Graph database applications with SQL/PGQ

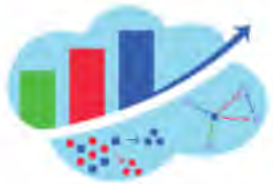
Jan Michels, Andy Witkowski
Oracle

Safe Harbor Statement

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.

Speaker Bios

- Andy Witkowski – Oracle – VP – Database SQL
  - 25 years experience designing and implementing SQL functionality in the Oracle database server
  - 48 US issued patents
- Jan Michels – Oracle – Principal Member of Technical Staff
  - 20 years experience designing extensions to SQL
    - XML, temporal, JSON, property graphs



Agenda

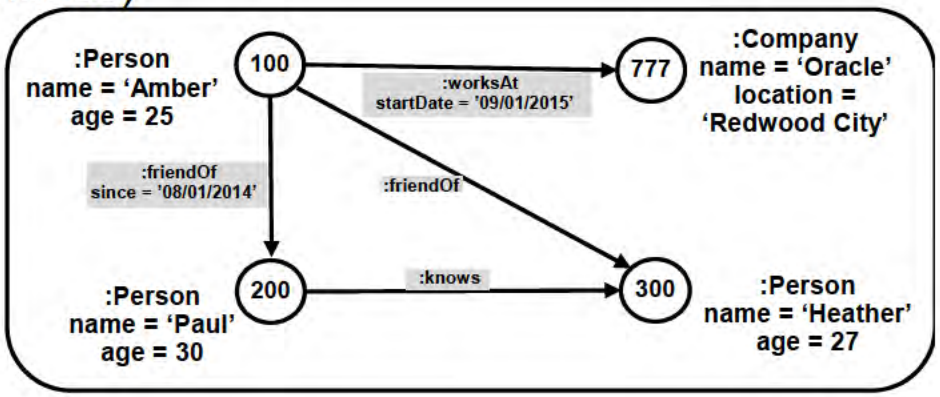
- SQL and Property Graphs
- How to use the SQL Extensions for Property Graphs
- Live Example

SQL and Property Graphs

- SQL – well-known – no further comment
- Property graphs:
  - Perform powerful analysis (regular path expression – Kleene * and shortest/cheapest paths) on data such as:
    - Networks: social, sensor, electrical grid, bill of materials, *etc.*
    - Financial transactions (*e.g.*, fraud detection, anti-money laundering)
    - Product recommendations (*e.g.*, friends of friends read similar books)
    - *Etc.*

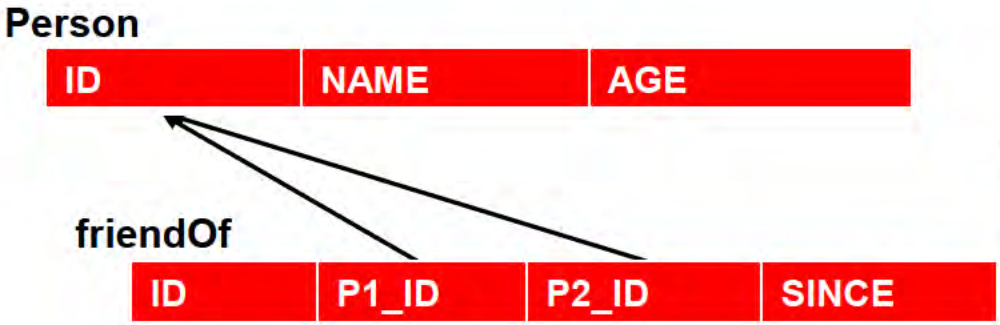
What is a property graph? (simplified)

- A set of vertices (sometimes called nodes)
- A set of (directed) edges
  - Edges connect vertices
- Vertices and edges can
  - Have labels (one or more)
    - A label is an identifier
    - Provides typing information
  - Have properties/attributes (zero or more)
    - Property definitions are associated with labels
    - A property is a typed key/value pair

