Supercharge Your Apps with Amazon Neptune Graph Database (Level 200)

Clayton Brown, Solution Architect

Fully Managed Database Options on AWS

Relational Databases

Amazon RDS

- Aurora
- Commercial: MySQL, Oracle, Microsoft SQL Server, PostgreSQL
- Community: MySQL, PostgreSQL, MariaDB

Amazon Redshift

- Data Warehouse

Non-Relational Databases

Amazon DynamoDB

- Key Value
- Document

Amazon ElastiCache

- In-Memory Data Store
- redis
- MEMCACHED

Amazon Neptune

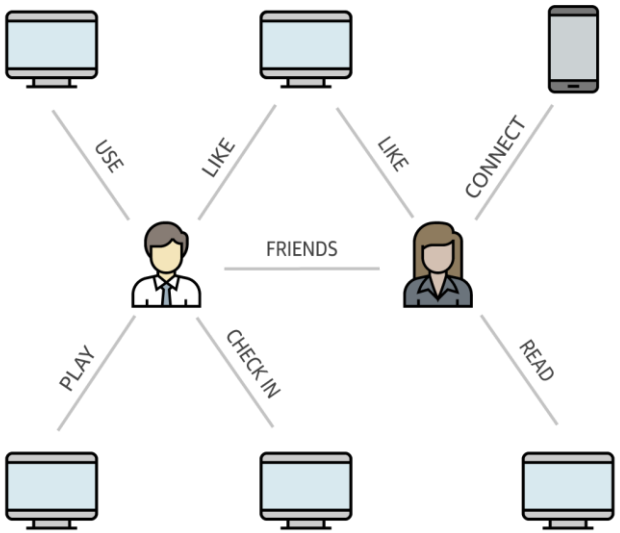
- Graph

AWS Database Migration Service

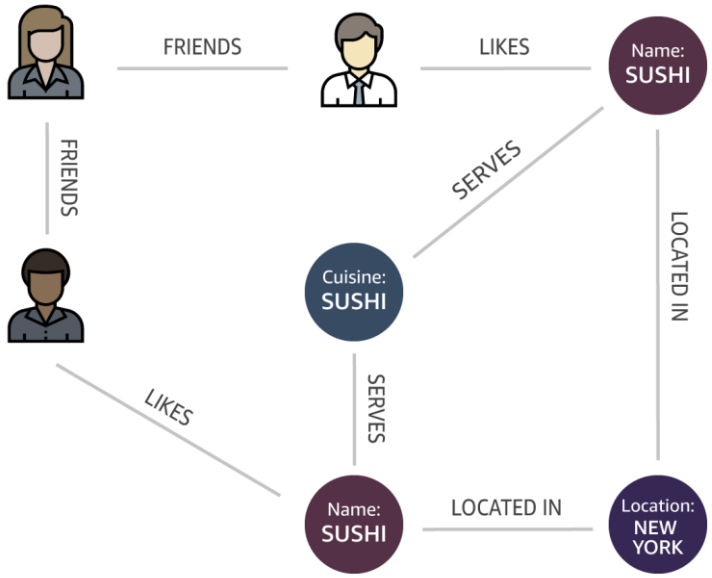
Agenda

- Building Applications on Highly Connected Data
- Types of Graphs and How to Query Them
- Property Graph and Apache TinkerPop Friend Recommendation Example
- RDF Knowledge Graph Example
- Overview of Amazon Neptune's Fully Managed, Enterprise-Ready Features

