

You Down With OPG? Leveraging Operational Property Graphs in Oracle 23c



31 May, 2024

Jim Czuprynski

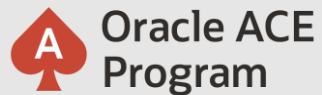
Chief StoryTeller

Zero Defect Computing, Inc.

Who Am I, and What Am I Doing Here?



- E-mail me at jim@jimthewhyguy.com
- Follow me on Mastodon (@JimTheWhyGuy@techfieldday.net)
- Connect with me on LinkedIn (Jim Czuprynski)



500+ technical experts helping peers globally

The **Oracle ACE Program** recognizes and rewards community members for their technical and community contributions to the Oracle community

3 membership tiers



For more details on Oracle ACE Program:
ace.oracle.com



Nominate

yourself or someone you know:

ace.oracle.com/nominate

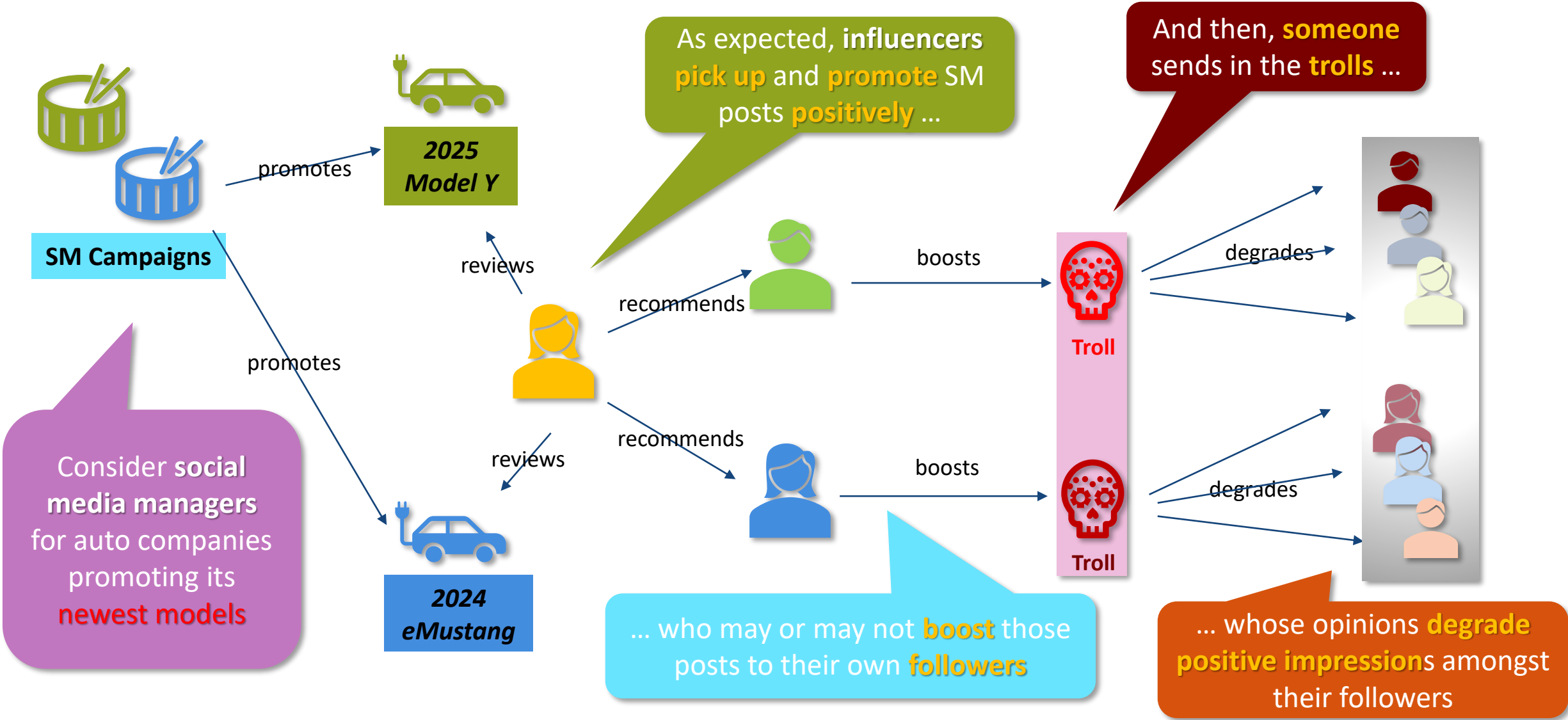
Connect: aceprogram_ww@oracle.com

Facebook.com/OracleACEs

[@oracleace](https://twitter.com/oracleace)



Property Graphs: Not About Data *Itself*, But How They're *Connected*



Business Case: New Product Launch Announcements



Automotive manufacturers have decided to release **new EV models**

Positive support from **key influencers** will be key to public acceptance because they'll highlight most compelling **new features**, thus **differentiating** them from competing models



It's **imperative** to promote releases via **social media** and measure messaging's **effectiveness**

How **positive** are positive reactions?



Are there any **negative** responses?



Are **negative** responses from **true influencers** or just **trolls**?

A Social Media Network: Nodes and Edges

```
CREATE TABLE IF NOT EXISTS entities (  
  e_id          NUMBER(09)      NOT NULL  
  ,e_gender     CHAR(01)        NOT NULL  
  ,e_name       VARCHAR2(40)    NOT NULL  
  ,e_country    CHAR(03)        NOT NULL  
  ,e_tz         CHAR(06)        NOT NULL  
  ,e_handle     CHAR(40)        NOT NULL  
  ,e_comment    VARCHAR2(512));
```

These two tables construct the **nodes**
of a simple **social media network** ...

```
CREATE TABLE IF NOT EXISTS messaging (  
  msg_id          NUMBER(13)      GENERATED ALWAYS  
                                     AS IDENTITY  
                                     (START WITH 1001)  
  ,msg_dtm        DATE            NOT NULL  
  ,msg_polarity   NUMBER(8,5)     DEFAULT 0  
  ,msg_subjectivity NUMBER(8,5)  DEFAULT 0  
  ,msg_text       VARCHAR2(4000)  NOT NULL);
```

A Social Media Network: Nodes and Edges

```
CREATE TABLE IF NOT EXISTS entities (  
  e_id          NUMBER(09)      NOT NULL  
  ,e_gender     CHAR(01)        NOT NULL  
  ,e_name       VARCHAR2(40)    NOT NULL  
  ,e_age        NUMBER(03)      NOT NULL  
  ,e_gender     CHAR(06)        NOT NULL  
  ,e_name       VARCHAR2(40)    NOT NULL  
  ,e_age        NUMBER(03)      NOT NULL  
  ,e_gender     CHAR(06)        NOT NULL  
  ,e_name       VARCHAR2(40)    NOT NULL  
  ,e_age        NUMBER(03)      NOT NULL
```

These two tables construct the **nodes** of a simple **social media network** ...