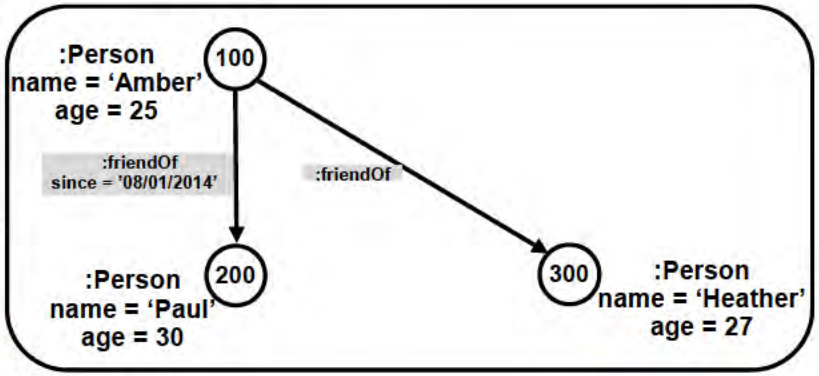


Relational vs. PG data model

- In relational DM, relationships are encoded implicitly (FK/PK, joins)
- In PGDM, relationships are explicit part of the DM in form of edges between entities (vertices)



VS.



Relational vs. Graph View

- Relational model is widely used for general data management
- However, the model is cumbersome for answering certain questions

Are Bob and Charlie gaming the system? Is there any money flow between them?

Is there any money flowing in cycles back to the originating owner (laundering)?

Account Table

Account ID	Owner ID	Creation Date
1111	200	2010-3-10
2222	100	2011-2-13
3333	400	2015-9-16
4444	300	2012-5-25
5555	100	…

Customer Table

Owner ID	Name
100	Alice
200	Bob
300	Charlie
400	Dave
…	

Transfer Table

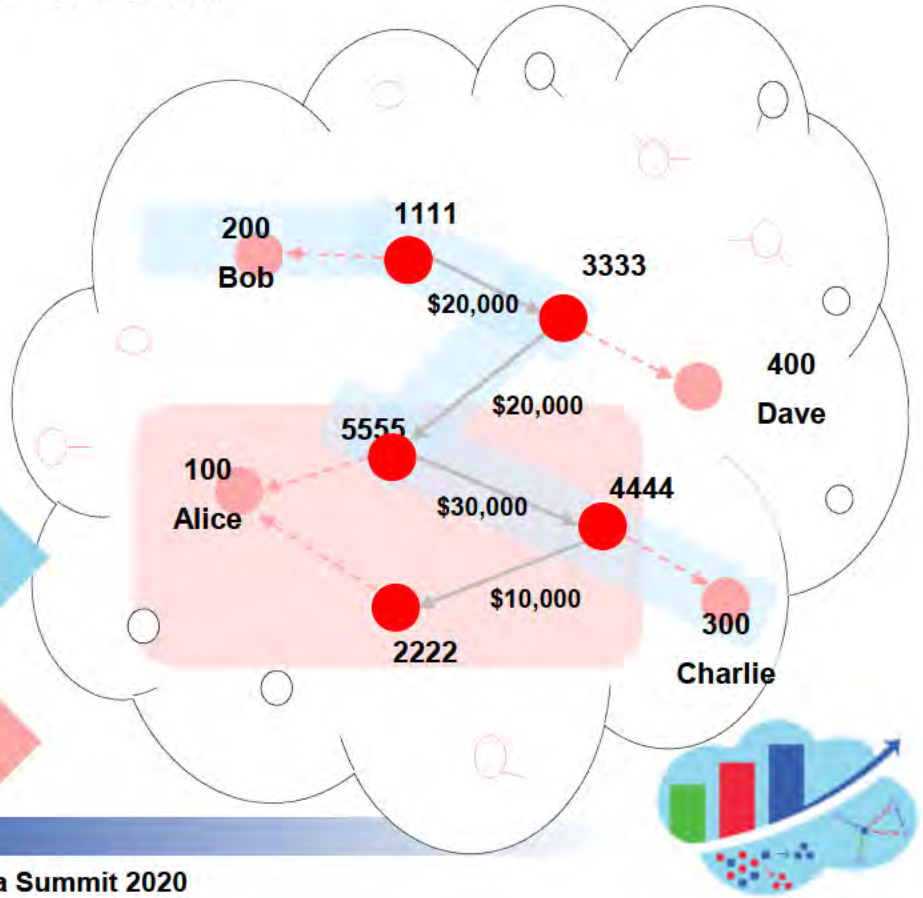
SRC	DEST	Type	Amount
1111	3333	Wire	$20,000
5555	4444	Wire	$30,000
4444	2222	Recurring	$10,000
3333	5555	Wire	$20,000
….			

