Large-Scale and Multi-Structured Databases

Graph Databases Design

Prof. Pietro Ducange

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Which Database?

- When dealing with NoSQL databases, it is ***not easy*** to decide which ***solution*** should be adopted.

- In general, data base designer should have in mind the main ***kinds of queries*** that the application will perform on the data.

- Graph databases are ***suitable*** for problem domains easily described in terms of entities and relations.

Typical Queries for Graph Databases

- How many hops (that is, edges) does it take to get from vertex A to vertex B?
- How many edges between vertex A and vertex B have a cost that is less than 100?
- How many edges are linked to vertex A?
- What is the centrality measure of vertex B?
- Is vertex C a bottleneck; that is, if vertex C is removed, which parts of the graph become disconnected?

Some Considerations

In the previous examples of queries, there is ***less emphasis*** on:

- ***selecting*** by particular properties (how many vertices have at least 10 edges?)

- ***aggregating*** values across a group of entities (selects all customer orders from the Northeast placed in the last month and sum the total value of those orders)

Moreover, the previous queries are ***fairly abstract (computer science/math domain).***

Designer needs to ***translate*** queries from the ***problem domain*** to the ***domain of experts*** working on graphs.

First Example: HollywoodDB

Let suppose to design a DB for handling ***movies***, ***actors*** and ***directors***.

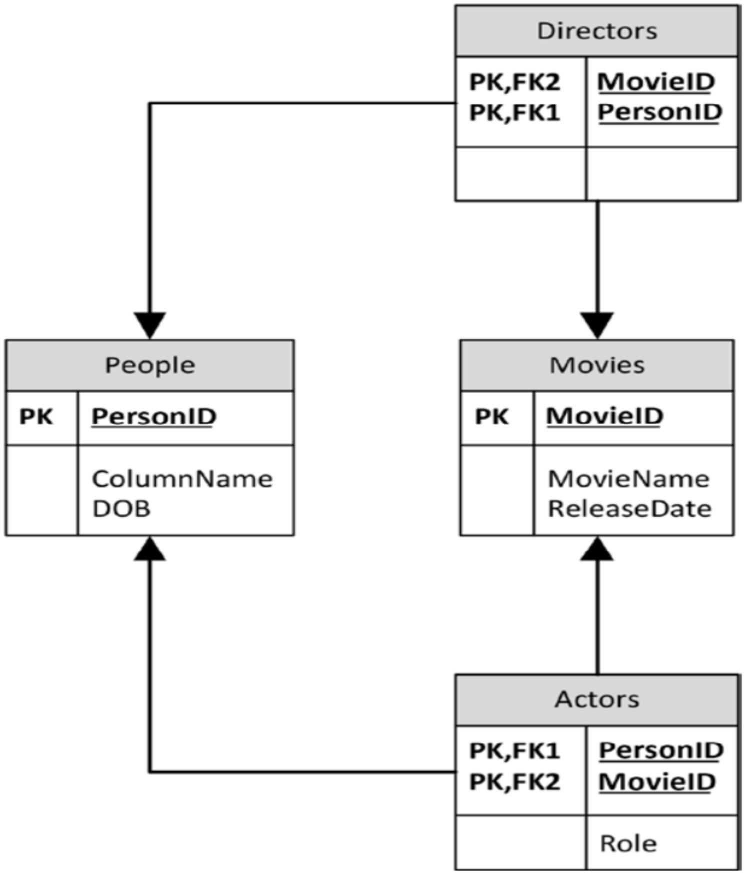
Let suppose to offer the following ***services*** for users:

1. Find all the actors that have ever appeared as a co-star in a movie starring a specific actor (for example: Keanu Reeves).