... while these two tables function as the **edges** connecting nodes within that **social media network**

```
CREATE TABLE IF NOT EXISTS relationships (  
  r_src         NUMBER(09)      NOT NULL  
  ,r_tgt        NUMBER(09)      NOT NULL  
  ,r_type       CHAR(20)        NOT NULL  
  ,r_comment    VARCHAR2(512));
```

```
CREATE TABLE IF NOT EXISTS messages (  
  msg_id        NUMBER(13)      NOT NULL  
  ,msg_dtm      DATE            NOT NULL  
  ,msg_polarity NUMBER          NOT NULL  
  ,msg_subjectivity NUMBER      NOT NULL  
  ,msg_text     VARCHAR2(4096)  NOT NULL
```

```
CREATE TABLE IF NOT EXISTS interactions (  
  smi_id        NUMBER(13)      GENERATED ALWAYS  
  AS IDENTITY  
  (START WITH 1001)  
  ,smi_e_id     NUMBER(09)      NOT NULL  
  ,smi_type     CHAR(12)        NOT NULL  
  ,smi_msg_id   NUMBER(13)      NOT NULL);
```

Photo by Antenna
on Unsplash

Demonstration: Exploring a Social Media Network



Creating an OPG Property Graph

```
CREATE OR REPLACE PROPERTY GRAPH smi_network
```

```
VERTEX TABLES (
```

```
  entities AS MESSENGERS
```

```
    KEY ( e_id )
```

```
    PROPERTIES ( e_id, e_gender, e_name, e_country, e_tz, e_handle )
```

```
, messaging AS MESSAGES
```

```
  KEY ( msg_id )
```

```
  PROPERTIES ( msg_id, msg_dtm, msg_polarity, msg_subjectivity, msg_text )
```

```
)
```

```
EDGE TABLES (
```

```
  relationships AS CONNECTIONS
```

```
    SOURCE KEY ( r_src ) REFERENCES messengers( e_id )
```

```
    DESTINATION KEY ( r_tgt ) REFERENCES messengers( e_id )
```

```
    PROPERTIES ( r_src, r_tgt, r_type)
```

```
, interactions AS INTERACTIONS
```

```
  SOURCE KEY ( smi_e_id ) REFERENCES messengers( e_id )
```

```
  DESTINATION KEY ( smi_msg_id ) REFERENCES messages( msg_id )
```

```
  PROPERTIES ( smi_id, smi_e_id, smi_msg_id, smi_type )
```

```
);
```

This describes the vertices (aka **nodes**) and their particular **attributes** ...

... while this describes the **edges** that connect the nodes and some of their pertinent **attributes**

Viewing OPG Attributes

```
SELECT * FROM user_pg_elements;
SELECT * FROM user_pg_edge_relationships;
SELECT * FROM user_pg_label_properties;
```

GRAPH_NAME	ELEMENT_NAME	ELEMENT_KIND	OBJECT_OWNER	OBJECT_NAME
SMI_NETWORK	MESSENGERS	VERTEX	HOL23C	ENTITIES
SMI_NETWORK	MESSAGES	VERTEX	HOL23C	MESSAGING
SMI_NETWORK	CONNECTIONS	EDGE	HOL23C	RELATIONSHIPS
SMI_NETWORK	INTERACTIONS	EDGE	HOL23C	INTERACTIONS

GRAPH_NAME	EDGE_TAB_NAME	VERTEX_TAB_NAME	EDGE_END	EDGE_COL_NAME	VERTEX_COL_NAME
SMI_NETWORK	INTERACTIONS	MESSENGERS	SOURCE	SMI_E_ID	E_ID
SMI_NETWORK	CONNECTIONS	MESSENGERS	SOURCE	R_SRC	E_ID
SMI_NETWORK	CONNECTIONS	MESSENGERS	DESTINATION	R_TGT	E_ID
SMI_NETWORK	INTERACTIONS	MESSAGES	DESTINATION	SMI_MSG_ID	MSG_ID

GRAPH_NAME	LABEL_NAME	PROPERTY_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	DATA_CHAR_LENGTH
SMI_NETWORK	MESSENGERS	E_ID	NUMBER	22	9	0	0
SMI_NETWORK	MESSENGERS	E_GENDER	CHAR	1	(null)	(null)	1
SMI_NETWORK	MESSENGERS	E_NAME	VARCHAR2	40	(null)	(null)	40
SMI_NETWORK	MESSENGERS	E_COUNTRY	CHAR	3	(null)	(null)	3
SMI_NETWORK	MESSENGERS	E_TZ	CHAR	6	(null)	(null)	6
SMI_NETWORK	MESSENGERS	E_HANDLE	VARCHAR2	40	(null)	(null)	40
SMI_NETWORK	MESSAGES	MSG_ID	NUMBER	22	13	0	0
SMI_NETWORK	MESSAGES	MSG_DTM	DATE	7	(null)	(null)	0
SMI_NETWORK	MESSAGES	MSG_POLARITY	NUMBER	22	8	5	0
SMI_NETWORK	MESSAGES	MSG_SUBJECTIVITY	NUMBER	22	8	5	0
SMI_NETWORK	MESSAGES	MSG_TEXT	VARCHAR2	4000	(null)	(null)	4000

Sifting Chaff from Grain: MATCH Clause Syntax (1)

Pattern	Describes	Example	Meaning
()	Nodes (aka vertices)	(n1)	Node labelled n1
[]	Edges	[e]	Edge labelled e
-[]->	Directional to	(n1) - [e] -> (n2)	Node n1 connected by edge e to node n2 in the direction of n2
<[]-	Directional from	(n1) <- [e] - (n2)	Node n1 connected by edge e to node n2 in the direction of n1
-[]-	Omni-directional	(n1) - [e] - (n2)	Node n1 connected by edge e to node n2 in either direction

