COMS E6998: **Microservices and Cloud Applications**

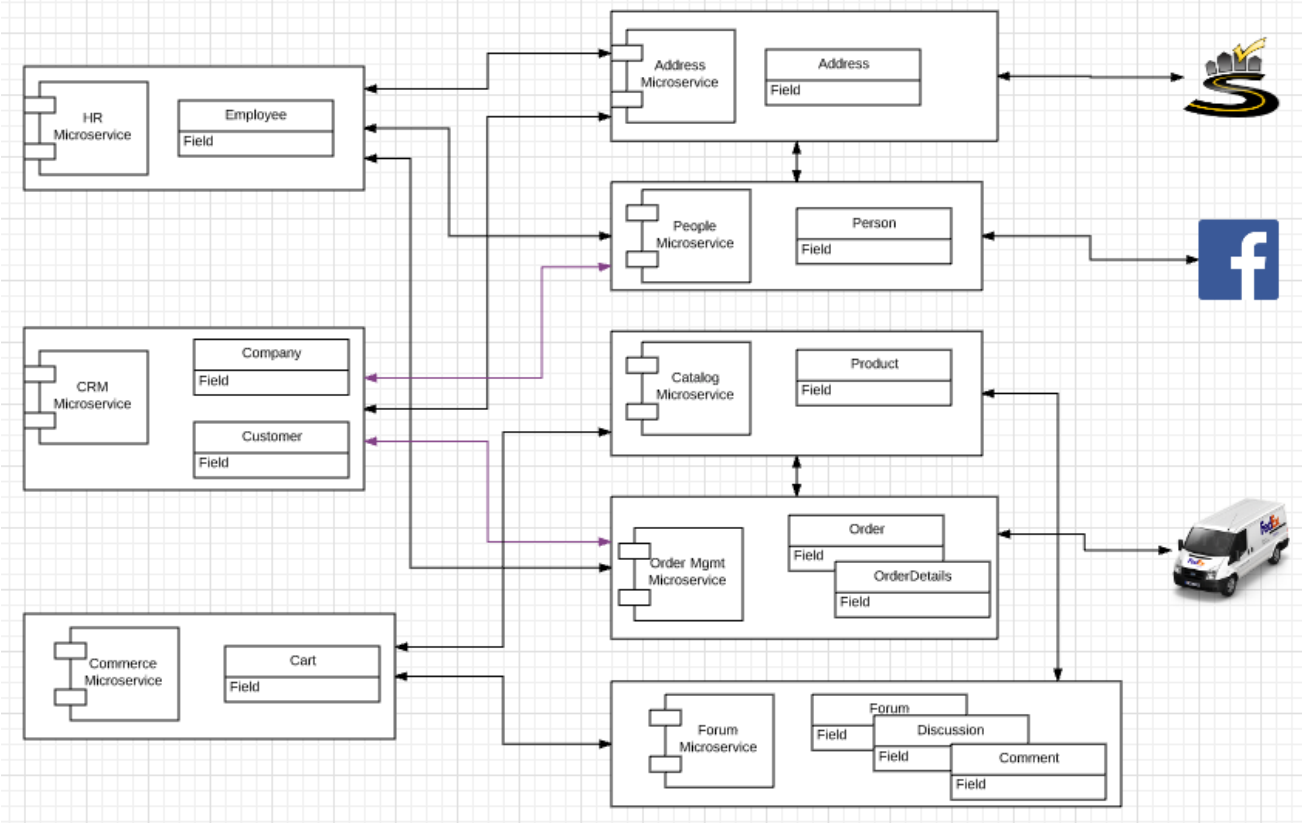
Lecture 10: Projects, 12 Factor Applications, Redis, Graph Databases

Dr. Donald F. Ferguson
dff9@columbia.edu

Comments Questions

Projects

Original Big Picture



0th Project

- Teams
  - Form your teams (approx. 5 people)
  - Identify contact focal point.
  - Give your team a “cool” name.
- Signup (reuse) and Amazon Web Service Account
  - Free Tier should be fine.
  - Provide access to team members.
- Create an Elastic Beanstalk instance/application.
  - Use one of the sample application.
  - Will have to make more sophisticated starting next week;
    I will use Node JS with Express for Elastic Beanstalk examples.

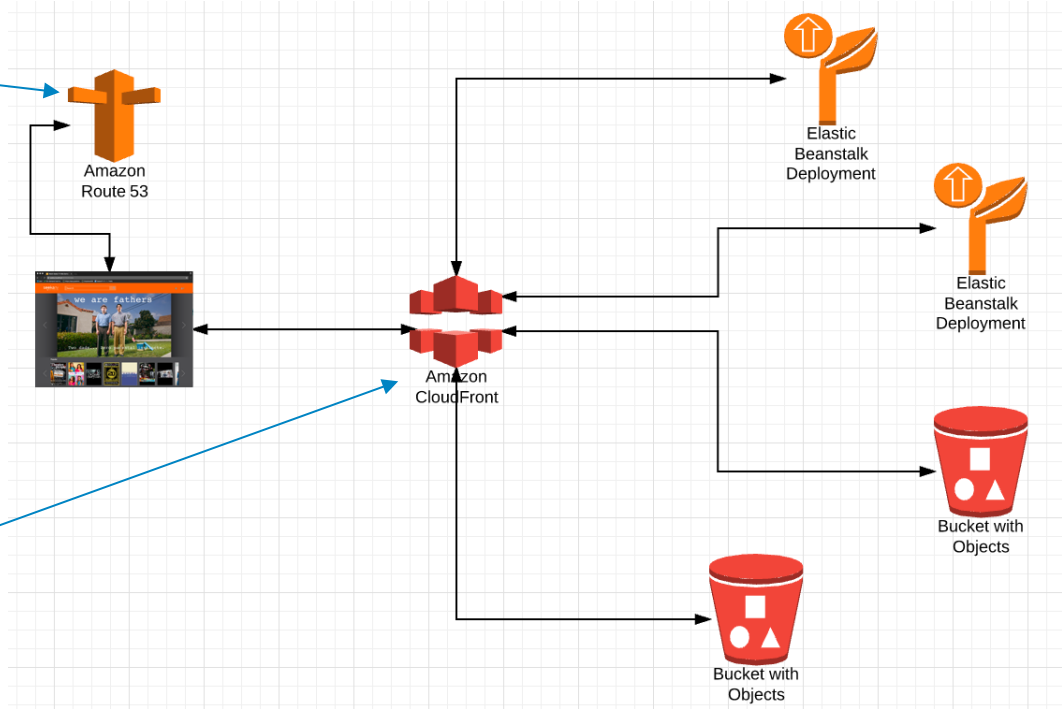
1st Project – Part 1

- Implement two distinct microservices
  - Person
  - Address
- Tasks
  - Use Swagger Editor to define and document REST APIs.
  - Implement an Elastic Beanstalk application (microservice) for each resource that implements the relevant REST API.
  - Each microservice should support
    - GET and POST on resource, e.g. /Person
    - GET, PUT, DELETE on resource/id, e.g. /Person/dff9
    - Simple query, e.g. /Person?lastName=Ferguson
    - Pagination
    - Relationship paths: /Person/dff9/address and /Addresses/someID/persons
    - HATEOAS links where appropriate.
  - Simple HTML/Angular demo UI.
- Due: 11:59 PM on 26-Sep-2017