Benefits from Graph View

- Represent the dataset as a graph
  - Entities become vertices
  - Relationships become edges
- Previous Qs more intuitive to answer with graph representation

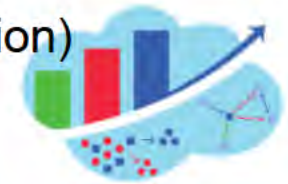
Are Bob and Charlie are gaming the system? Is there any money flow between them?

Is there any money flowing in cycles back to the originating owner (laundering)?



Why integrate with SQL?

- Data is already stored in relational databases
  - Tables model vertices and edges
  - Table columns model properties
- SQL has powerful (analytical) functionality
  - No need to duplicate functionality in a stand-alone property graph data base
  - *E.g.*, GROUP BY, row pattern matching, window functions, SQL+analytics functions, etc.
  - *E.g.*, security model, transactions, manageability, metadata
- Easily join graph data with relational (non-graph) data
  - No need to ship data from graph system to relational system or vice versa
- Graph definition in SQL dictionary (along with SQL schema definition)



Graph Application Example: Anti-Money Laundering (AML)

- Anti-Money Laundering (AML)
  - A big application in financial domain
  - Detect and report suspicious activity including offenses to **money laundering**
- **Graph** is very useful in building AML solutions
  - Money laundering activities are spread across many transactions between multiple entities
  - Need to gather up actives that look suspicious individually and analyze inter-connections among those

AML Example

- How it works with graph (a rough sketch)

2. Process transaction records

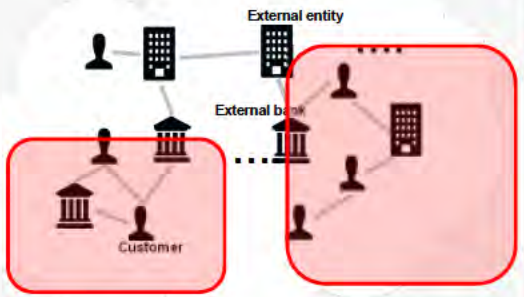
- ***Alerts*** **are created around** ***entities*** **(i.e. beneficiary, originator, intermediary)**
- **if the transaction violates certain rules**
- **E.g. amount too large, transaction too frequent, entity blacklisted, …**

Originator	Beneficiary	Time Stamp	Amount	…
Paul	Zion Bank	6/19 5:59:59.336 UTC+8	$ 30,000	… ✓
Jack	E-Weddingbands LLC	6/19 5:59:59.516 UTC+8	$ 112,000	… ⚠
James	Provo Bank	6/19 6:01:20.222 UTC+8	$ 150.00	… ✓
Steve	Linda	6/19 6:02:55.222 UTC+8	$ 999.30	… ✓
…	…	…	…	…

- **Created alerts are attached to entity graph**

1. Represent the financial data set as a Graph

- **A graph among financial entities as vertices (e.g. customers, external bank, company, etc.)**
- **Various relationships are captured as edges (e.g. has transaction history, ownership, etc …)**



Entity Graph

3. Identify correlated alerts

- **From the graph, the** ***groups of correlated alerts*** **can be identified**
- **Each group is evaluated and scored for their potential risks.**
- **High scored groups are promoted as** ***cases*** **that requires further investigation from human experts**

SQL Extensions for Property Graphs

- Graph - first class database object
  - View-like object (initially no-on disk materialization)
  - Created using DDL statements out of relational tables
    - Tables hold data representing vertices & edges
    - No restriction on the number of vertex and edge tables in a given graph
    - No restriction on the number of graphs in a database
- Primarily aimed at supporting graph querying over existing schemas
- Data Model + DDL
- Graph Pattern Matching + SQL query syntax
- Designed for and in close collaboration with the US and ISO SQL standards committees.