2. Find all movies and actors directed by a specific director (for example Andy Wachowski).

3. Find all movies in which specific actor has starred in, together with all co-stars and directors

HollywoodDB: Relational Models



Problems:

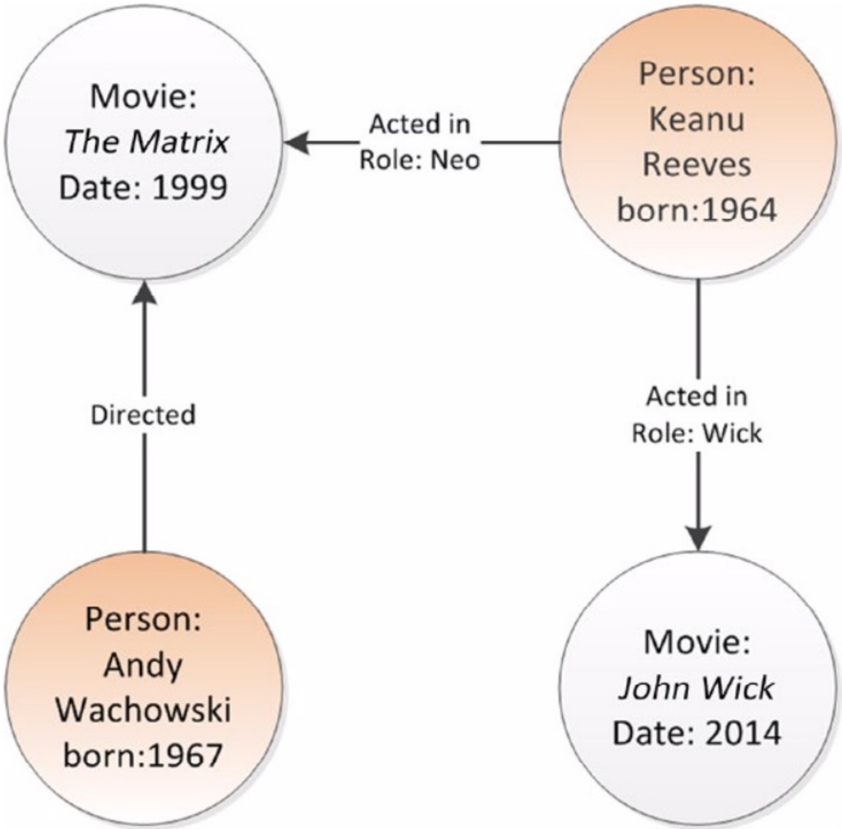
- ***SQL lacks the syntax*** to easily perform ***graph traversal***, especially traversals where the depth is unknown or ***unbounded***.
- ***Performance degrades*** quickly as we traverse the graph. Each ***level of traversal*** adds significantly response time to a specific query .

HollywoodDB: Relational Models

In the following, we show a SQL query for finding all actors who have ever worked with Keanu Reeve.

```sql
SELECT p2.personname, m1.movieName
  FROM people p1
       JOIN actors a1 ON (p1.personid = a1.personid)
       JOIN movies m1 ON (a1.movieid = m1.movieid)
       JOIN actors a2 ON (a2.movieid = m1.movieid)
       JOIN people p2 ON (p2.personid = a2.personid)
 WHERE p1.personname = 'Keanu Reeves';
```