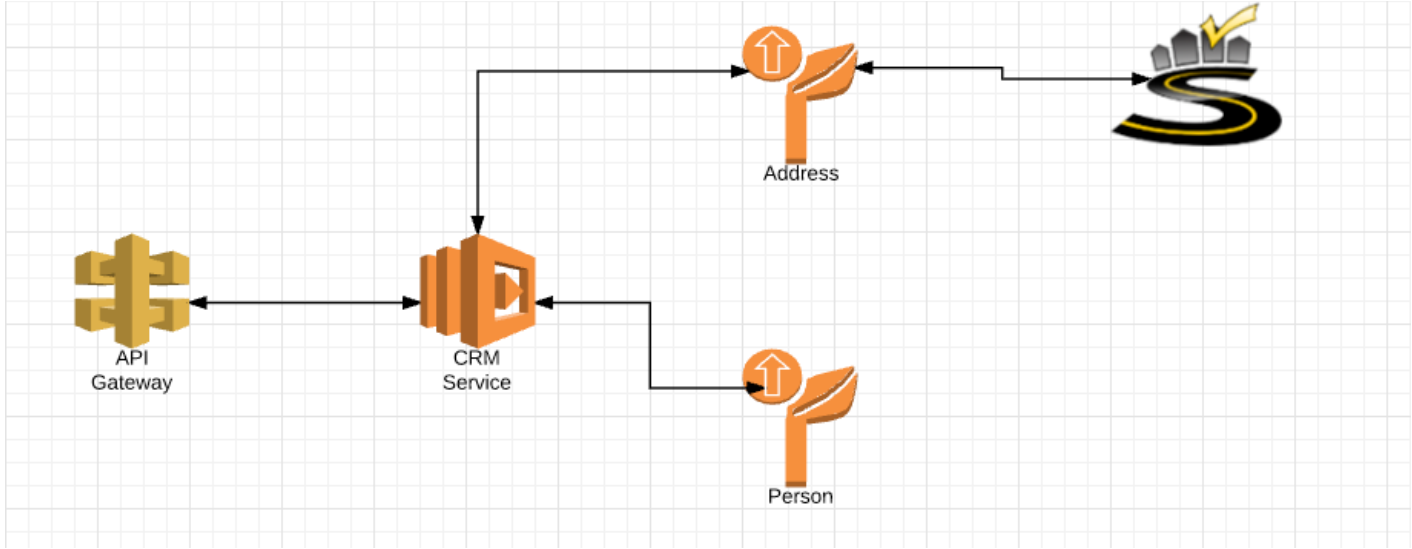
Single Site

- DNS
  - Resolve dff-company.com to
  - Something.amazon.com
  - Under the covers
- Map
  - /api/person
  - /api/address
  - /app
    - /js
    - /views
    - /app-content
  - /digital-assets
    - /images
    - /videos

To correct IP addresses and sub-paths



Project 2 – Part 1



Build on 1st to microservices

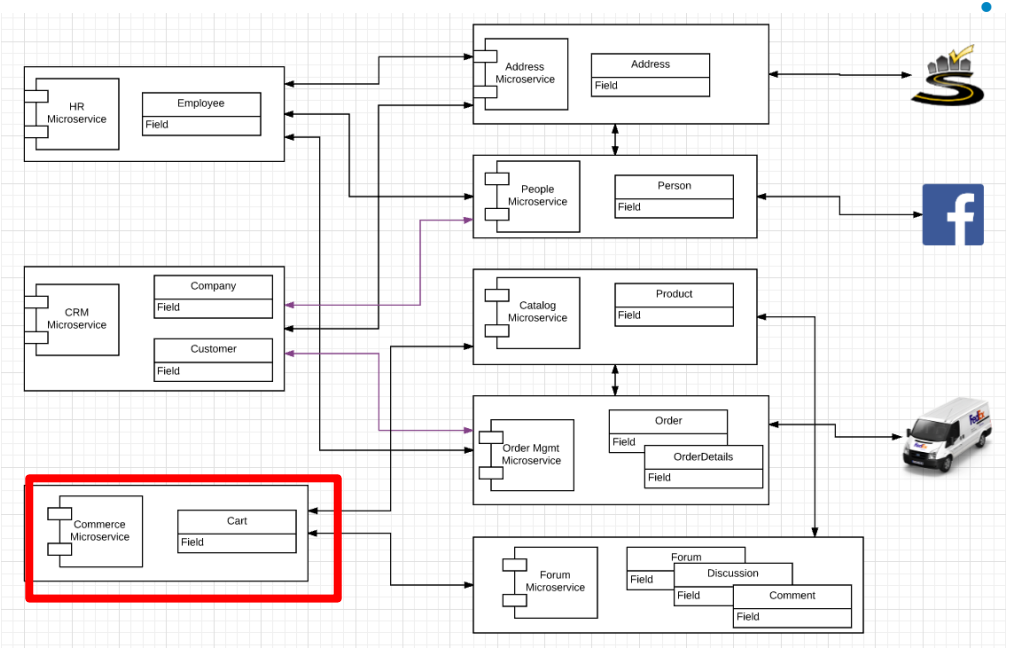
- Implement the CRM Service (and HR Service) using Lambda functions, and orchestration approach from previous lecture.
- Integrate with SmartyStreets
- Deploy all 3 microservices via API Gateway.

- Deliver web content via CloudFront and S3.

Project 2 – Part 2

- Implement middleware plugins and microservices implementing
  - ETag generation and processing.
  - Idempotent functions.
  - SNS Event generation on PUT, POST and DELETE.
- Plug/deploy the plugin layer in
  - Each of the Beanstalk Microservices
  - The CRM Lambda function.
- Write an empty, placeholder Lambda function that reacts to the SNS events. We will do some interesting things with this later.

Project 3 4 – Start/Expand Commerce Microservice



- Use Lambda function

Customers may

- Register with
- And subsequently logon with
- Facebook or Twitter

Just

- Implement register/logon
- We will later
  - Use Step Functions for composing microservices and APIs
  - Publish commerce actions to FB and Twitter.

Also, write

- A placeholder
- API Gateway Custom Authorizer
- Which we will later use to manager authorization to orders.

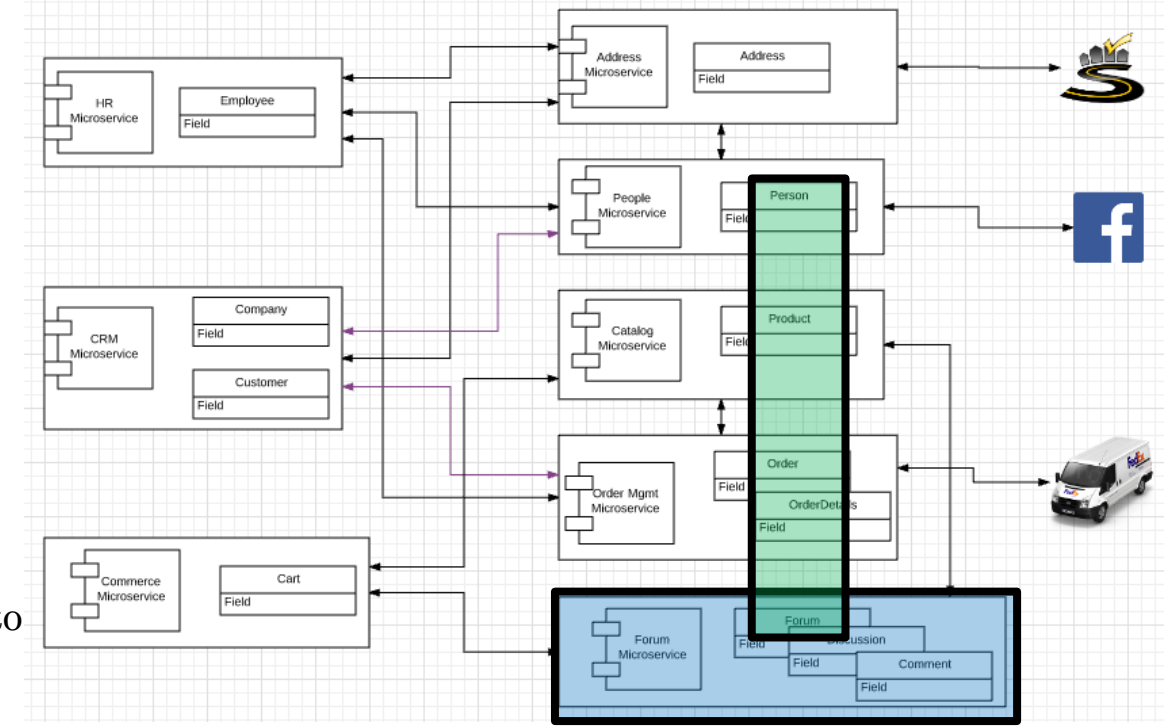
Last Week's Lecture – CAP and New Datamodels

The initial idea was

- DynamoDB for the forum
- Neo4J to track/query
  - Who bought what?
  - Who has bought things similar to whom?
  - Who commented on what?
  - etc.
- Use Redis to optimize
  - Idempotency
  - Etag

But

- we do not have enough time to do in context of solution.
- Will have to do smaller scenarios and use cases.