OPGs, GRAPH_TABLE, and MATCH

```
SELECT *
FROM GRAPH_TABLE(
  smi_network
  MATCH (m1) - [p IS CONNECTIONS] -> (m2)
  COLUMNS (
    p.r_src AS source
  , m1.e_handle AS src_handle
  , m1.e_country AS src_from
  , m1.e_tz AS src_tz
  , P.r_tgt AS target
  , m2.e_handle AS dst_handle
  , m2.e_country AS dst_from
  , m2.e_tz AS dst_tz
  );
```

The new **GRAPH_TABLE** operator lets me look for relationships within an OPG

This MATCH statement tells OPG to connect **members (nodes)** to their **payload activity (edges)**

Attributes of each **node** as well as their connecting **edges** are **prefixed** to **differentiate** them from each other

OPGs, GRAPH_TABLE, and MATCH

```
SELECT *
FROM GRAPH_TABLE(
  smi_network
  MATCH (m1) - [p IS CONNECTIONS] -> (m2)
);
```

The new **GRAPH_TABLE** operator lets me look for relationships within an OPG

COL	SOURCE	SRC_HANDLE	SRC_FROM	SRC_TZ	TARGET	DST_HANDLE	DST_FROM	DST_TZ
p	1001	EatTheRich14	RUS	UTC+3	1002	ProudSCBoy	ROM	UTC+2
,	m	1002	ProudSCBoy	ROM	UTC+2	1003	TSLAHater21	RUS
,	m	1003	TSLAHater21	RUS	UTC+3	1004	BiteMeElon12	BOL
,	m	1004	BiteMeElon12	BOL	UTC-5	1001	EatTheRich14	RUS
,	P	2001	AntiVaxx42	DRK	UTC+11	2002	LogBurner18	CHI
,	m	2002	LogBurner18	CHI	UTC+9	2003	SorryHenryFord29	CHI
,	m	2003	SorryHenryFord29	CHI	UTC+8	2004	2A3Percent	PER
,	m	2004	2A3Percent	PER	UTC-6	2001	AntiVaxx42	DRK
,	m	101	2bhntn	GBR	UTC+0	2	FordMediaMgr	USA
,	m	102	akko	USA	UTC-5	149	getreadytopaymore	IRE
,	m	102	akko	USA	UTC-5	155	hotdogwater	USA
,	m	102	akko	USA	UTC-5	157	jack_horner	USA
,	m	102	akko	USA	UTC-5	226	texanbychoice2015	GBR
,	m	103	alberightback080	USA	UTC-3	133	cpt-will	USA
,	m	103	alberightback080	USA	UTC-3	192	OrangeRadio	USA
,	m	104	America#1	USA	UTC-5	233	trivium525	USA
,	m	105	another_nc_guy156	USA	UTC-7	205	rdsouza	USA
,	m	106	AodhOrdnigh	IRE	UTC+0	135	DamnItJimImJustACountryDoctor!	USA

Traversing Network “Hops”: MATCH Clause Syntax (2)

Pattern	Describes	Example	Meaning
$-\dots-\>^*$	Directional with zero or more hops	$(n1) - [e] -\>^* (n2)$	Node n1 connected by edge e to node n2 in n2 's direction with any number of hops in between
$-\dots-\>^+$	Directional with one or more hops	$(n1) - [e] -\>^+ (n2)$	Node n1 connected by edge e to node n2 in n2 's direction with one or more hops in between
$-\dots-\>\{n-m\}$	Directional with exactly n to m hops	$(n1) - [e] -\> \{2-4\} (n2)$	Node n1 connected by edge e to node n2 in n2 's direction with exactly two to four hops in between

Who Are the Most Interactive Members?

```
SELECT src_id, src_handle, COUNT(1) AS multihop_chains
FROM GRAPH_TABLE(
  smi_network
  MATCH (m1 IS MESSENGERS)-[p IS CONNECTIONS]->{1,3}(m2 IS MESSENGERS)
  COLUMNS (
    m1.e_id AS src_id, m1.e_handle AS src_handle,
    m2.e_id AS tgt_id, m2.e_handle AS tgt_handle
  )
)
GROUP BY src_id, src_handle
ORDER BY multihop_chains DESC
FETCH FIRST 15 ROWS ONLY;
```

... in this case, between **one (1)** and **three (3)** hops

This **MATCH clause** filters the graph for any social media member who has communicated to any other social media member through multiple hops ...

Who Are the Most Interactive Members?

```

SELECT src_id, src_handle, COUNT(1) AS multihop_chains
FROM GRAPH_TABLE(
  smi_network
  MATCH (m1 IS MESSENGERS)-[p IS ...]
  COLUMNS (
    m1.e_id AS src_id, m1.e_handle AS src_handle,
    m2.e_id AS tgt_id, m2.e_handle AS tgt_handle
  )
)
GROUP BY src_id, src_handle
ORDER BY multihop_chains DESC
FETCH FIRST 15 ROWS ONLY;
    
```

SRC_ID	SRC_HANDLE	MULTIHOP_CHAINS
175	mcriderjm	139
241	virtuafighter	66
246	wng239	60
179	mnmoon	55
133	cpt-will	52
182	msattler066	52
148	gagaga	50
231	tjmaxx948	48
121	caspiansails	44
218	smarterthandems	44
202	purplereign615	44
199	Plato57	44
153	HighDensityEVParkingSafe	44
135	DamnItJimImJustACountryDoctor!	44
244	whitey	42

These are the members who've been most active in the social media network whose activities have reached **beyond one to three other** members

... the media indicated media hops ...

Finding Potential Trolls

```
SELECT
  followers.src_id, followers.src_handle
, followed.followed_by, followers.following
FROM
  (SELECT src_id, src_handle, COUNT(src_id) AS following
   FROM GRAPH_TABLE(
     smi_network
     MATCH (m1 IS MESSENGERS)-[p IS CONNECTIONS]->(m2 IS MESSENGERS)
     COLUMNS (m1.e_id AS src_id, m1.e_handle as src_handle)
   ) GROUP BY src_id, src_handle ) followers
, (SELECT tgt_id, COUNT(tgt_id) AS followed_by
   FROM GRAPH_TABLE(
     smi_network
     MATCH (m1 IS MESSENGERS)-[p IS CONNECTIONS]->(m2 IS MESSENGERS)
     COLUMNS (m2.e_id AS tgt_id)
   ) GROUP BY tgt_id) followed
WHERE followers.src_id = followed.tgt_id
  AND followed_by >= 5
  AND following <= 5
ORDER BY followed_by DESC;
```

The **results** returned here ...

... combine the results of who is being **followed** by multiple other members ...