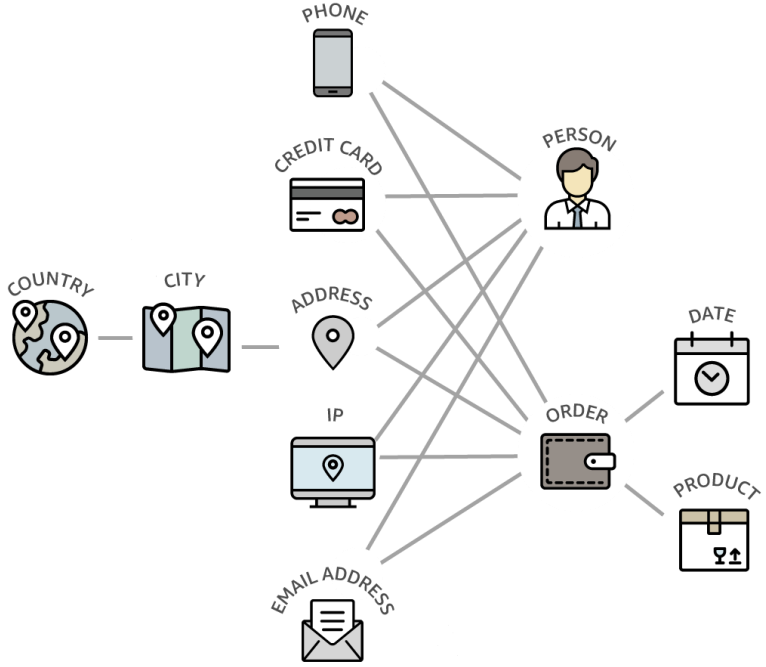
Highly Connected Data



Social Networks



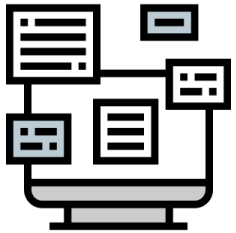
Restaurant Recommendations



Retail Fraud Detection

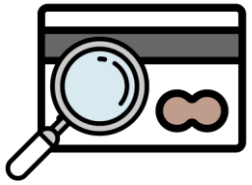
Use Cases for Highly Connected Data

Social Networking

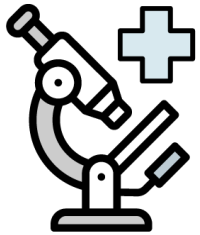


Recommendations

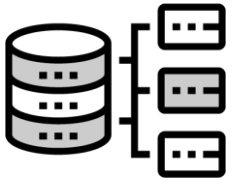
Knowledge Graphs



Fraud Detection

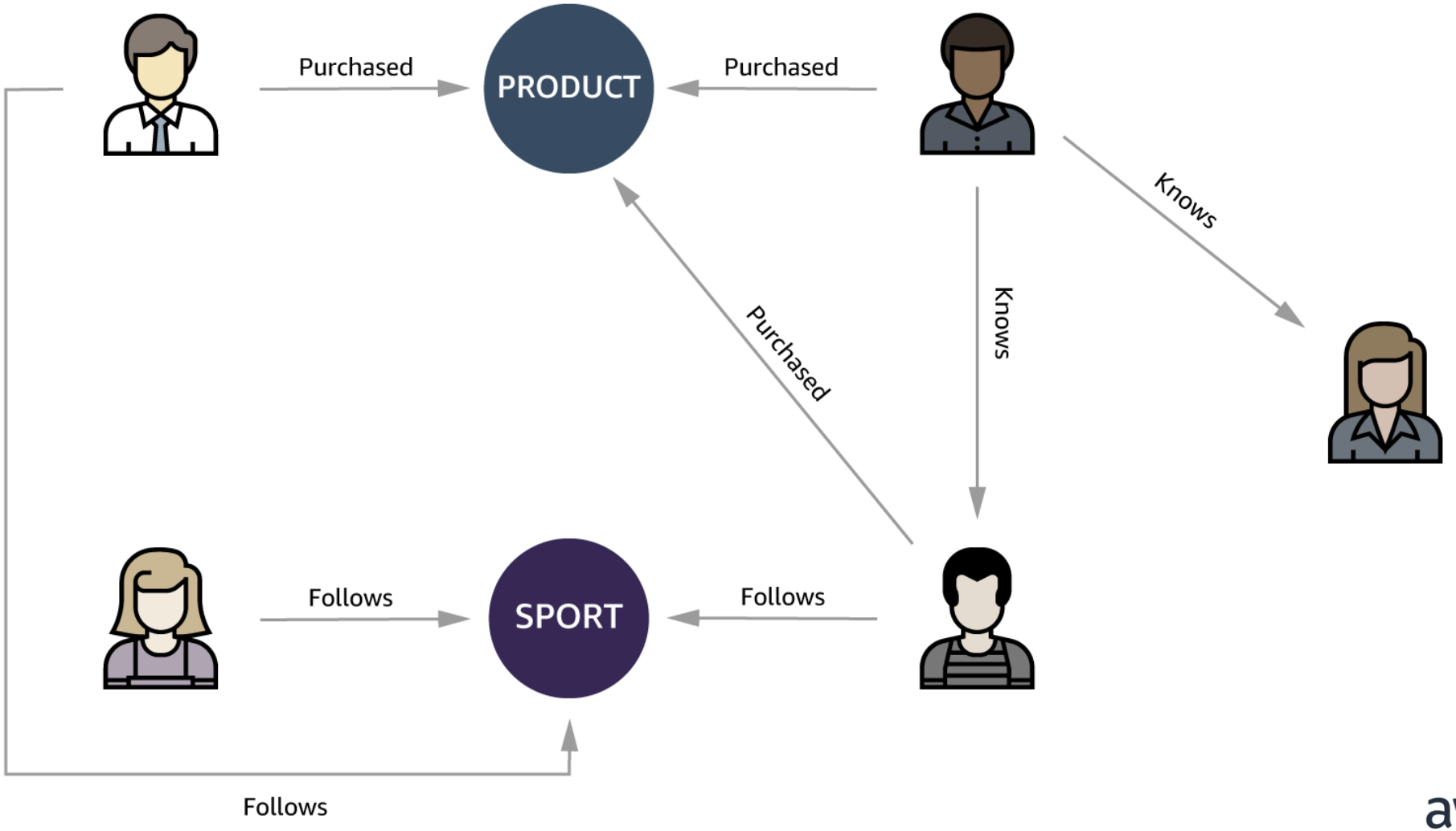


Life Sciences

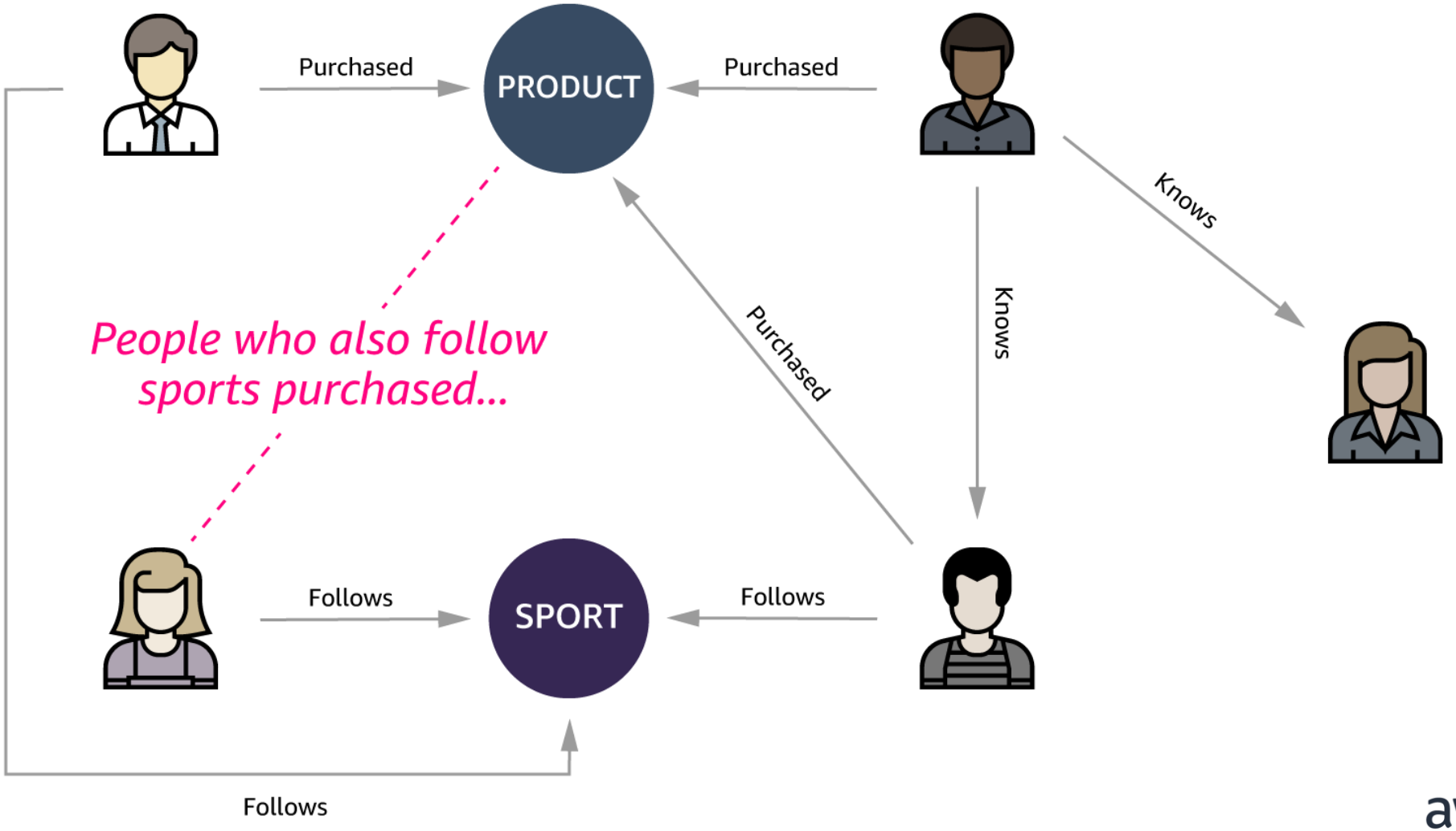


Network & IT Operations

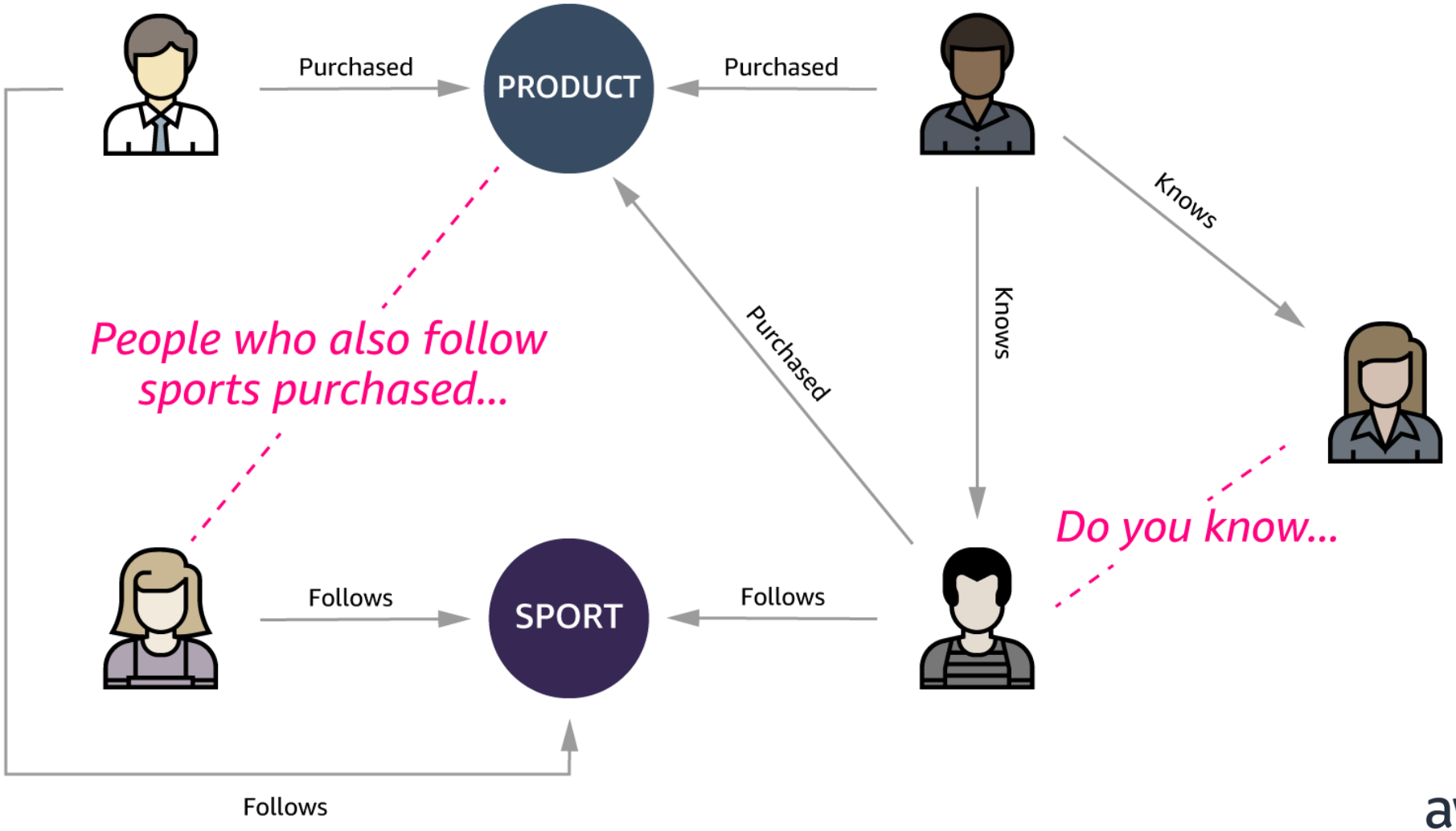
Recommendations Based on Relationships



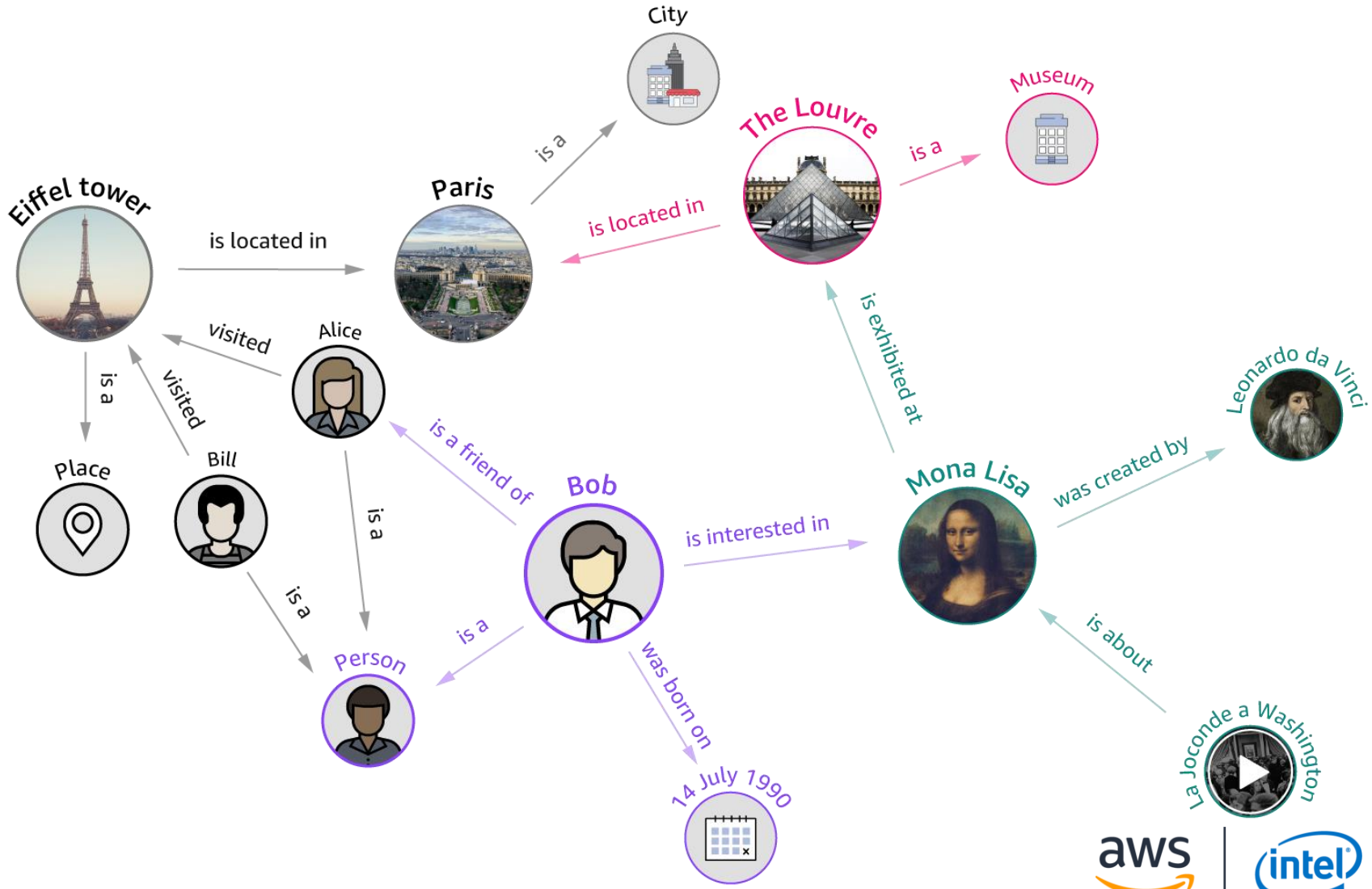
Recommendations Based on Relationships



Recommendations Based on Relationships

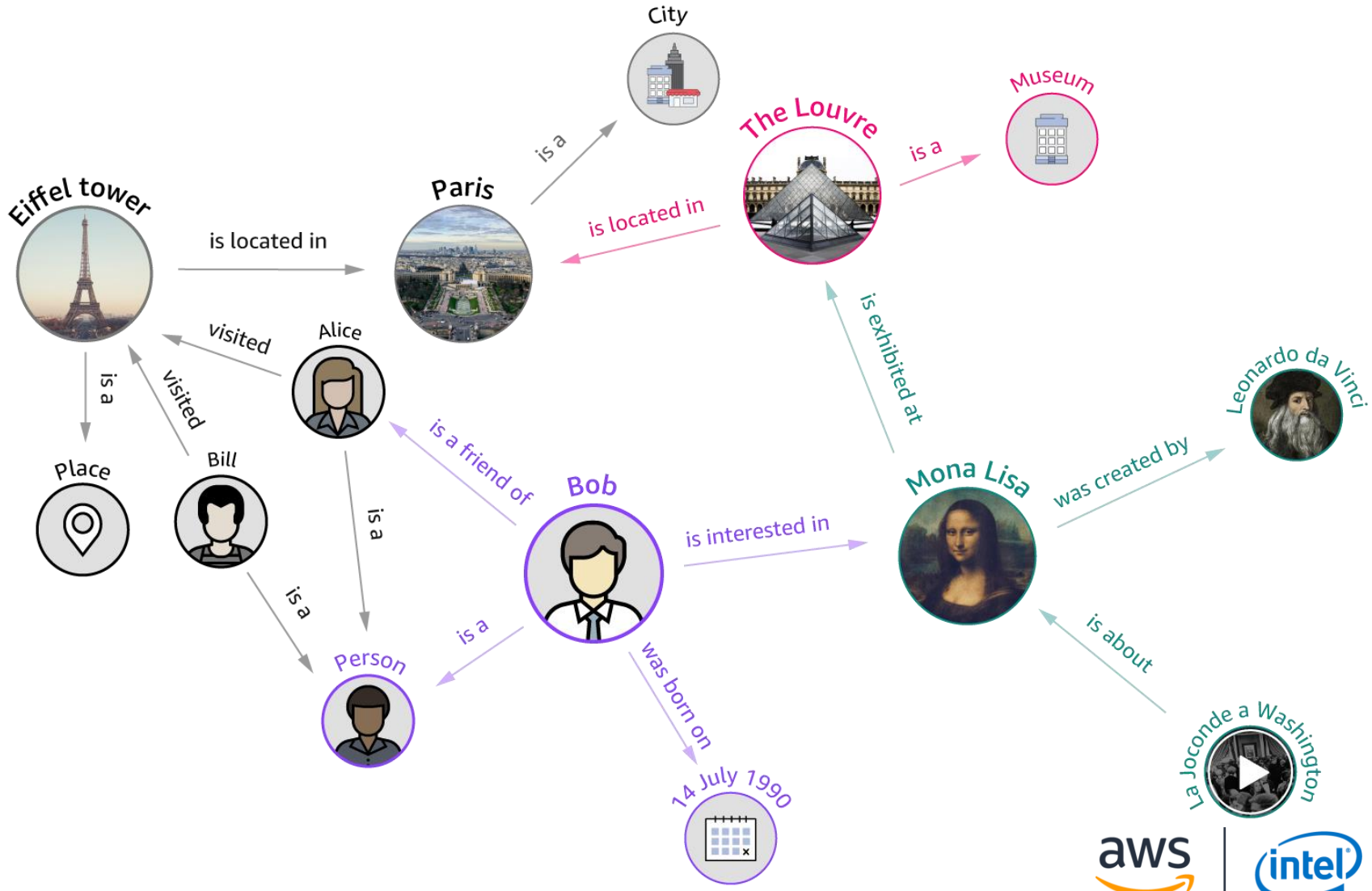


Knowledge Graph Applications



Knowledge Graph Applications

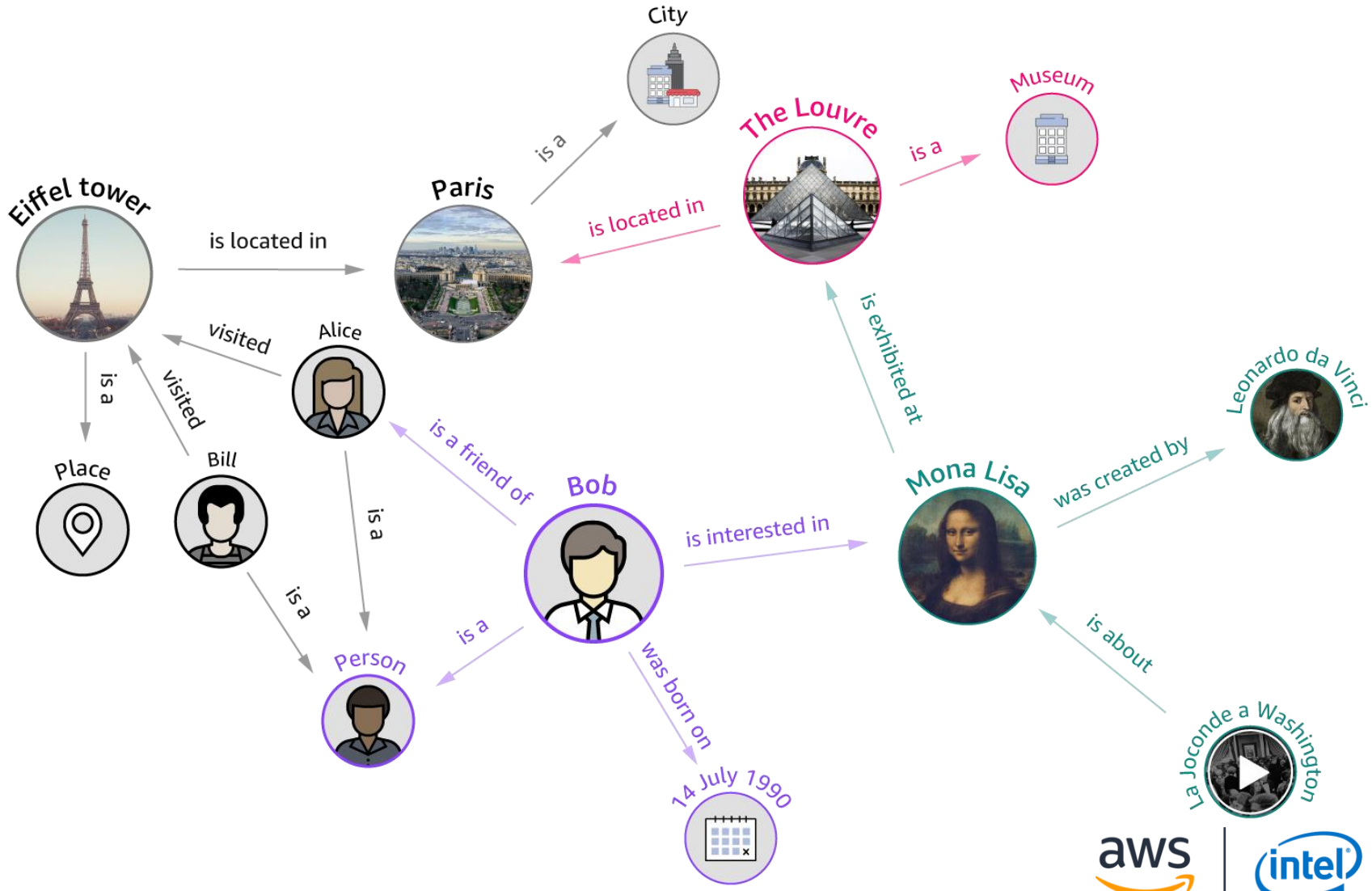
Who painted the Mona Lisa?



Knowledge Graph Applications

Who painted the Mona Lisa?

What museums should Alice visit while in Paris?

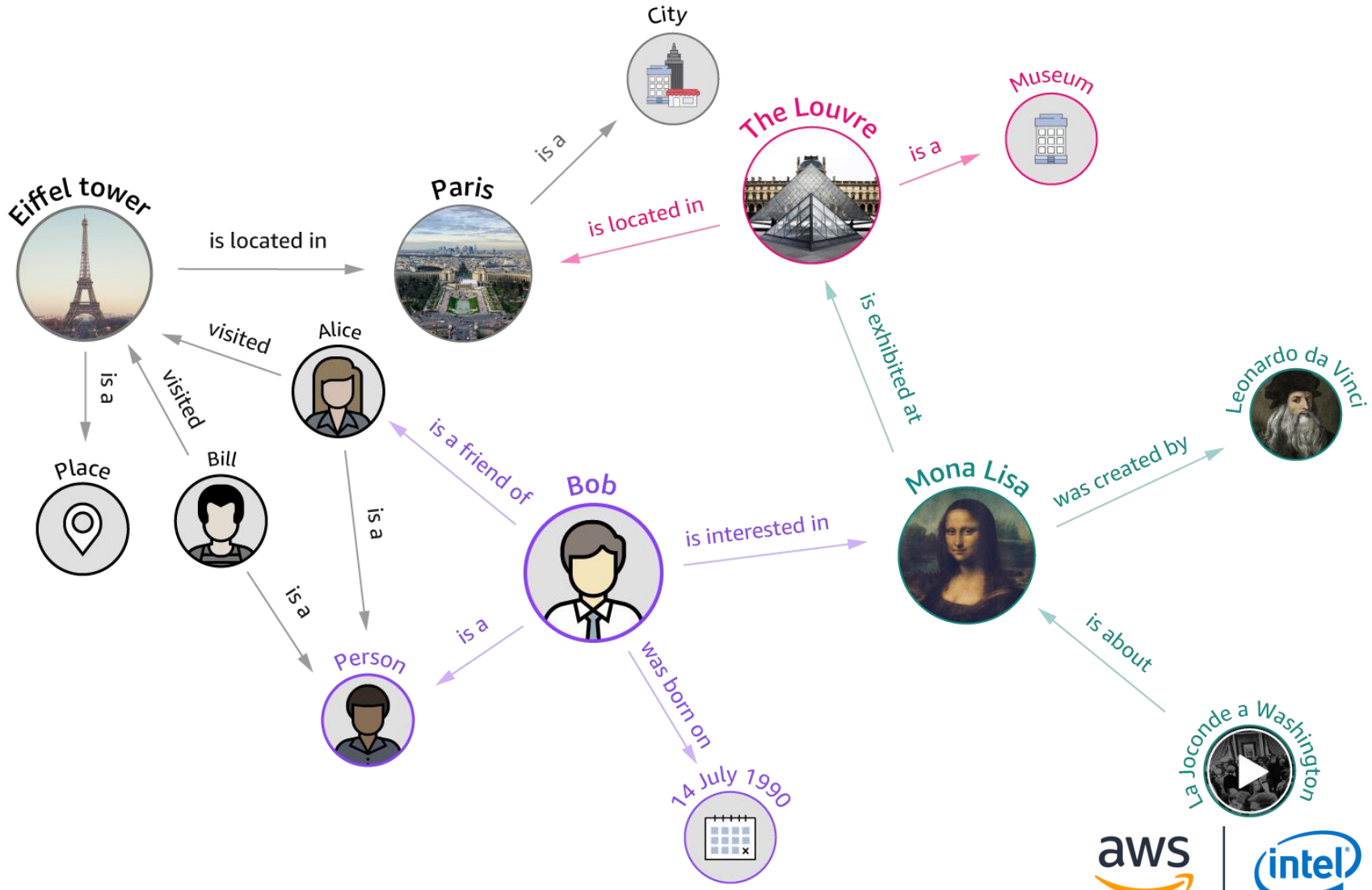


Knowledge Graph Applications

Who painted the Mona Lisa?