HollywoodDB: Graph Model

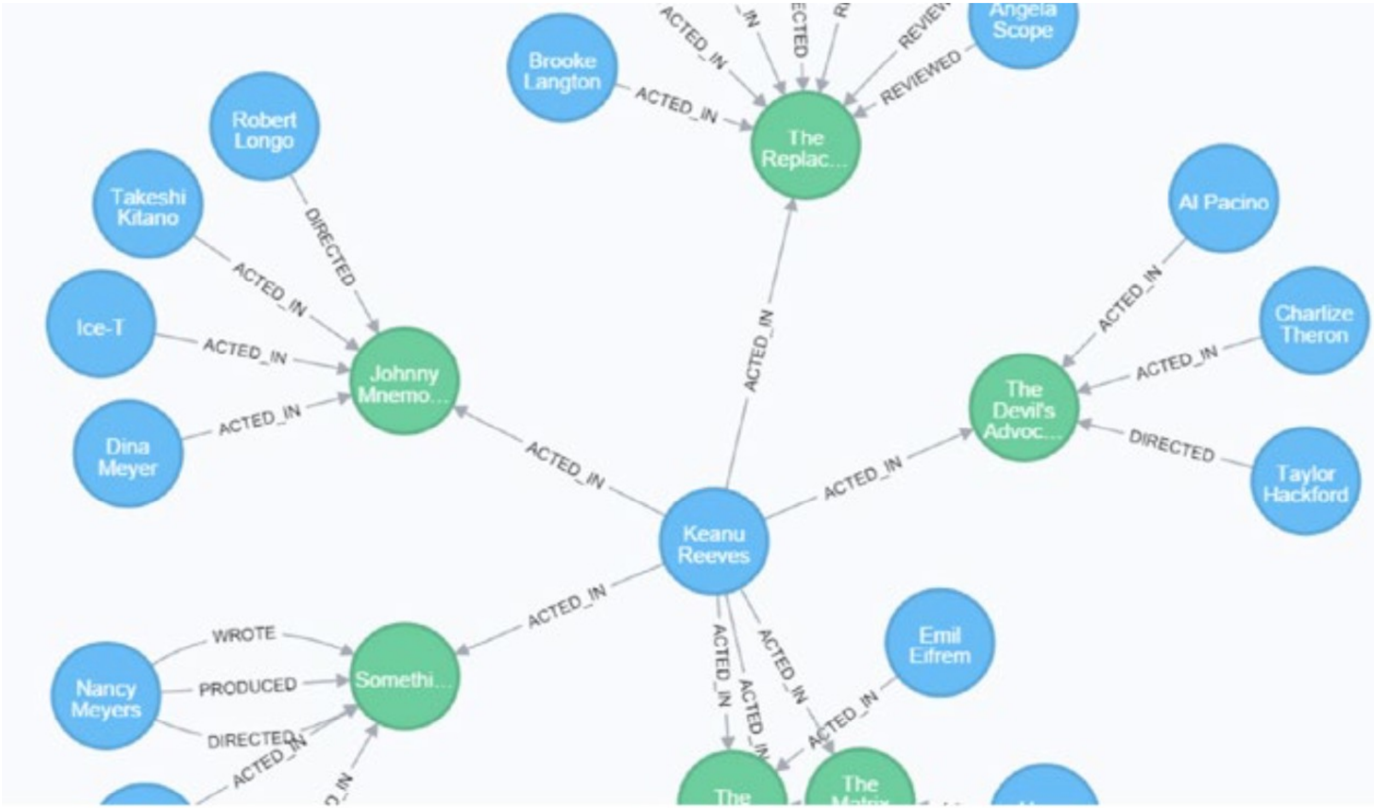


DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

HollywoodDB: Graph Model




```
neo4j-sh (?)$ MATCH (kenau:Person {name:"Keanu Reeves"})
               -[:ACTED_IN]->(movie)<-[:ACTED_IN]-(coStar)
              RETURN coStar.name;
```

DESIGNING A SOCIAL NETWORK FOR NOSQL DATABASE DEVELOPERS

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

Use Cases

Let consider the database developer as the main actor of the social network. The main use cases may be:

1. ***Join*** and ***leave*** the site
2. ***Follow*** the postings of other developers
3. ***Post*** questions for others with expertise in a particular area
4. ***Suggest*** new connections with other developers based on shared interests
5. ***Rank*** members according to their number of connections, posts, and answers

Entities and their Properties

Developers:

- Name
- Location
- NoSQL databases used
- Years of experience
- Areas of interest

Posts:

- Date created
- Topic keywords
- Post type (for example, question, tip, news)
- Title
- Body of post

