in Example 2-1.

Example 2-1. Bob’s friends

SELECT p1.Person  
FROM Person p1 JOIN PersonFriend  
  ON PersonFriend.FriendID = p1.ID  
JOIN Person p2  
  ON PersonFriend.PersonID = p2.ID  
WHERE p2.Person = 'Bob'
Example 2-2. Who is friends with Bob?

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
    ON PersonFriend.PersonID = p1.ID
JOIN Person p2
    ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice’s friends-of-friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
    ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
    ON pf2.PersonID = pf1.FriendID
JOIN Person p2
    ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```
Graph Database Example

![Graph Database Example Diagram](image)

*Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph*
Partner and Vukotic’s experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

<table>
<thead>
<tr>
<th>Depth</th>
<th>RDBMS execution time (s)</th>
<th>Neo4j execution time (s)</th>
<th>Records returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.016</td>
<td>0.01</td>
<td>~2500</td>
</tr>
<tr>
<td>3</td>
<td>30.267</td>
<td>0.168</td>
<td>~110,000</td>
</tr>
<tr>
<td>4</td>
<td>1543.505</td>
<td>1.359</td>
<td>~600,000</td>
</tr>
<tr>
<td>5</td>
<td>Unfinished</td>
<td>2.132</td>
<td>~800,000</td>
</tr>
</tbody>
</table>
Modeling Order History as a Graph

Figure 2-6. Modeling a user’s order history in a graph
Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here’s the equivalent ASCII art representation in Cypher:

(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
Like most query languages, Cypher is composed of clauses. The simplest queries consist of a START clause followed by a MATCH and a RETURN clause (we’ll describe the other clauses you can use in a Cypher query later in this chapter). Here’s an example of a Cypher query that uses these three clauses to find the mutual friends of user named Michael:

```
START a=node:user(name='Michael')
MATCH (a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```
WHERE
    Provides criteria for filtering pattern matching results.

CREATE and CREATE UNIQUE
    Create nodes and relationships.

DELETE
    Removes nodes, relationships, and properties.

SET
    Sets property values.

FOREACH
    Performs an updating action for each element in a list.

UNION
    Merges results from two or more queries (introduced in Neo4j 2.0).

WITH
    Chains subsequent query parts and forward results from one to the next. Similar to piping commands in Unix.
Property Graph Example – Shakespeare

![Diagram](image)

**Figure 3-6. Three domains in one graph**
Creating the Shakespeare Graph

CREATE (shakespeare { firstname: 'William', lastname: 'Shakespeare' }),
( juliusCaesar { title: 'Julius Caesar' }),
( shakespeare)::WROTE_PLAY { year: 1599 }>>>(juliusCaesar),
( theTempest { title: 'The Tempest' }),
( shakespeare)::WROTE_PLAY { year: 1610 }>>>(theTempest),
( rsc { name: 'RSC' }),
( production1 { name: 'Julius Caesar' }),
( rsc)::PRODUCED>>>(production1),
( production1)::PRODUCTION_OF>>>(juliusCaesar),
( performance1 { date: 20120729 } ),
( performance1)::PERFORMANCE_OF>>>(production1),
( production2 { name: 'The Tempest' } ),
( rsc)::PRODUCED>>>(production2),
( production2)::PRODUCTION_OF>>>(theTempest),
( performance2 { date: 20061121 } ),
( performance2)::PERFORMANCE_OF>>>(production2),
( performance3 { date: 20120730 } ),
( performance3)::PERFORMANCE_OF>>>(production1),
( billy { name: 'Billy' } ),
( review { rating: 5, review: 'This was awesome!' } ),
( billy)::WROTE_REVIEW>>>(review),
( review)::RATED>>>(performance1),
( theatreRoyal { name: 'Theatre Royal' } ),
( performance1)::VENUE>>>(theatreRoyal),
( performance2)::VENUE>>>(theatreRoyal),
( performance3)::VENUE>>>(theatreRoyal),
( greyStreet { name: 'Grey Street' } ),
( theatreRoyal)::STREET>>>(greyStreet),
( newcastle { name: 'Newcastle' } ),
( greyStreet)::CITY>>>(newcastle),
( tyneAndWear { name: 'Tyne and Wear' } ),
( newcastle)::COUNTY>>>(tyneAndWear),
( england { name: 'England' } ),
( tyneAndWear)::COUNTRY>>>(england),
( stratford { name: 'Stratford upon Avon' } ),
( stratford)::COUNTRY>>>(england),
( rsc)::BASED_IN>>>(stratford),
( shakespeare)::BORN_IN>>>(stratford)
Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
     newcastle=node:city(name='Newcastle'),
     bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[:STREET|CITY*1..2]-(theater)
    <[:VENUE]-()-[:PERFORMANCE_OF]->()-[:PRODUCTION_OF]->
    (play)<-[w:WROTE_PLAY]-(bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play
```

Adding this WHERE clause means that for each successful match, the Cypher execution engine checks that the WROTE_PLAY relationship between the Shakespeare node and the matched play has a year property with a value greater than 1608. Matches with a WROTE_PLAY relationship whose year value is greater than 1608 will pass the test; these plays will then be included in the results. Matches that fail the test will not be included in the results. By adding this clause, we ensure that only plays from Shakespeare’s late period are returned:

```
+-----------------------+
| play                  |
+-----------------------+
| "The Tempest"         |
+-----------------------+
1 row
```
Another Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
    newcastle=node:city(name='Newcastle'),
    bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[[:STREET][CITY*1..2]]-(theater)
    <-[[:VENUE]-()-[p:PERFORMANCE_OF]-(play)<-[[:WROTE_PLAY]]-(bard)
RETURN play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC
```

The RETURN clause here counts the number of PERFORMANCE_OF relationships using the identifier p (which is bound to the PERFORMANCE_OF relationships in the MATCH clause) and aliases the result as performance_count. It then orders the results based on performance_count, with the most frequently performed play listed first:

```
+-----------------+------------------+
| play            | performance_count |
+-----------------+------------------+
| "Julius Caesar" | 2                 |
| "The Tempest"   | 1                 |
+-----------------+------------------+
2 rows
```
Building Application Example – Collaborative Filtering

Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```cypher
START reader=node:users(name={readerName})
MATCH reader[:LIKES]->book<-[:LIKES]-other_readers[:LIKES]->books
RETURN books.title
```
Chaining on the Query

```
START bard=node:author(lastname='Shakespeare')
MATCH (bard)-[w:WROTE_PLAY]->(play)
WITH play
ORDER BY w.year DESC
RETURN collect(play.title) AS plays
```

Executing this query against our sample graph produces the following result:

```
<table>
<thead>
<tr>
<th>plays</th>
</tr>
</thead>
<tbody>
<tr>
<td>[&quot;The Tempest&quot;,&quot;Julius Caesar&quot;]</td>
</tr>
</tbody>
</table>
```

1 row
What's this query for?

```
START bob=node:user(username='Bob')
MATCH (bob)-[:SENT]->(email)-[:CC]->(alias),
    (alias)-[:ALIAS_OF]->(bob)
RETURN email
```
How to make graph database fast?

Figure 6-1. Nonnative graph processing engines use indexing to traverse between nodes
Use Relationships, not indexes, for fast traversal
Storage Structure Example

Figure 6-4. Neo4j node and relationship store file record structure
Nodes and Relationships in the Object Cache

Figure 6-6. Nodes and relationships in the object cache
An Emerging Benchmark

Test Set:
data generator of full social media activity simulation of any number of users
Questions?