What museums should Alice visit while in Paris?

What artists have paintings in The Louvre?



Navigate a Web of Global Tax Policies

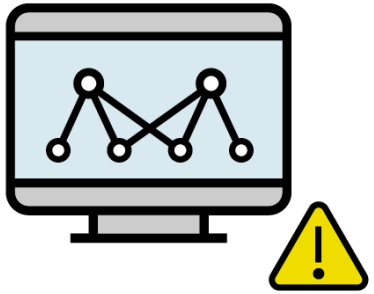
THOMSON REUTERS

"Our customers are increasingly required to navigate a complex web of global tax policies and regulations. We need an approach ***to model the sophisticated corporate structures*** *of our largest clients and deliver an end-to-end tax solution. We use a microservices architecture approach for our platforms and are beginning to* ***leverage Amazon Neptune as a graph-based system*** *to quickly create links within the data."*

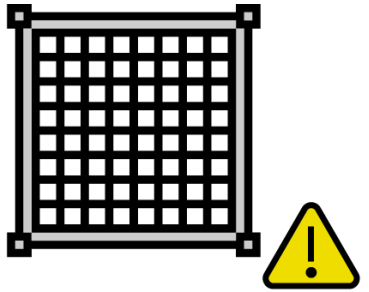
Tim Vanderham, Chief Technology Officer, Thomson Reuters Tax & Accounting

Relational Database Challenges Building Apps with Highly Connected Data

Unnatural for querying graph

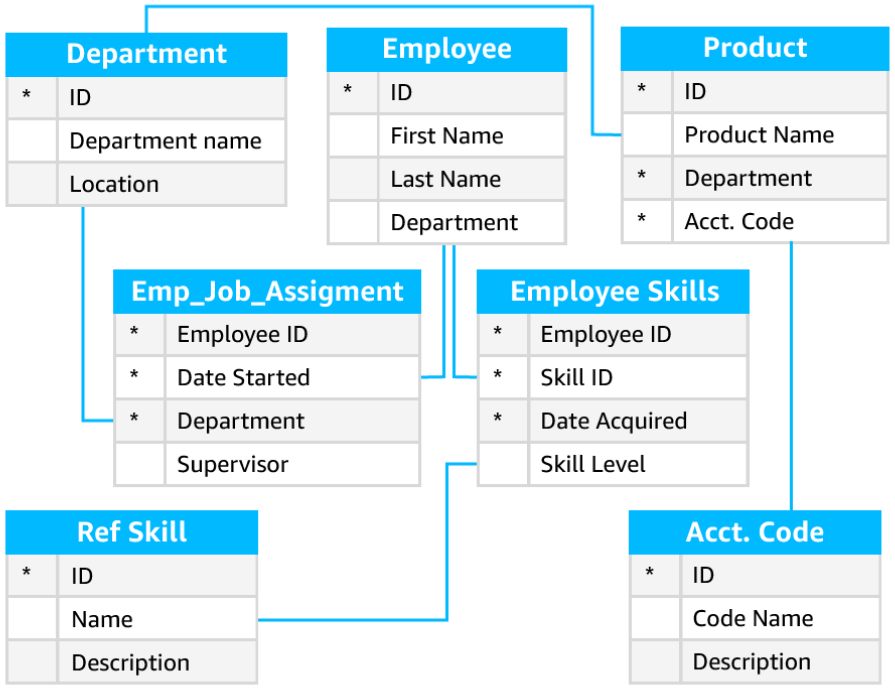


Inefficient graph processing

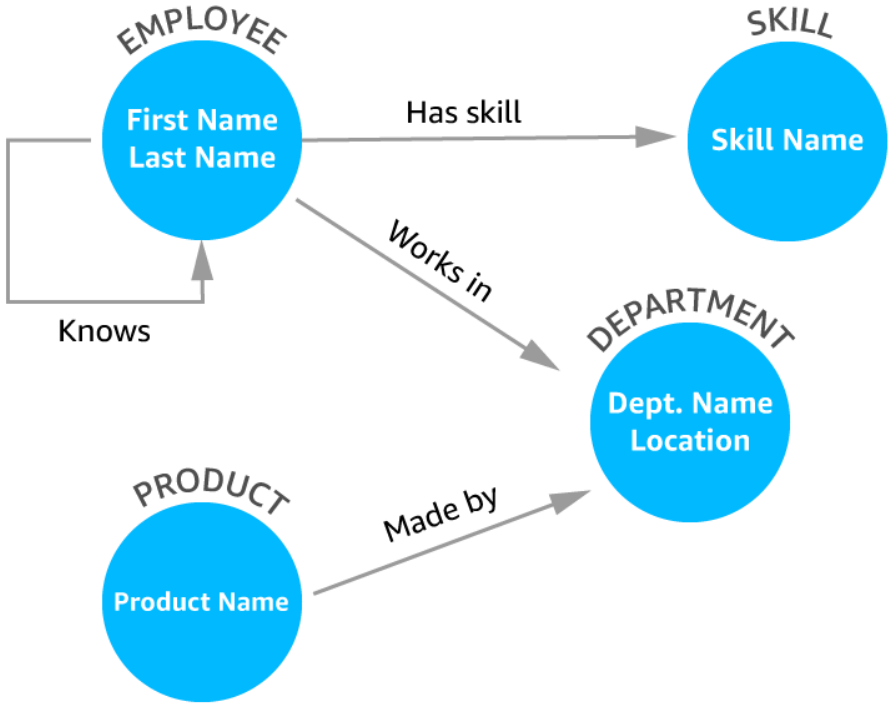


Rigid schema inflexible for changing data

Different Approaches for Highly Connected Data

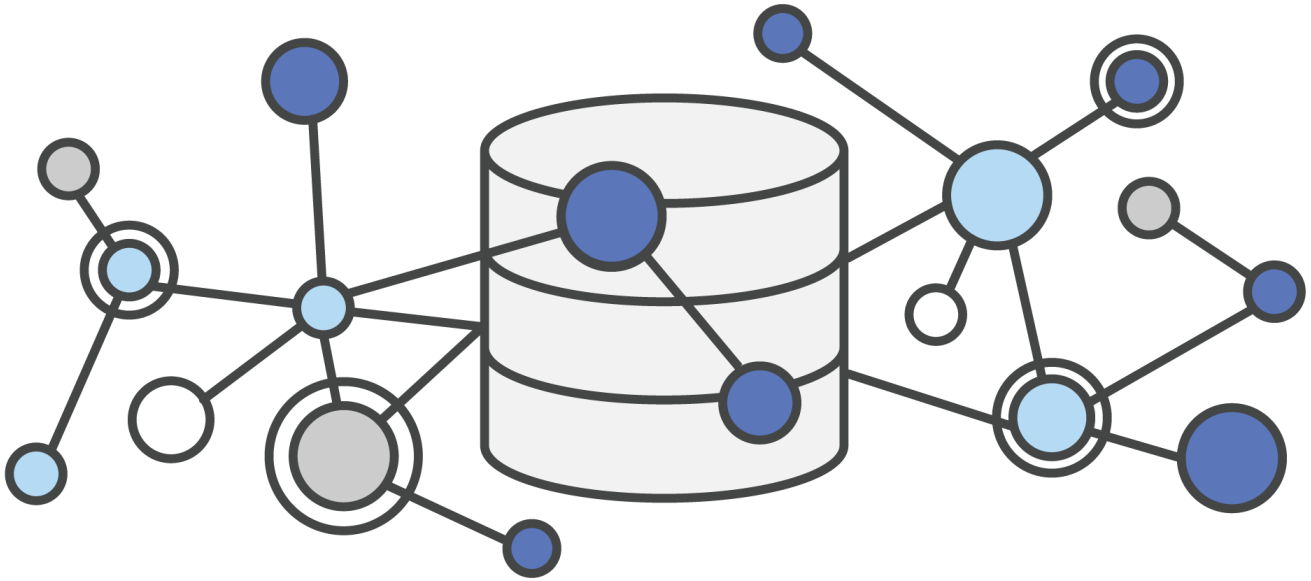


Purpose-built for a business process



Purpose-built to answer questions about relationships

A Graph Database is Optimized for Efficient Storage and Retrieval of Highly Connected Data

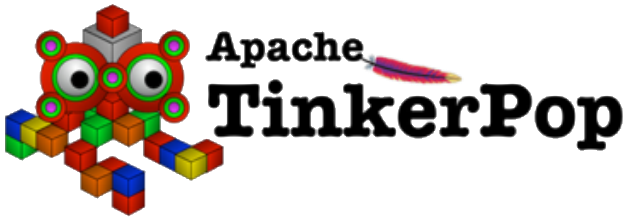


Leading Graph Models and Frameworks

PROPERTY GRAPH

Open Source Apache TinkerPop™

Gremlin Traversal Language



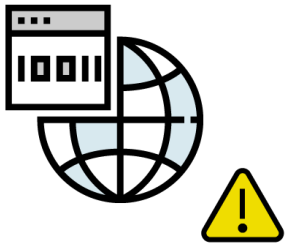
RESOURCE DESCRIPTION FRAMEWORK (RDF)

W3C Standard

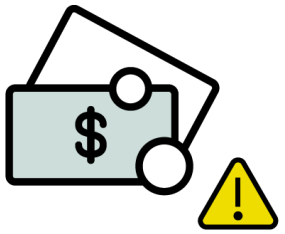
SPARQL Query Language

Challenges of Existing Graph Databases

Difficult to scale



Difficult to maintain high availability



Too expensive

Limited support for open standards

Amazon Neptune - Fully Managed Graph Database

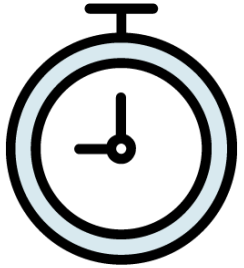
OPEN

Supports Apache TinkerPop & W3C RDF graph models

FAST

Query billions of relationships with millisecond latency

RELIABLE

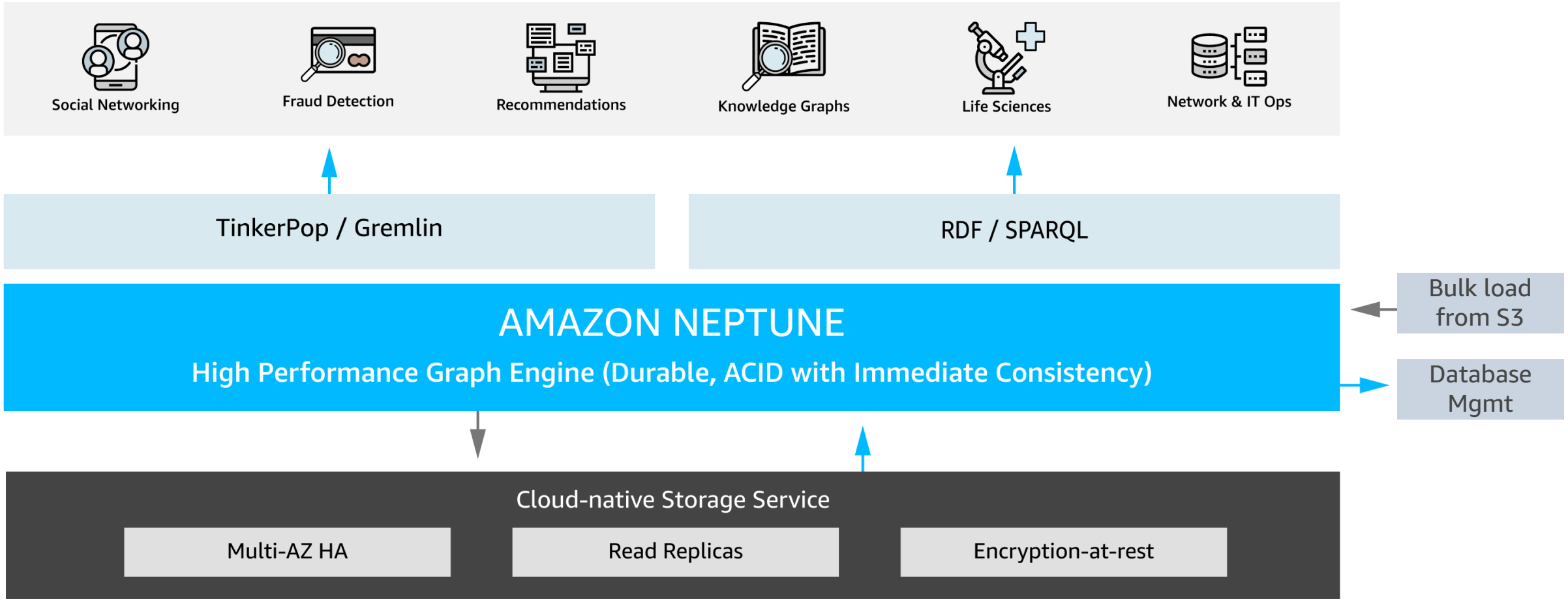


6 replicas of your data across 3 AZs with full backup and restore

EASY

Build powerful queries easily with Gremlin and SPARQL

Amazon Neptune High Level Architecture

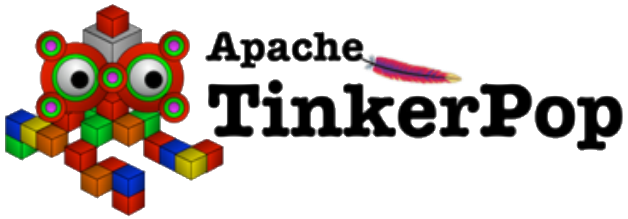


Types of Graphs and How to Query Them

PROPERTY GRAPH

Open Source Apache TinkerPop™

Gremlin Traversal Language



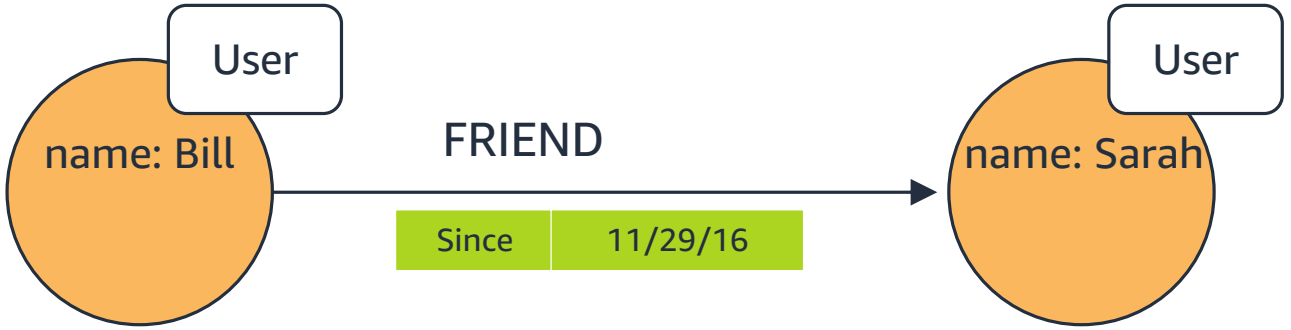
RESOURCE DESCRIPTION FRAMEWORK (RDF)