Sample Graph Data – Underlying Tables

customer

CID	NAME	CITY
100	Joe	San Jose
200	Jane	Santa Clara
300	Jeremy	San Francisco
400	Jessica	Redwood Shores
500	Fletcher	San Jose

account

AID
10
20
30
40
50

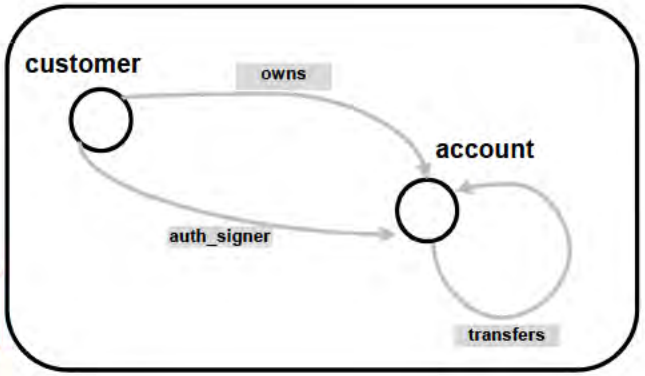
owns

OID	CID	AID	SINCE
110	100	10	1/1/2019
220	200	20	2/2/2019
330	300	30	3/3//2019
440	400	40	4/4/2019
550	500	50	5/5/2019
510	500	10	1/1/2019

transfers

TID	FROM_ID	TO_ID	WHEN	AMOUNT
102001	10	20	1/1/2020	5000
103001	10	30	1/1/2020	15000
104001	10	40	1/1/2020	20000
105001	10	50	1/1/2020	25000
304001	30	40	1/2/2020	11000
305001	30	50	1/2/2020	4000
403001	40	30	1/3/2020	15000
305002	30	50	1/3/2020	14000

Property Graph Definition (DDL) - Example

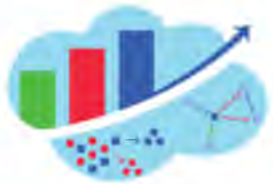


Defaults apply for label and all properties.

Explicit label and properties options for customer

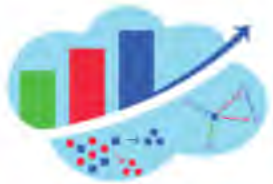
```
CREATE PROPERTY GRAPH aml
   VERTEX TABLES ( account
                 , customer
                    LABEL customer PROPERTIES ( cid, name, city ))
      EDGE TABLES ( owns SOURCE customers DESTINATION accounts
                         PROPERTIES ( since )
                  , auth_signer SOURCE customer DESTINATION account
                  , transfers
                    SOURCE KEY ( from_id ) REFERENCES accounts ( aid )
                    DESTINATION KEY (to_id) REFERENCES accounts ( aid )
                    LABEL transfers PROPERTIES ( when, amount) )
```

Columns when and amount are exposed as properties. Columns tid, from_id, and to_id are **not**.