... with those members who **follow hardly anyone else**

Finding Potential Trolls

```
SELECT
  followers.src_id, followers.src_handle
, followed.followed_by, followers.following
FROM
  (SELECT src_id, src_handle, COUNT(src_id) AS following
    FROM GRAPH_TABLE(
      smi_network
      MATCH (m1 IS MESSENGERS)-[p IS CONNECTED]->(m2 IS MESSENGERS)
      COLUMNS (m1.e_id AS src_id, m1.e_handle AS src_handle, m2.e_id AS tgt_id)
    ) GROUP BY src_id, src_handle ) followers
, (SELECT tgt_id, COUNT(tgt_id) AS followed_by
    FROM GRAPH_TABLE(
      smi_network
      MATCH (m1 IS MESSENGERS)-[p IS CONNECTED]->(m2 IS MESSENGERS)
      COLUMNS (m2.e_id AS tgt_id)
    ) GROUP BY tgt_id) followed
WHERE src_id = 1
AND followed_by > 5
ORDER BY following
```

The **results** returned here ...

SRC_ID	SRC_HANDLE	FOLLOWED_BY	FOLLOWING
1	TeslaMediaMgr	18	1
2001	AntiVaxx42	11	1
2002	LogBurner18	10	1
2004	2A3Percent	10	1
2003	SorryHenryFord29	10	1
2	FordMediaMgr	10	1
196	pat11602	8	2
1001	EatTheRich14	7	1
1002	ProudSCBoy	6	1
1004	BiteMeElon12	6	1
1003	TSLAHater21	6	1
148	gagaga	5	4

While some of these members are **known**, others are **followed by** several members ... but are **following hardly any others!**



Demonstration: Visualizing Key Influencers

Populating a GVT Region: Option 1

The screenshot shows the Oracle APEX configuration page for a region titled 'Messaging'. The region type is 'Graph Visualization (Pr)'. The 'Source' section is highlighted with a red box and contains the following configuration:

- Location: Local Database
- Type: Property Graph
- Graph Owner: Parsing Schema
- Graph Name: SMI_NETWORK
- Match Clause: `(m1 IS MESSENGERS)-[p IS INTERACTIONS]->(m2 IS MESSAGES)`
- Columns Clause: `m1.e_handle, p.smi_type, to_char(p.smi_e_id) AS source, TO_CHAR(m2.msg_dtm, 'YYYY-MM-DD HH24:MI:SS') AS sent_on, TO_CHAR(m2.msg_polarity) AS polarity, p.smi_type AS msg_type`
- Where Clause: `p.smi_type = 'Post'`

... choosing the Graph Name, columns, and even selection criteria ...

You can build a GVT region by supplying key information ...

Populating a GVT Region: Option 1

The screenshot shows the configuration interface for a GVT region named 'Social Media Messaging'. The 'Source' section is highlighted with a red box and contains the following configuration:

- Location: Local Database
- Type: Property Graph
- Graph Owner: Parsing Schema
- Graph Name: SMI_NETWORK
- Match Clause: `(m1 IS MESSENGERS)-[p IS INTERACTIONS]->(m2 IS MESSAGES)`
- Columns Clause: `m1.e_handle, p.smi_type, to_char(p.smi_e_id) AS source, TO_CHAR(m2.msg_dtm, 'YYYY-MM-DD HH24:MI:SS') AS sent_on, TO_CHAR(m2.msg_polarity) AS polarity, p.smi_type AS msg_type`
- Where Clause: `p.smi_type = 'Post'`

The 'Sub Regions' section is also highlighted with a red box and contains the following configuration:

- Sub Region: Messaging
- Columns: SOURCE, E_HANDLE, SMI_TYPE, SENT_ON, POLARITY, MSG_TYPE

Two green callout boxes provide additional context:

- Top callout: "... choosing the Graph Name, columns, and even selection criteria ..."
- Bottom callout: "... which automatically builds the list of columns to return to the GVT tool"

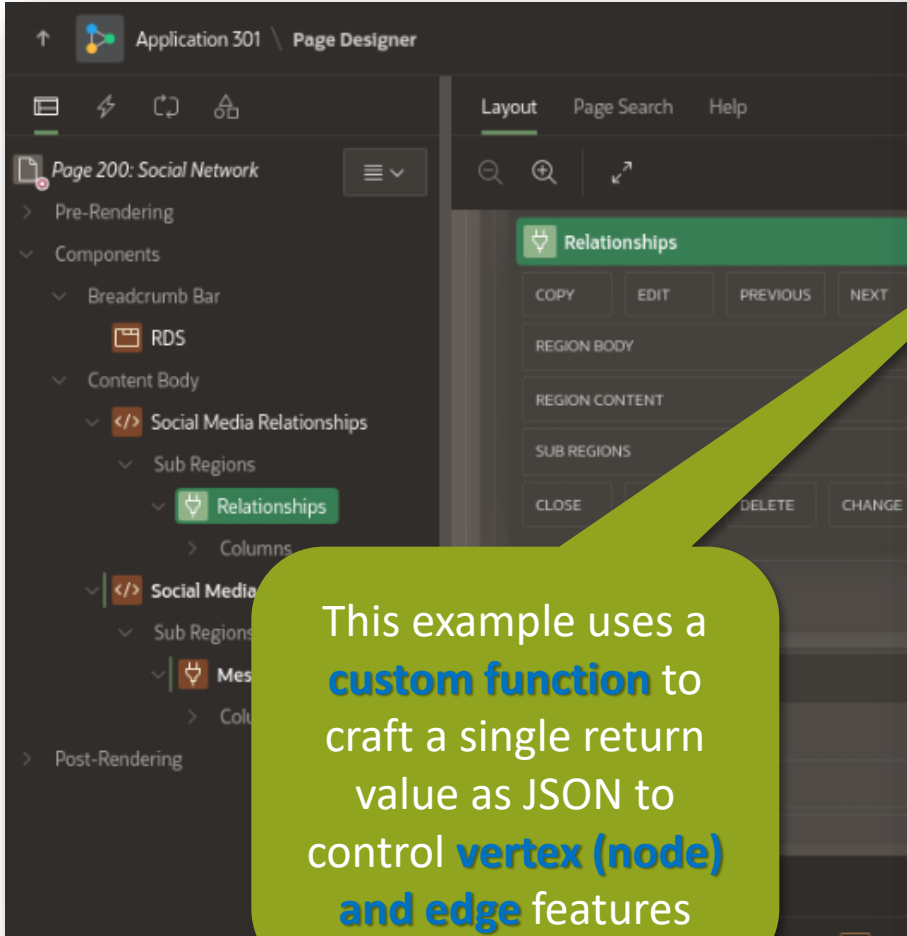
Populating a GVT Region: Option 2

The screenshot shows the Oracle Page Designer interface for 'Application 301' in 'Page Designer' mode. The main workspace displays a 'Relationships' region configuration. The 'Source' section is highlighted with a red box and contains the following SQL query:

```
SELECT v1, e, v2
FROM GRAPH_TABLE(
sm1_network
MATCH (m1 IS MESSENGERS)-[p IS
CONNECTIONS]->(m2 IS MESSENGERS)
COLUMNS (
p.r_src AS src_id
, p.r_tgt AS dst_id
, vertex_id(m1) AS v1
, edge_id(p) as e
, vertex_id(m2) as v2)
```