W3C Standard

SPARQL Query Language

Property Graph

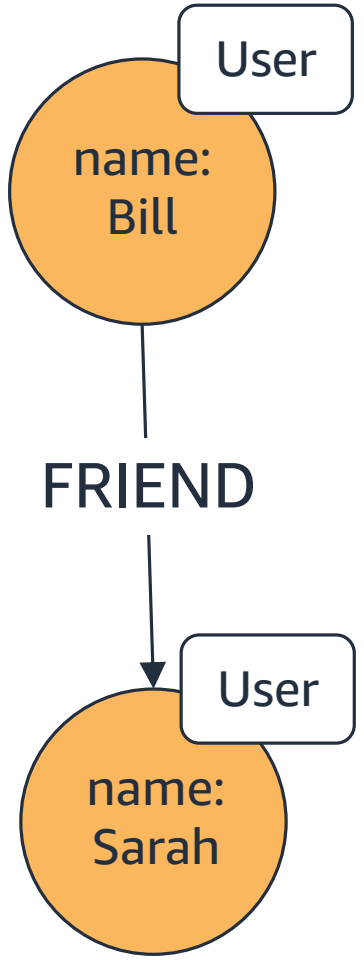
- A property graph is a set of vertices and edges with respective properties (i.e. key/value pairs)
- **Vertex** represents entities/domains
- **Edge** represents directional relationship between vertices.
  - Each edge has a label that denotes the type of relationship



- Each vertex & edge has a unique identifier
- Vertex and edges can have **properties**
- **Properties** express non-relational information about the vertices and edges

Creating a TinkerPop Graph

```
//Connect to Neptune and receive a remote graph, g.

user1 = g.addVertex (id, 1,  label, "User", "name", "Bill");
user2 = g.addVertex (id, 2,  label, "User", "name", "Sarah");

...

user1.addEdge("FRIEND", user2, id, 21);
```



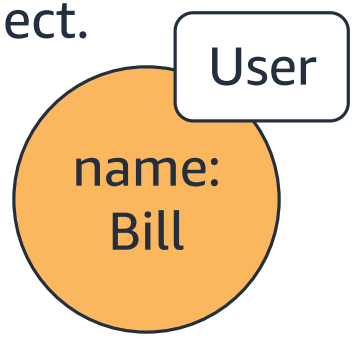
Apache TinkerPop

Gremlin (Apache TinkerPop 3.3)

RDF Graphs

- **RDF Graphs** are described as a collection of triples: subject, predicate, and object.

subject → <http://www.socialnetwork.com/person#1>
predicate → rdf:type contacts:User;
Object (literal) → contact:name: "Bill" .



- **Internationalized Resource Identifiers (IRIs)** uniquely identify subjects.

IRI → <http://www.socialnetwork.com/person#1>

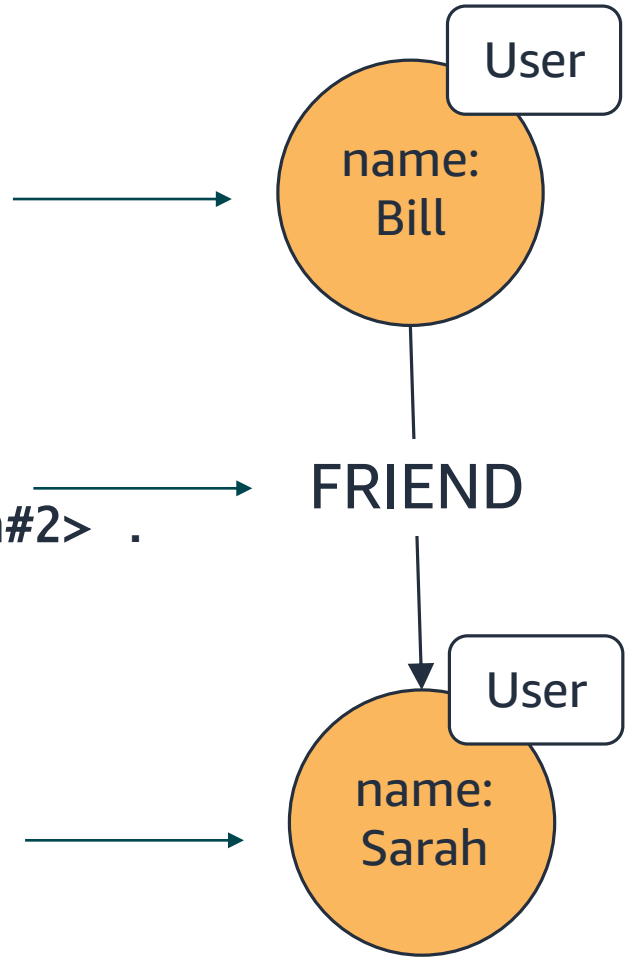
- The Object can be an IRI or Literal.
  - A Literal in RDF is like a property and RDF supports the XML data types.
  - When the Object is an IRI, it forms an "Edge" in the graph.

subject → <http://www.socialnetwork.com/person#1>
predicate → contacts:friend
Object (IRI) → <http://www.socialnetwork.com/person#2> .

“There’s No Trouble with Triples”: RDF Example

```
@prefix contacts: <http://www.socialnetwork.com/people#>.
<http://www.socialnetwork.com/person#1>
       rdf:type contacts:User;
       contact:name: ”Bill” .

<http://www.socialnetwork.com/person#1>
       contacts:friend <http://www.socialnetwork.com/person#2> .
```

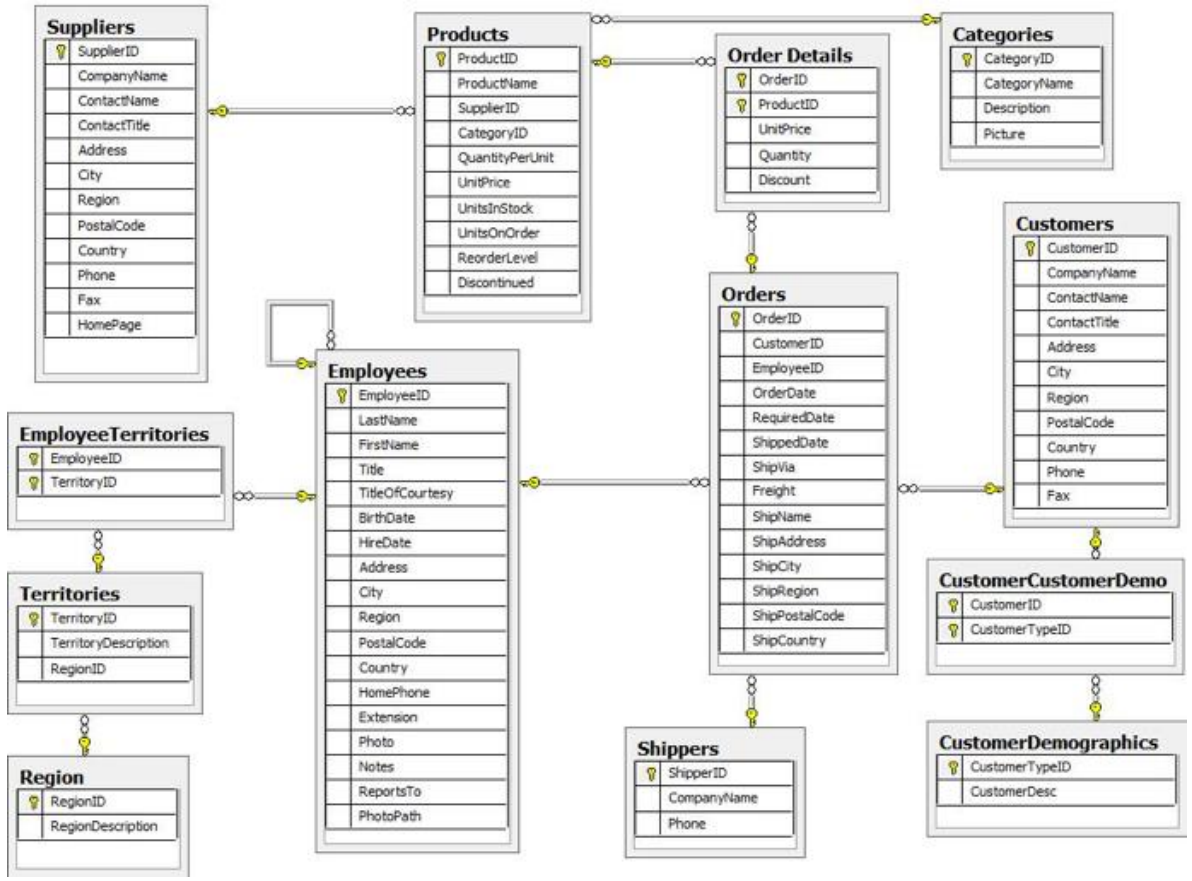
- <http://www.socialnetwork.com/person#2>
- rdf:type contacts:User;
- contact:name: ”Sarah” .



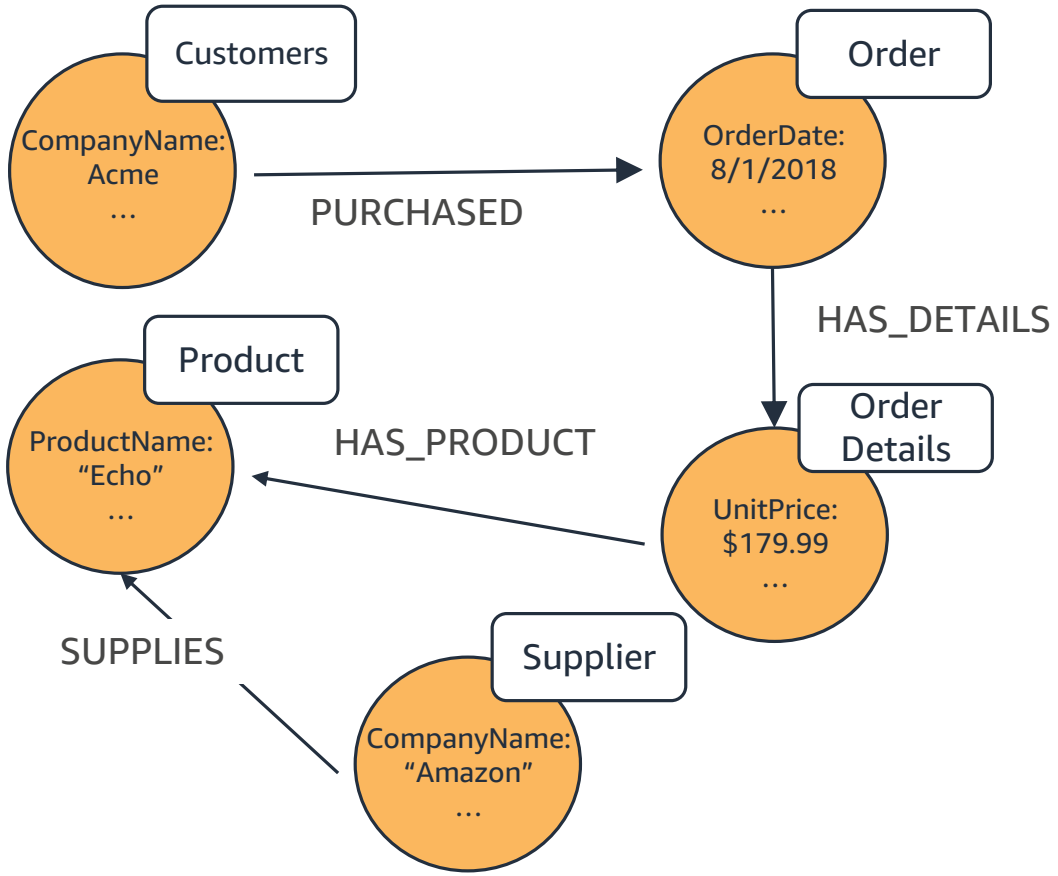
RDF
(Turtle Serialization)

Graph vs. Relational Database Modeling

Relational model



Graph model subset



* Source : http://www.playnexacro.com/index.html#show:article

SQL Relational Database Query

Find the name of companies that purchased the ‘Echo’.

```
SELECT distinct c.CompanyName
FROM customers AS c
•      JOIN orders AS o ON                          /* Join the customer from
  the order */
•             (c.CustomerID = o.CustomerID)
•      JOIN order_details AS od    /* Join the order details from the order */
              ON (o.OrderID = od.OrderID)
•      JOIN products as p        /* Join the products from the order details */
              ON (od.ProductID = p.ProductID)
•      WHERE p.ProductName = 'Echo';       /* Find the product named ‘Echo’ */
```

SPARQL Declarative Graph Query

Find the name of companies that purchased the ‘Echo’.

```
PREFIX sales_db: <http://sales.widget.com/>
SELECT distinct ?comp_name WHERE {
•     ?customer    <sales_db:HAS_ORDER> ?order ;       #customer graph pattern
•                  <sales_db:CompanyName> ?comp_name .       #orders graph
  pattern
•     ?order       <sales_db:HAS_DETAILS> ?order_d .  #order details graph
  pattern
•     ?order_d     <sales_db:HAS_PRODUCT> ?product .       #products graph
  pattern
      ?product     <sales_db:ProductName> “Echo” .
}
```