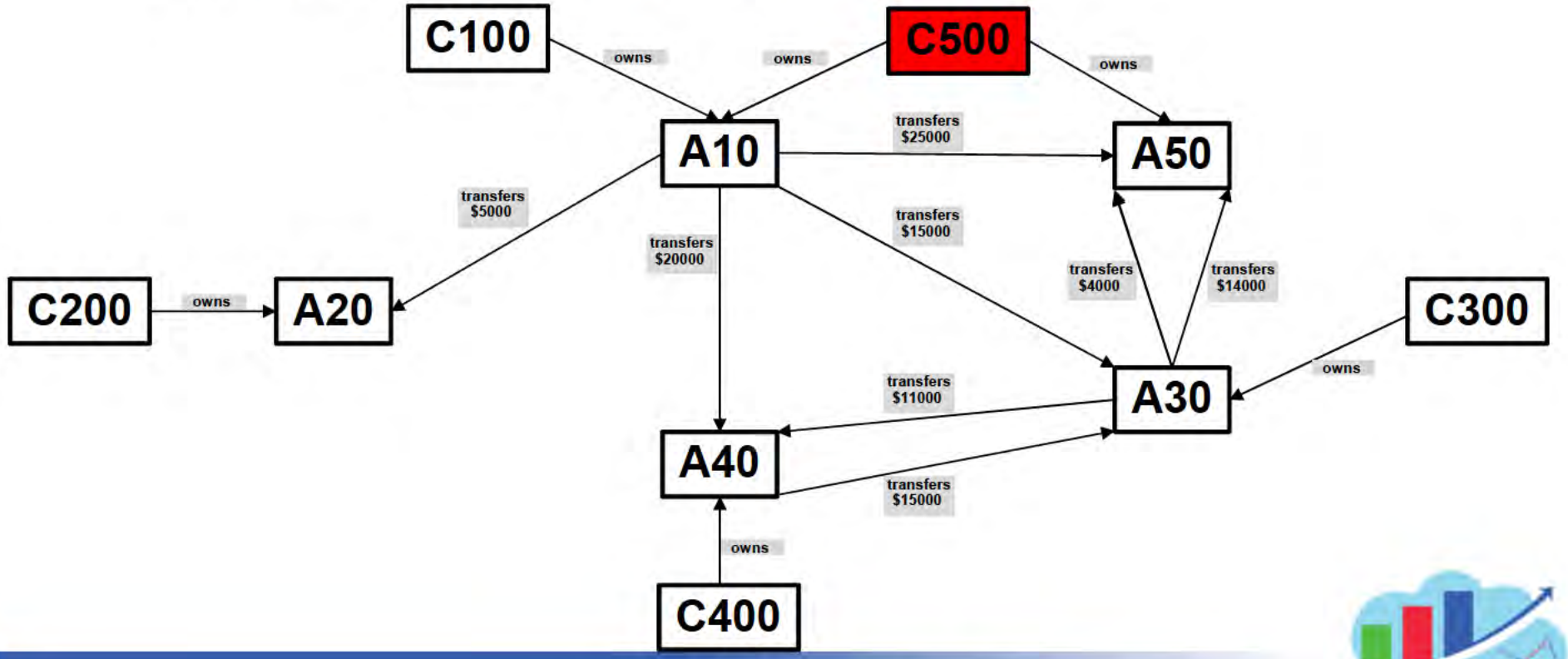


Property Graph Definition (2)

- Existing tables (or views): customers, accounts, owns, auth_signer, transfers
- User can specify options for
  - Labels (1 or more per vertex/edge table)
  - Properties (0 or more per label), can rename properties
  - Keys (single or multi-column key)
- If not specified, defaults apply:
  - Single label defaults to table name/alias
  - All (non-hidden) columns are exposed as properties for a given label
  - Keys are inferred from primary/foreign keys of underlying tables.
  - PK-FK determines connection between vertices via edges (e.g., customer –[owns]-> account)
- User can mix and match within a single PG definition:
  - Explicit options, and
  - Implicit defaults



Sample Graph Data



Querying PGs – Example 1

Retrieve the **info of all customers** who got **more than $10,000 from customer 100**.

New operator* applied to graph (aml), returns table

```
SELECT gt.cid, gt.name, gt.city, gt.amount
FROM GRAPH_TABLE ( aml,
  MATCH
   ( c1 IS customer ) -[ IS owns ]->
        ( IS account ) -[ t1 IS transfers ]->
        ( IS account ) <-[ IS owns ]- ( c2 IS customer )
  WHERE c1.cid = 100
    AND t1.amount > 10000
  COLUMNS ( c2.cid
          , c2.name
          , c2.city
          , t1.amount )
) gt
```

Edge pattern enclosed in -[]->

Vertex pattern enclosed in ()

COLUMNS defines the shape of the output table. Properties projected out of the MATCH.