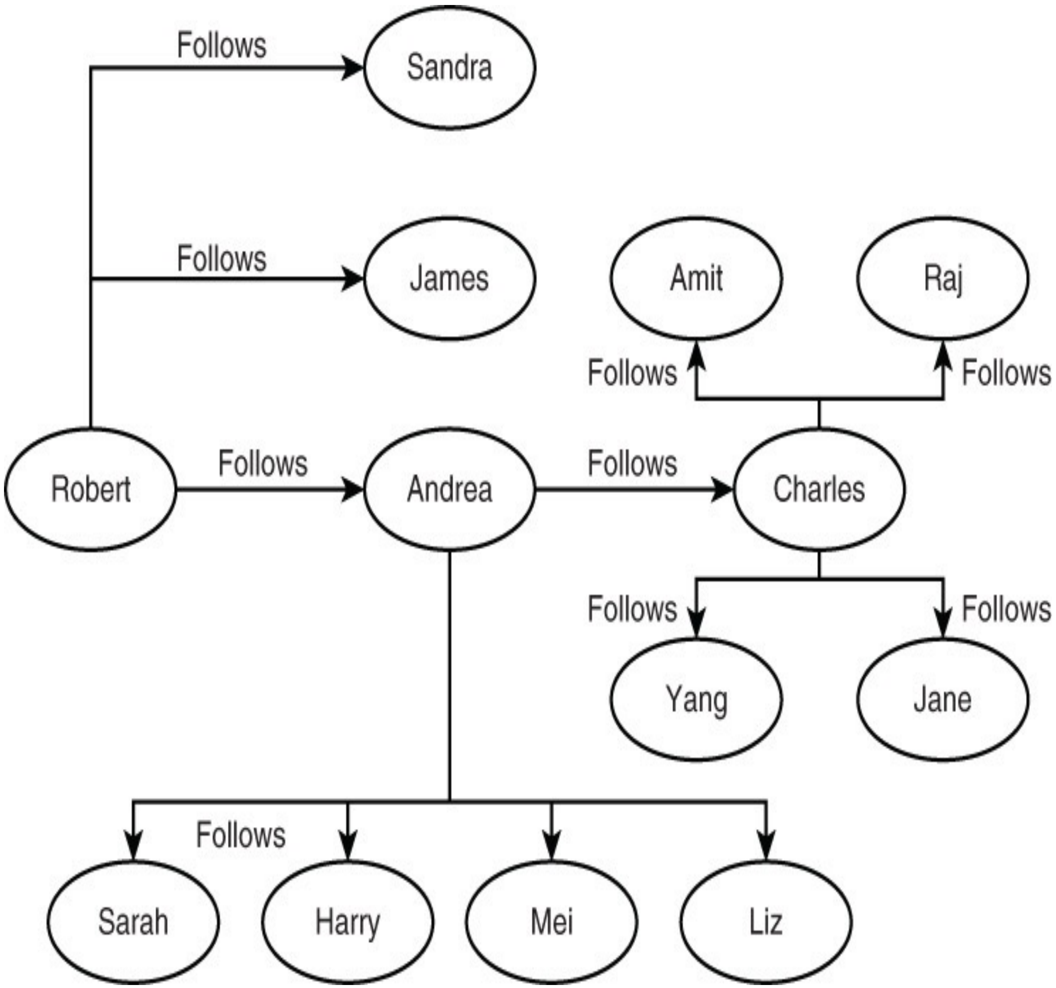
Analyzing Relations

Considering the two entities in our domain, the following ***candidate*** relations can be analyzed:

- Developer - Developer
- Developer - post
- Post - Developer
- Post - Post

Notice that, instead of considering ***all the possible combinations*** for the identification of candidate relations, as the ***number of entities grows***, it is better to focus on combinations that are ***reasonably*** likely to ***support*** queries related to the ***application requirements***.