My View on Status

Project	Subtask	Element
Project 0	Set up team Run ELB	
Project 1	Implement 2 microservices	• **Implement** • **REST API/HATEOAS** • **Query, relationships**
	Swagger	• **Document, test** • Integrate with API GW, development tools, …
Project 2	Cloud APIs	• **SmartyStreets (UI, backend)**
	Orchestration	• In code
	Single Site Image	• S3 for content • CloudFront • API Gateway
	Middleware	• **ETag and idempotency technical microservice** • **SNS event generation**
Project 3	OAuth2	• **Logon, Register** • **Custom API GW Authorizer → Authorization**
Additional Goals		• **Step functions** • **SNS to integrate registration, commerce views, etc. with Neo4J** • **SNS → Swagger via WebHooks/Lambda** • **Redis: Optimize data sharing for middleware scenarios** • **Rules engines** • **Text search**

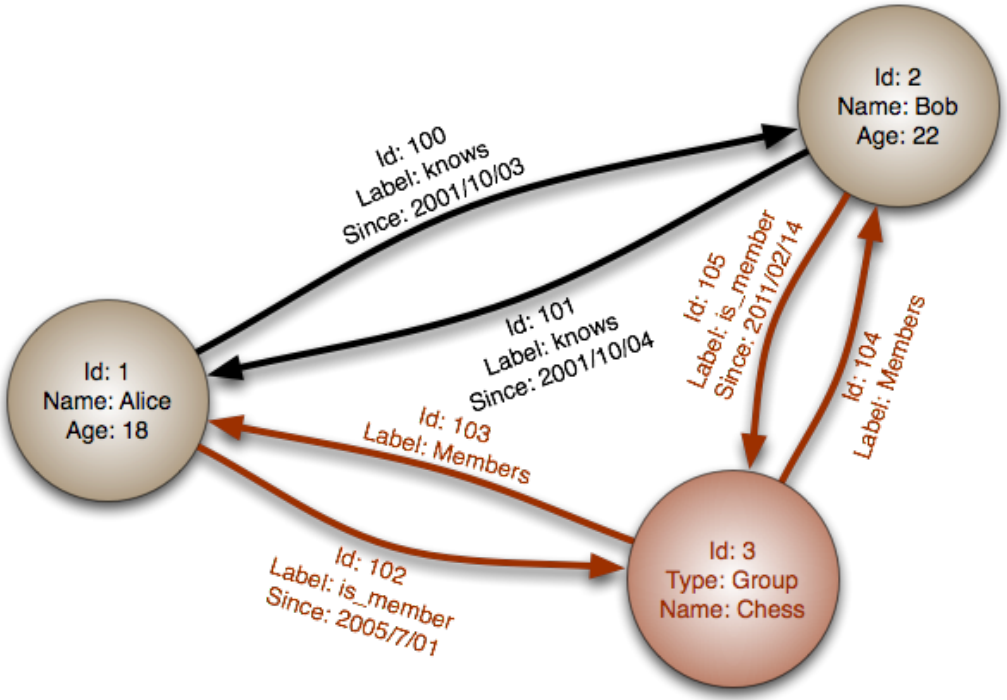
Timetable

- Project 2 and project 3
  - Complete (initial) reviews by 05-Dec.
  - Use 27-Nove and 05-Dec lectures if possible, then office hours, then extra OH.
- We will hold one final project review with each team
  - Covering all elements of projects 1, 2 and 3
    - Including modification/correction based on feedback from prior reviews.
    - I will provide a list of completion requirements metrics.
  - Project 4, which has two small subelements
    - Simple idempotency check microservice → Redis
      - POST token returns OK if token not previously seen.
      - POST returns error if token previously recorded.
    - Simple graph microservice on Neo4J using sample data.
      - POST a simple query command in JSON.
      - Returns results from Neo4J.

Graph Database

Graph Database

- Exactly what it sounds like
- Two core types
  - Node
  - Edge (link)
- Nodes and Edges have
  - Label(s) = “Kind”
  - Properties (free form)
- Query is of the form
  - p1(n)-p2(e)-p3(m)
  - n, m are nodes; e is an edge
  - p1, p2, p3 are predicates on labels



Why Graph Databases?

(https://www.slideshare.net/debanjanmahata/an-introduction-to-nosql-graph-databases-and-neo4j)

- Schema Less and Efficient storage of Semi Structured Information
- No O/R mismatch – very natural to map a graph to an Object Oriented language like Ruby.
- Express Queries as Traversals. Fast deep traversal instead of slow SQL queries that span many table joins.
- Very natural to express graph related problem with traversals (recommendation engine, find shortest parth etc..)
- Seamless integration with various existing programming languages.
- ACID Transaction with rollbacks support.
- Whiteboard friendly – you use the language of node,properties and relationship to describe your domain (instead of e.g. UML) and there is no need to have a complicated O/R mapping tool to implement it in your database. You can say that Neo4j is "Whiteboard friendly" !(http://video.neo4j.org/JHU6F/live-graph-session-how-allison-knows-james/)

Social Network "path exists" Performance

- Experiment:
  - ~1k persons
  - Average 50 friends per person
  - `pathExists(a,b)` limited to depth 4

	# persons	query time
Relational database	1000	2000ms
Neo4j	1000	2ms
Neo4j	1000000	2ms

Graph databases are

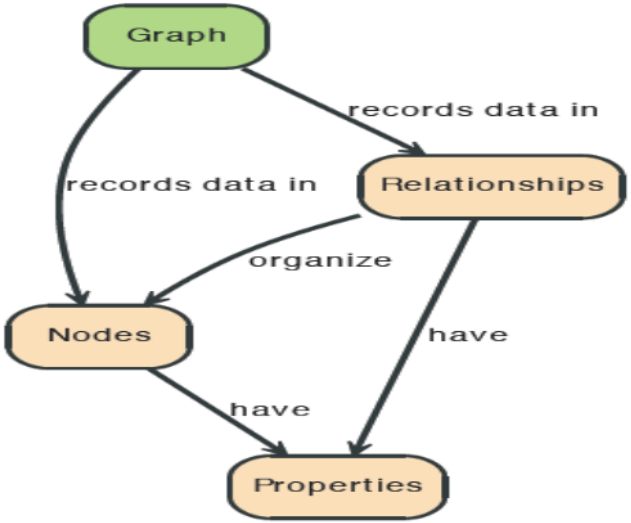
- Extremely fast for some queries and data models.
- Implement a language that vastly simplifies writing queries.

What are graphs good for?

- Recommendations
- Business intelligence
- Social computing
- Geospatial
- Systems management
- Web of things
- Genealogy
- Time series data
- Product catalogue
- Web analytics
- Scientific computing (especially bioinformatics)
- Indexing your *slow* RDBMS
- And much more!