For finer control, you can issue a SQL statement that describes the specific nodes and edges you want to display

Populating a GVT Region: Option 2

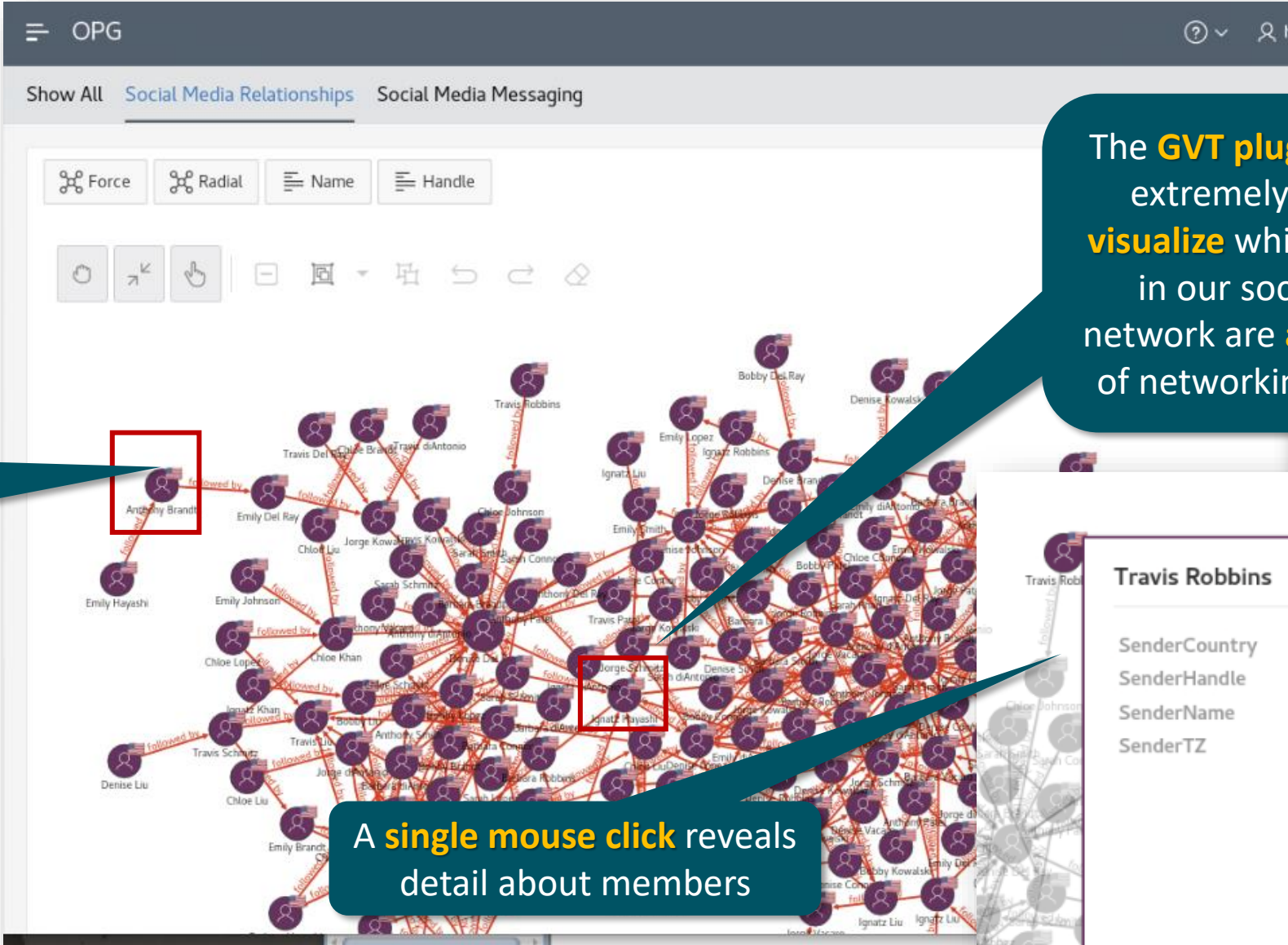


The screenshot shows the Page Designer interface for 'Application 301'. The left sidebar contains a tree view with components like 'Page 200: Social Network', 'Pre-Rendering', 'Components', 'Breadcrumb Bar', 'RDS', 'Content Body', 'Social Media Relationships', 'Sub Regions', and 'Relationships'. The 'Relationships' panel is active, showing options like COPY, EDIT, PREVIOUS, NEXT, REGION BODY, REGION CONTENT, SUB REGIONS, and CLOSE, DELETE, CHANGE.

This example uses a **custom function** to craft a single return value as JSON to control **vertex (node) and edge** features

```
Code Editor - SQL Query
1 SELECT FNC_SMI_SQLGRAPH_JSON(
2 'SELECT v1, e, v2
3 FROM GRAPH_TABLE(
4 smi_network
5 MATCH (m1 IS MESSENGERS)-[p IS INTERACTIONS]->(m2 IS MESSAGES)
6 COLUMNS (
7 p.smi_e_id AS source
8 , p.smi_msg_id AS target
9 , m1.e_handle AS SentBy
10 , TO_CHAR(m2.msg_dtm, 'YYYY-MM-DD HH24:MI:SS') AS SentAsOf
11 , m2.msg_polarity AS Polarity
12 , p.smi_type AS MsgType
13 , SUBSTR(m2.msg_text, 1, 40) || '...' AS MsgAbvText
14 ,vertex_id(m1) AS v1
15 ,edge_id(p) as e
16 ,vertex id(m2) as v2)
17 )'
18 ) AS result_column FROM DUAL;
19
```

Probing Social Media Networks Graphically (1)