Developer –Follows- Developer

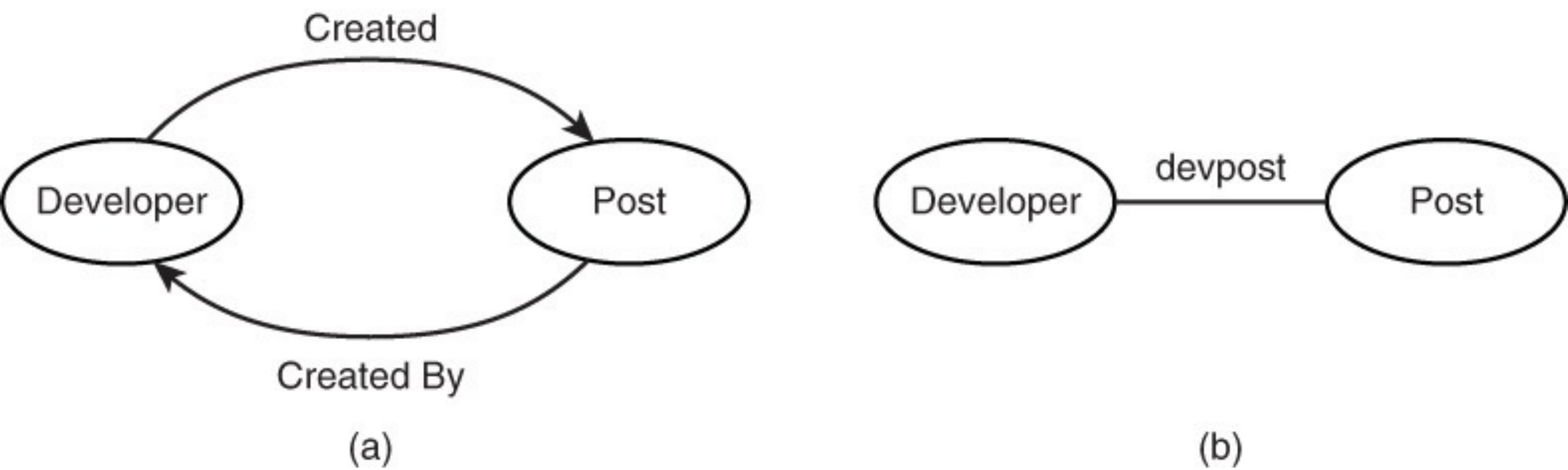


If a developer ***follows*** another developer, it is reasonable that he/she is ***interested*** in reading his/her ***posts***.