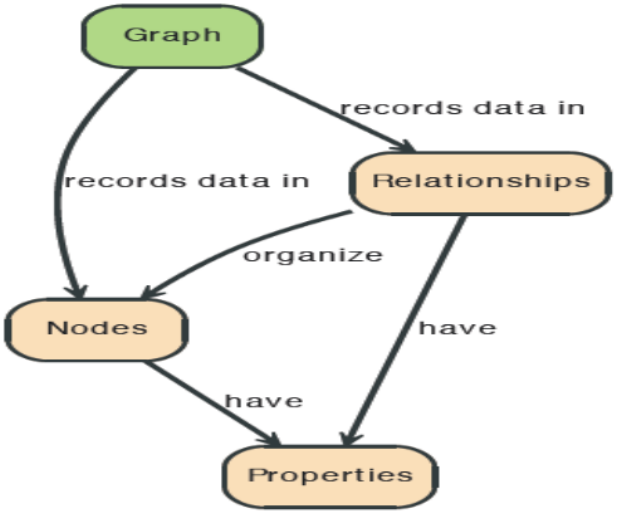
Graphs

- "A Graph —records data in→ Nodes —which have→ Properties"



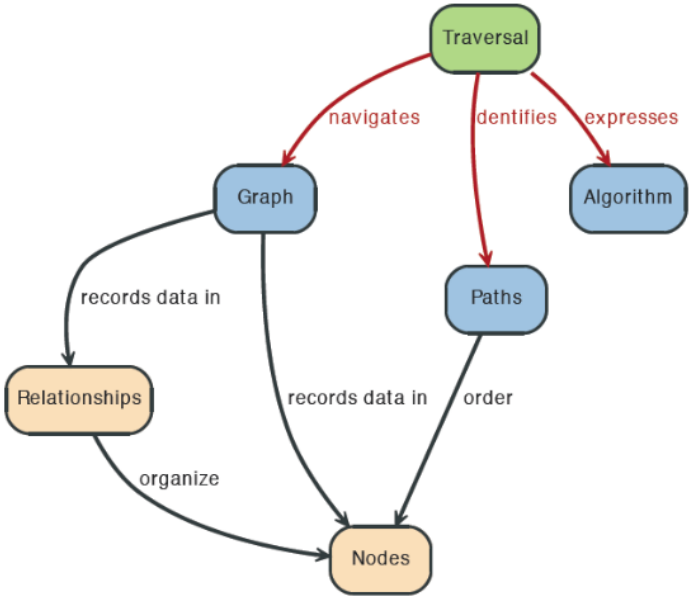
Graphs

- "Nodes —are organized by→ Relationships — which also have→ Properties"



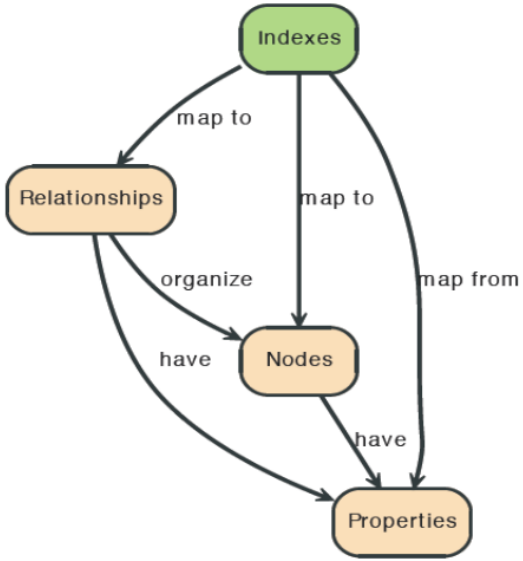
Query a graph with Traversal

- "A Traversal —navigates→ a Graph; it —identifies→ Paths —which order→ Nodes"

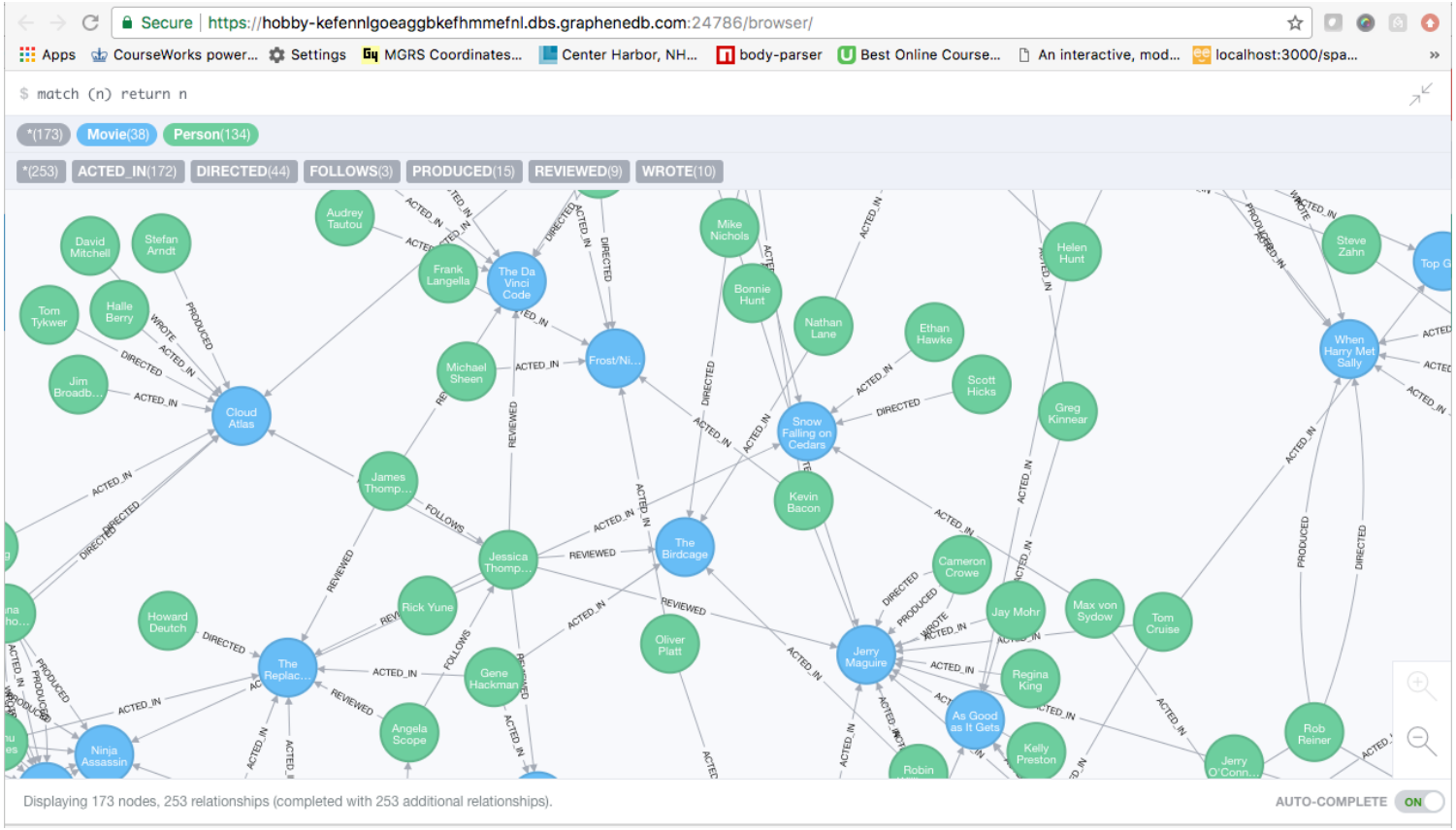


Indexes

- "An Index —maps from→ Properties —to either→ Nodes or Relationships"