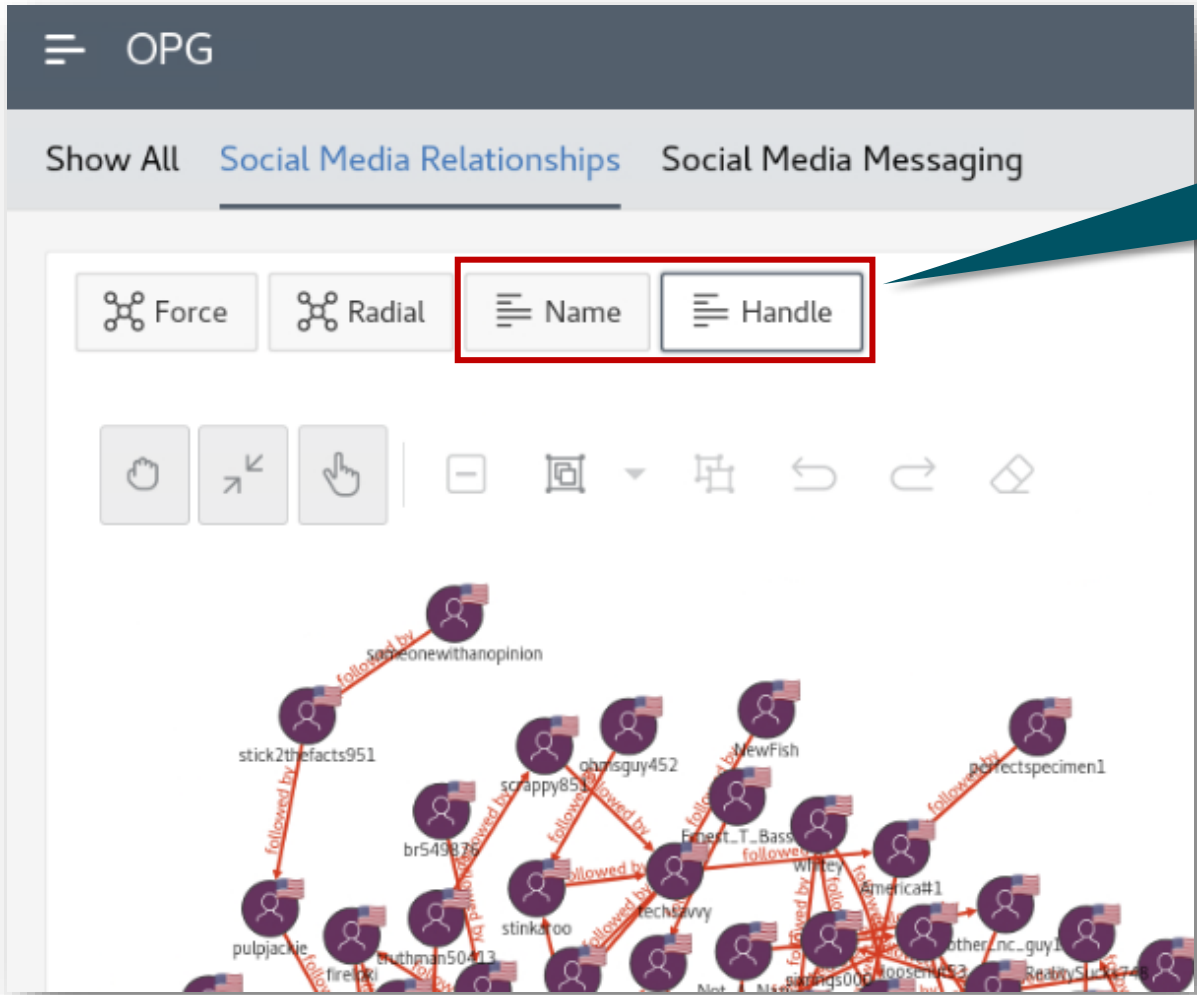


... as well as members who **aren't**

The **GVT plug-in** makes it extremely simple to **visualize** which members in our social media network are **at the center** of networking activity ...