* Exact syntax, placement of arguments, *etc.* currently under discussion/in flux.

Example 1 - Output

CID	NAME	CITY	AMOUNT
300	Jeremy	San Francisco	15000
400	Jessica	Redwood Shores	20000
500	Fletcher	San Jose	25000

Example 1 – using plain SQL

Retrieve the **info of all customers** who got **more than $10,000 from customer 100**.

```
SELECT c2.cid, c2.name, c2.city, t1.amount
FROM customer c1, owns o1, account a1, transfers t1
       , account a2, owns o2, customer c2
WHERE c1.cid = o1.cid
  AND o1.aid = a1.aid
  AND a1.aid = t1.from_id
  AND t1.to_id = a2.aid
  AND a2.aid = o2.aid
  AND o2.cid = c2.cid
  AND c1.cid = 100
  AND t1.amount > 10000
```

Querying PGs – Example 1a*

Retrieve the **info of all customers** who got **more than $10,000 from customer 100 via 1 intermediary**.

```
SELECT gt.cid, gt.name, gt.city, gt.amount1, gt.amount2
FROM GRAPH_TABLE ( aml,
  MATCH
   ( c1 IS customer ) -[ IS owns ]->
        ( IS account ) -[ t1 IS transfers ]->
        ( IS account ) -[ t2 IS transfers ]->
        ( IS account ) <-[ IS owns ]- ( c2 IS customer )
  WHERE c1.cid = 100
    AND t1.amount > 10000
    AND t2.amount > 10000
  COLUMNS ( c2.cid
          , c2.name
          , c2.city
          , t1.amount AS amount1
          , t2.amount AS amount2 )
 ) gt
```

* Equivalent plain SQL query not shown. But straightforward extension of the previous one.

Example 1a - Output

CID	NAME	CITY	AMOUNT1	AMOUNT2
400	Jessica	Redwood Shores	15000	11000
300	Jeremy	San Francisco	20000	15000
500	Fletcher	San Jose	15000	14000

Querying PGs – Example 1b

Retrieve the info of all customers who got more than $10,000 from customer 100 via **2** intermediary.

```
SELECT gt.cid, gt.name, gt.city, gt.amount1, gt.amount2
FROM GRAPH_TABLE ( aml,
  MATCH
    ( c1 IS customer ) -[ IS owns ]->
          ( IS account ) -[ t1 IS transfers ]->
          ( IS account ) -[ IS transfers ]->
          ( IS account ) -[ t2 IS transfers ]->
          ( IS account ) <-[ IS owns ]- ( c2 IS customer )
  WHERE c1.cid = 100
    AND t1.amount > 10000
    AND t2.amount > 10000
  COLUMNS ( c2.cid
            , c2.name
            , c2.city
            , t1.amount AS amount1
            , t2.amount AS amount2 )
 ) gt
```

Example 1b - Output

CID	NAME	CITY	AMOUNT1	AMOUNT2
300	Jeremy	San Francisco	15000	15000
400	Jessica	Redwood Shores	20000	11000
500	Fletcher	San Jose	20000	14000

Querying PGs – Example 2

Retrieve the **info of all unique customers** who are connected to fraudster 500 **within four hops** via money transfers (incoming, outgoing, or both).

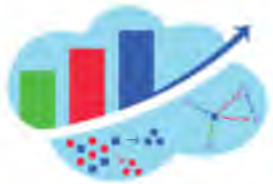
```
SELECT DISTINCT gt.cid, gt.name, gt.city
FROM GRAPH_TABLE ( aml,
  MATCH
   ( c1 IS customer ) -[ IS owns ]->
        ( IS account ) -[ IS transfers ]- {1,4}
        ( IS account ) <-[ IS owns ]- ( c2 IS customer )
  WHERE c1.cid = 500
  COLUMNS ( c2.cid
          , c2.name
          , c2.city )
 ) gt
```

Repeating edge (1 to 4 times)

Example 2 – using plain SQL

Retrieve the **info of all unique customers** who are connected to fraudster **500** within four hops via money transfers (incoming, outgoing, or both).