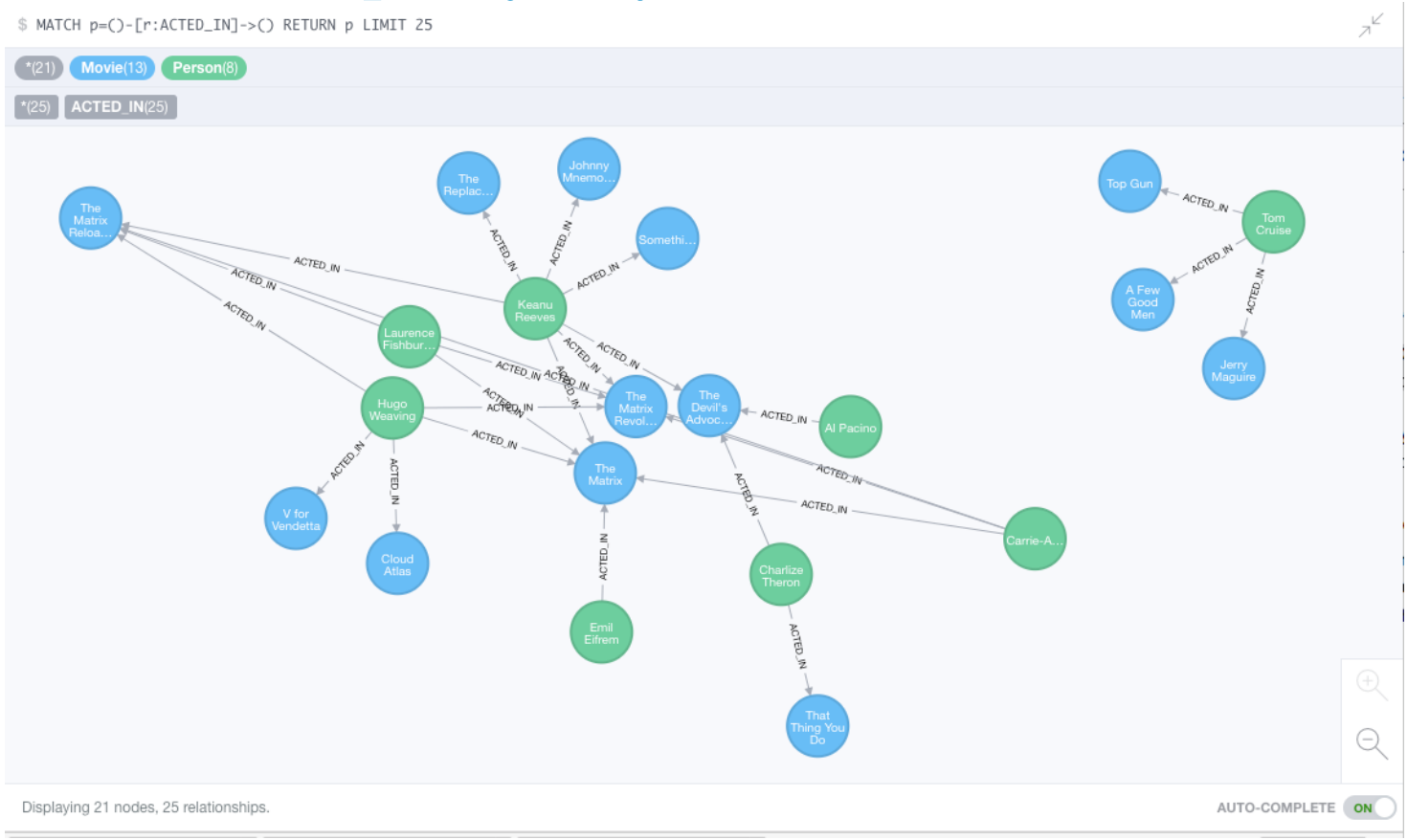


A Graph Database (Sample)



Neo4J Graph Query

Who acted in which movies?



Big Deal. That is just a JOIN.

- Yup. But that is simple.
- Try writing the queries below in SQL.

The Movie Graph

Recommend

Let's recommend new co-actors for Tom Hanks. A basic recommendation approach is to find connections past an immediate neighborhood which are themselves well connected.

For Tom Hanks, that means:

1. Find actors that Tom Hanks hasn't yet worked with, but his co-actors have.
2. Find someone who can introduce Tom to his potential co-actor.

Extend Tom Hanks co-actors, to find co-co-actors who haven't work with Tom Hanks...