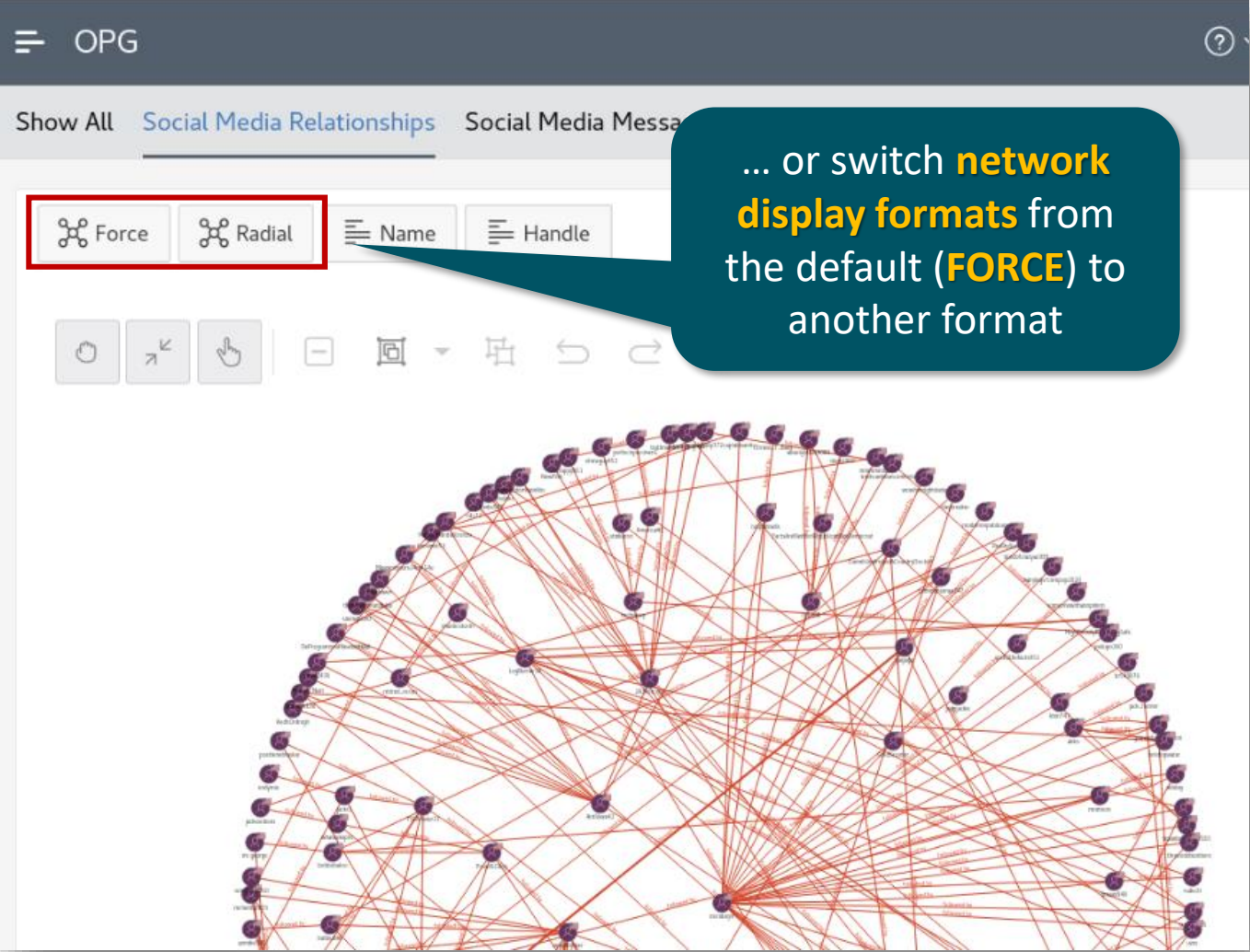
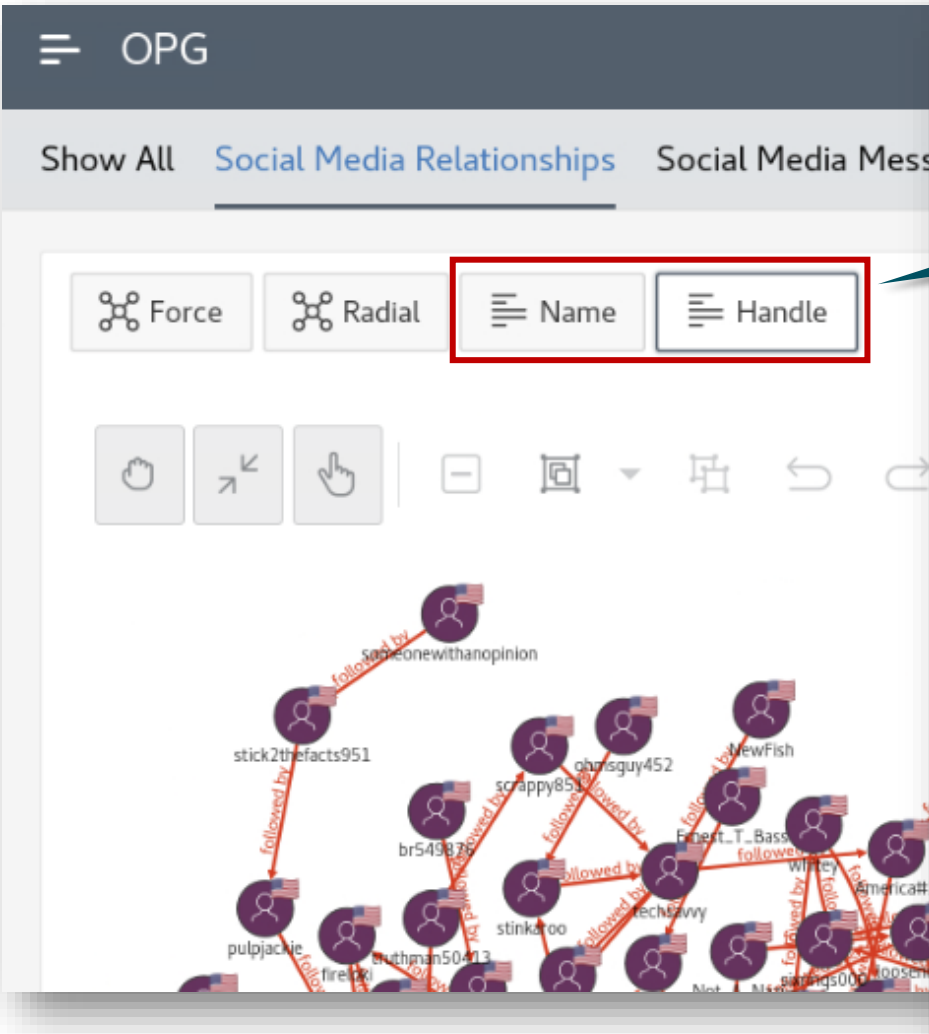
A **single mouse click** reveals detail about members

Probing Social Media Networks Graphically (2)



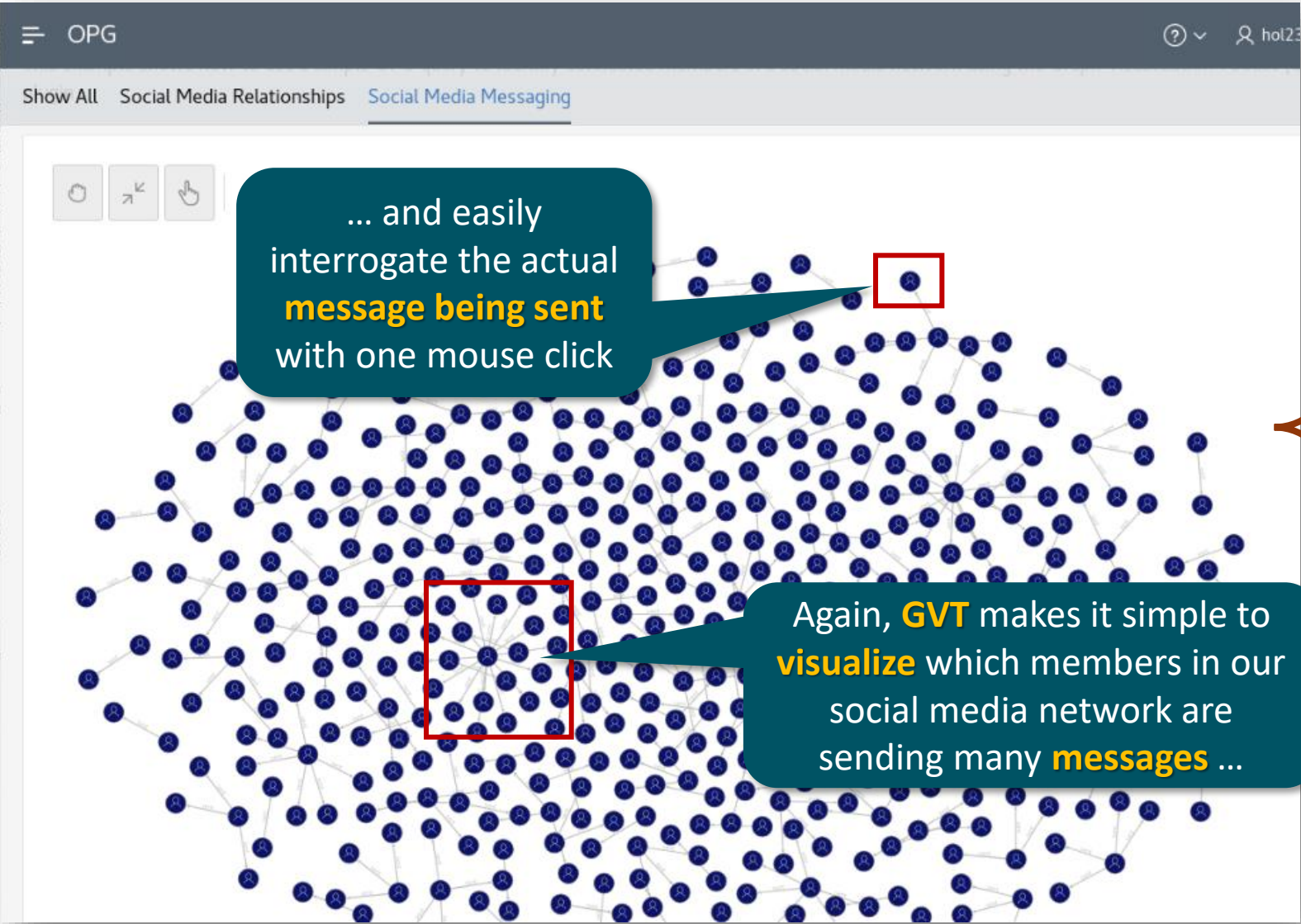
The plug-in can be **customized** with dynamic actions to switch **node and edge attributes ...**