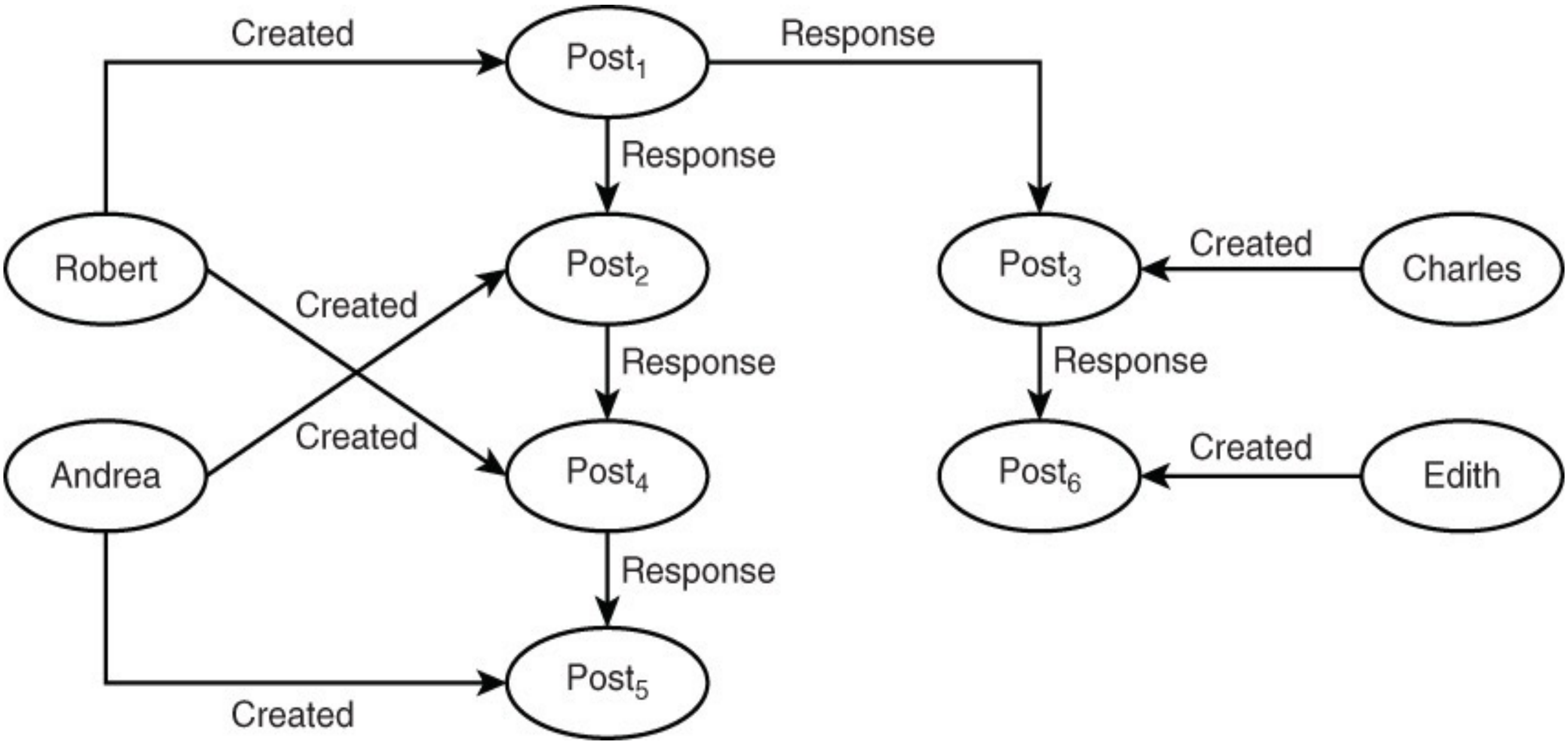
The ***depth of the path***, namely how many edges to traverse for identifying a followed developer, is a parameter that can be defined and handled at the level of ***application code***.

Relations between Developer and Post

In the figure below, we show a Developer and a Post with ***two directed*** edges (a) or ***one*** edge ***not direct***



Post-Post Relation



A post may be created in ***response*** to another post!

Mapping Queries

Remember queries drive the design of graph databases.

Domain-specific queries must ***be translated*** into ***Graph-centric*** queries, after the ***identification*** of ***entities*** and ***relations*** between entities.

Graph-Centric Query	Domain-Specific Query
How many hops (that is, edges) does it take to get from vertex A to vertex B?	How many follows relations are between Developer A and Developer B?
How many incoming edges are incident to vertex A?	How many developers follow Andrea Wilson?
What is the centrality measure of vertex B?	How well connected is Edith Woolfe in the social network?
Is vertex C a bottleneck; that is, if vertex C is removed, which parts of the graph become disconnected?	If a developer left the social network, would there be disconnected groups of developers?

Basic Steps for Designing GraphDB

- Identify the ***queries*** you would like to perform.
- Identify ***entities*** in the graph.
- Identify ***relations*** between entities
- ***Map*** domain-specific ***queries*** to more abstract queries
- ***Implement*** graph ***queries*** and use graph ***algorithms***