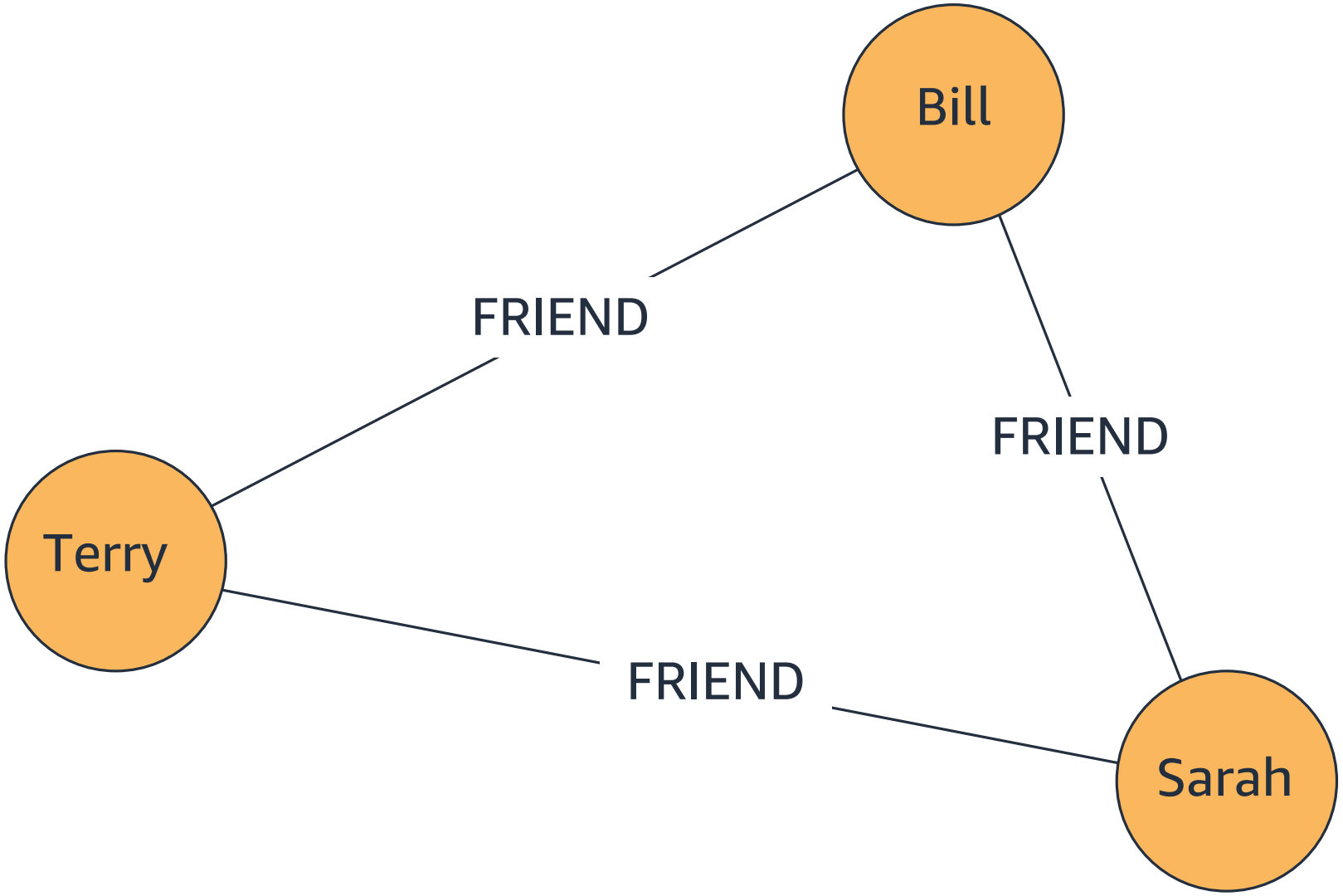
Gremlin Imperative Graph Traversal

Find the name of companies that purchased the ‘Echo’.

```
/* All products named ”Echo” */
g.V().hasLabel(‘Product’).has('name','Echo')
      .in(’HAS_PRODUCT') /* Traverse to order details */
      .in(‘HAS_DETAILS’) /* Traverse to order */
      .in(’HAS_ORDER’) /* Traverse to Customer */
      .values(’CompanyName’).dedup()  /* Unique Company Name */
```

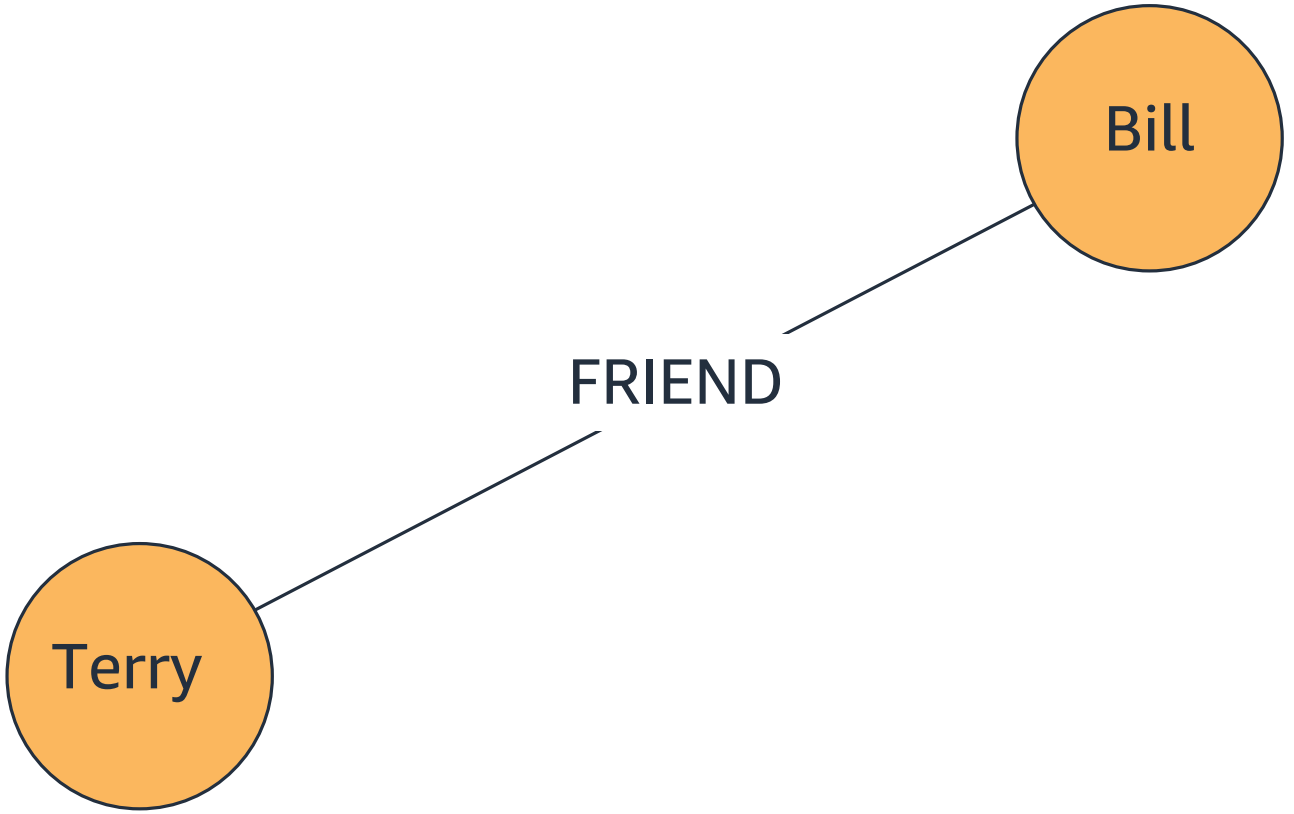
TinkerPop Social Network Friend Recommendation Example

Triadic Closure – Closing Triangles

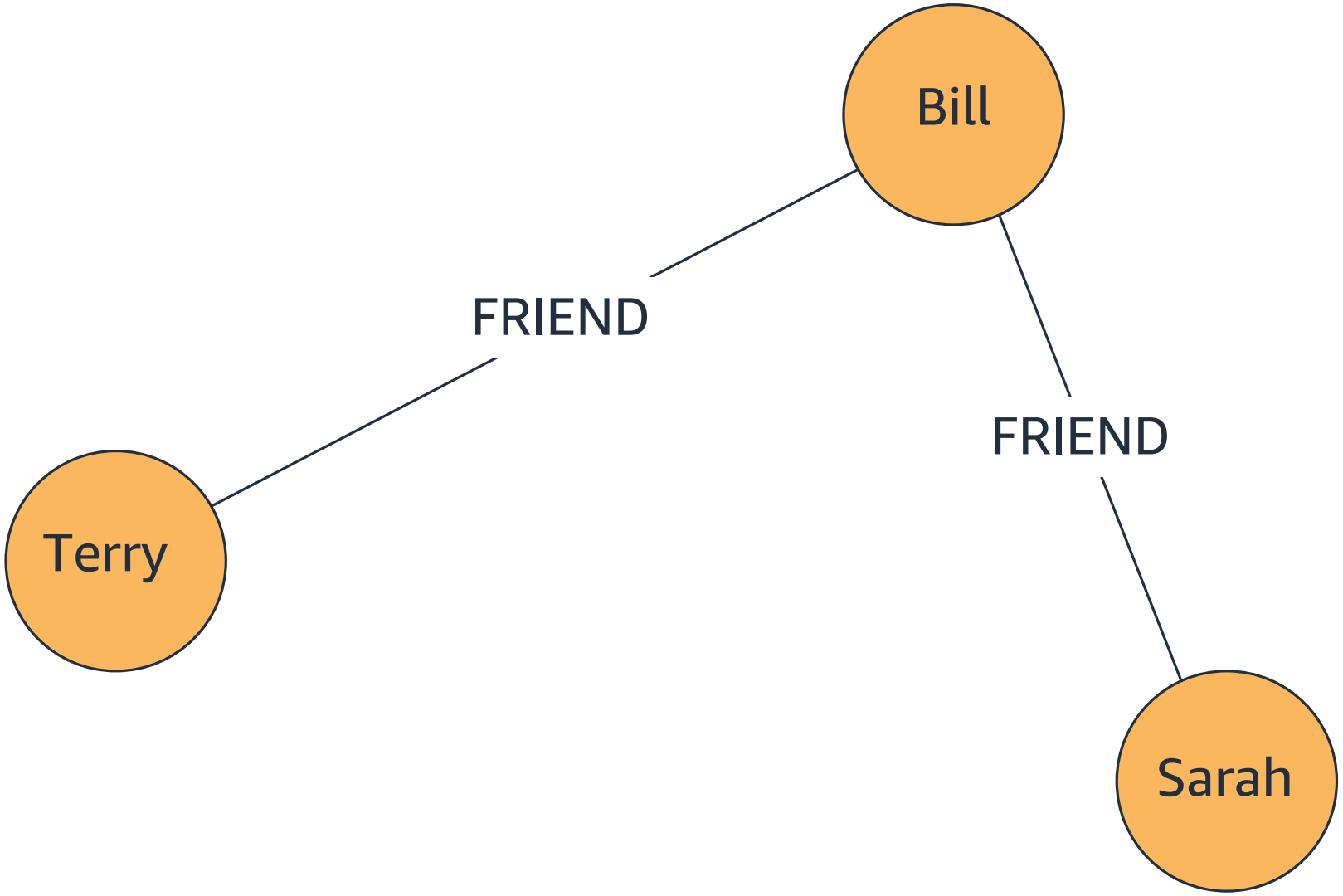


Recommending New Connections

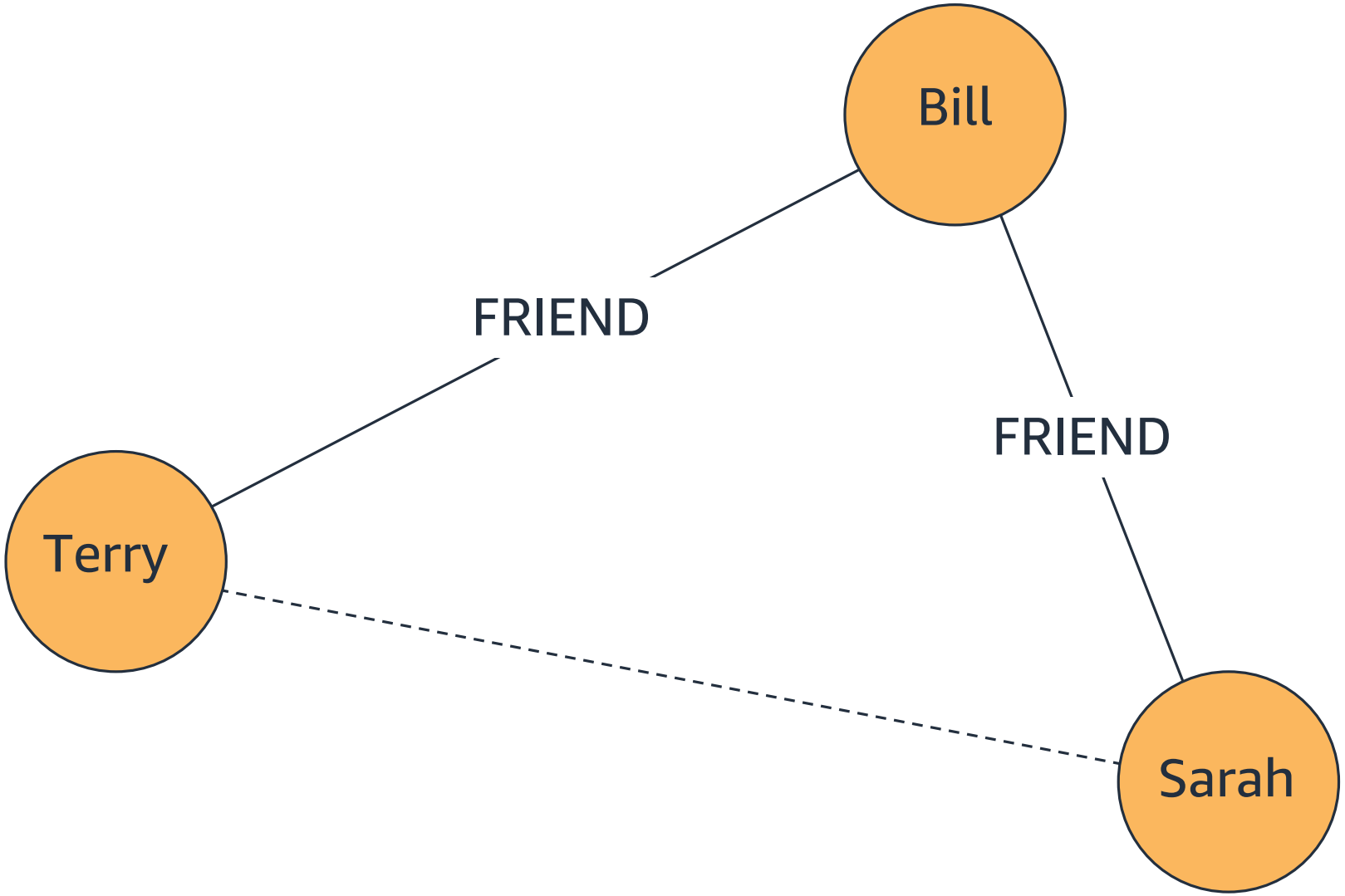
Immediate Friendships



Means and Motive



Recommendation



Recommend New Connections

```
g = graph.traversal()

g.V().has('name','Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(neq('user')).where(neq('friends')).
  groupCount().by('name').
  order(local).by(values, decr)
```

Find Terry

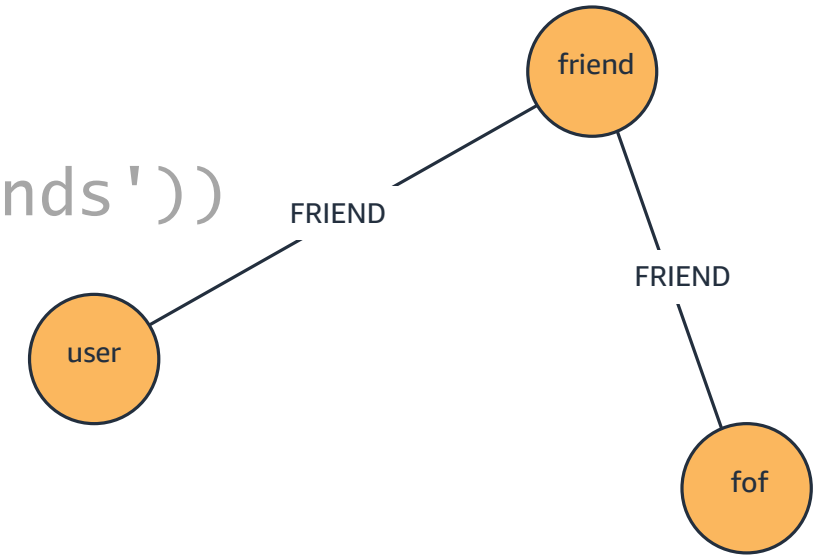
```
g = graph.traversal()

g.V().has('name','Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(neq('user')).where(neq('friends')).
  groupCount().by('name').
  order(local).by(values, decr)
```