- Can be done, but query would fill several slides
- Left as an exercise for the audience
  - Hint: can use recursive WITH or built result iteratively



Querying PGs – Example 2a

Retrieve the **info of all customers** who are connected to fraudster **500** within four hops via money transfers (incoming, outgoing, or both), and **their closest distance from 500**.

```
SELECT gt.cid, gt.name, gt.city, MIN(gt.hops)
FROM GRAPH_TABLE ( aml,
  MATCH
    ( c1 IS customer ) -[ IS owns ]->
        ( IS account ) -[ t1 IS transfers ]- {1,4}
        ( IS account ) <-[ IS owns ]- ( c2 IS customer )
  WHERE c1.cid = 500
  COLUMNS ( c2.cid
          , c2.name
          , c2.city
          , COUNT (t1.*) AS hops )
 ) gt
GROUP BY gt.id, gt.name, gt.city
```

Aggregate function inside COLUMNS clause.

Querying PGs – Example 3

Find the (shortest) loops where money flows back to the original sender (not necessarily the same account).

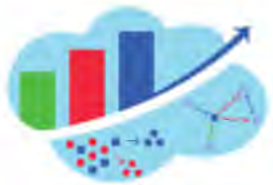
```
SELECT DISTINCT gt.cid, gt.name, gt.city, gt.min_hops
FROM GRAPH_TABLE ( aml,
  MATCH SHORTEST
   ( c1 IS customer ) -[ IS owns ]->
        ( IS account ) -[ t1 IS transfers ]->*
        ( IS account ) <-[ IS owns ]- ( c1 ) )
  COLUMNS ( c1.cid
          , c1.name
          , c1.city
          , COUNT(t1.*) AS min_hops)
 ) AS gt
```

SHORTEST makes an unbounded upper limit computable.

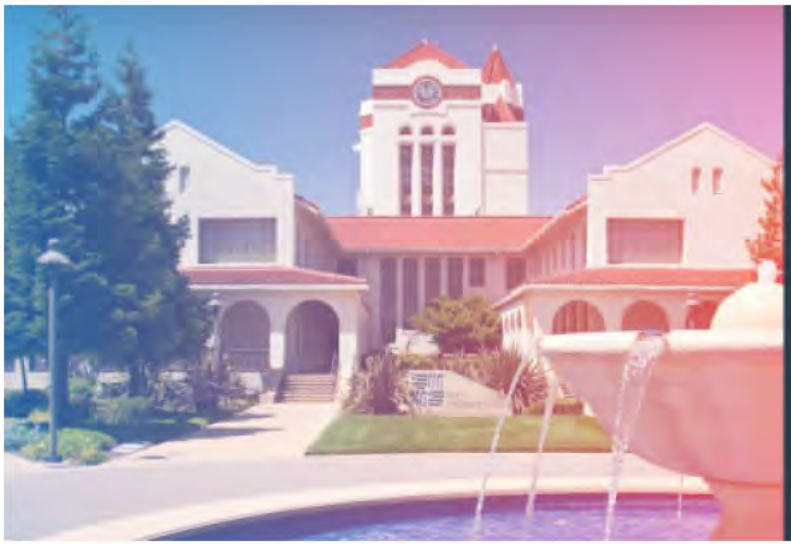
Kleene * - 0 to unbounded iterations

Key Takeaways

- New database object
  - Property graph
- Functionality of finding complex path (variable length/shortest path) in SQL
  - Without resorting to procedural language embedding SQL



Questions & Answers



ANALYTICS AND DATA SUMMIT 2020

All Analytics. All Data.
No Nonsense.

February 25-27, 2020