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
      (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->(m2)
RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC
```

Find someone to introduce Tom Hanks to Tom Cruise

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
      (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

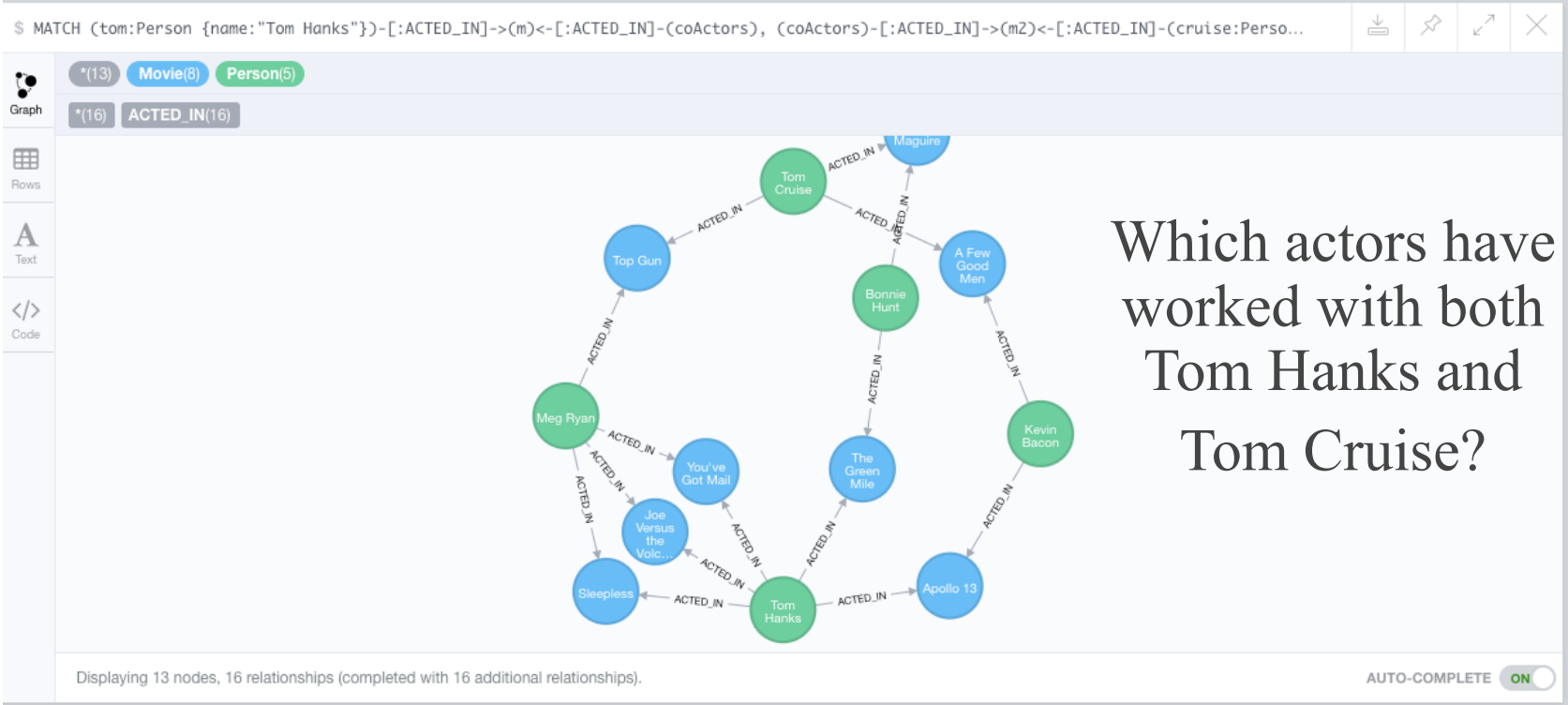
```
1 MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
2       (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
3 WHERE NOT (tom)-[:ACTED_IN]->(m2)
4 RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC
```

$ MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors), (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors) ...

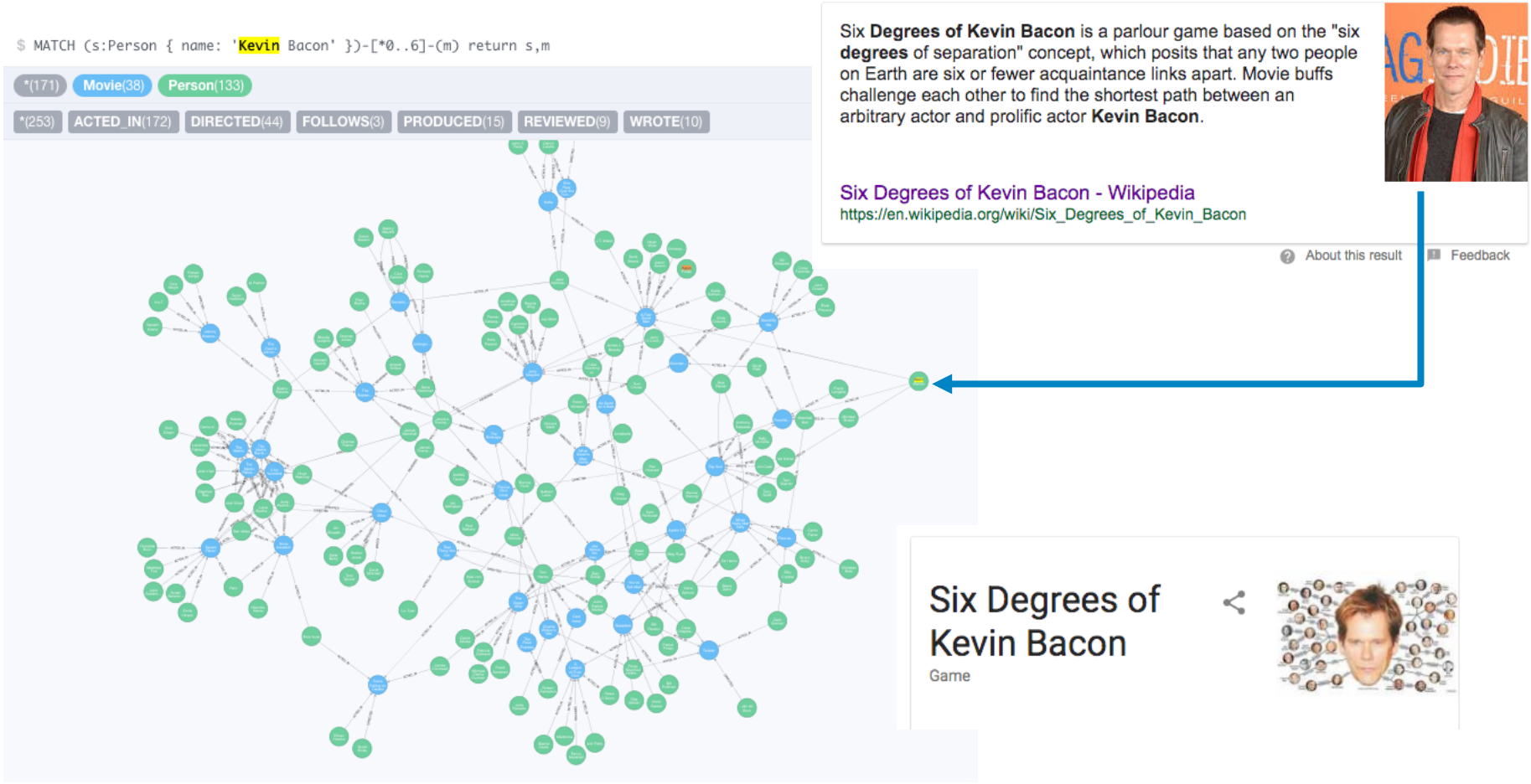
Rows | Text | Code

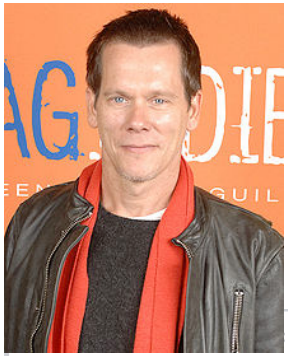
Recommended	Strength
Tom Cruise	5
Zach Grenier	5
Helen Hunt	4
Cuba Gooding Jr.	4
Keanu Reeves	4
Tom Skerritt	3
Carrie-Anne Moss	3
Val Kilmer	3
Bruno Kirby	3
Philip Seymour Hoffman	3
Billy Crystal	3
Carrie Fisher	3

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
      (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

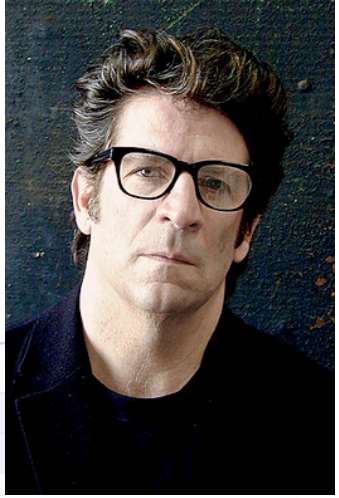


Which actors have worked with both Tom Hanks and Tom Cruise?

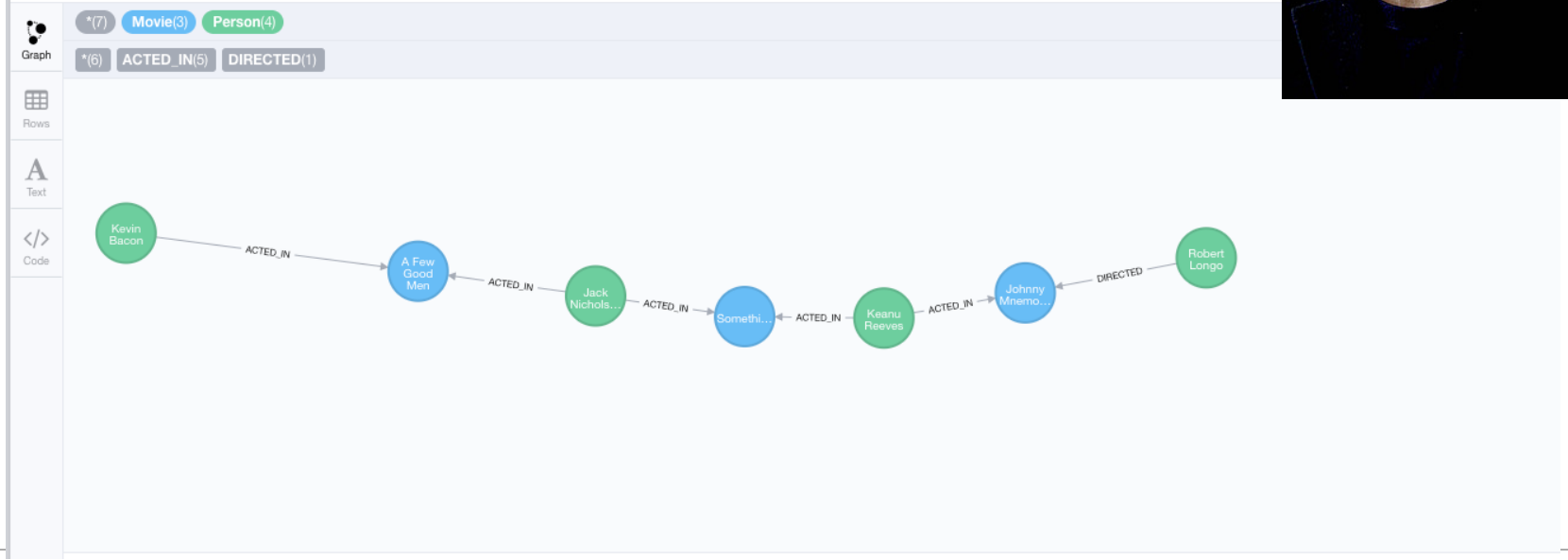




How do you get from Kevin Bacon to Robert Longo?



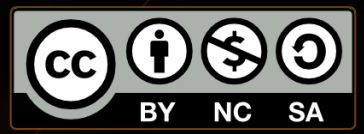
```
'Kevin Bacon' }),(robert:Person { name: 'Robert Longo' }), p = shortestPath((kevin)-[*..15]-(robert)) RETURN p
```



Redis

Redis

Redis Key-Value Database: Practical Introduction



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

SOFTWARE UNIVERSITY FOUNDATION

Redis

Ultra-Fast Data Structure Server

What is Redis?

- Redis is:
  - Ultra-fast in-memory key-value data store
  - Powerful data structure server
  - Open-source software: http://redis.io
- Redis stores data structures:
  - Strings, lists, hashes, sets, sorted sets
  - Publish / subscribe messaging

Redis: Features

- Redis is really fast
  - Non-blocking I/O, single threaded
  - 100,000+ read / writes per second

- Redis is not a database
  - It complements your existing data storage layer
  - E.g. StackOverflow uses Redis for data caching
- For small labs Redis may replace entirely the database
  - Trade performance for durability → data is persisted immediately

Installing Redis

- To install Redis on Linux / Mac OS X:
  - Download from http://redis.io/download
  - Install it and run it
- To install Redis on Windows:
  - Download https://github.com/MSOpenTech/redis
    - Compile the solution in the **`msvs`** folder and build it with VS
  - The easier way: use the Windows package manager Chocolatey
    - http://chocolatey.org