Find Terry's Friends

```
g = graph.traversal()

g.V().has('name','Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(neq('user')).where(neq('friends')).
  groupCount().by('name').
  order(local).by(values, decr)
```

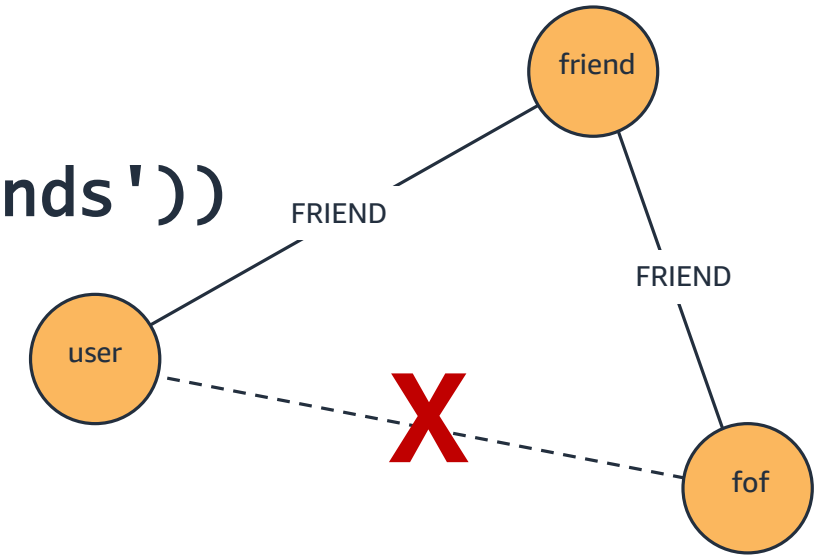
And The Friends of Those Friends

```
g = graph.traversal()

g.V().has('name','Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(neq('user')).where(neq('friends'))
  groupCount().by('name').
  order(local).by(values, decr)
```



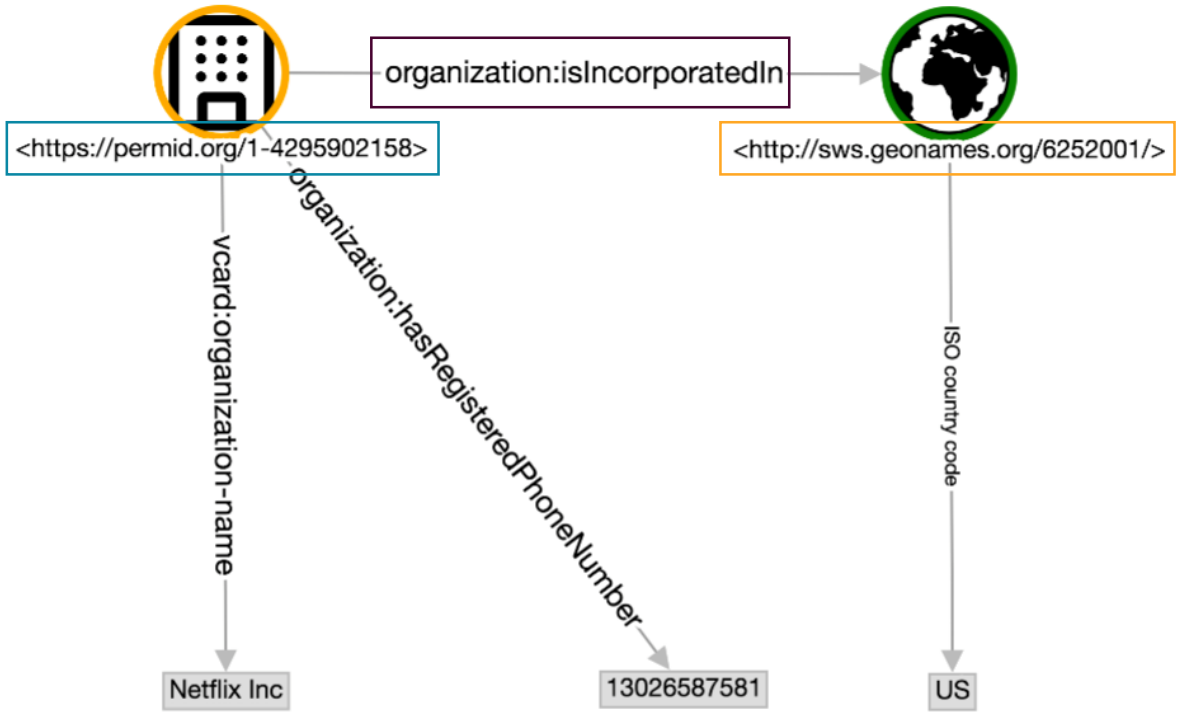
…Who Aren't Terry and Aren't Friends with Terry

```
g = graph.traversal()

g.V().has('name','Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(neq('user')).where(neq('friends'))
  groupCount().by('name').
  order(local).by(values, decr)
```



RDF Knowledge Graph Example

URIs as Globally Unique Identifiers



URIs to identify nodes and edge labels

`<https://permid.org/1-4295902158>`

=> identifies the company “Netflix Inc”

`organization:isIncorporatedIn`[1]

=> identifies the relationship “is incorporated in”

`<http://sws.geonames.org/6252001/>`

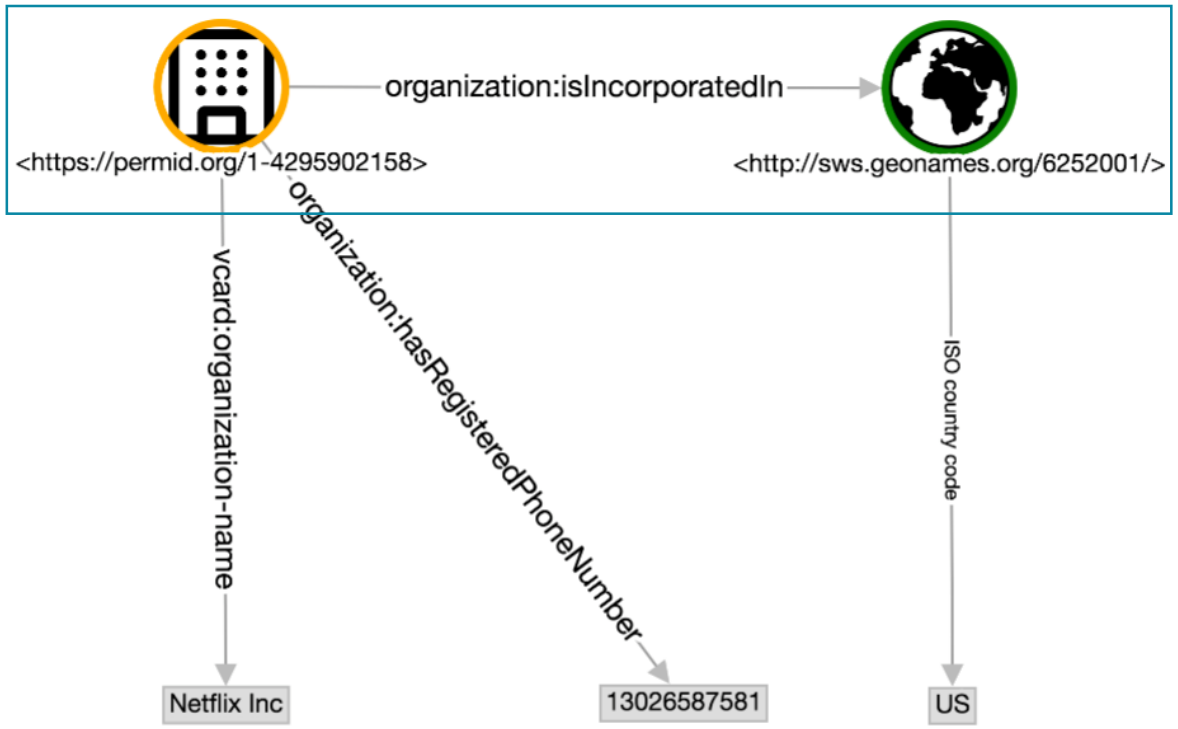
=> identifies country “USA”

[1] This is a shortcut for

`<http://permid.org/ontology/organization/isIncorporatedIn>`.

RDF uses XML prefix notation, where the prefix `organization` is a shortcut for `<http://permid.org/ontology/organization/>`.

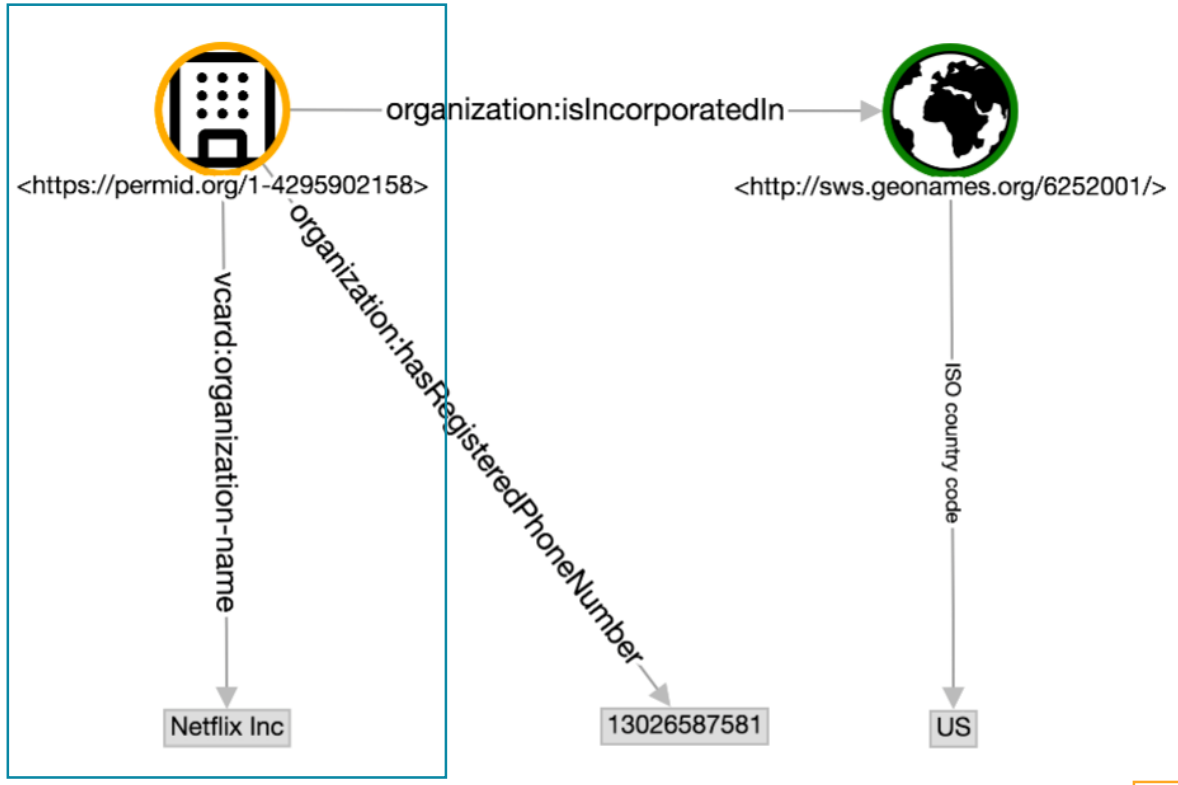
RDF Graph: Collection of Triples (1)



- # Every edge in the RDF graph is represented as
- # a (subject, predicate, object) triple

- <https://permid.org/1-4295902158> ← subject
- organization:isIncorporatedIn ← predicate
- <http://sws.geonames.org/6252001/> . ← object

RDF Graph: Collection of Triples (2)



- `# Every edge in the RDF graph is represented as`
- `# a (subject, predicate, object) triple`

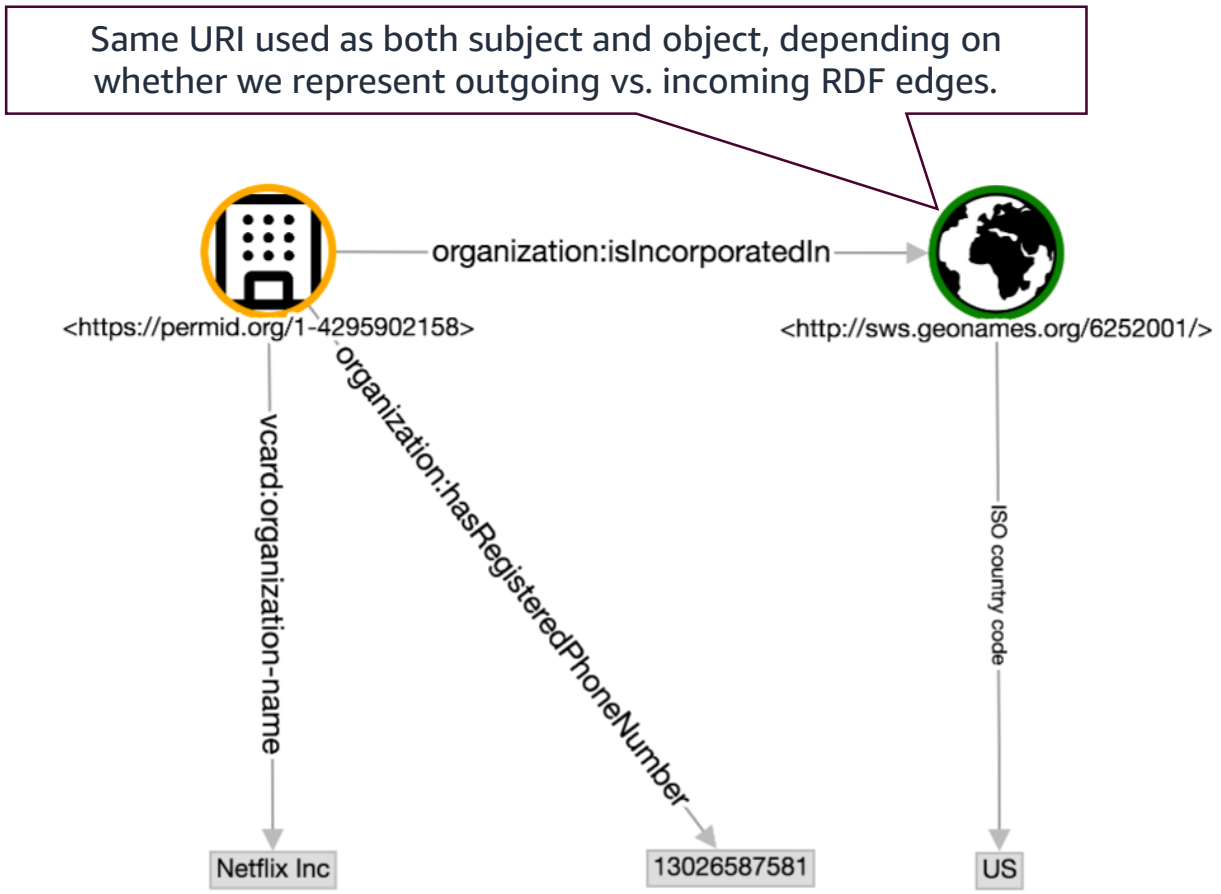
- `<https://permid.org/1-4295902158>` ← subject
- `  organization:isIncorporatedIn` ← predicate
- `  <http://sws.geonames.org/6252001/> .` ← object (URI)

- `<https://permid.org/1-4295902158>` ← subject
- `  vcard:organization-name` ← predicate
- `  "Netflix Inc" .` ← object (literal)

Literals are “sinks” in the graph. They do not have any outgoing edges.