Some Advices

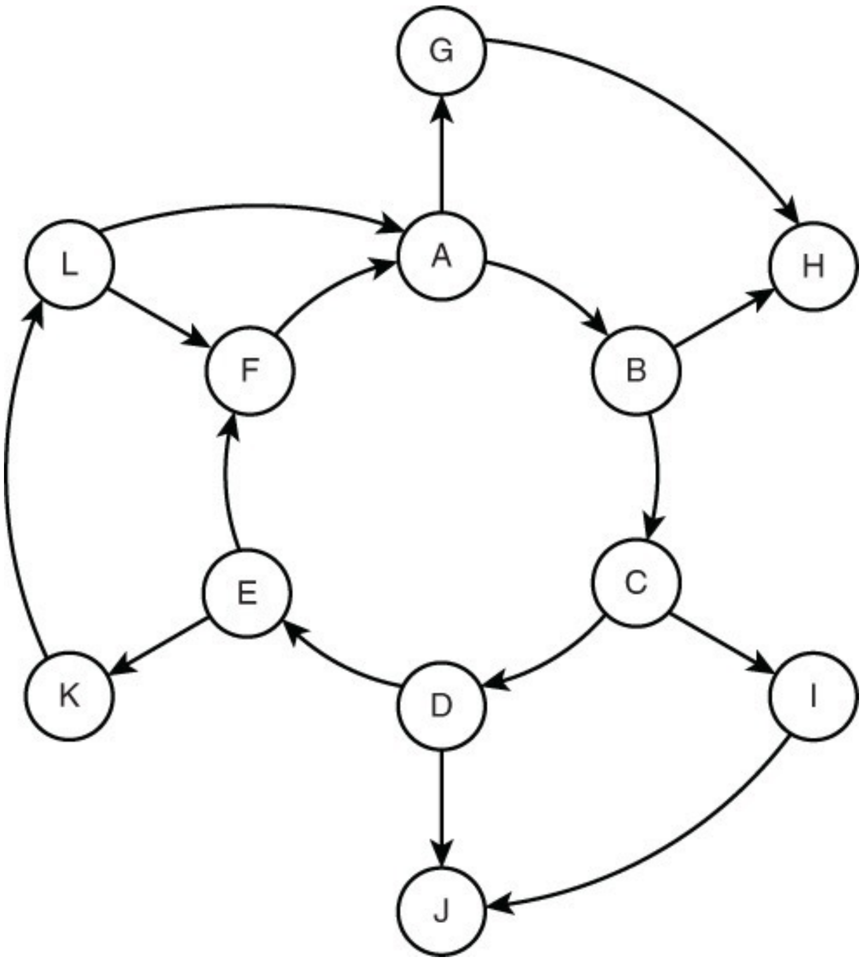
1. Pay attention to the ***dimension*** of the graph: the bigger the graph the more ***computational expensive*** will be performing queries.

2. If available, define and use ***indexes*** for improving retrieval time.

3. Use ***appropriate*** type of ***edges***:
   a. In case of symmetrical relations, use undirected edge, otherwise use direct edges.
   b. Properties requires memory! If possible, ***use simple edge*** without properties. Pay also attention to the data type of the properties.

4. ***Watch for cycles*** when traversing graphs: in order to avoid endless cycles, use appropriate data structure for ***keeping track*** of visited nodes.

Cycles in Graphs



Pay attention if we need to start a traversal at vertex A and then followed a path through vertices B, C, D, E, and F, you would end up back at A.

If there is ***no indication*** that we have already visited A, we could find us in an endless cycle of revisiting the same six nodes over and over again.

Scalability of Graph Databases

Most of the famous implementation of graph databases do not consider the deployment on cluster of servers (***no data partitions nor replicas***).

Indeed, they are designed for running on a ***single server*** and can be only ***vertically scaled***.

Thus, the developer must take a special attention to the growing of:

- The number of nodes and edges
- The number of users
- The number and size of properties

Execution Times of Two Typical Algorithms

Number of Vertices	Time to Find Shortest Path	Time to Find Maximal Clique
1	2	2
10	200	1,024
20	800	1,048,576
30	1,800	1,073,741,824
40	3,200	1,099,511,627,776
50	5,000	1,125,899,906,842,620

Maximal clique: largest group of people who all follow each other

Suggested Readings

Chapter 14 of the book “*Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015*”.

Chapter 5 of the book “*Guy Harrison, Next Generation Databases, Apress, 2015*”.

Images

If not specified, the images shown in this lecture have been extracted from:

“Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015”