```
choco install redis
```

Running Redis

- On Linux / Mac OS X start / stop the Redis service

```
sudo service redis_6379 start
```

```
sudo service redis_6379 stop
```

  - The default Redis port is 6379 (the service name is **redis_6379**)

- On Windows start / stop the "**redis**" service

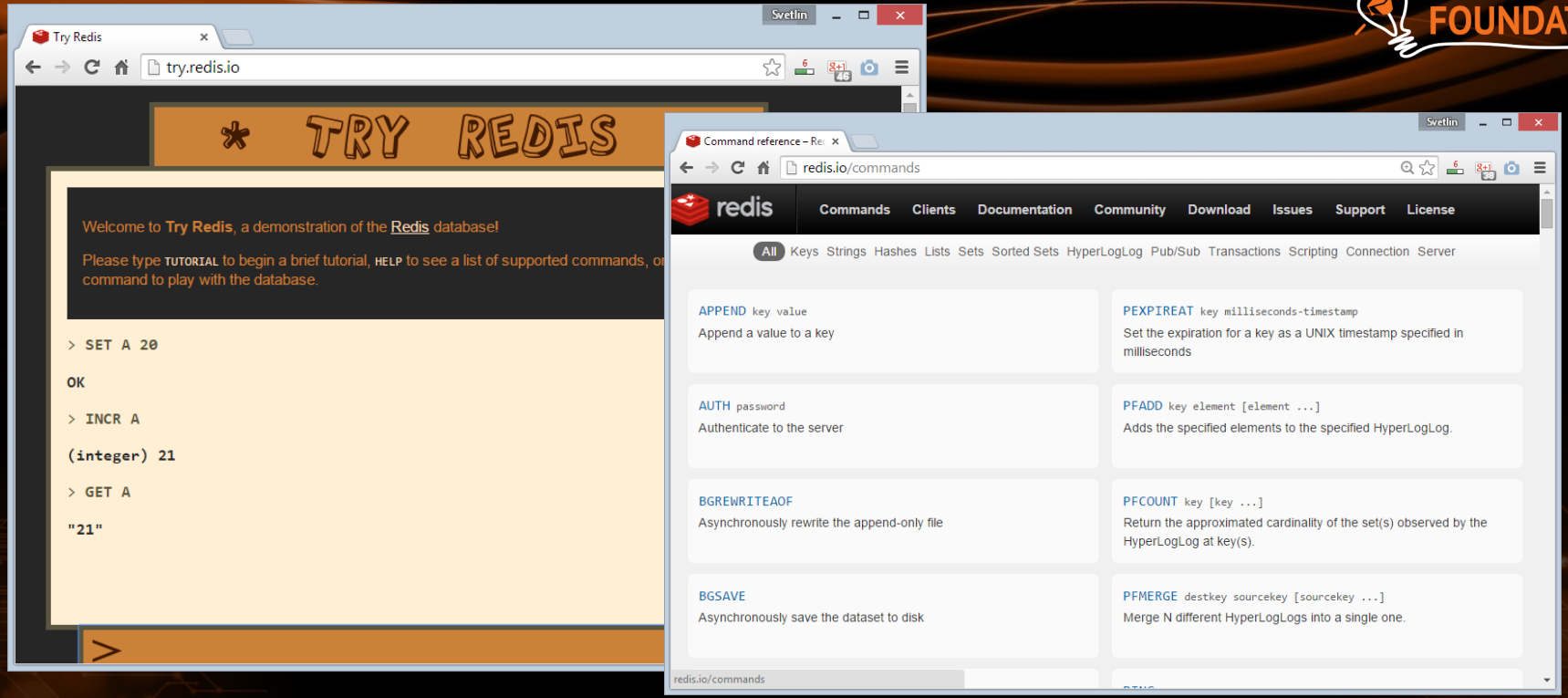
```
sc start redis
```

```
sc stop redis
```

- Use the console-based (CLI) client or just connect with Telnet:

```
redis-cli
```

```
telnet localhost 6379
```



Redis: Basic Commands

Redis: Data Model

- Redis keeps key-value pairs
  - Every item is stored as **`key`** + **`value`**
- Keys are unique identifiers
- Values can be different data structures:
  - Strings (numbers are stored as strings)
  - Lists (of strings)
  - Hash tables: string → string
  - Sets / sorted sets (of strings)

key	value
firstName	Bugs
lastName	Bunny
location	Earth

Redis: Commands

- Redis works as interpreter of commands:

```
SET name "Nakov"
OK
GET name
"Nakov"
DEL name
(integer) 1
GET name
(nil)
```

- Play with the command-line client **`redis-cli`**
- Or play with Redis online at http://try.redis.io

String Commands

- `SET [key] [value]`
  - Assigns a string value in a key
- `GET [key]` / `MGET [keys]`
  - Returns the value by key / keys
- `INCR [key]` / `DECR [key]`
  - Increments / decrements a key
- `STRLEN [key]`
  - Returns the length of a string

```
SET name
"hi"
OK

GET name
"hi"

SET name abc
OK

GET "name"
"abc"

GET Name
(nil)
```

```
SET a 1234
OK

INCR a
(integer) 1235

GET a
"12345"

MGET a name
1) "1235"
2) "asdda"

STRLEN a
(integer) 4
```

Working with Keys

- `EXISTS [key]`
  - Checks whether a key exists
- `TYPE [key]`
  - Returns the type of a key
- `DEL [key]`
  - Deletes a key
- `EXPIRE [key] [t]`
  - Deletes a key after `t` seconds