RDF supports strings and other XML datatypes (bool, integer, dates, floats, doubles, …)

RDF Graph: Collection of Triples (3)



- `# Every edge in the RDF graph is represented as`
- `# a (subject, predicate, object) triple`

- `<https://permid.org/1-4295902158>`
- `  organization:isIncorporatedIn`
- `  <http://sws.geonames.org/6252001/> .`

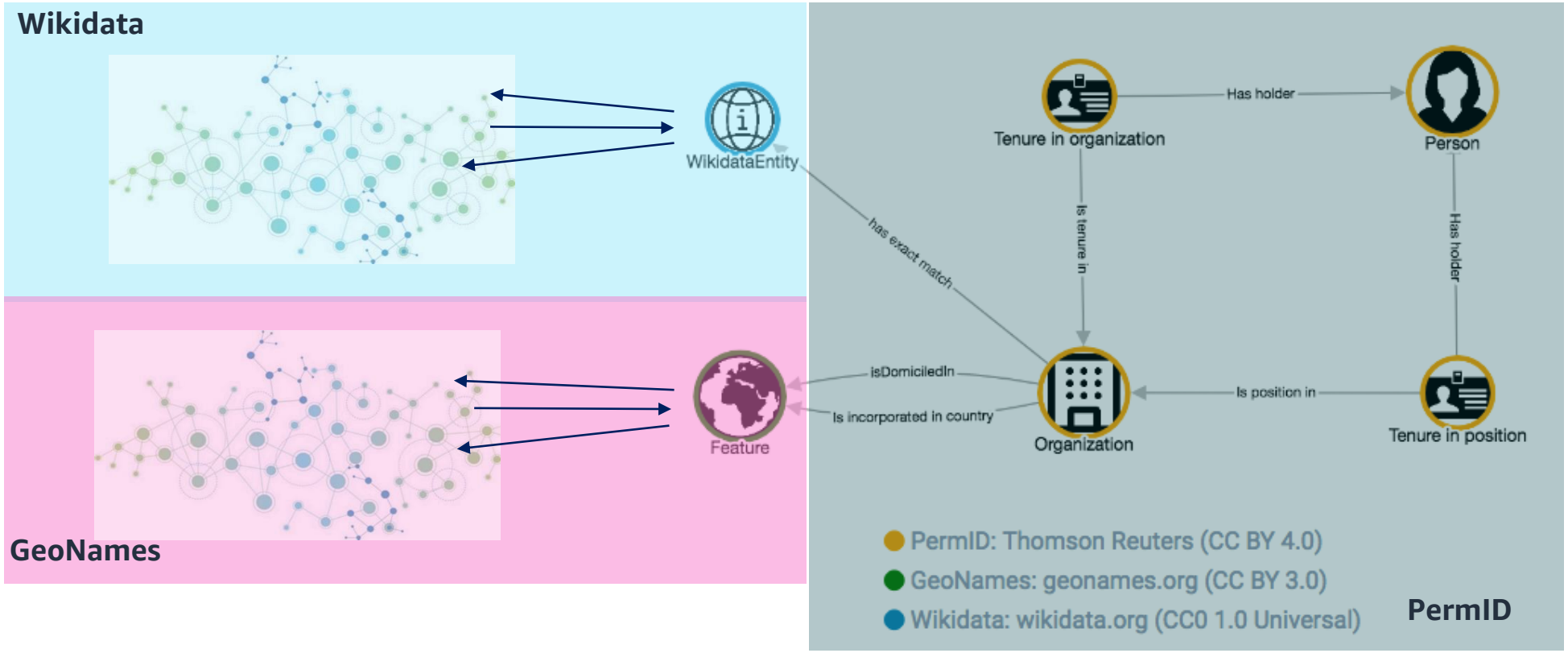
- `<https://permid.org/1-4295902158>`
- `   vcard:organization-name`
- `   "Netflix Inc" .`

- `<https://permid.org/1-4295902158>`
- `   organization:hasRegisteredPhoneNumber`
- `   "13026587581" .`

- `<http://sws.geonames.org/6252001/>`
- `  iso:countryCode`
- `  ”US” .`

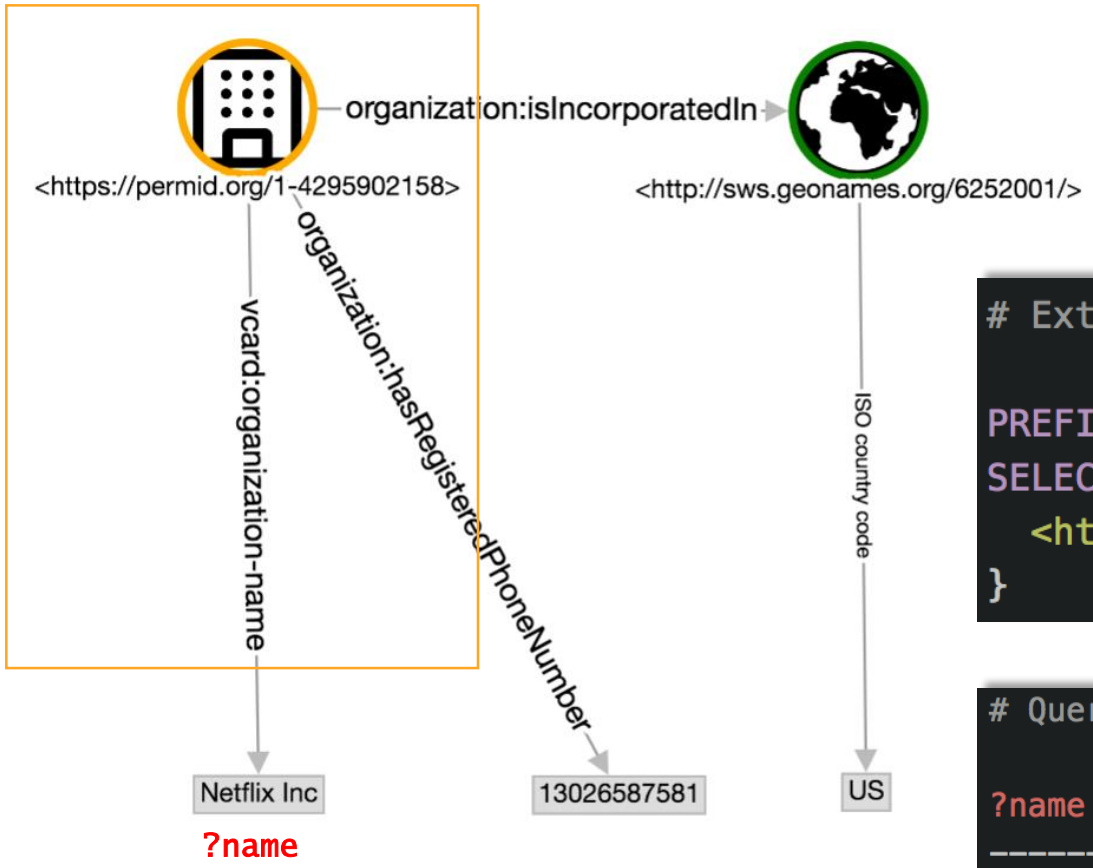
The Benefit of URIs: Linked Data

Linking across datasets by referencing globally unique URIs



Example: PermID (re)uses `<http://sws.geonames.org/6252001/>` as a global Identifier for the USA, which is an identifier rooted in GeoNames.

Querying RDF Using SPARQL (1)

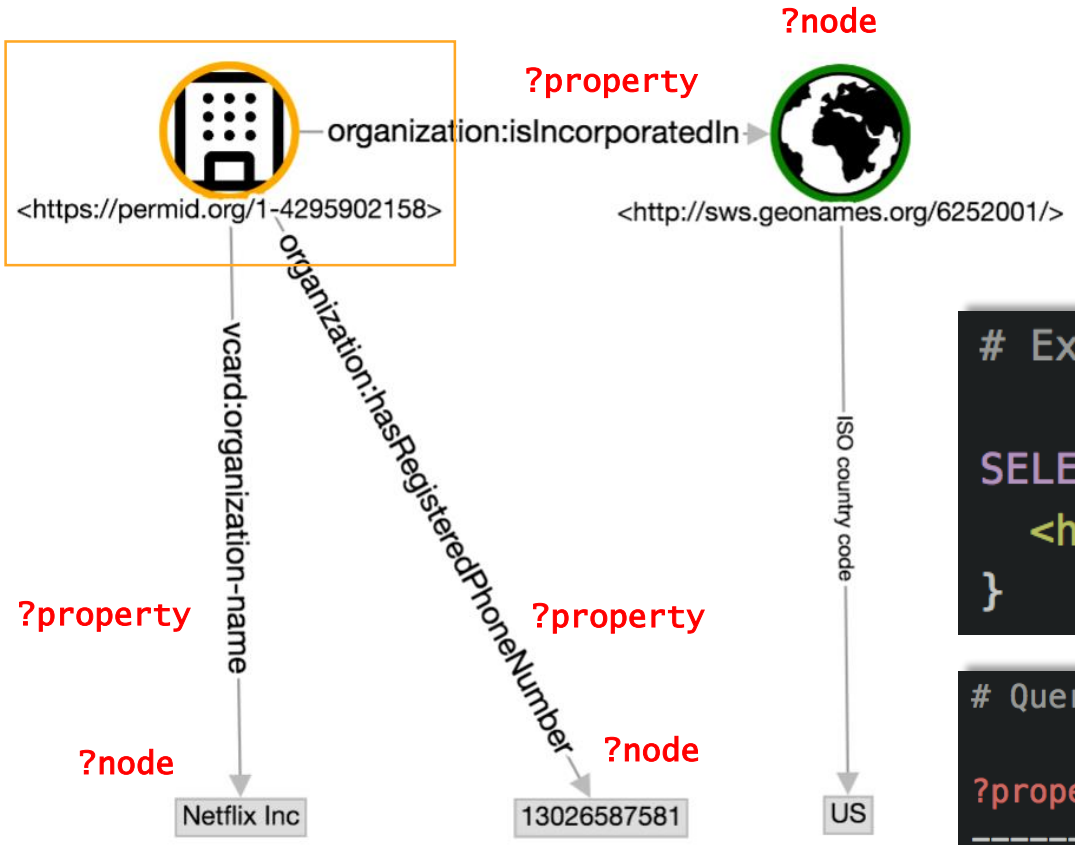


Variables are prefixed with "?".

```
# Extract the name of company Netflix (using its URI)

PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
SELECT ?name WHERE {
  <https://permid.org/1-4295902158> vcard:organization-name ?name
}
```

```
# Query result:

?name
-------------
"Netflix Inc"
```

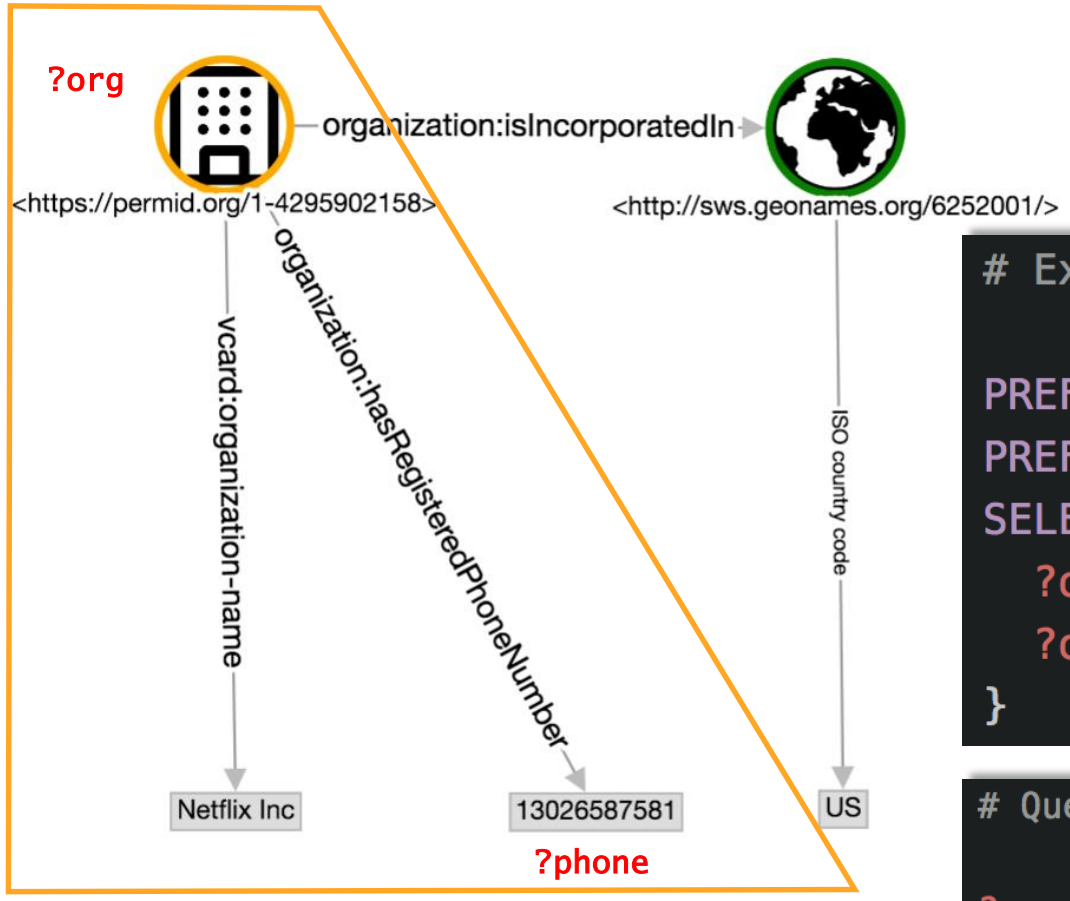
Querying RDF Using SPARQL (2)




```
# Extract outgoing predicate-to-object pairs for Netflix

SELECT ?property ?node WHERE {
  <https://permid.org/1-4295902158> ?property ?node
}
```

```
# Query result:

?property                                | ?node
-----------------------------------------|---------------------
vcard:organization-name                  | "Netflix Inc"
organization:hasRegisteredPhoneNumber    | "13026587581"
organization:isIncorporatedIn            | <http://sws/geo...>
```

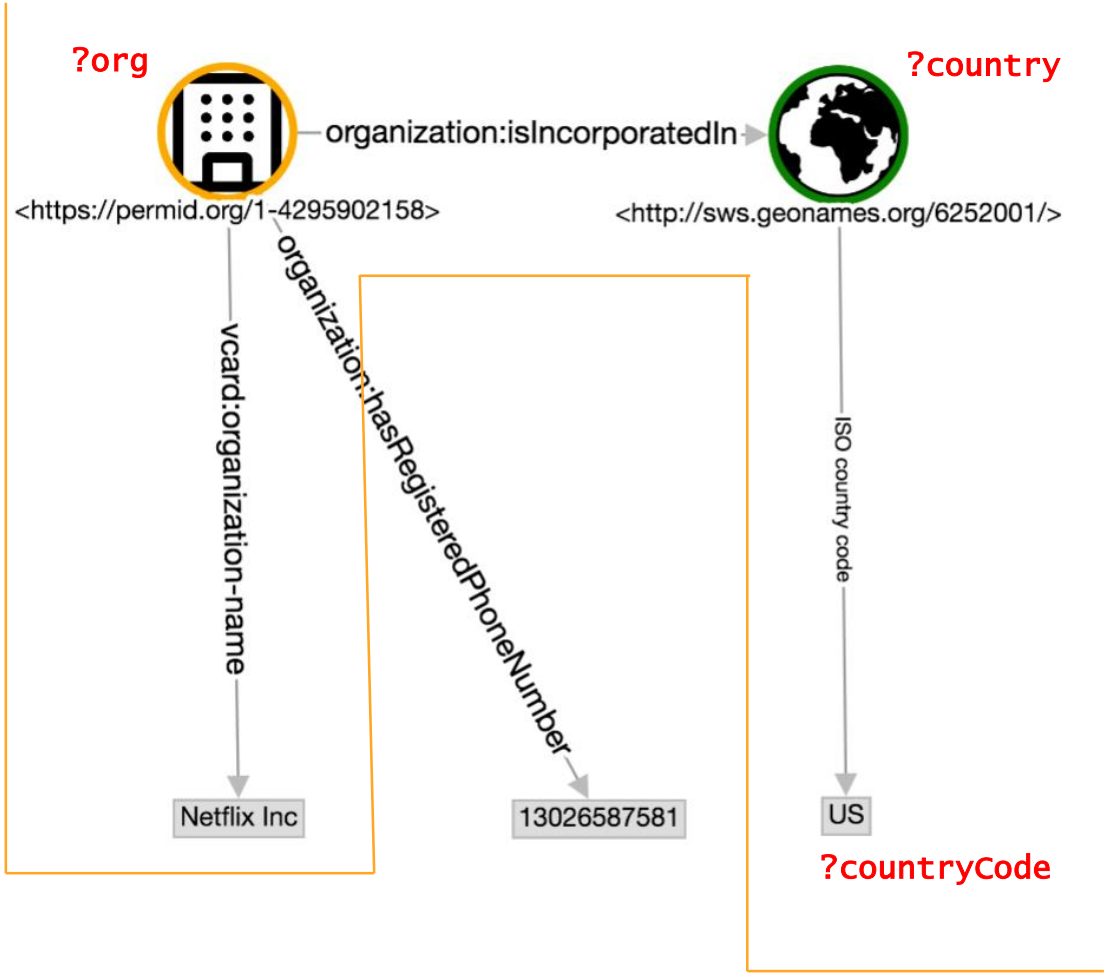
Querying RDF Using SPARQL (3)




```
# Extract URI and phone number of company "Netflix Inc"

PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX org: <http://permid.org/ontology/organization/>
SELECT ?org ?phone WHERE {
  ?org vcard:organization-name "Netflix Inc" .
  ?org org:hasRegisteredPhoneNumber ?phone
}
```

```
# Query result:

?org                                | ?phone
------------------------------------|---------------
<https://permid.org/1-4295902158> | "13026587581"
```

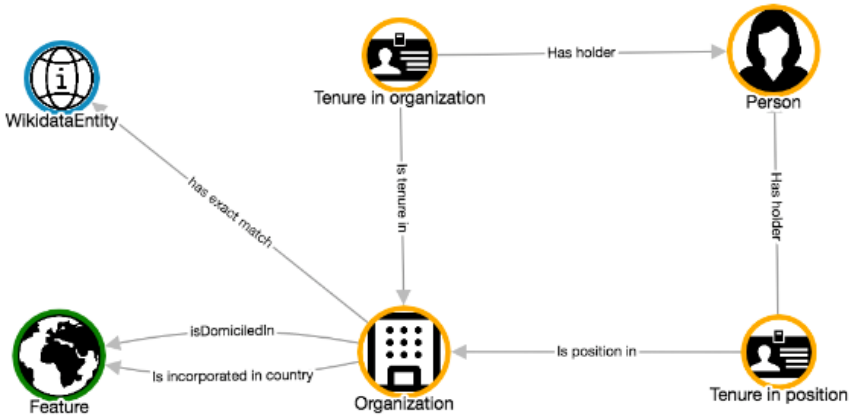
Querying RDF Using SPARQL (4)




```
# Extract the organization ID and country code of
# company "Netflix Inc"

PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX org: <http://permid.org/ontology/organization/>
PREFIX geo: <http://www.geonames.org/ontology#>

SELECT ?org ?countryCode WHERE {
  ?org vcard:organization-name "Netflix Inc" .
  ?org org:isIncorporatedIn ?country .
  ?country geo:countryCode ?countryCode
}
```

```
# Query result:

?org                                    | ?countryCode
----------------------------------------|--------------
<https://permid.org/1-4295902158>       | "US"
```

The Power of Linked Data



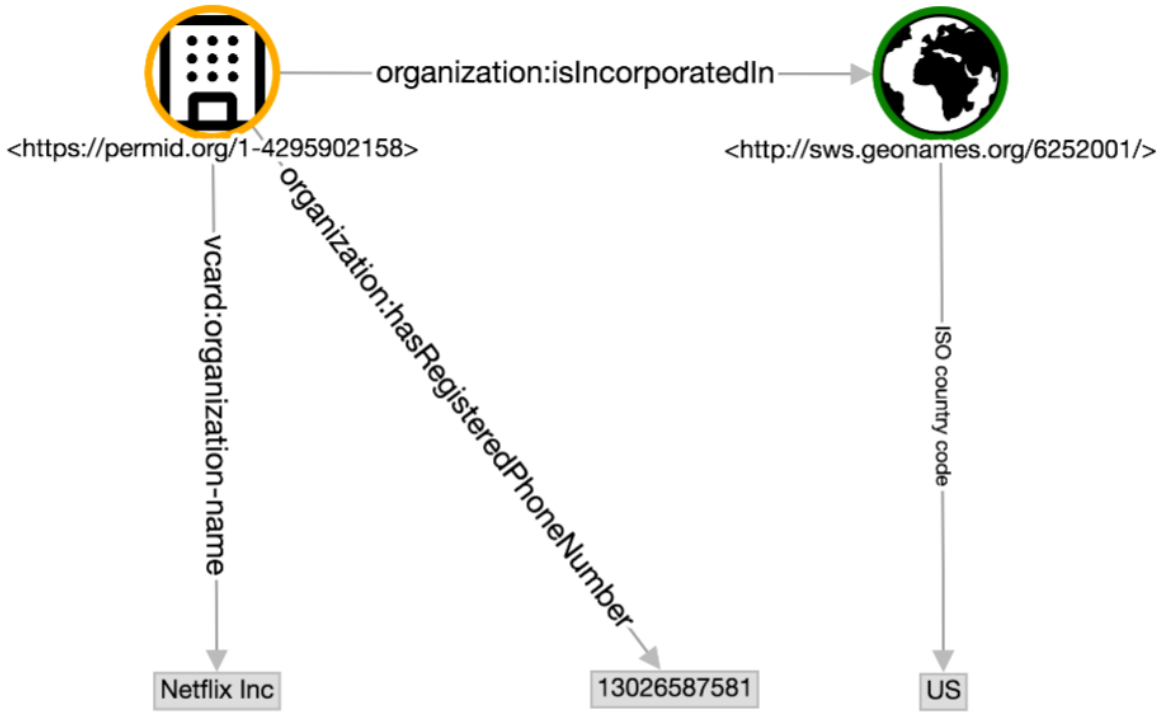
Data from PermID

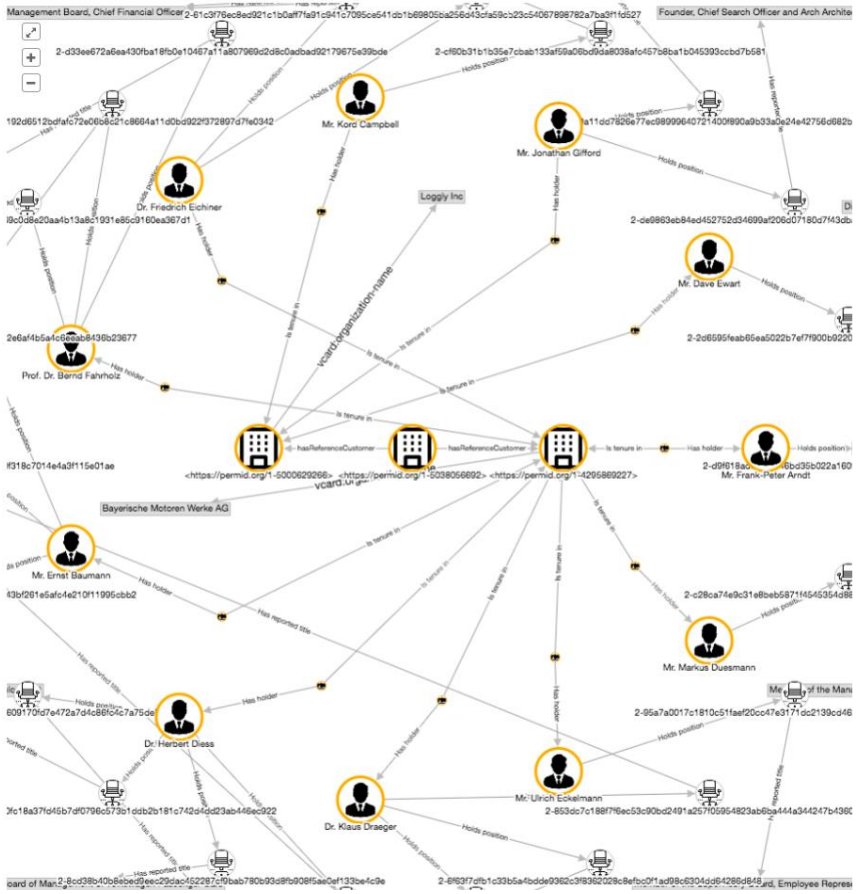
Data from Wikidata

Data from GeoNames

```
SELECT DISTINCT ?companyName ?companyId
                (SAMPLE(?stockExchangeWD) AS ?sampleStockExchangePerWD)
                (MAX(?employeeNumberWD) AS ?maxEmployeeNumberPerWD)
                ?locationEnglishName ?locationChineseName
WHERE {

  # extract company name, Wikidata URI, and location from PermID
  ?companyId vcard:organization-name ?companyName .
  ?companyId skos:exactMatch ?orgInWikidata .
  ?companyId fibo:isDomiciledIn ?location .

  # filter by companies with more than one hundred thousand employees
  # and extract stock exchange information from Wikidata
  ?orgInWikidata <http://www.wikidata.org/prop/P1128> ?employeeStmt .
  ?employeeStmt <http://www.wikidata.org/prop/statement/P1128> ?employeeNumberWD .
  FILTER (?employeeNumberWD > 10000)

  ?orgInWikidata <http://www.wikidata.org/prop/direct/P414> ?stockExchange .
  ?stockExchange rdfs:label ?stockExchangeWD .

  # extract the company location's English and Chinese name
  ?location geo:alternateName ?locationEnglishName .
  FILTER((LANG(?locationEnglishName)) = "en")
  ?location geo:alternateName ?locationChineseName .
  FILTER((LANG(?locationChineseName)) = "zh")

} GROUP BY ?companyName ?companyId ?location
           ?locationEnglishName ?locationChineseName
```

RDF Graphs

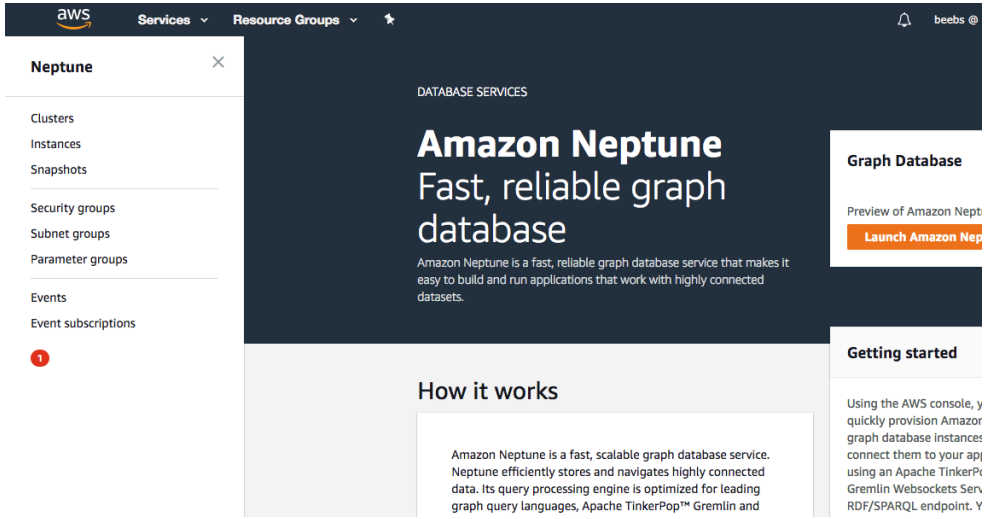


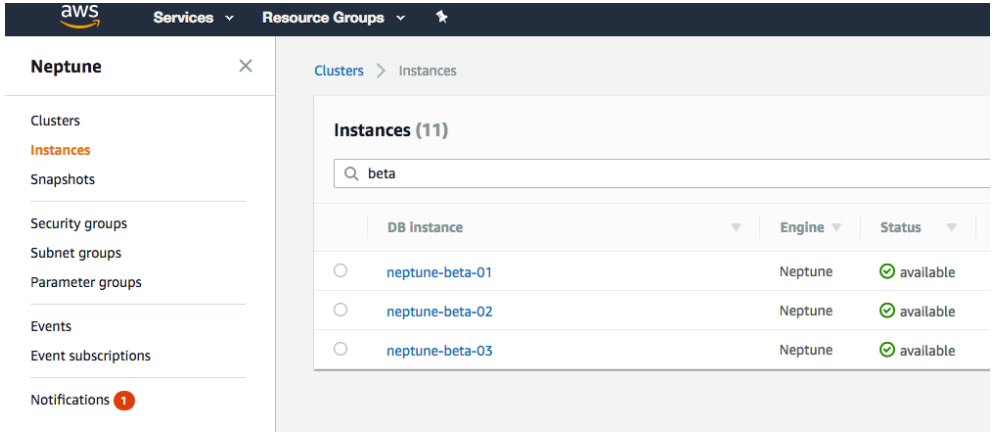


Fully Managed Service

BENEFITS

- Easily configurable via the Console
- Multi-AZ High Availability, ACID
- Support for up to 15 read replicas
- Supports Encryption at rest
- Supports Encryption in transit (TLS)
- Backup and Restore, Point-in-time Recovery





https://aws.amazon.com/neptune/

Learn from AWS experts. Advance your skills and knowledge. Build your future in the AWS Cloud.

Digital Training

Free, self-paced online courses built by AWS experts

Classroom Training

Classes taught by accredited AWS instructors

AWS Certification

Exams to validate expertise with an industry-recognized credential

Ready to begin building your cloud skills?
Get started at: https://www.aws.training/

With deep expertise on AWS, APN Partners can help your organization at any stage of your Cloud Adoption Journey.

AWS Managed Service Providers

APN Consulting Partners who are skilled at cloud infrastructure and application migration, and offer proactive management of their customer's environment.

AWS Competency Partners

APN Partners who have demonstrated technical proficiency and proven customer success in specialized solution areas.

AWS Marketplace

A digital catalog with thousands of software listings from independent software vendors that make it easy to find, test, buy, and deploy software that runs on AWS.

AWS Service Delivery Partners

APN Partners with a track record of delivering specific AWS services to customers.

Ready to get started with an APN Partner?
Find a partner: https://aws.amazon.com/partners/find/
Learn more at the AWS Partner Network Booth

Thank You for Attending AWS Innovate

We hope you found it interesting! A kind reminder to **complete the survey.**

Let us know what you thought of today's event and how we can improve the event experience for you in the future.


aws-apac-marketing@amazon.com


twitter.com/AWSCloud


facebook.com/AmazonWebServices


youtube.com/user/AmazonWebServices


slideshare.net/AmazonWebServices


twitch.tv/aws