Probing Social Media Networks Graphically (2)



... or switch network display formats from the default (FORCE) to another format

Probing Social Media Networks Graphically (3)



MSG_DTM	2024-02-08T00:03:07
MSG_ID	6507
MSG_POLARITY	4
MSG_SUBJECTIVITY	0
MSG_TEXT	I understand your point but I have to ask...have you actually priced new vehicles in the last three years? ALL of them are insanely priced. If this is a low-40k vehicle...it would definitely be in-line with a comparably equipped ICE version of an SUV especially after the tax credit.

OPG vs PGX: Advantages and Drawbacks

Advantage	PGX	OPG
Access to visualization via GraphWiz plug-ins and tools	YES	YES
Easily accessible via SQL clients (e.g. SQL Developer)	YES	YES
Easily accessible via APEX	NO	YES
Tuned to handle high transaction volumes in real time	NO	YES
Direct access to PGX analytic functions *	YES	NO
Direct access to PGX ML models *	YES	NO

*There are **additional limitations** for **PGQL queries** when accessing SQL property graphs!

Plans for Future Experimentation

```
WITH
query AS (
  -- source and target constitutes the edge direction and thus
  -- represents the reporting structure of employee to their managers.
  SELECT
    source
    ,target
  FROM GRAPH_TABLE(
    smi_network
    MATCH (m1) -[p IN CONNECTIONS]->(m2)
    COLUMNS (p_r_src AS source, P_r_tgt AS target)
  )
  --WHERE src_id IN (1,2) AND dst_id between 100 and 125
),
page AS (
  -- Pagination
  SELECT *
  FROM query
  ORDER BY
    source
    ,target
  -- OFFSET :page_start ROWS --:page_start
  -- FETCH NEXT :page_size ROWS ONLY --:page_size
)
,vertices AS (
  -- Fetch node details from OPGraph to construct JSON
  SELECT
```

Build more complex **OPG queries** to solve even more complex **business problems**

Top 12 Vertices Based on PageRank

```
%pgql-pgx
SELECT x.pagerank, x.handle, x.followers, x.following
  FROM MATCH (x)
     ON nw_complex
 ORDER BY pagerank DESC
 LIMIT 12
```

Apply **PGQL** via **Graph Studio** to OPGs

PGX Analytic Functions: Results

```
%pgql-pgx
SELECT
  | x.handle as "Handle"
  ,x.followers as "# of Followers"
  ,x.following as "# of Following"
  ,x.pagerank as "PageRank"
  ,x.betweenness as "DistBtwn"
  ,x.authority as "Hits - Authority"
  ,x.hubs as "Hits - Hubs"
  FROM MATCH (x)
     ON nw_complex
 WHERE x.betweenness > 0
 ORDER BY x.betweenness DESC
```

Apply PGX **analytic and ML functions** to OPG graphs in **(nearly) real time**

Useful References

Property Graph Developer's Guide

<https://docs.oracle.com/en//database/oracle/property-graph/23.3/spgdg/index.htm>

Using the APEX Graph Visualization Plug-In

<https://docs.oracle.com/en//database/oracle/property-graph/23.3/spgdg/visualizing-sql-graph-queries-using-apex-graph-visualization-plug.html>

Oracle Graph JavaScript API Reference for Property Graph Visualization

<https://docs.oracle.com/en/database/oracle/property-graph/23.4/pgjsd/index.html>

APEX and Property Graphs in Oracle Database 23c (video)

<https://www.youtube.com/watch?v=DODoJI3sR14>

Valuable In-Depth Training and Techniques

Oracle Graph Learning Path

<https://blogs.oracle.com/database/post/oracle-graph-learning-path>

Powering Network Topology Planning and Administration with Oracle Graph

<https://blogs.oracle.com/database/post/powering-network-topology-planning-and-administration-with-oracle-graph>

Oracle Graph on Medium

<https://medium.com/tag/oracle-graph>

Additional Data Sources and Use Cases

Welcome to Our New ‘Bespoke’ Realities:

<https://www.nytimes.com/2023/11/30/opinion/political-reality-algorithms.html>

Fame for sale: Efficient detection of fake Twitter followers

<https://www.sciencedirect.com/science/article/abs/pii/S0167923615001803>

Social Media Sentiment Analysis

<https://towardsdatascience.com/sentiment-analysis-74624b075842>

Detecting Fake Users on Social Media with a Graph Database

<https://journals.uvic.ca/index.php/arbutus/article/view/20027>

Tesla sweeps the 2023 Cars.com American-Made Index

<https://www.foxnews.com/auto/tesla-sweeps-2023-cars-com-american-made-index>