hroug 24. godišnja konferencija

Graph Analytics
A New Way to Understand your Data

Albert Godfrind

Spatial and Graph Solutions Architect
Oracle
October 2019

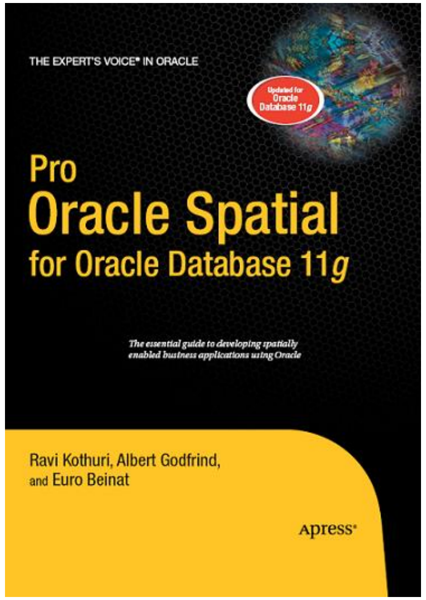
@agodfrin
albert.godfrind@oracle.com

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

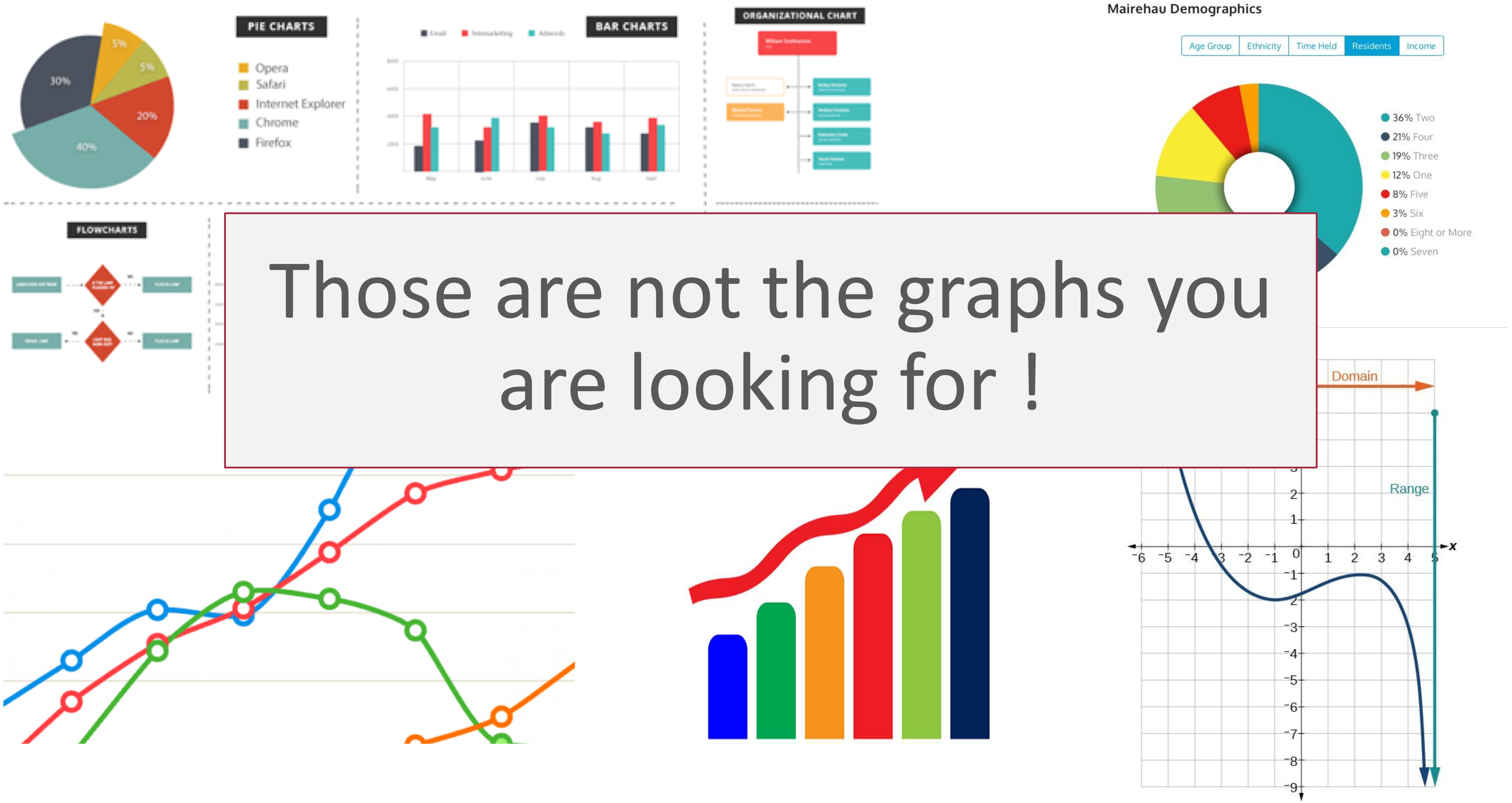
Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

- In IT for way too long!
- With Oracle for ever
- Oracle Spatial Evangelist
- Graph Evangelist
- Author



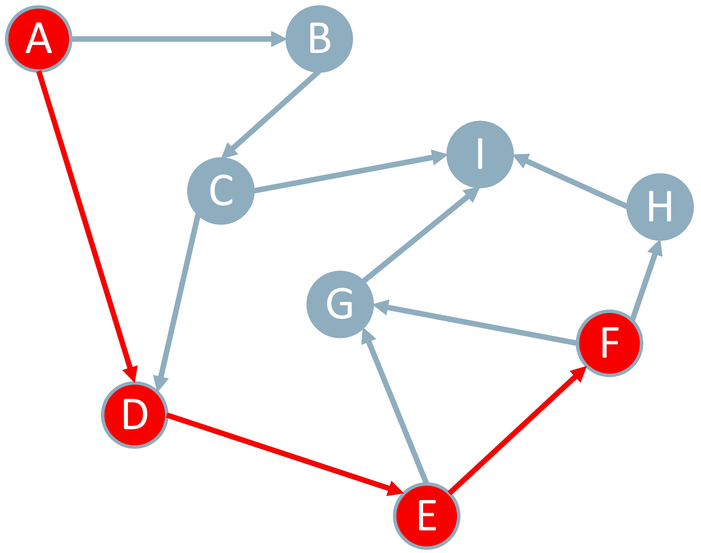
ORACLE®

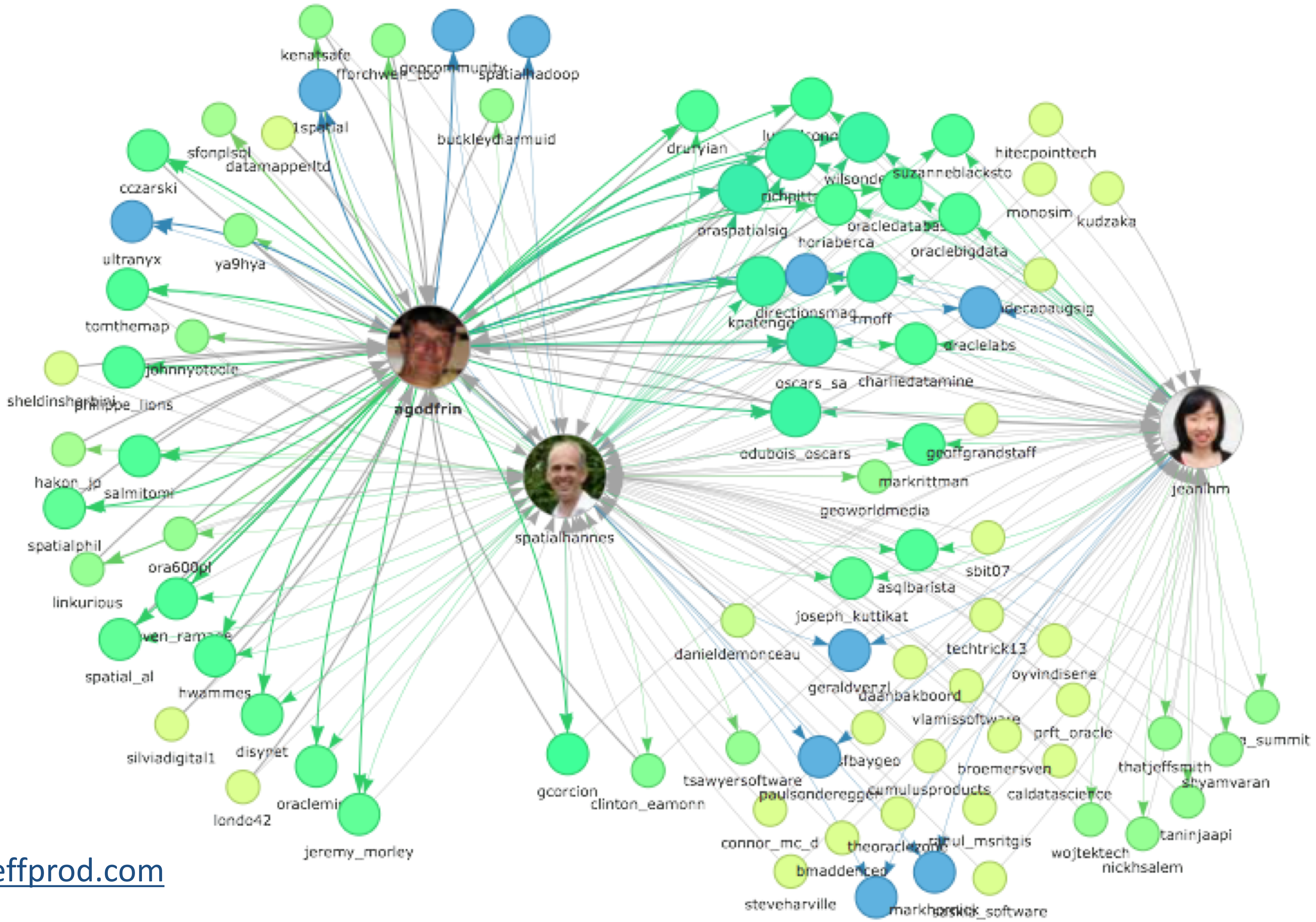
Those are not the graphs you are looking for !



ID	DEPENDS_ON
A	B
A	D
B	C
C	I
C	D
D	E
E	F
E	G
F	G
F	H
G	I
H	I

Does A depend on F ?





https://twitter.jeffprod.com

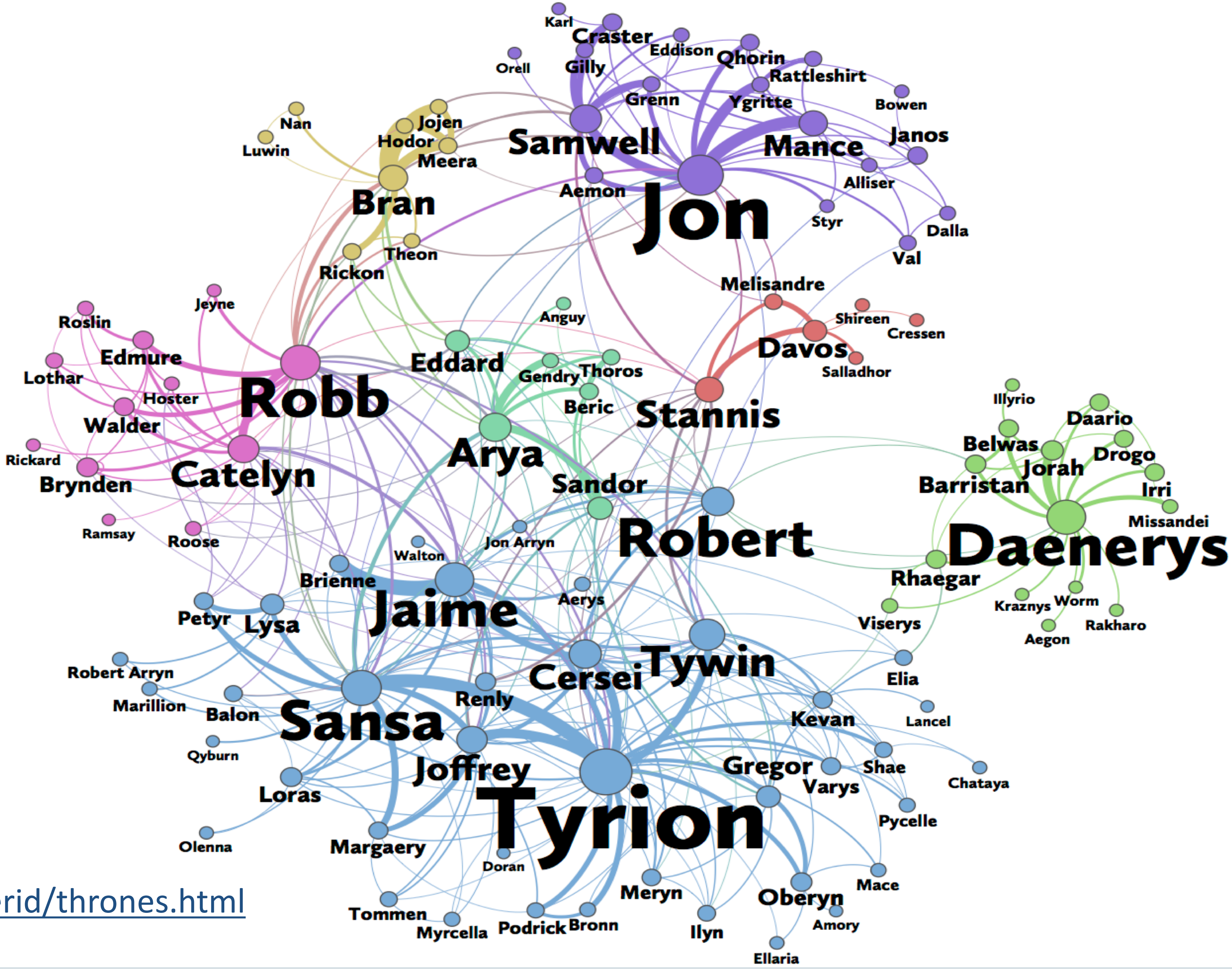
ORACLE

A social network generated from A Storm of Swords.

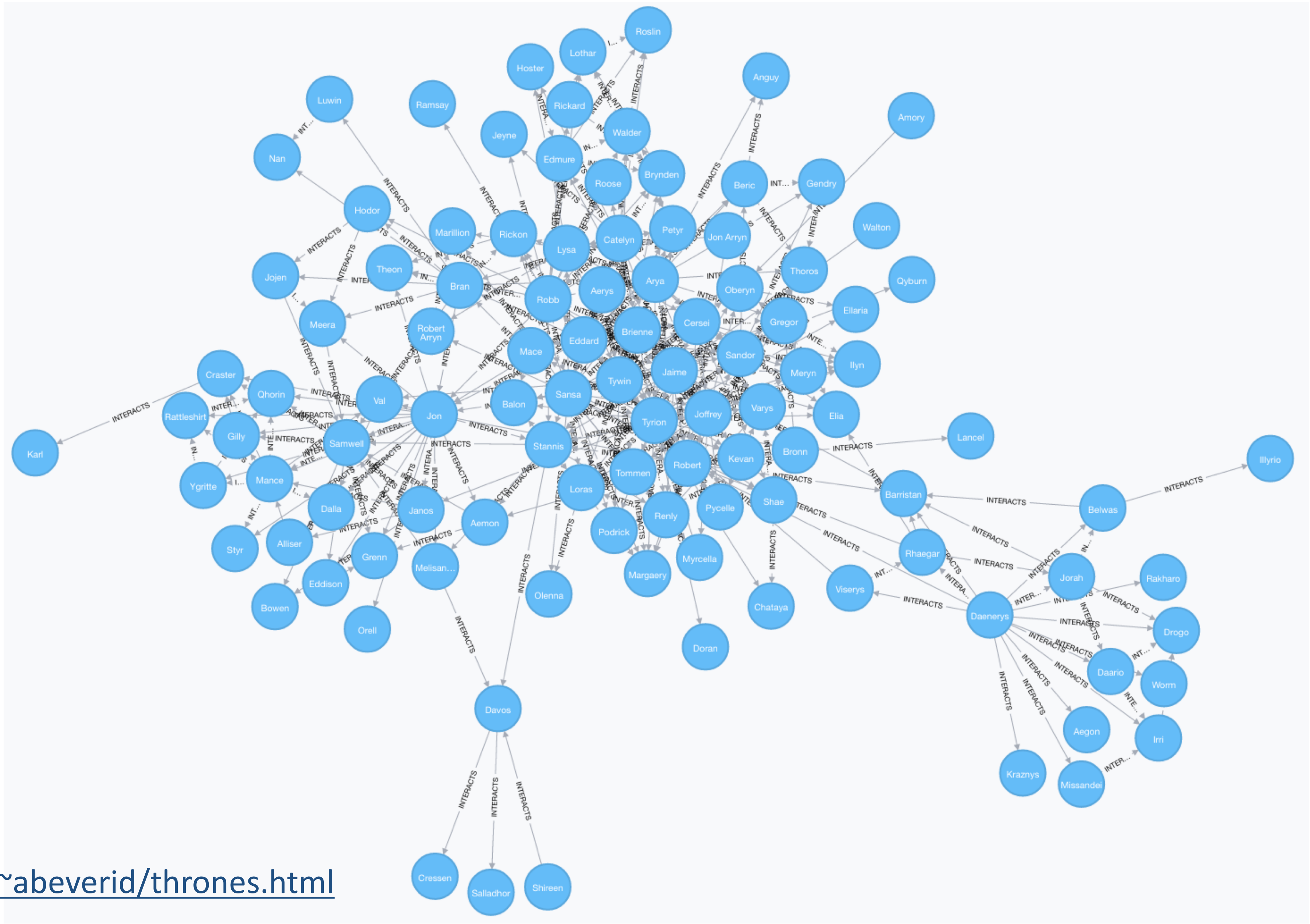
The color of a vertex indicates its community.

The size of a vertex corresponds to its PageRank value, and the size of its label corresponds to its betweenness centrality.

An edge's thickness represents its weight.



https://www.macalester.edu/~abeverid/thrones.html



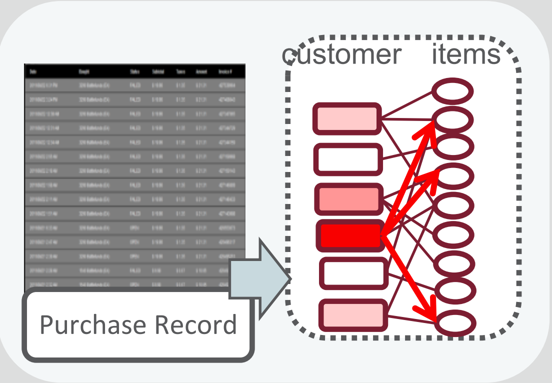
https://www.macalester.edu/~abeverid/thrones.html

ORACLE

Common Graph Analytics Use Cases

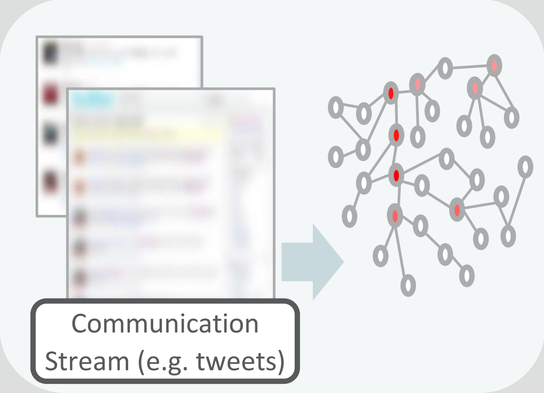
Recommend the most *similar* item purchased by *similar* people

Product Recommendation



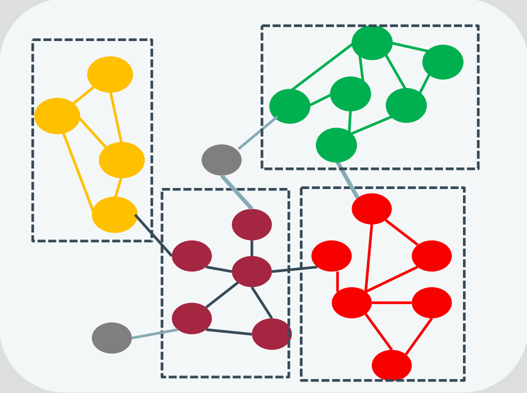
Find out people that are *central* in the given network – e.g. influencer marketing

Influencer Identification



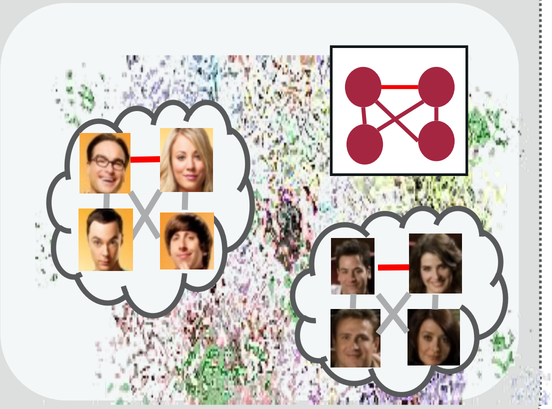
Identify group of people that are close to each other – e.g. target group marketing

Community Detection



Find out all the sets of entities that match to the given pattern – e.g. fraud detection

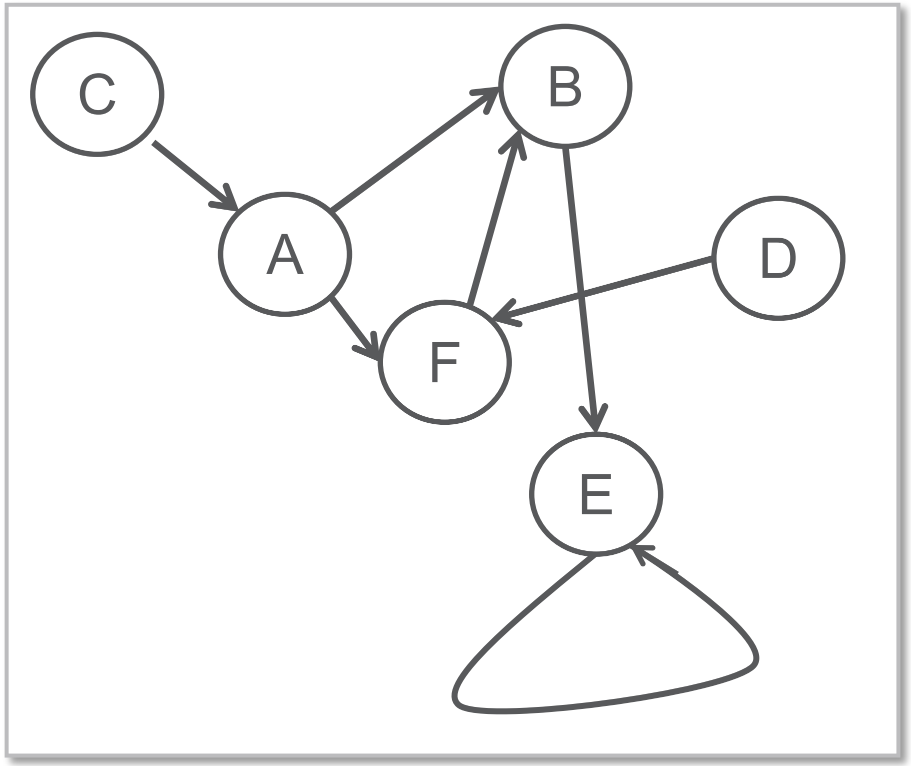
Graph Pattern Matching



ORACLE

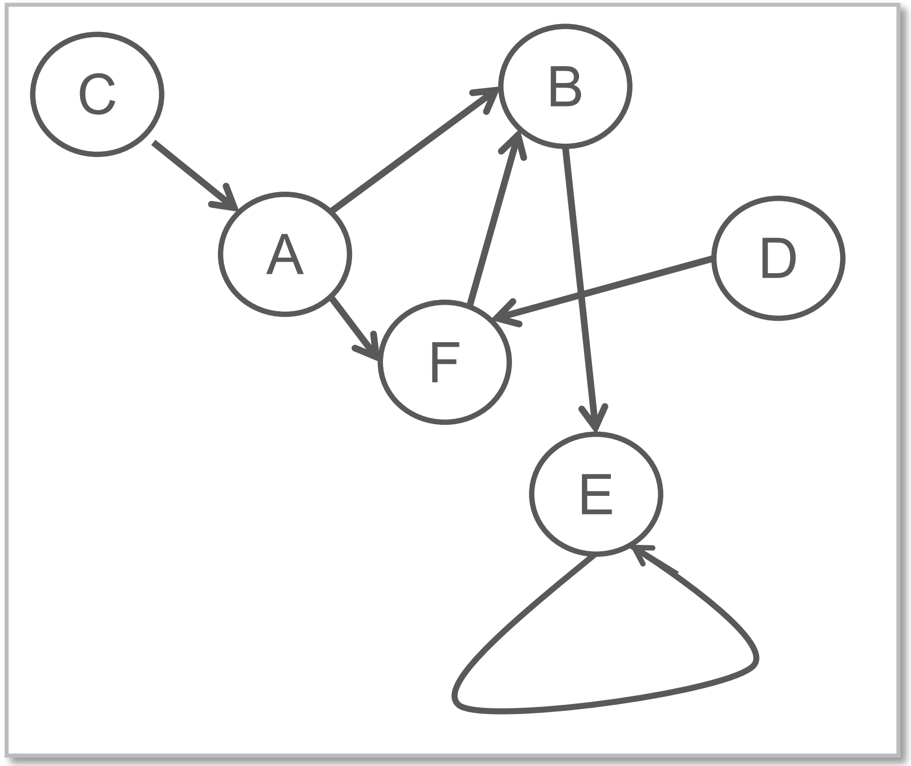
Graph Data Model

- What is a graph?
  - Data model representing entities as vertices and relationships as edges
  - Optionally including attributes
  - Also known as „linked data“
- What are typical graphs?
  - Social Networks
    - LinkedIn, facebook, Google+, ...
  - IP Networks, physical networks, ...
  - Knowledge Graphs
    - Apple SIRI, Google Knowledge Graph, ...



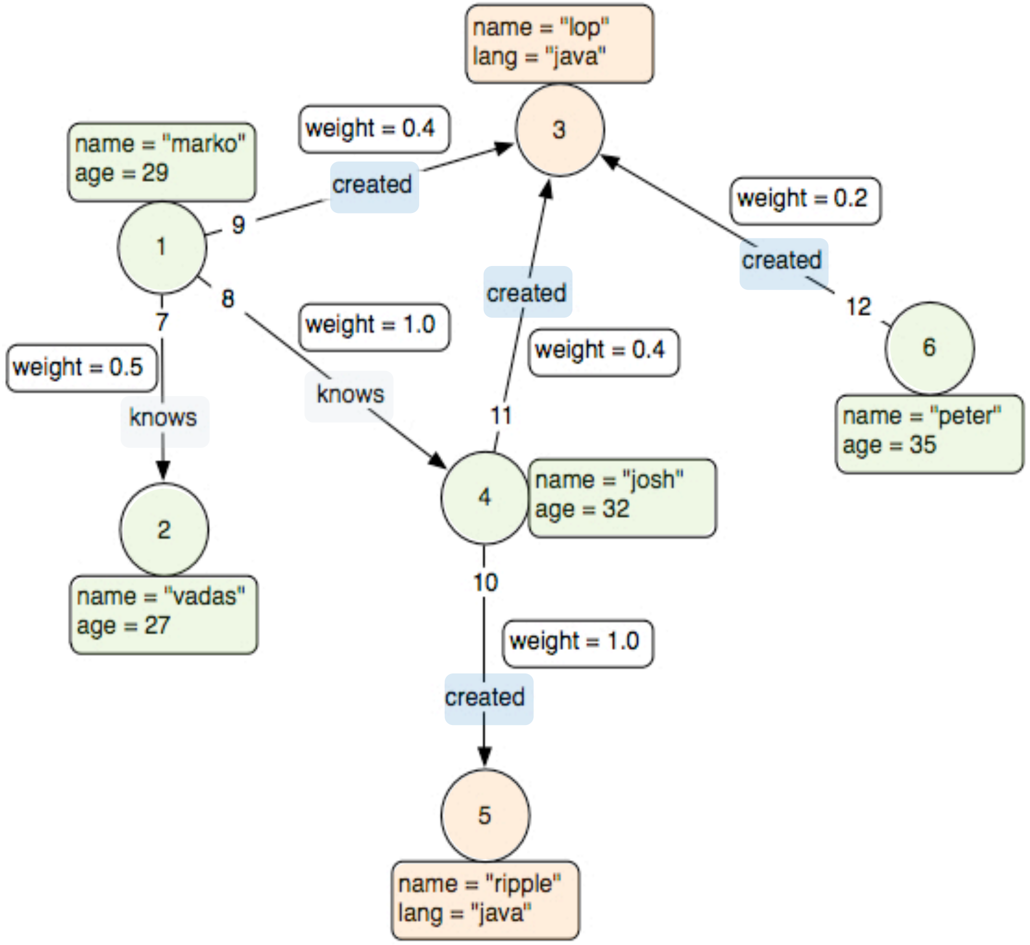
Graph Data Model

- Why are graphs popular?
  - Easy data modeling
    - „whiteboard friendly“
  - Flexible data model
    - No predefined schema, easily extensible
    - Particularly useful for sparse data
  - Insight from graphical representation
    - Intuitive visualization
  - **Enabling new kinds of analysis**
    - Overcoming some limitations in relational technology
    - Basis for Machine Learning (Neural Networks)



ORACLE

The Property Graph Data Model



- A set of **vertices** (or nodes)
  - each vertex has a unique identifier.
  - each vertex has a set of in/out edges.
  - each vertex has a collection of **key-value** properties.
- A set of **edges** (or links)
  - each edge has a unique identifier.
  - each edge has a head/tail vertex.
  - each edge has a label denoting type of relationship between two vertices.
  - each edge has a collection of **key-value** properties.

https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model

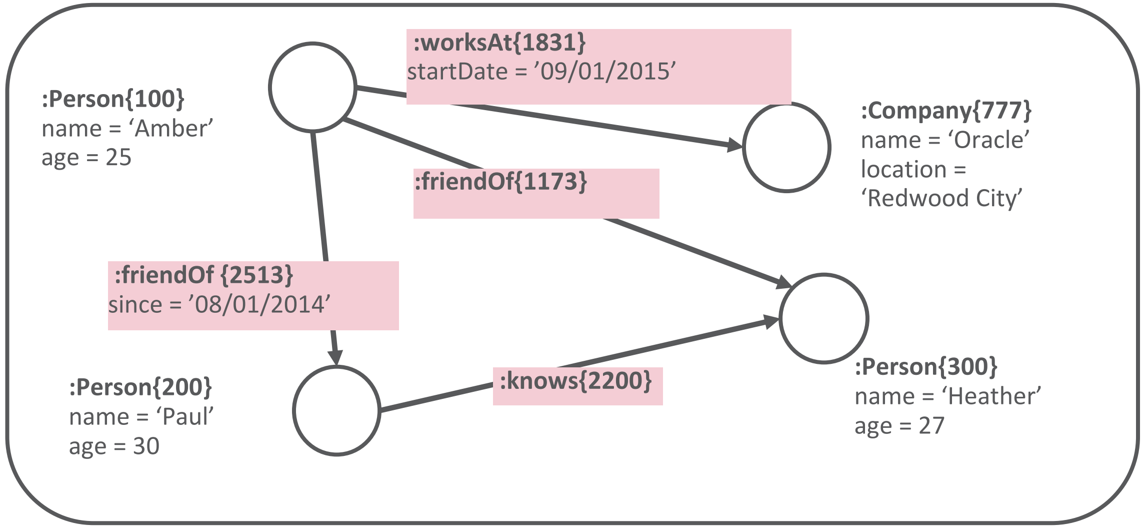
Two Categories of Graph Analysis

Computational Graph Analytics

- Apply well known algorithms
- Traversing graph or iterating over graph (usually repeatedly)
- Procedural logic
- Examples:
  - Shortest Path, PageRank, Weakly Connected Components, Centrality, ...

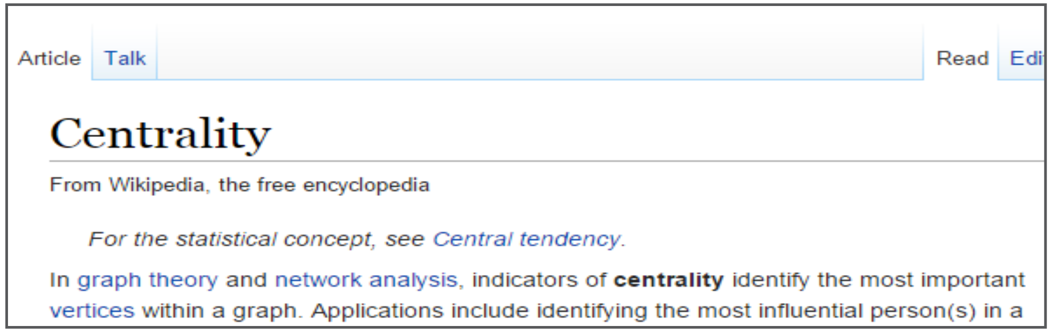
Graph Pattern Matching

- Based on description of pattern
- Find all matching sub-graphs



Graph Analysis: Influencer Identification

- Requirement:
  - Identify entities from a graph dataset that are relatively more important than others (from topology)
- Approaches:
  - Determine centrality of entities (concept based on graph theory)

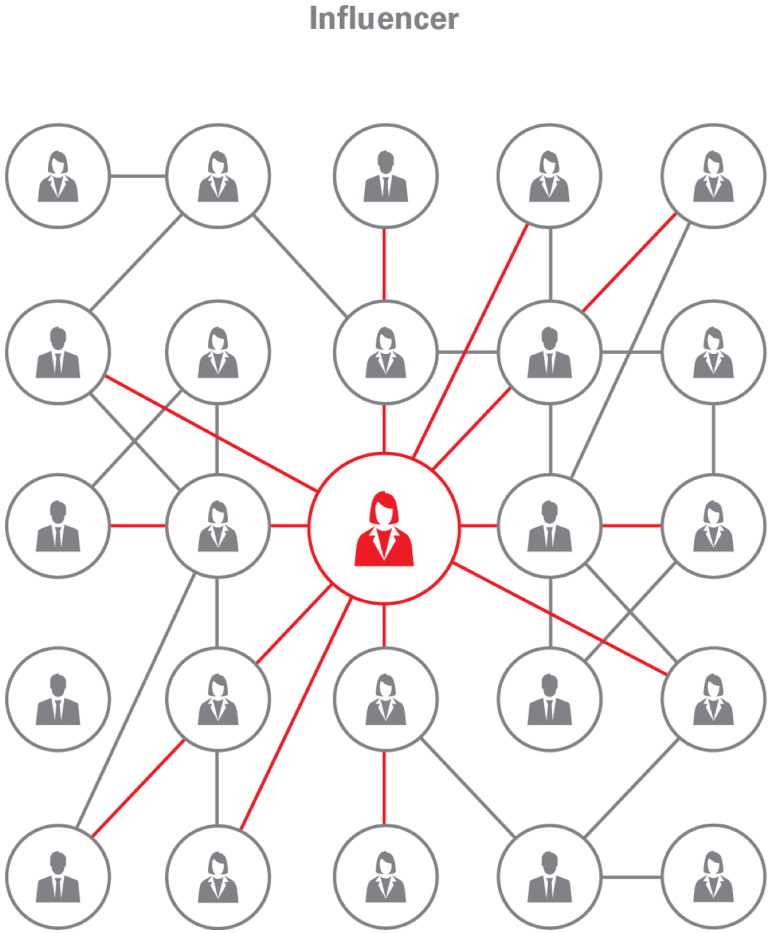

Article Talk Read Edi

Centrality

From Wikipedia, the free encyclopedia

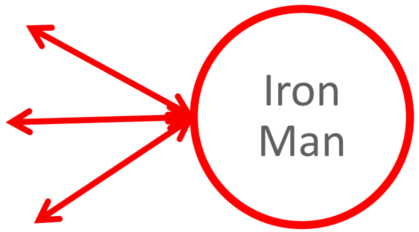
For the statistical concept, see Central tendency.

In graph theory and network analysis, indicators of **centrality** identify the most important vertices within a graph. Applications include identifying the most influential person(s) in a

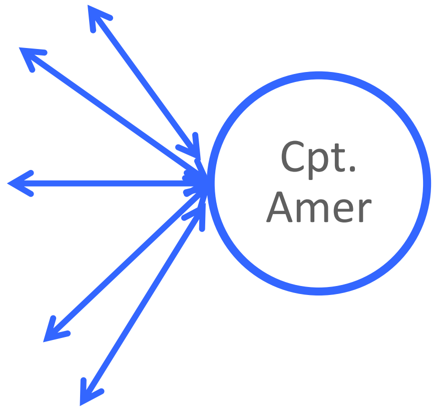


Importance as **Degree Centrality**

- The more **edges** a vertex has, the higher its **degree**



- The greater the degree, the more important the vertex is
- This is one way to look at importance

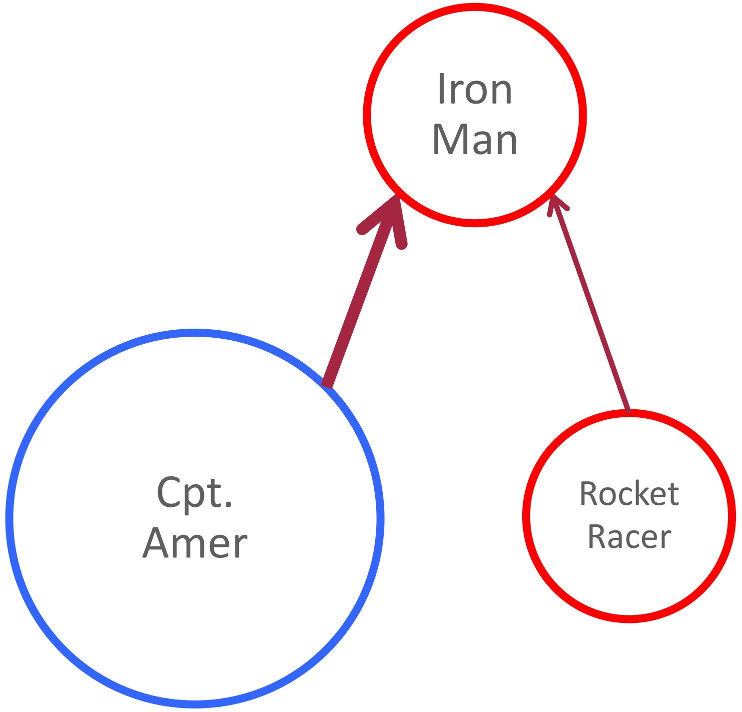


- Is your most connected customer most important?

ORACLE

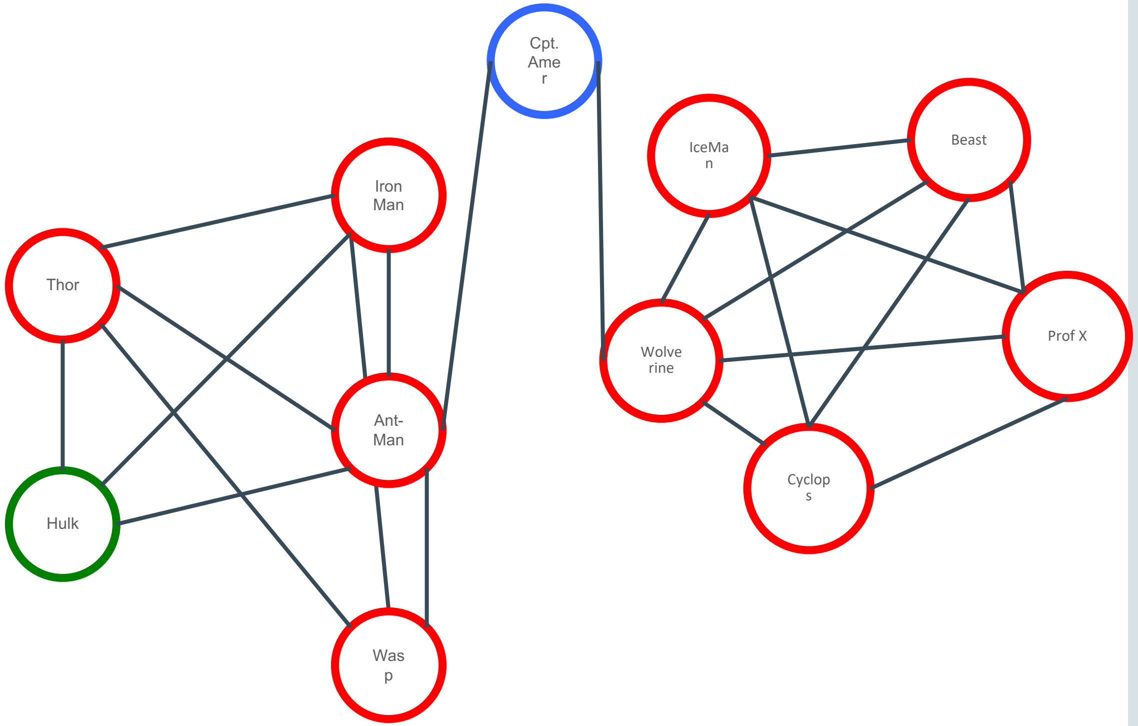
Importance as **Page Rank**

- Importance can flow **through** a graph
- A node connected to by important nodes is **also** important
- This is importance as a measure of
  - Trust
  - Prominence
- Thinking about customers in a graph requires multiple definitions of importance



Importance as **Betweenness**

- Importance can be how often you're on the critical path
- **Betweenness** is the number of shortest paths a node is part of
- E.g. The superhero on all the teams
- E.g. The player in all the scoring sequences

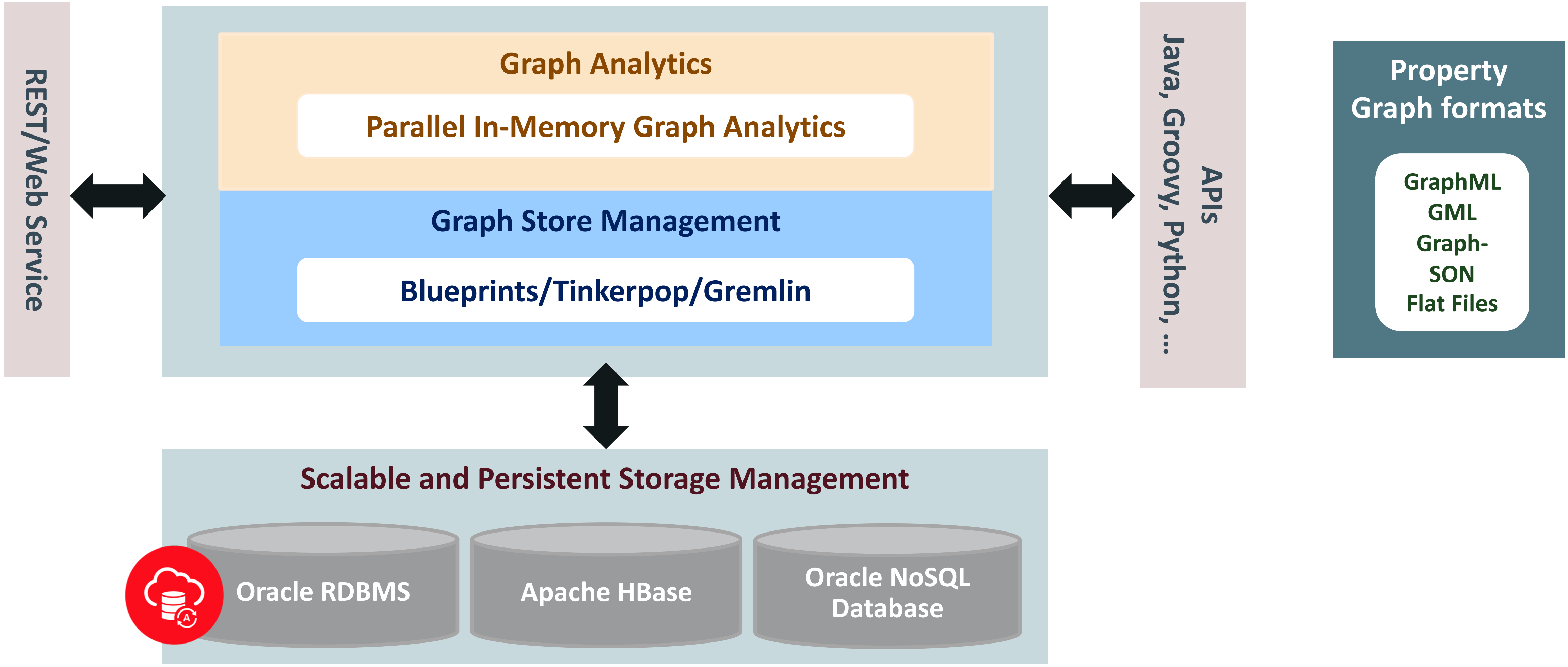


Targeted Marketing in Telco

- Model each **subscriber** as a **vertex** in the graph
- **Interactions** between subscribers are represented by **edges**
  - Based on call data records for voice, SMS, MMS ...
- Using **centrality** algorithms to determine important customers
- Target these customers with marketing campaigns for retention

ORACLE

Architecture of Oracle Property Graph



Constructing a Graph

- From a relational model
  - Rows in **tables** usually become **vertices**
  - **Columns** become **properties** on vertices
  - **Relationships** become **edges**
  - Join tables in n:m relations are transformed into relationships, columns become properties on edges
- Model may depend on requirements
  - Pattern matching, analysis, visualization, data integration, ...
- Modeling can involve trial-and-error approach
  - Unlike classical ER-modeling with its strict theoretical underpinning
- Graph can **evolve**, data model is not static
  - Add new vertex types, new edge types, new properties, ...

VIDEO_SALES_ORDERS

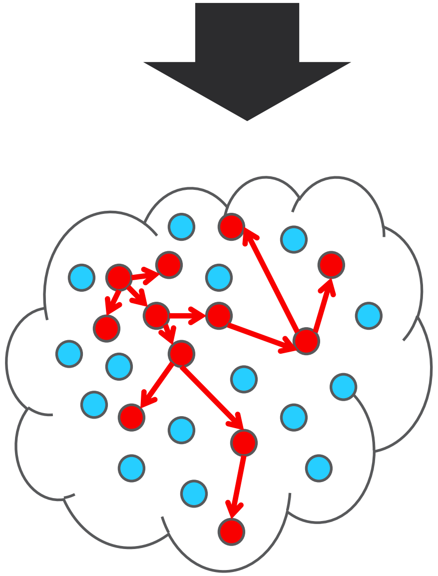
SALES_ID	CUST_NAME
10	SMITH
20	JONES
30	TURNER
40	ADAMS

SALES_ORDER_LINE_ITEMS

SALES_ID	LINE_ID	PROD_ID
10	1	1000
10	2	3000
20	1	4000
20	2	3000
20	3	2000
30	1	1000
30	2	1000
40	1	4000

VIDEO_PRODUCTS

PROD_ID	PROD_DESC
1000	TOY STORY
2000	TRUE LIES
3000	POPCORN
4000	STARGATE



Graph Schema for Oracle Database

Vertex Table: “<graph>VT$”

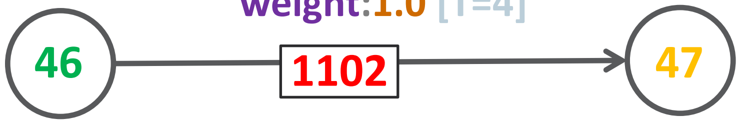
```
Name   Null?     Type
-----  --------  -----------------------
VID    NOT NULL  NUMBER
K                NVARCHAR2(3100)
T                INTEGER
V                NVARCHAR2(15000)
VN               NUMBER
VT               TIMESTAMP WITH TIMEZONE
```

name: Matthew McConaughey [T=1]
age: 47 [T=2]
birth-date:1969-11-04 12:00:00.0 [T=5]

Edge Table: “<graph>GE$”

```
Name     Null?     Type
-------  --------  -------------------------
EID      NOT NULL  NUMBER
SVID     NOT NULL  NUMBER
DVID     NOT NULL  NUMBER
EL                 NVARCHAR2(3100)
K                  NVARCHAR2(3100)
T                  INTEGER
V                  NVARCHAR2(15000)
VN                 NUMBER
VT                 TIMESTAMP WITH TIME ZONE
```

admires
weight:1.0 [T=4]



Source Tables

PERSONS		
PERSON_ID	NOT NULL	NUMBER
NAME		VARCHAR2(27)
COMPANY		VARCHAR2(27)
SHOW		VARCHAR2(23)
OCCUPATION		VARCHAR2(80)
TEAM		VARCHAR2(13)
RELIGION		VARCHAR2(22)
CRIMINAL_CHARGE		VARCHAR2(58)
MUSIC_GENRE		VARCHAR2(9)
ROLE		VARCHAR2(30)
POLITICAL_PARTY		VARCHAR2(34)
COUNTRY		VARCHAR2(14)

ORGANIZATIONS		
ORG_ID	NOT NULL	NUMBER
NAME		VARCHAR2(27)
TYPE		VARCHAR2(59)
RELIGION		VARCHAR2(22)
GENRE		VARCHAR2(15)
COUNTRY		VARCHAR2(14)

RELATIONS		
RELATION_ID	NOT NULL	NUMBER
FROM_ID		NUMBER
FROM_TYPE		VARCHAR2(12)
TO_ID		NUMBER
TO_TYPE		VARCHAR2(12)
RELATION_TYPE		VARCHAR2(12)

```
RELATION_ID    FROM_ID FROM_TYPE         TO_ID TO_TYPE      RELATION_TYP
----------- ---------- ------------ ---------- ------------ ------------
       1007          6 person                7 person       collaborates
       1017         10 person                1 person       feuds
       1019         11 person                1 person       collaborates
       1025         12 organization         14 person       collaborates
       1029         17 person               15 organization collaborates
...
       1116          3 person               55 person       collaborates
       1118          3 person               45 organization collaborates
       1122          5 person               58 person       collaborates
       1124         57 person               58 person       collaborates
       1130         56 person               57 person       collaborates
       1142         37 organization         66 organization feuds
```

Loading the Vertices

- Load vertices for **PERSONS**

```
insert into connectionsvt$ (vid,k,t,v)
select person_id, replace(lower(k),'_',' ') as k, 1 as t, v
from persons
  unpivot (
    v for (k) in (
      name,
      company,
      show,
      occupation,
      team,
      religion,
      criminal_charge,
      music_genre,
      role,
      political_party,
      country
    )
  )
order by person_id, k;
```

- Load vertices for **ORGANIZATIONS**

```
insert into connectionsvt$ (vid,k,t,v)
select org_id, replace(lower(k),'_',' ') as k, 1 as t, v
from organizations
  unpivot (
    v for (k) in (
      name,
      type,
      religion,
      genre,
      country
    )
  )
order by org_id, k;
```

Use the **UNPIVOT** function to turn columns into rows

ORACLE

Interacting with Graphs

- Access through **APIs**
  - Implementation of Apache Tinkerpop Blueprints APIs
  - Based on Java, REST plus SolR Cloud/Lucene support for text search
- **Scripting**
  - Groovy, Python, Javascript, ...
  - Notebooks: Apache Zeppelin, Jupyter
- **Graphical UIs**
  - Cytoscape
  - Commercial Tools such as TomSawyer Perspectives
  - Oracle pgVIZ

Pre-Built Analytics

- Local Clustering Coefficient
- Matrix Factorization (Gradient Descent)
- PageRank and variants
- Center
- Closeness Centrality and variants
- Degree Centrality and variants
- Periphery
- Radius
- Random Walk with Restart
- Degree Distribution and variants
- Diameter
- Dijkstra's Algorithm and variants
- SALSA and variants
- SSSP (Bellman Ford) and variants
- SSSP (Hop Distance) and variants
- Bidirectional Dijkstra's Algorithm (and variants)
- Eigenvector Centrality
- Fattest-Path
- Strongly Connected Components (Kosaraju)
- Strongly Connected Components (Tarjan)
- Triangle Counting
- Filtered BFS
- Hyperlink-Induced Topic Search
- K-Core
- Vertex Betweenness Centrality and variants
- Weakly Connected Components

Using the Groovy Shell for Analytics

- Start the shell

```
$ cd /opt/oracle/oracle-spatial-graph/property_graph/pgx/bin
$ ./pgx

PGX Shell 3.1.0
type :help for available commands
variables instance, session and analyst ready to use
pgx>
```

- Load the graph in memory

```
pgx> pg = session.readGraphWithProperties("connections_config.json");
==> PgxGraph[name=connections,N=78,E=164,created=1488925033245]
```

Page Rank

- Compute Pagerank values

```
rank=analyst.pagerank(graph:pg, max:1000);
==> VertexProperty[name=approx_pagerank,type=double,graph=connections]
```

- Show the top influencers

```
rank.getTopKValues(3).each{println it.key.getProperty("name")+" "+it.value}

Barack Obama 0.0608868998919989
Nicolas Maduro 0.03445628038301776
NBC 0.027831790283775117
```

ORACLE

Community Detection

- Run the Weakly Connected Components algorithm

```
wcc = analyst.wcc(pg)
==> ComponentCollection[name=compproxy_10,graph=connections]
```

- Run label propagation

```
partition = analyst.communitiesLabelPropagation(pg)
==> VertexCollectionWrap[name=compproxy_10,graph=connections]
...
==> VertexCollectionWrap[name=compproxy_10,graph=connections]
```

- Each vertex now has a new property: **label_propagation**
  - Values 0 to N (number of partitions)

ORACLE

Community Detection

- Which community contains the vertex “Alibaba” ?

```
v = pg.getVertices(new VertexFilter("vertex.name = 'Alibaba'"))[0];
==> PgxVertex[ID=65]
vc = partition.getPartitionByVertex(v);
```

- Who else is in that community ?

```
vc.each{println it.getId()+" "+it.getProperty("name")}
73 Nest
71 Pony Ma
64 Jeff Bezos
68 Jack Ma
37 Amazon
66 eBay
70 Carl Icahn
65 Alibaba
67 Google
69 Tencent
72 Facebook
```

ORACLE

PGQL · Property Graph Query Language   1.1 Specification   1.0 Specification   White Paper   Parser   PGX

PGQL · Property Graph Query Language

An SQL-like query language for graphs

Try It

Graphs + SQL

PGQL is a graph pattern matching query language for the property graph data model, inspired by Cypher, SQL, and G-CORE. PGQL combines Cypher-like ASCII art syntax with familiar constructs from SQL, such as `SELECT`, `FROM` and `WHERE`. PGQL also provides powerful constructs for matching regular path expressions (e.g. `PATH`).

An example PGQL query is as follows:

```
SELECT p2.name AS friend_of_friend
  FROM facebook_graph                             /* In the Facebook graph..   */
 MATCH (p1:Person) -/:friend_of{2}/-> (p2:Person) /* ..match two-hop friends.. */
 WHERE p1.name = 'Mark'                           /* ..of Mark.                */
```

See PGQL 1.1 Specification for a detailed specification of the language.

Graph Pattern Matching

PGQL uses ASCII art syntax for matching vertices, edges, and paths:

- `(n:Person)` matches a **vertex** (node) `n` with label `Person`
- `-[e:friend_of]->` matches an **edge** `e` with label `friend_of`
- `-/:friend_of+/->` matches a **path** consisting of one or more (`+`) edges, each with label `friend_of`

SQL Capabilities

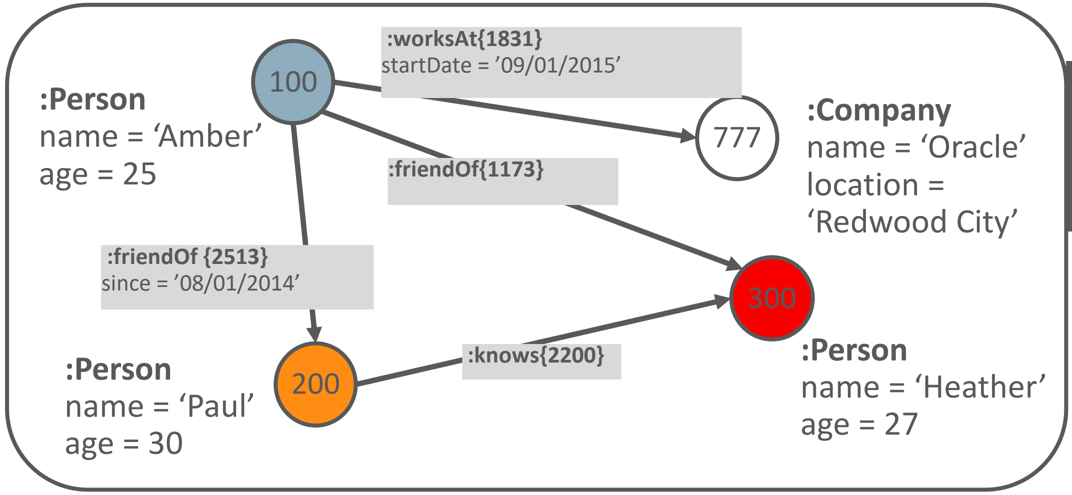
http://pgql-lang.org

ORACLE

Basic graph pattern matching

- Find all instances of a given pattern/template in the data graph

```
SELECT v3.name, v3.age
  FROM socialNetworkGraph
 MATCH (v1:Person) -[:friendOf]-> (v2:Person) -[:knows]-> (v3:Person)
 WHERE v1.name = ‘Amber’
```



socialNetwork Graph

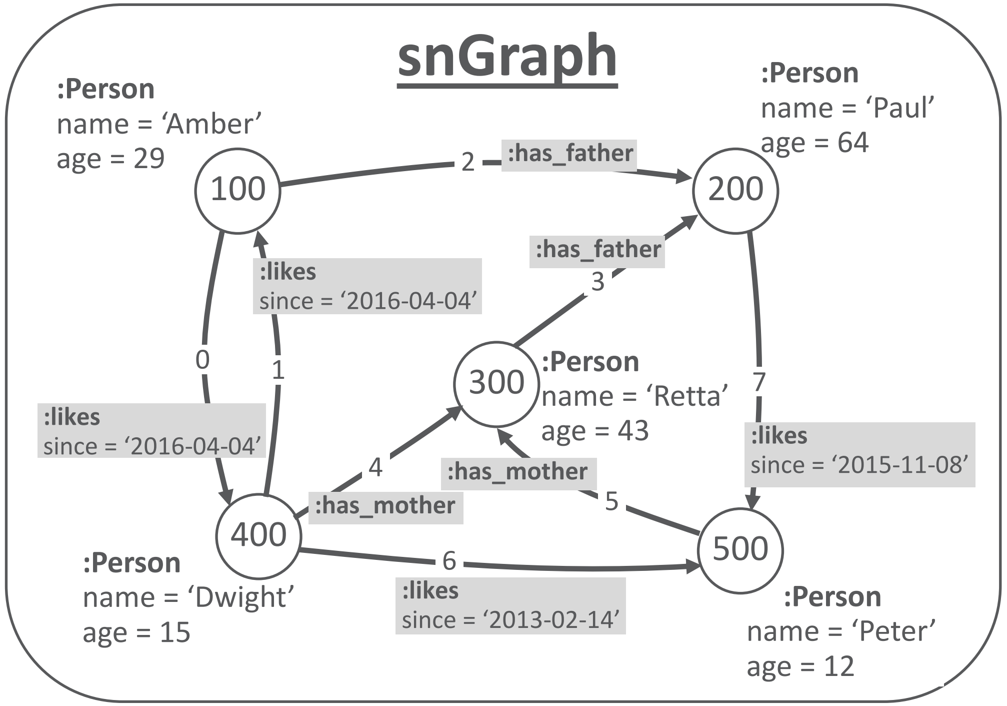
Query: Find all people who are known by friends of ‘Amber’.

Regular path expressions

```
  PATH has_parent AS (child) -[:has_father|has_mother]-> (parent)
SELECT x.name, y.name, ancestor.name
  FROM snGraph
 MATCH (x:Person) -/:has_parent+/-> (ancestor)
     , (y:person) -/:has_parent+/-> (ancestor)
 WHERE x.name = 'Peter' AND x <> y
```

Matching a pattern repeatedly

- Define a **PATH** expression at the top of a query
- Instantiate the expression in the **MATCH** clause
- Match **repeatedly**, e.g. zero or more times (*) or one or more times (+)



Executing PGQL

```
session.queryPgql(" \
  SELECT x.name, y.name \
  FROM connections
  MATCH (x) -[:leads]-> (y) \
  ORDER BY x.name, y.name \
").print()
```

```
+------------------------------------+
| x.name           | y.name          |
+------------------------------------+
| Bobby Murphy     | Snapchat        |
| Ertharin Cousin  | World Food Programme |
| Evan Spiegel     | Snapchat        |
| Google           | Nest            |
| Jack Ma          | Alibaba         |
| Jeff Bezos       | Amazon          |
| Pony Ma          | Tencent         |
| Pope Francis     | The Vatican     |
| Tony Fadell      | Nest            |
+------------------------------------+
```

ORACLE

Using Analytics Results

- Page Rank

```
pg.queryPgql("""
  SELECT id(v) as id, v.name, v.pagerank
  MATCH (v)
  ORDER by v.pagerank DESC
  LIMIT 4
""").print()
```

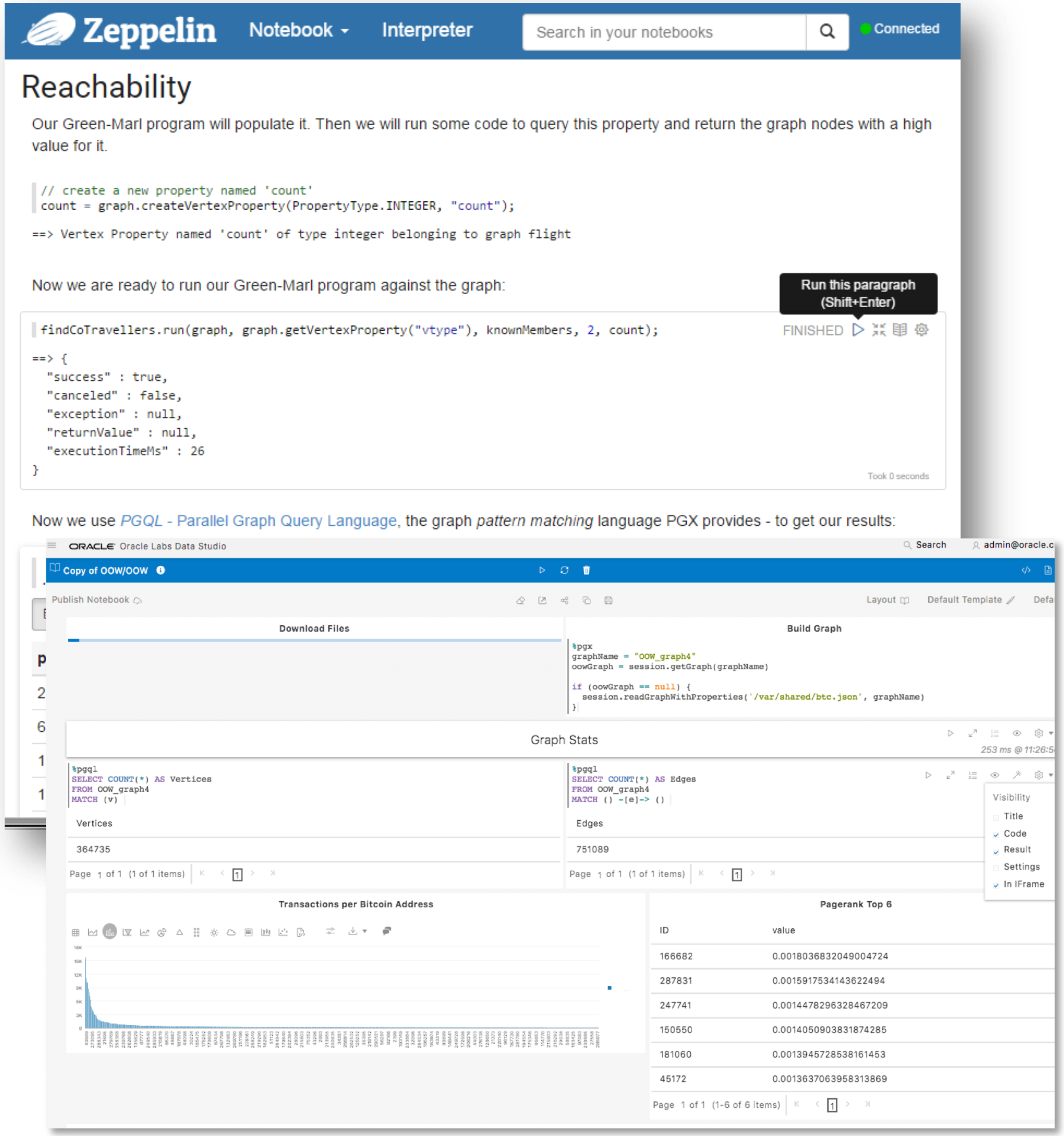
```
pg.queryPgql("""
  SELECT id(v) as id, v.name, v.pagerank
  MATCH (v:person)
  ORDER by v.pagerank DESC
  LIMIT 4
""").print()
```

```
+---------------------------------------------+
| id | v.name         | v.pagerank            |
+---------------------------------------------+
| 1  | Barack Obama   | 0.0608868998919989    |
| 60 | Nicolas Maduro | 0.03445628038301776   |
| 42 | NBC            | 0.027831790283777511  |
| 37 | Amazon         | 0.025848763054771094  |
+---------------------------------------------+
```

```
+---------------------------------------------+
| id | v.name         | v. pagerank           |
+---------------------------------------------+
| 1  | Barack Obama   | 0.0608868998919989    |
| 60 | Nicolas Maduro | 0.03445628038301776   |
| 5  | Pope Francis   | 0.022681215952817014  |
| 3  | Charlie Rose   | 0.021665073518388547  |
+---------------------------------------------+
```

Notebook integration

- Multi-purpose notebook for data analysis and visualization
  - Browser-based script and query execution
- For documentation and interactive analysis
  - Typically used by Data Scientists
- Interpreters for graph analysis and graph pattern matching
  - PGX, PGQL, Markdown
- Graph visualization
- Integrated with Graph Cloud Service



Zeppelin Notebook

Search your Notebooks

anonymous

Game Of Thrones

default

We have loaded the original data the a FINISHED ... Note that the number of edges is doubled due to how PGX internally represents an *undirected* graph.

One of the first measures of **centrality** - which characters are most important - mentioned in the article is *eigenvector centrality*.

Eigenvector centrality is an interesting measure of centrality because *a vertex with few edges can be highly ranked* if other highly ranked vertices are connected through it. So eigenvector centrality finds you characters that are the linchpins to the story even though they may not appear very often.

We will run the built-in algorithm for that, and visualize the results:

```
eigenvector = analyst.eigenvectorCentrality(undirected);
eigenvector.getTopKValues(10);
```

FINISHED

settings

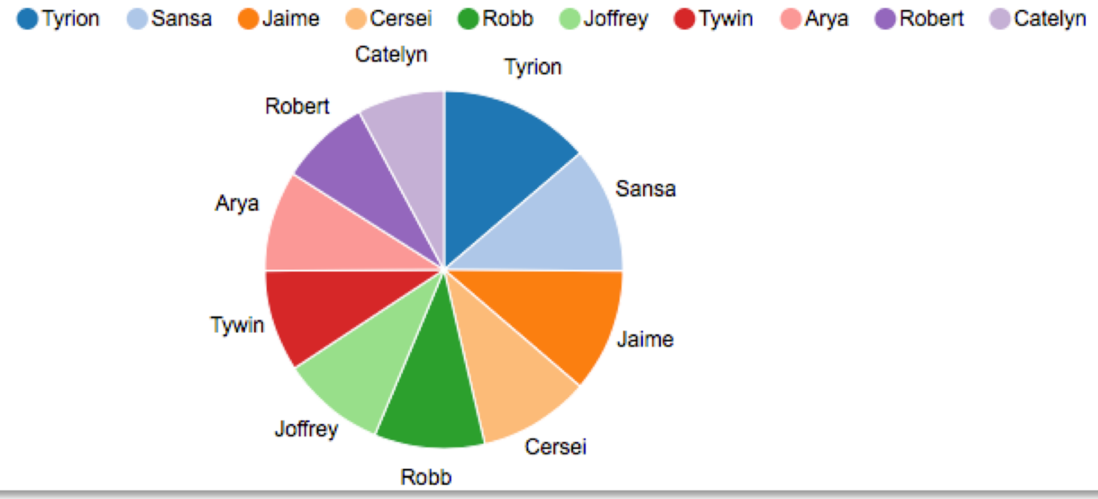
All fields:

ID value

Keys: ID

Groups

Values: value SUM



Zeppelin Notebook

Search your Notebooks

anonymous

Updated: Bitcoin Use Case - DBS

default

```
%pgx
//create the graph
G = session.readGraphWithProperties("/opt/data/bitcoin/small_config_new.json")
```

READY

==> PGX Graph named 'small_bitcoin_edges_new_2' bound to PGX session '4ad3dfc9-d5ed-4f79-b095-a30182ff7672' registered at PGX Server Instance running in embedded mode

Top nodes by degree centrality

READY

```
has been a party to in the transaction network.
centrality = analyst.degreeCentrality(G)
topDegrees = G.queryPgql("SELECT x.name AS NodeID, x.degree AS Top_Degree WHERE
(x) ORDER BY x.degree DESC LIMIT 10"  )
```

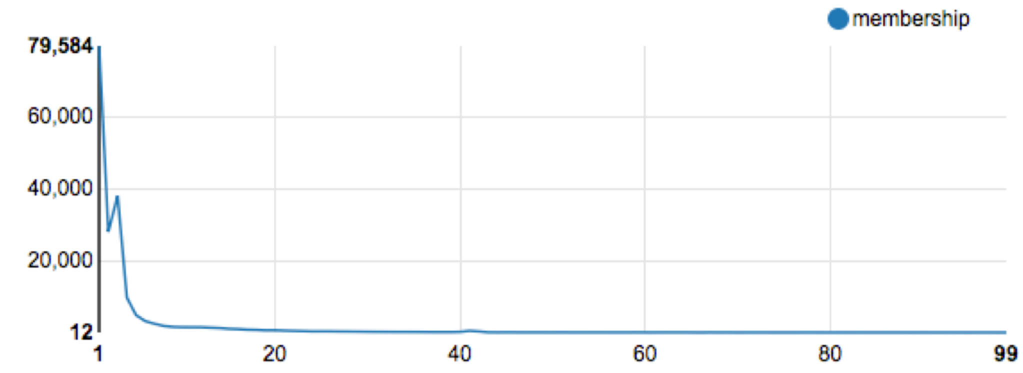
NodeID	Top_Degree
6	149,573
537,797	50,497
701	38,016
14	32,646
1	29,765

```
%pgx
//display the degree distribution as histogram

topDegrees = G.queryPgql("SELECT  x.degree AS degree, COUNT(*) as membership
```

READY

settings



Above, the left panel examines the top nodes by degree, and the right panel plots the distribution. The bitcoin network is a peer-to-peer platform, but a number of brick-and-m READY such as restaurants, apartments, law firms, and popular online services have been increasingly venturing into the space. Many of the highest degree nodes will belong to this category. On the other hand, the majority of nodes have very few connections. These are the individuals making isolated purchases.

ORACLE

Combining **in-memory** analytics engine with **Oracle Database**

- Extremely **fast** graph processing
  – In-memory algorithms
  – Parallel processing
- **Graph-specific** storage model
  – Highly compact
  – Designed for graph traversal
- **Dedicated APIs** and language bindings
  – Blueprints APIs, Java, R, Node.js, ...
- **Synchronization** with data store
- **Many** deployment options
  – Embedded, shared, distributed

- **Scaleable** graph store
  – Supporting extremely large graphs
- High **availability**
  – Using all capabilities of Oracle Database
- **Security**
  – Enterprise-grade access control
- **In-database graph analysis**
  – Using SQL and PGQL
- Combination with **RDF graphs**
  – Integration of semantic technologies
- Integration with enterprise data

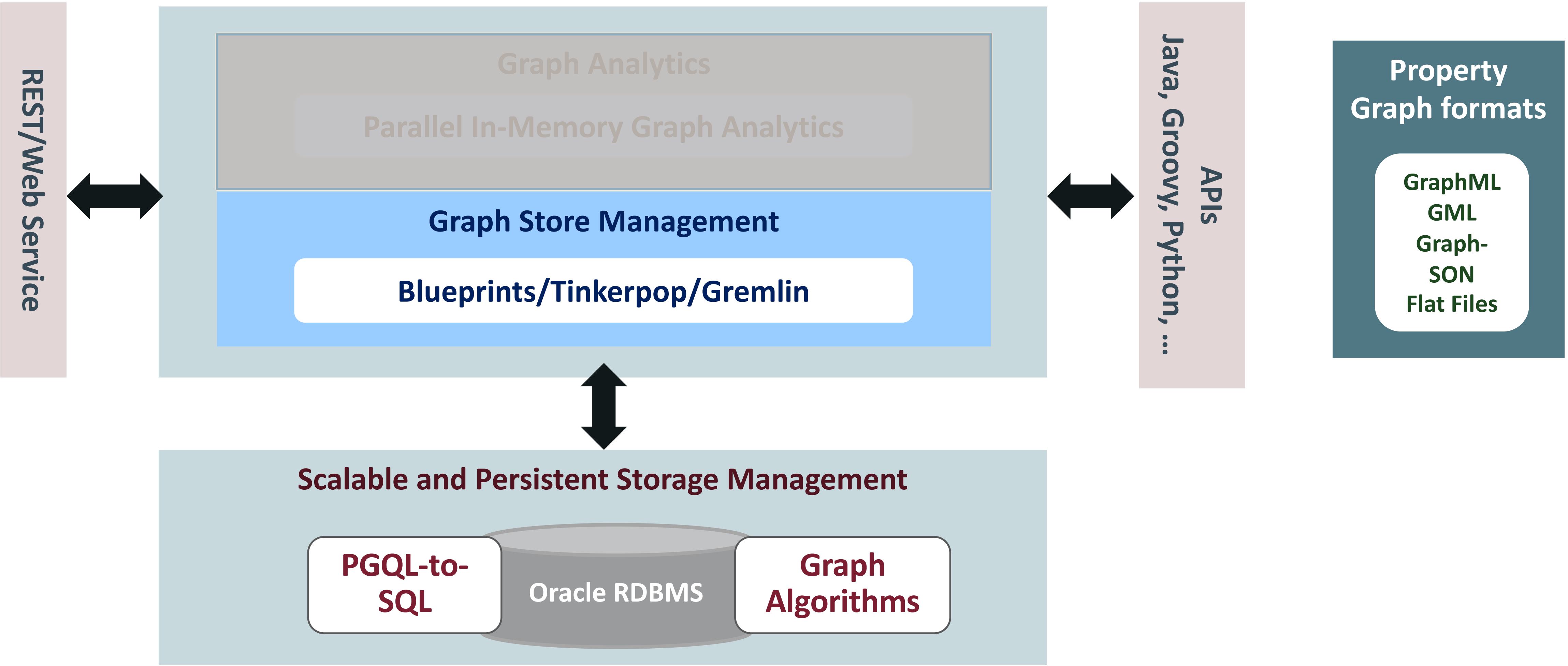
ORACLE

Graph Analytics **in the Database**

- Useful for **combined** graph and relational queries
  - Geospatial analysis, text analysis, ...
- Appropriate if data **changes rapidly** (real-time changes)
  - No need to update snapshots in analytics engine
- When graph is too **large** to fit in memory
- Major graph algorithms available in **PL/SQL**
  - Shortest path, PageRank, Triangle counting, Connected Components, Sparsification, Sub-graph generation
- PGQL translated to SQL
  - Graph analysis using CONNECT BY (NOCYCLE) PRIOR or recursive WITH

ORACLE

Graph Analytics in the Database



PGQL-to-SQL …

- It is much easier and more natural to express graph queries with **PGQL** than with SQL
- SQL translation is performed **automatically** behind the scenes
- Users don’t need to worry about writing complex SQL
- Frequently updated data can be queried without the need to constantly push updates to in-memory PGX snapshots
- We can **leverage the Oracle SQL engine**, which is mature and highly optimized

PGQL to SQL: Filter and Aggregate

PGQL:

```
SELECT count(d) AS cnt
WHERE (n WITH fname='The Academy') -[:admires] -> (d)
```

Find how many people "The Academy" admires

SQL:

```
SELECT 2 AS "cnt$T",
  to_nchar(COUNT(T0.DVID),'TM9','NLS_Numeric_Characters=''.,''') AS "cnt$V",
  COUNT(T0.DVID)        AS "cnt$VN",
  to_timestamp_tz(null) AS "cnt$VT"
FROM "GRAPH1GT$" T0,
     "GRAPH1VT$" T1
WHERE T1.K=n'fname' AND T0.SVID=T1.VID
  AND (T1.T = 1 AND T1.V = n'The Academy')
  AND (T0.EL = n'admires')
```

Value columns:

- **$T - value type**
- **$V - string value**
- **$VN - number value**
- **$VT - timestamp value**

ORACLE

PGQL to SQL: Deep Path Query

PGQL:

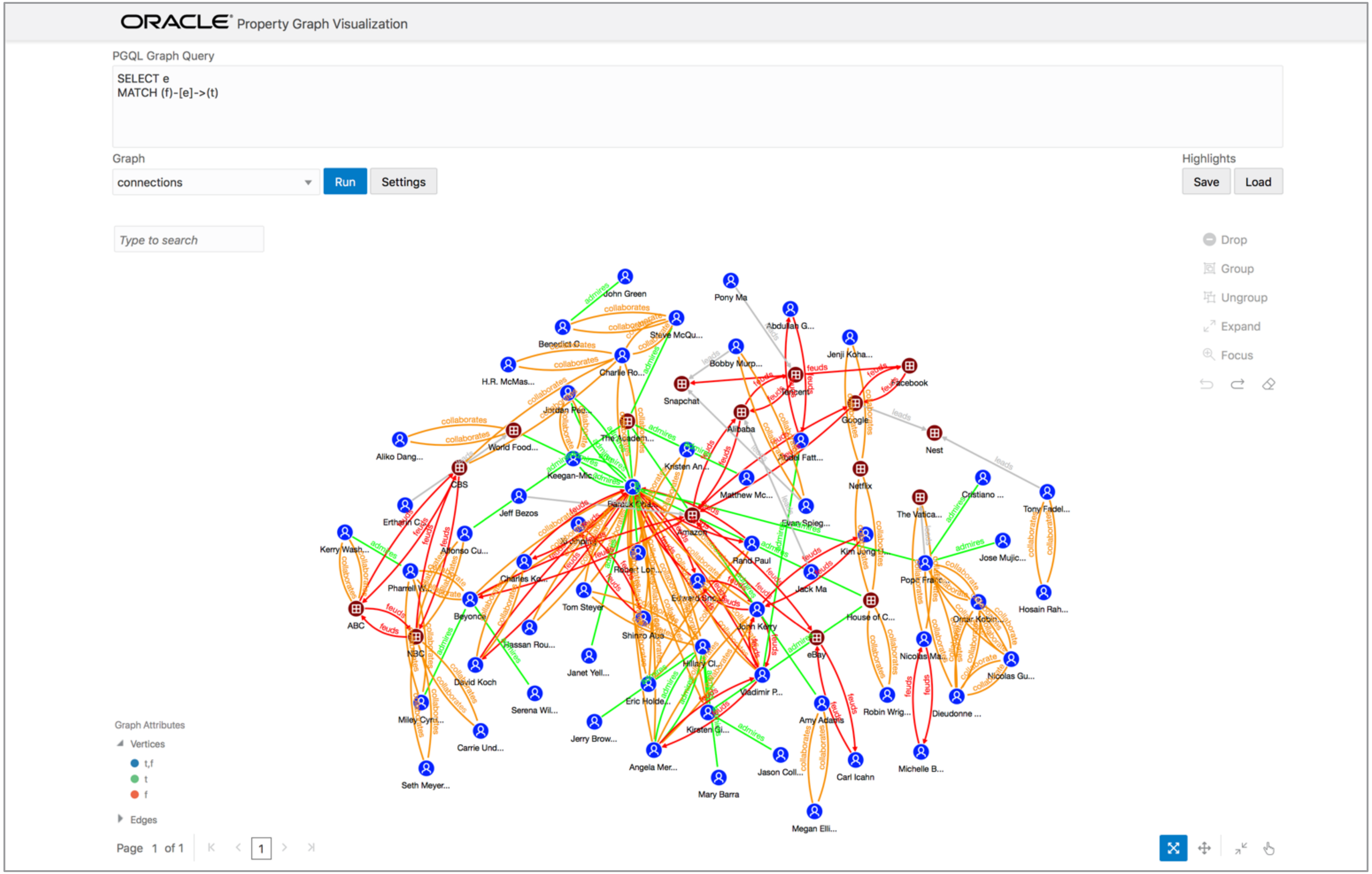
```
PATH knows_path := () -[:knows]-> ()
SELECT s1.fname, s2.fname
WHERE (s1) -/:knows_path*/-> (o) <-/:knows_path*/-(s2)
ORDER BY s1,s2
```

Find the **pairs of people** who are connected to a common person through the "**knows**" relation

SQL:

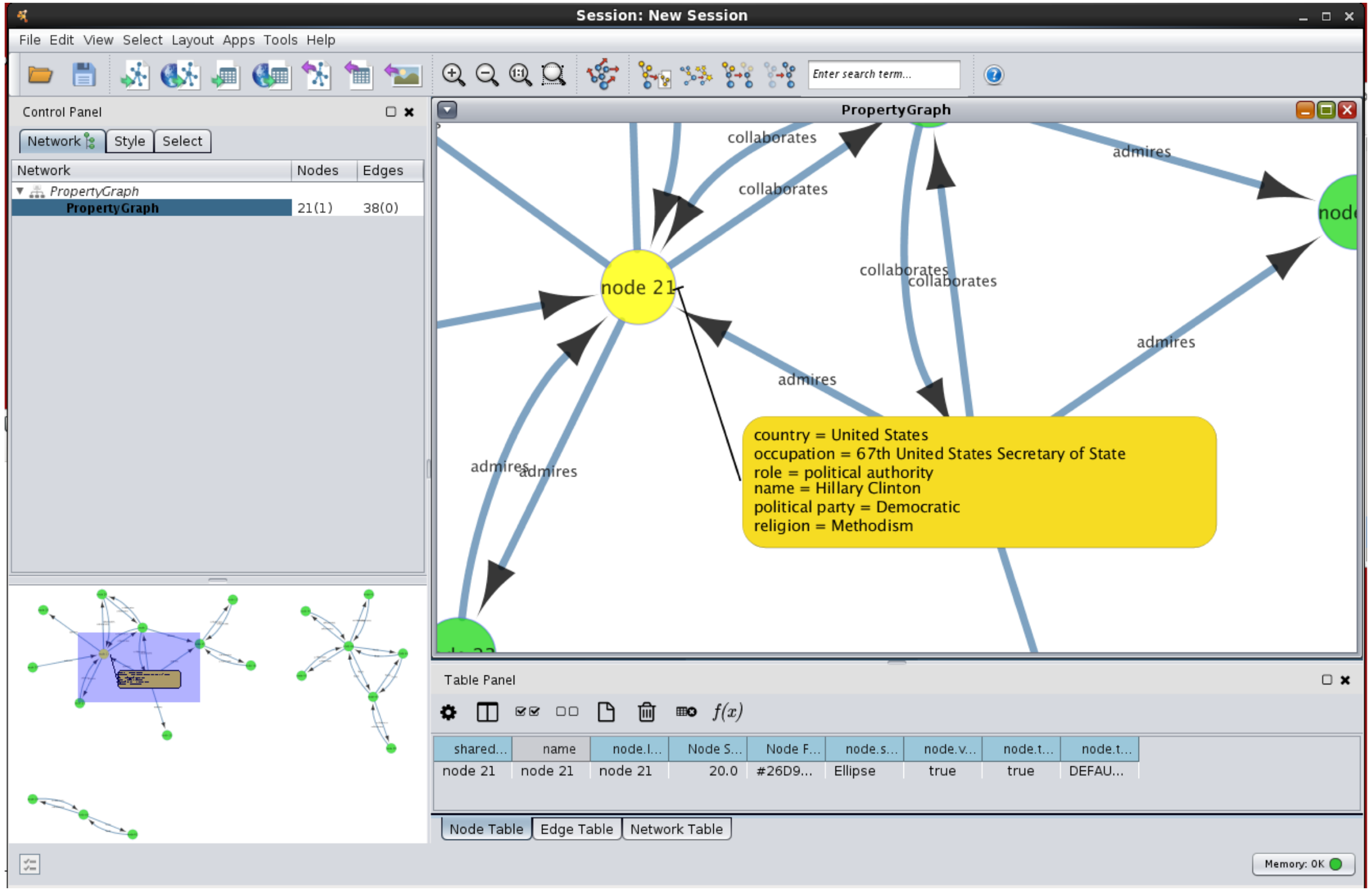
```
SELECT T2.T AS "s1.fname$T",T2.V AS "s1.fname$V",T2.VN AS "s1.fname$VN",T2.VT AS "s1.fname$VT",
       T3.T AS "s2.fname$T",T3.V AS "s2.fname$V",T3.VN AS "s2.fname$VN",T3.VT AS "s2.fname$VT"
FROM (/*Path[*/SELECT DISTINCT SVID, DVID FROM ( SELECT VID AS SVID, VID AS DVID FROM "GRAPH1VT$" UNION ALL SELECT SVID,DVID
      FROM (WITH RW (ROOT, SVID, DVID, LVL) AS ( SELECT ROOT, SVID, DVID,  LVL FROM (SELECT SVID ROOT, SVID, DVID, 1 LVL
      FROM (SELECT T0.SVID AS SVID, T0.DVID AS DVID FROM "GRAPH1GT$" T0 WHERE (T0.EL = n'knows'))
      ) UNION ALL SELECT DISTINCT RW.ROOT, R.SVID, R.DVID, RW.LVL+1 FROM (SELECT T1.SVID AS SVID,
        T1.DVID AS DVID FROM "GRAPH1GT$" T1 WHERE (T1.EL = n'knows')) R, RW WHERE RW.DVID = R.SVID )
      CYCLE SVID SET cycle_col TO 1 DEFAULT 0 SELECT ROOT SVID, DVID FROM RW ))/*]Path*/) T6,
     (/*Path[*/SELECT DISTINCT SVID, DVID FROM ( SELECT VID AS SVID, VID AS DVID FROM "GRAPH1VT$" UNION ALL SELECT SVID,DVID
      FROM (WITH RW (ROOT, SVID, DVID, LVL) AS ( SELECT ROOT, SVID, DVID,  LVL FROM (SELECT SVID ROOT, SVID, DVID, 1 LVL
      FROM (SELECT T4.SVID AS SVID, T4.DVID AS DVID FROM "GRAPH1GT$" T4 WHERE (T4.EL = n'knows'))
      ) UNION ALL SELECT DISTINCT RW.ROOT, R.SVID, R.DVID, RW.LVL+1 FROM (SELECT T5.SVID AS SVID,
        T5.DVID AS DVID FROM "GRAPH1GT$" T5 WHERE (T5.EL = n'knows')) R, RW WHERE RW.DVID = R.SVID )
      CYCLE SVID SET cycle_col TO 1 DEFAULT 0 SELECT ROOT SVID, DVID FROM RW ))/*]Path*/) T7,
"GRAPH1VT$" T2, "GRAPH1VT$" T3
WHERE T2.K=n'fname' AND T3.K=n'fname' AND T6.SVID=T2.VID AND T6.DVID=T7.DVID AND T7.SVID=T3.VID
ORDER BY T6.SVID ASC NULLS LAST, T7.SVID ASC NULLS LAST
```

Viewing Graphs



Cytoscape

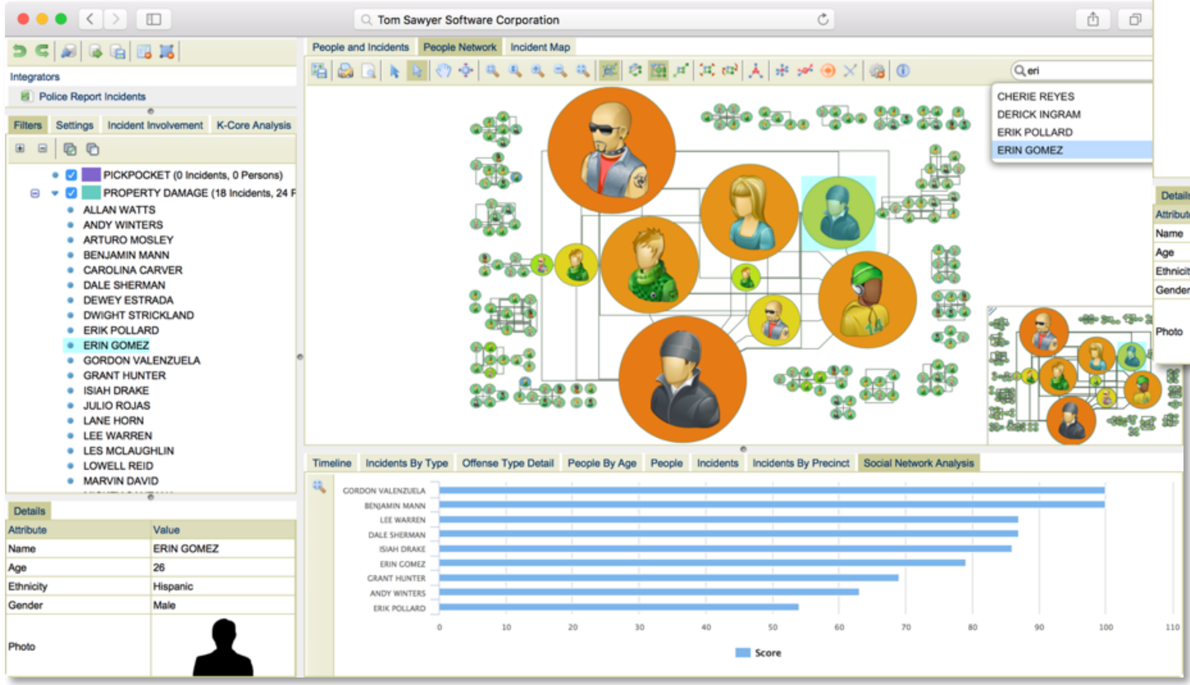
- Open Source
- Originally for biological research
- Now a general platform for complex graph analysis and visualization
- Desktop / Pure Java
- Extensible via Plug-ins
- Also Javascript library (cytoscape.js)

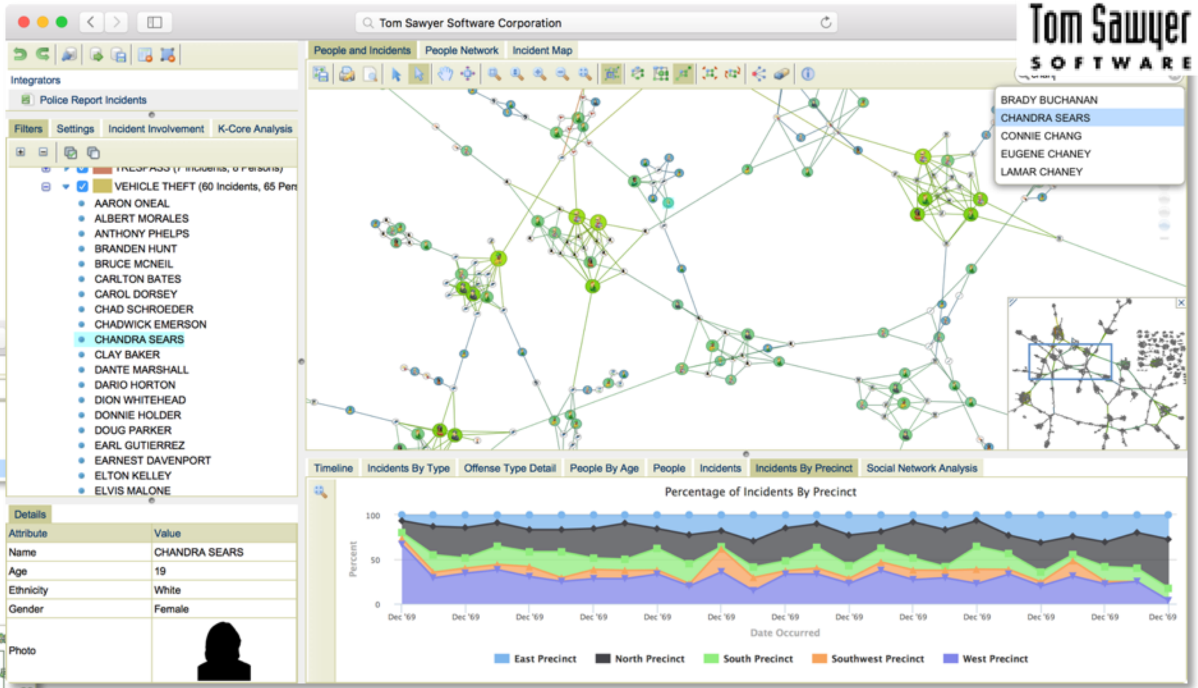


http://www.cytoscape.org

Tom Sawyer Perspectives

- Desktop and Web
- Powerful and flexible
- Full Oracle Integration





Tom Sawyer SOFTWARE

http://www.tomsawyer.com

ORACLE

Platform Sizing

Graph Size	Recommended Physical Memory to be Dedicated	Recommended Number of CPU Processors
10 to 100M edges	Up to 14 GB RAM	2 to 4 processors, and up to 16 processors for more compute-intensive workloads
100M to 1B edges	14 GB to 100 GB RAM	4 to 12 processors, and up to 16 to 32 processors for more compute-intensive workloads
Over 1B edges	Over 100 GB RAM	12 to 32 processors, or more for especially compute-intensive workloads

Can fit a graph with ~23bn edges into one BDA node

Not necessary to load the FULL graph in memory: only load sub-graphs as needed

ORACLE

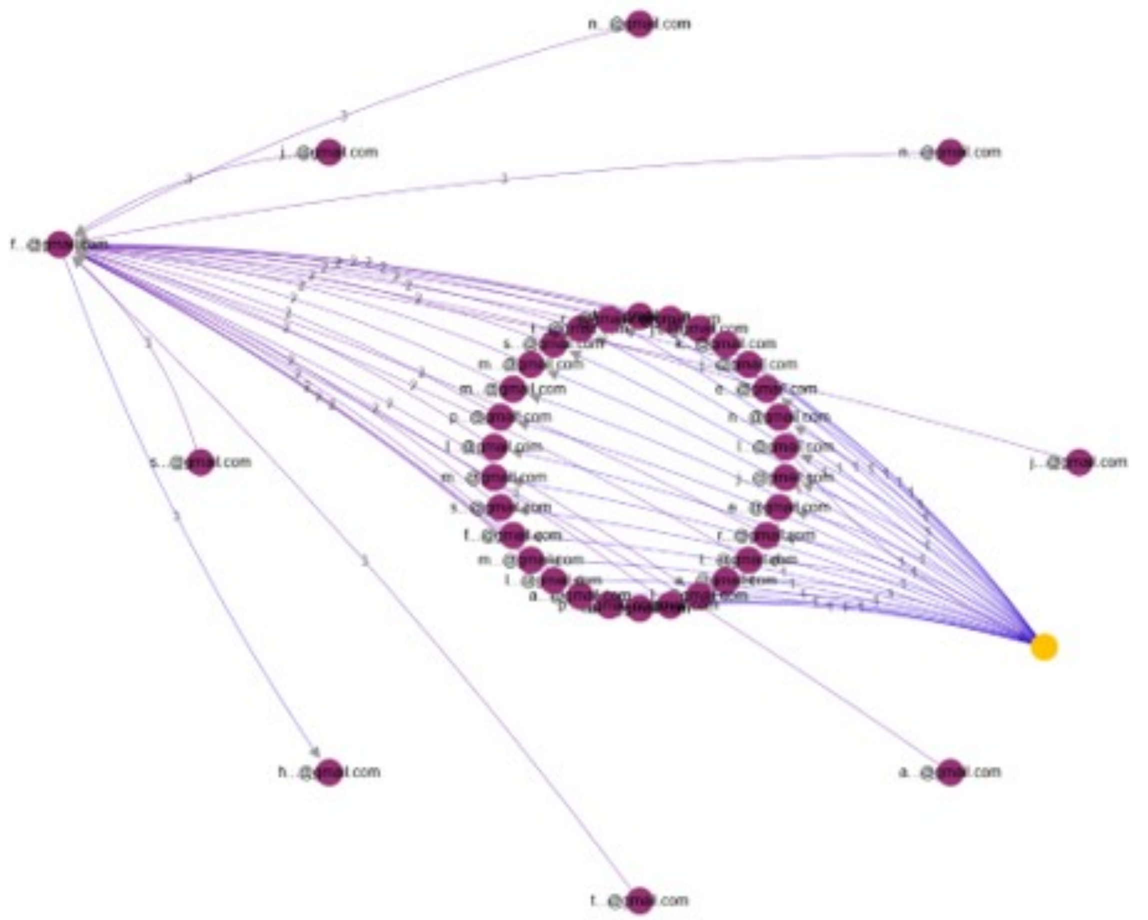
Paysafe

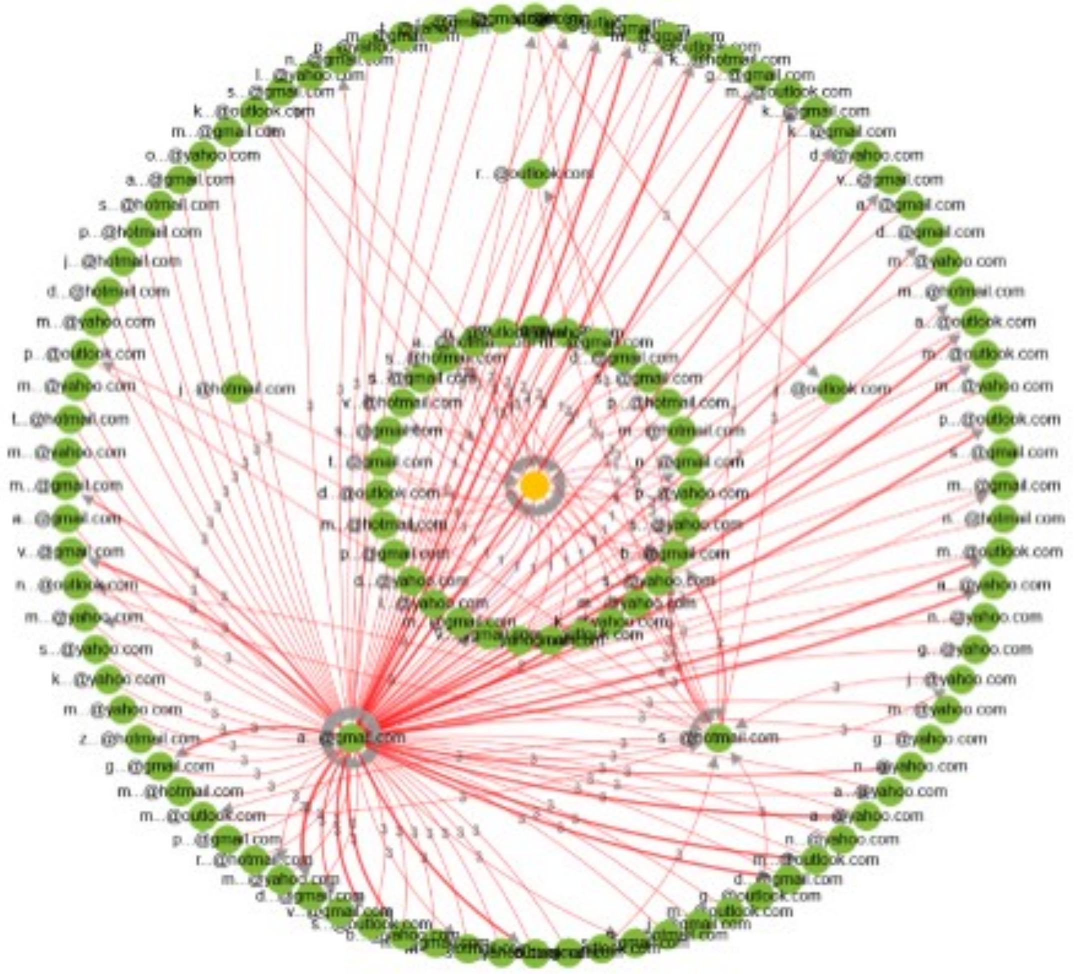
Paysafe:

- Providing online payment solutions
  - Real-time payments, e-Wallets
  - 1bn revenue/yr
  - 500000 payments/day
- Strong demand for fraud detection
  - Only feasible with graph data
  - In real-time, upon money movement
  - During account creation
  - In investigation, visualizing payment flows
- Storing payments in database
  - Refreshing graph using delta update

ORACLE

Suspicious patterns in e-payments





Summary

- Graphs are powerful tools, **complementing** relational databases
  - Especially strong for analysis of graph topology and connectedness
- Graph analytics offer **new insights**
  - Especially relationships, dependencies and behavioural patterns
- Oracle Graph technology offers
  - **Integration** with relational database
  - Scalable parallel **processing**
  - Secure and scalable graph **storage** using Oracle Database
- Available both on-premise and in the Cloud

Graph – an important growth area for data & analytics

Gartner Identifies Top 10 Data and Analytics Technology Trends for 2019

Gartner

Trend No. 5: Graph

Graph analytics is a set of analytic techniques that allows for the exploration of relationships between entities of interest such as organizations, people and transactions.

The application of graph processing and graph DBMSs will grow at 100 percent annually through 2022 to continuously accelerate data preparation and enable more complex and adaptive data science.

Graph data stores can efficiently model, explore and query data with complex interrelationships across data silos, but the need for specialized skills has limited their adoption to date, according to Gartner.

Graph analytics will grow in the next few years due to the need to ask complex questions across complex data, which is not always practical or even possible at scale using SQL queries.

Source: Gartner press release, 18/2/2019, www.gartner.com/en/newsroom/press-releases/2019-02-18-gartner-identifies-top-10-data-and-analytics-technology

ORACLE

Resources

- Oracle Spatial and Graph product page:

  www.oracle.com/database/technologies/spatialandgraph/property-graph-features.html

  – White papers, software downloads, documentation and videos

- Graph Analytics Explained:

  www.oracle.com/technologies/developer-tools/parallel-graph-analytix.html

- Tutorials:

  docs.oracle.com/cd/E56133_01/latest/

@OracleBigData, @agodfrin @SpatialHannes, @JeanIhm

Oracle Spatial and Graph Group

ORACLE

Introduction to Graph analytics

Youtube videos

- What is Oracle Big Data Spatial and Graph?
  https://youtu.be/t9pJJhzZKOE

  How can graph analytics help my business?
  https://youtu.be/0dJNzBi7B-k

  Detecting anomalies with Oracle Big Data Spatial and Graph
  https://youtu.be/nfP6HDOImjY

  Generating recommendations with Oracle Big Data Spatial and Graph
  https://youtu.be/9LRlF3of-Hs

ORACLE

Thank You

Albert Godfrind

Spatial and Graph Solutions Architect
Oracle
October 2019

@agodfrin
albert.godfrind@oracle.com