```
SET count 5
OK
TYPE count
string
EXISTS count
(integer) 1
DEL count
(integer) 1
EXISTS count
(integer) 0
```

```
SET a 1234
OK
GET a
"1234"
EXPIRE a 5
(integer) 1
EXISTS a
(integer) 1
EXISTS a
(integer) 0
```

Working with Hashes (Hash Tables)

- `HSET [key] [field] [value]`
  - Assigns a value for given field
- `HKEYS [key]`
  - Returns the fields (keys) is in a hash
- `HGET [key] [field]`
  - Returns a value by fields from a hash
- `HDEL [key] [field]`
  - Deletes a fields from a hash

```
HSET user name "peter"
(integer) 1

HSET user age 23
(integer) 1

HKEYS user
1) "name"
2) "age"

HGET user age
"23"

HDEL user age
(integer) 1
```

Working with Lists

- `RPUSH` / `LPUSH [list] [value]`
  - Appends / prepend an value to a list
- `LINDEX [list] [index]`
  - Returns a value given index in a list
- `LLEN [list]`
  - Returns the length of a list
- `LRANGE [list] [start] [count]`
  - Returns a sub-list (range of values)

```
RPUSH names "peter"
(integer) 1

LPUSH names Nakov
(integer) 1

LINDEX names 0
"Nakov"

LLEN names
(integer) 2

LRANGE names 0 100
1) "Nakov"
2) "peter"
```

Working with Sets

- `SADD [set] [value]`
  - Appends a value to a set
- `SMEMBERS [set]`
  - Returns the values from a set
- `SREM [set] [value]`
  - Deletes a value form a set
- `SCARD [set]`
  - Returns the stack size (items count)

```
SADD users "peter"
(integer) 1

SADD users "peter"
(integer) 0

SADD users maria
(integer) 1

SMEMBERS users
1) "peter"
2) "maria"

SREM users maria
(integer) 1
```

Working with Sorted Sets


```
ZADD myzset 1 "one"
(integer) 1

ZADD myzset 1 "uno"
(integer) 1

ZADD myzset 2 "two" 3 "three"
(integer) 2

ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
…
```

- Learn more at http://redis.io/commands#sorted_set

Publish / Subscribe Commands

- First user subscribes to certain channel "**news**"

```
SUBSCRIBE news
1) "subscribe"
2) "news"
3) (integer) 1
```

- Another user sends messages to the same channel "**news**"

```
PUBLISH news "hello"
(integer) 1
```

- Learn more at http://redis.io/commands#pubsub

Using Redis as a Database

SOFTWARE UNIVERSITY FOUNDATION

- Special naming can help using Redis as database

```
SADD users:names peter
HSET users:peter name "Peter Petrov"
HSET users:peter email "pp@gmail.com"

SADD users:names maria
HSET users:maria name "Maria Ivanova"
HSET users:maria email "maria@yahoo.com"

SMEMBERS users:names
HGETALL users:peter
```

Add a user "**peter**".

Use "**users:peter**" as key to hold user data

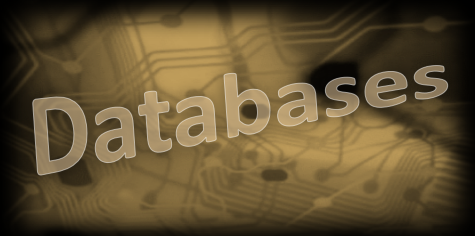
Add a user "**maria**".

List all users

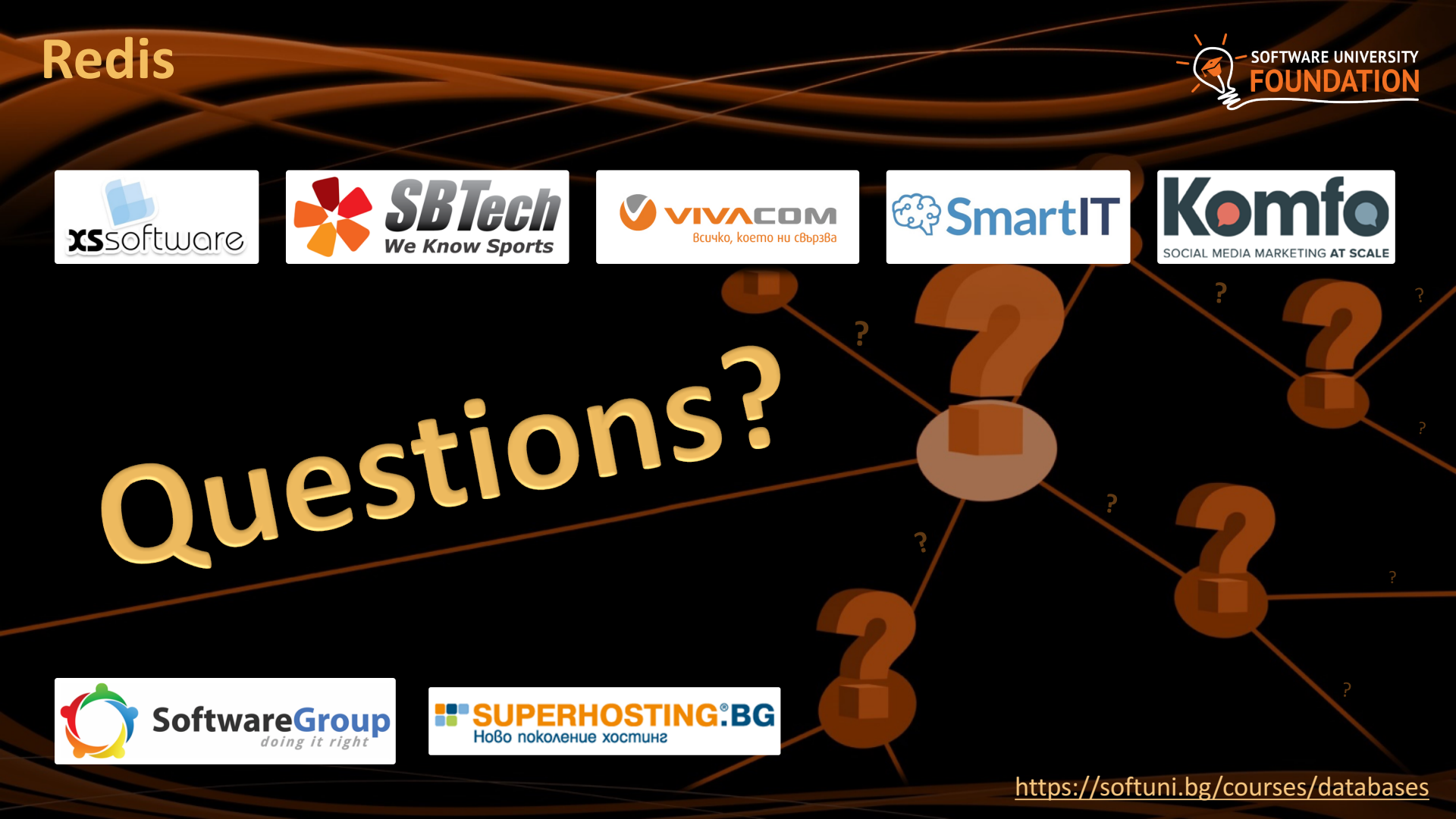
List all properties for user "**peter**"

Summary

1. Redis is ultra-fast in-memory data store
   - Not a database, used along with databases
2. Supports strings, numbers, lists, hashes, sets, sorted sets, publish / subscribe messaging
3. Used for caching / simple apps



Redis



https://softuni.bg/courses/databases

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Databases" course by Telerik Academy under CC-BY-NC-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg

12 Factor Applications

https://12factor.net/

https://www.slideshare.net/bifer/twelve-factor-apps

The 12 Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. LogsTreat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes