ORACLE®

# PGQL: a Property Graph Query Language

Oskar van Rest
Sungpack Hong
Jinha Kim
Xuming Meng
Hassan Chafi

Oracle Labs
June 24, 2016

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Safe Harbor Statement

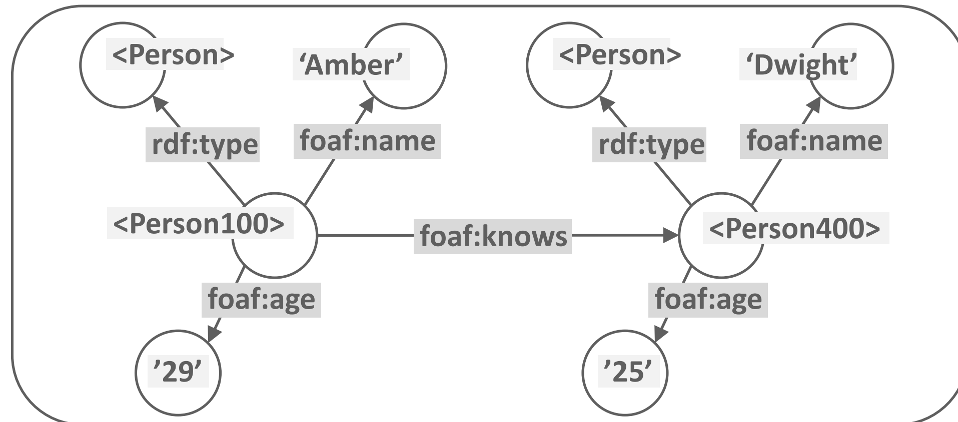
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development,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

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Query Languages for Graphs

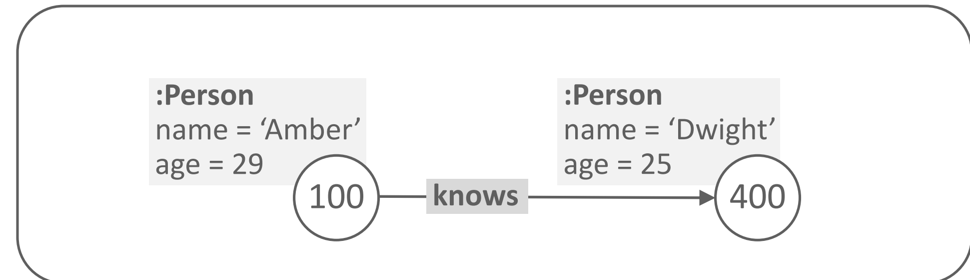
## RDF Graphs




- Standard query language:
  - W3C **SPARQL 1.1**

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

## Property Graphs




- No standard query language
- Existing languages:
  - PGQL (Oracle) ← this presentation
  - Cypher (Neo4j)
  - Gremlin (Tinkerpop)
  - *... (LDBC)*

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

https://github.com/oracle/pgql-lang

# PGQL: a Property Graph Query Language

- Very close to SQL
- Aligned with existing database management systems

```
SELECT y.name, e, p, GRAPH{x -[e]-> y}
FROM snGraph IN { SELECT … FROM … WHERE … }
WHERE
   (x:Person WITH name = 'Paul') -[e:likes]-> (y),
   (z:Person WITH name = 'Amber') -/p:likes*/-> (y),
   x.age > y.age
GROUP BY
ORDER BY
LIMIT
OFFSET
```



ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Limitations of other languages (1): Path Queries

- SPARQL
  - Reachability only; no path finding
- Cypher
  - Find *all* paths as default semantics (too expensive)
  - Limited RPQs
- SQL
  - Doesn't allow for naturally expressing path queries (i.e. recursive SQL)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Limitations of other languages (2): Graph Construction

- SPARQL
  - Query either returns a result set or a graph → no uniform interface
  - Queries that return graphs cannot be composed
- Cypher
  - Can't construct graphs
- (SQL: can construct tables naturally so no limitation)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# PGQL: a Property Graph Query Language

1. Graph Pattern Matching Queries
2. Path Queries
3. Graph Construction Queries

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# PGQL: a Property Graph Query Language

1. Graph Pattern Matching Queries
2. Path Queries
3. Graph Construction Queries

ORACLE

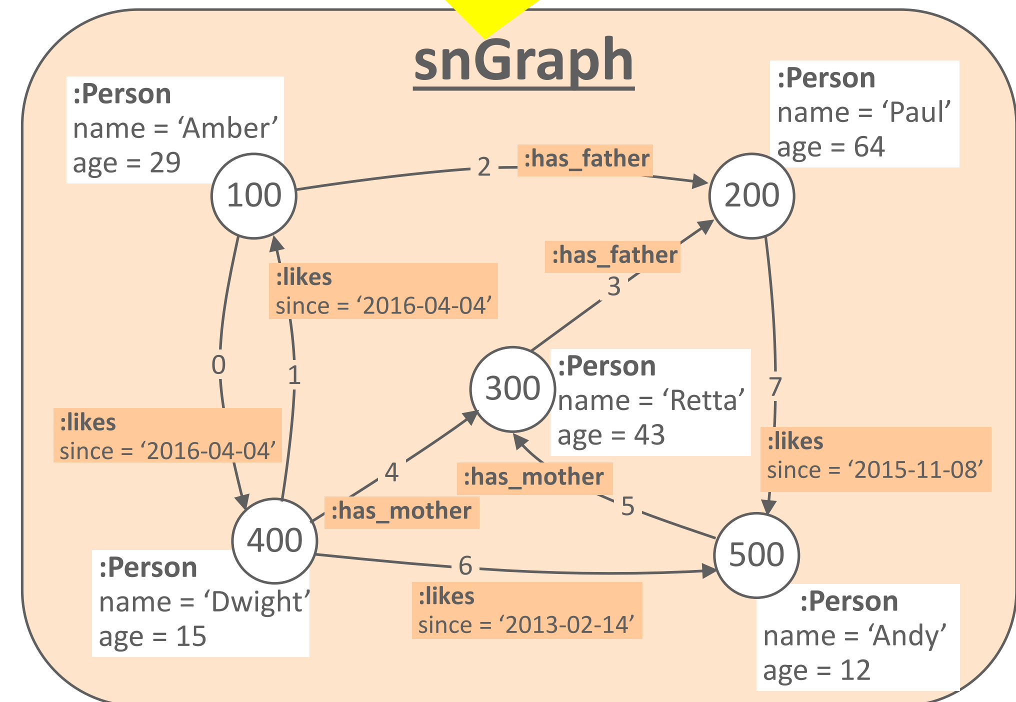
Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Property Graph Data Model

- (Named) graph with vertices and edges
- Vertex
  - Unique identifier
  - Set of labels
  - Properties (arbitrary key-value pairs)
- Edge
  - Unique identifier
  - Single label
  - Properties (arbitrary key-value pairs)
  - Direction

A value can be an object of any type (e.g. Integer, String, Set<String>, etc.)

Name of graph ( “Social Network Graph” ➔ snGraph)




Property graph can be schema-less

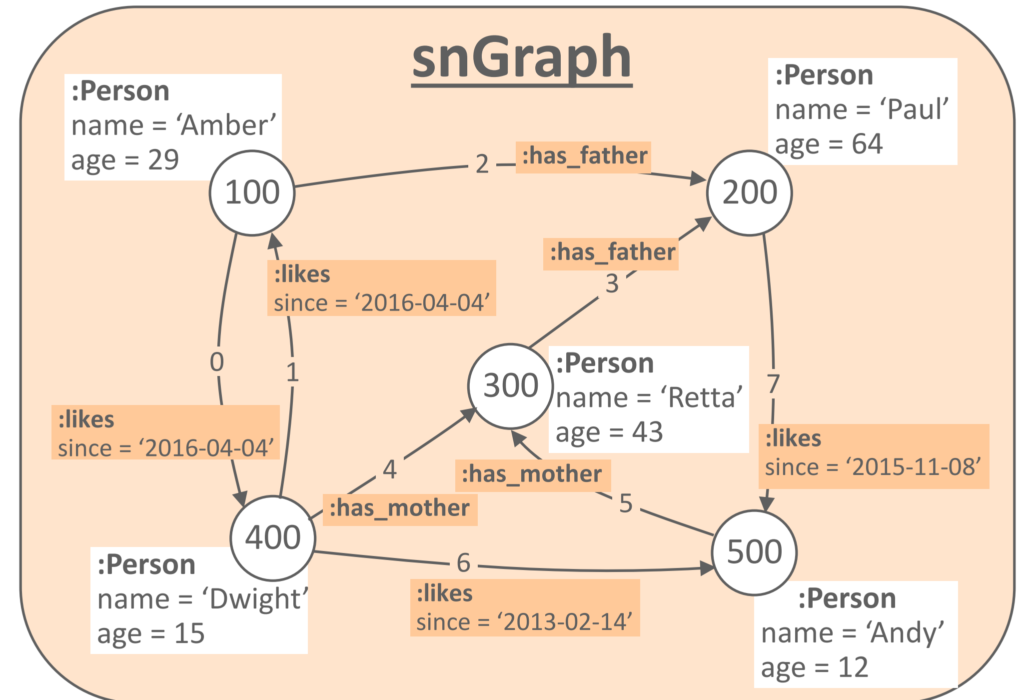
ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Example Query

"Find friends of Paul that are also friends, or friends of friend, of Amber. The friend should be older than Amber."

```
SELECT y.name, y.age
FROM snGraph
WHERE
  (x:Person WITH name = 'Paul') -[:likes]-> (y:Person)
  (z:Person WITH name = 'Amber') -/:likes*1..2/-> (y)
  z.age > y.age
```






Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

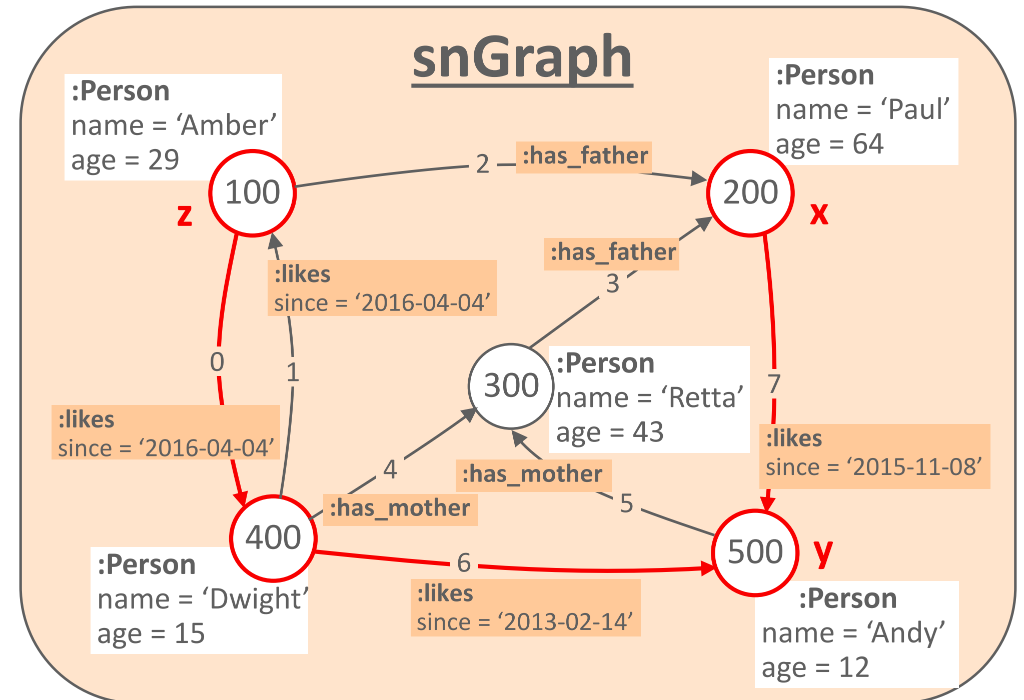
# Example Query

"Find friends of Paul that are also friends, or friends of friend, of Amber. The friend should be younger than Amber."

```
SELECT y.name, y.age
FROM snGraph
WHERE
   (x:Person WITH name = 'Paul') -[:likes]-> (y:Person)
   (z:Person WITH name = 'Amber') -/:likes*1..2/-> (y)
   z.age > y.age
```

Result set

| y.name | y.age |
|---|---|
| 'Andy' | 12 |




ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Type System

- Built-in datatypes:
  - **Basic types**: Integer, String, Double, Set<String>, ...
  - **Graph-specific types**: Vertex, Edge, Path, Graph
- Query result is always a "**result set**"
  - Even graphs are returned in the result set
  - Like in SQL, result set may contain duplicate rows ("bag semantics")

**Built-in functions:**
Vertex: `id(), labels()`
Edge: `id(), label()`
Path: `length(), weight()`
Graph: `numNodes(), numEdges()`

A graph inside the result set

```
SELECT y.name, y.id(), y, e, p, snGraph
FROM snGraph
WHERE
  (x:Person WITH name = 'Paul') -[e:likes]-> (y)
  (z:Person WITH name = 'Amber') -/p:likes*1..2/-> (y)
  z.age > y.age
```

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Common Query Functionality

- Grouping and aggregation
  - Similar to SQL

```
SELECT x.age, COUNT(*)
FROM snGraph
WHERE
  (x:Person)
GROUP BY x.age AS xAge
HAVING ...
```

- Order by, offset, limit
  - Similar to SQL
- Union (all)
  - Similar to SQL

- Subqueries
  - Bottom-up semantics

```
SELECT x.name
FROM snGraph
WHERE
   (x:Person), x.age = maxAge,
   {
     SELECT MAX(x.age) AS maxAge
     WHERE (x:Person)
   }
```

- Filtering using graph patterns
  - EXISTS & NOT EXISTS (like SPARQL)

```
SELECT y.name
FROM snGraph
WHERE
 (:Person WITH name = 'Paul') -[:likes]-> (y)
 NOT EXISTS {
   (:Person WITH name = 'Amber') -[:likes]-> (y)
 }
```

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# PGQL: a Property Graph Query Language

1. Graph Pattern Matching Queries
2. Path Queries
3. Graph Construction Queries

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Path Queries

- Test for existence of, or retrieve, **arbitrary-length** paths
  - This is a very natural operation on graphs: relationships can be traversed without computing joins
- Two types of path queries:
  - **Reachability queries** *test* for the existence of a paths between pairs of vertices
  - **Path finding queries** *retrieve* paths between pairs of vertices
- Three challenges to overcome (next three slides)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Challenge 1: Supporting Regular Path Queries (RPQs)

- What are RPQs?
  - RPQs specify a regular expression over the edges along a path
    - Reachability: test if some path satisfies the RPQ
    - Path finding: return one or more paths that satisfy the RPQ
- Example RPQs in SPARQL 1.1:
  - `{ <http://example/thing> rdf:type/rdfs:subClassOf* ?type }`

    “find all types of http://example/thing”
  - `{ ?x rdf:type|^rdf:type ?y }`

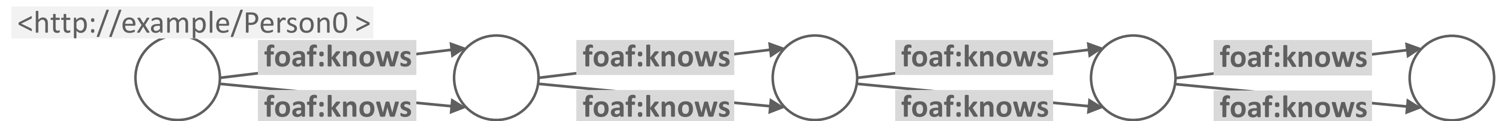
    “find all vertices *x* and *y* such that *x* is a type of *y* or *y* is a type of *x*”
- In the past, RPQs were mostly applied to “labelled graph” (e.g. RDF)
- How to support RPQs over <u>property</u> graphs?

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Challenge 2: Avoiding expensive “all paths” enumeration

- Assume simple SPARQL query:
  - `{ <http://example/Person0> foaf:knows* ?y }`
  - Reachability: O(n + m)
  - All path finding: O(n + $2^m$)




- Should not rely on query planner to translate e.g. an ‘all path finding query’ into a ‘reachability query’
- Use cases that require materialization of all paths are very uncommon

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Challenge 3: Choosing a path finding semantic

- Different use cases demand for different semantics:
  - Find **single shortest** paths (weighted / minimal hop)
  - Find ***k* shortest** paths (weighted / minimal hop)
  - Find ***k* dissimilar** paths
  - Find ??? paths
- However, multiple “built-in semantics” make the language too complex

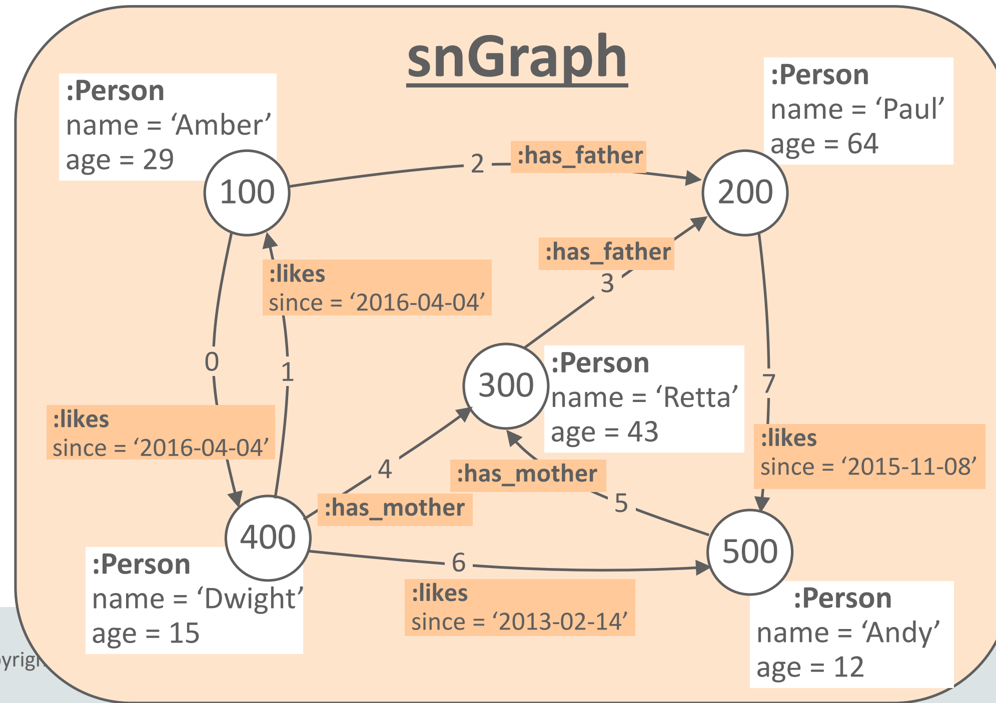
ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# RPQs for Property Graphs: Composable Path Patterns (1)

- **PATH** construct allows for specifying a path pattern
  - Graph patterns (**WHERE**) can be composed of path patterns
  - Path patterns (**PATH**) can be composed of other path patterns

```
PATH has_parent := (:Person) -[:has_father|has_mother]-> (:Person)
PATH has_ancestor := (:Person) -/has_parent+/-> (:Person)
SELECT x.id(), y.id()
FROM snGraph
WHERE
  (x:Person WITH name = 'Andy') -/has_ancestor/-> (ancestor)
  (y) -/has_ancestor/-> (ancestor),
  x.id() > y.id()
```




ORACLE®

Copyrig

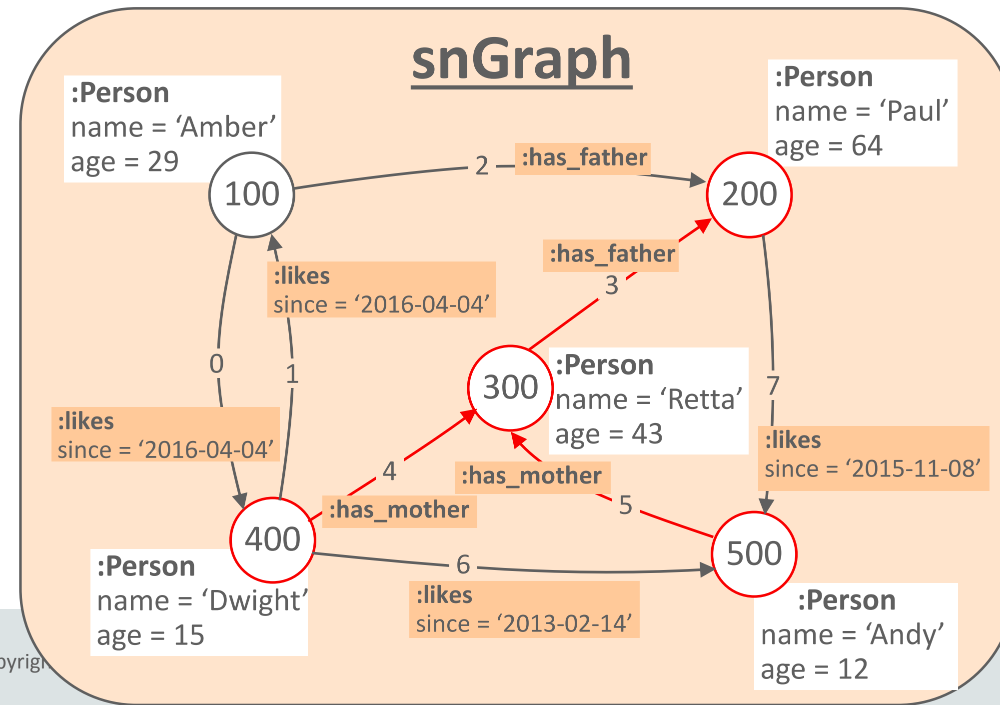
# RPQs for Property Graphs: Composable Path Patterns (1)

- **PATH** construct allows for specifying a path pattern
  - Graph patterns (**WHERE**) can be composed of path patterns
  - Path patterns (**PATH**) can be composed of other path patterns

```
PATH has_parent := (:Person) -[:has_father|has_mother]-> (:Person)
PATH has_ancestor := (:Person) -/has_parent+/-> (:Person)
SELECT x.id(), y.id()
FROM snGraph
WHERE
   (x:Person WITH name = 'Andy') -/has_ancestor/-> (ancestor)
   (y) -/has_ancestor/-> (ancestor),
   x.id() > y.id()
```

Result set

| x.id() | y.id() |
| --- | --- |
| 500 | 400 |
| 500 | 300 |




ORACLE
Copyrig

# RPQs for Property Graphs: Composable Path Patterns (2)

- Not just edge labels, but also properties:
  - Inlined property constraint

```
PATH has_new_friend := () -[:likes WITH since > '2016-01-01']-> ()
```

  - Cross-constraint on properties

```
PATH has_older_friend := (x) -[:likes]-> (y) WHERE y.age > x.age
```

- Cross-constraints in path pattern allow for **comparing data along paths**
  - More expressive than traditional RPQs (e.g. SPARQL)
- All constraints can be processed *during* path traversal
  - No need to find all paths and then filter out invalid paths

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Path Finding (1)

- PGQL's semantics:
  - **Reachability**

```
SELECT y, y.id()
WHERE
  (x@100) -/:likes+/-> (y)
```

(x@100) is shorthand for (x **WITH** id() = 100)

  - Single **min-hop shortest** path

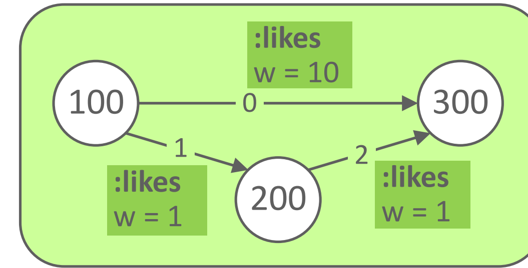
```
SELECT p, p.length()
WHERE
  (x@100) -/p:likes+/-> (y)
```

  - Single **weighted shortest** path

```
SELECT p, p.length(), p.weight()
WHERE
  (x@100) -/p'w':likes+/-> (y)
```

Edge property to minimize. More complex weight functions can be specified by constructing "weighted graph" via subquery.

Input graph




| y | y.id() |
|---|---|
| 300 | 300 |
| 200 | 200 |

| p | p.length() |
|---|---|
| 100 —0→ 300 | 1 |
| 100 —1→ 200 | 1 |

| p | p.length() | p.weight() |
|---|---|---|
| 100 —1→ 200 —2→ 300 | 2 | 2 |
| 100 —1→ 200 | 1 | 1 |


rights reserved. |

# Path Finding (2)

- What about returning *multiple* paths?
  - User-defined path-finding functions?

```
SELECT $kDissimilarPaths(snGraph, x, y, 20)
FROM snGraph
WHERE
  (x@1) -/:likes*/-> (y)
```

Example: for each source-destination pair x, y, return k = 20 *dissimilar* paths

  - Not entirely clear yet

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# PGQL: a Property Graph Query Language

1. Graph Pattern Matching Queries
2. Path Queries
3. Graph Construction Queries

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Graph Construction

- Graph construction queries allow for:
  - Decomposing complex query into simpler ones
  - Creating SQL-like Views
  - Running queries in sequence
- SPARQL allows for constructing a graph, but:
  - Query either returns a result set or a graph ➔ no uniform interface
  - Queries can only be composed of queries that return tables; not of queries that return graphs

Graph construction in SPARQL

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT   { <http://example.org/person#Alice> vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

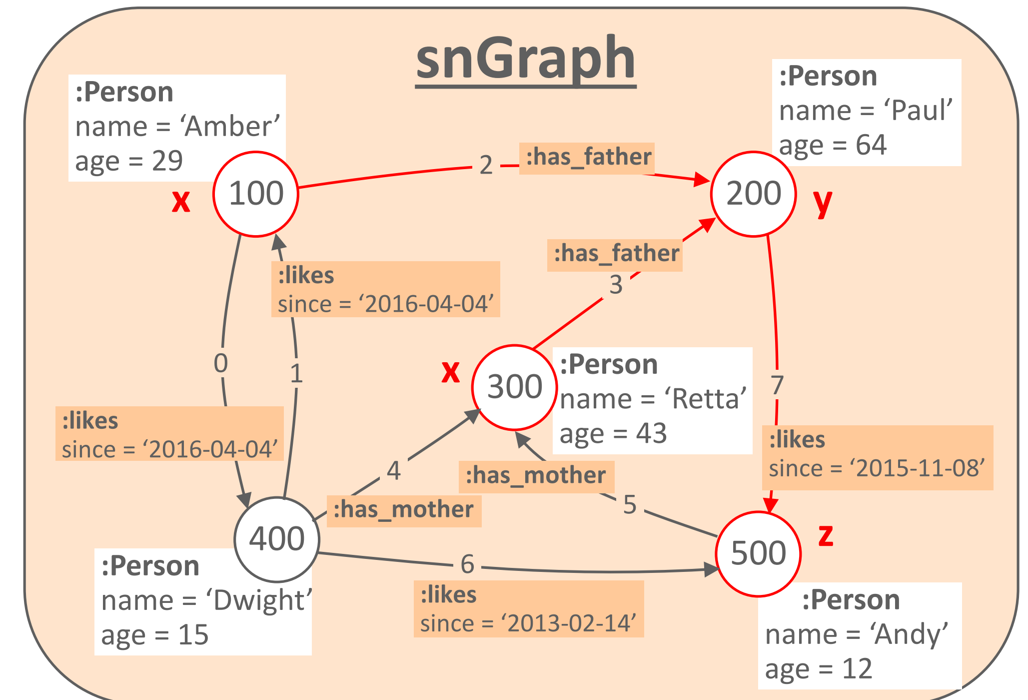
# The GRAPH Construct

- The GRAPH construct allows for specifying a graph production pattern with:
  - Vertices / edges / paths from WHERE clause
  - New vertices / edges / properties (not shown)

```
SELECT GRAPH{ (x) -[e]-> (y) } AS myGraph
FROM snGraph
WHERE
  (x) -[e:has_father|has_mother]-> (y),
  (y) -[:likes]-> (z)
```

myGraph

| |
|---|
| :Person name = 'Amber' age = 29 (100) —2 :has_father→ (200) :Person name = 'Paul' age = 64 |
| :Person name = 'Retta' age = 43 (300) —3 :has_father→ (200) :Person name = 'Paul' age = 64 |




Graph are returned inside the result set!

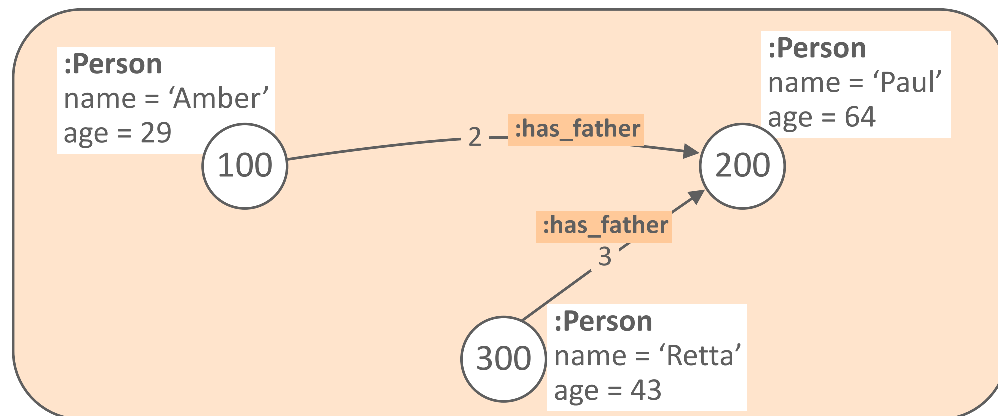
ORACLE

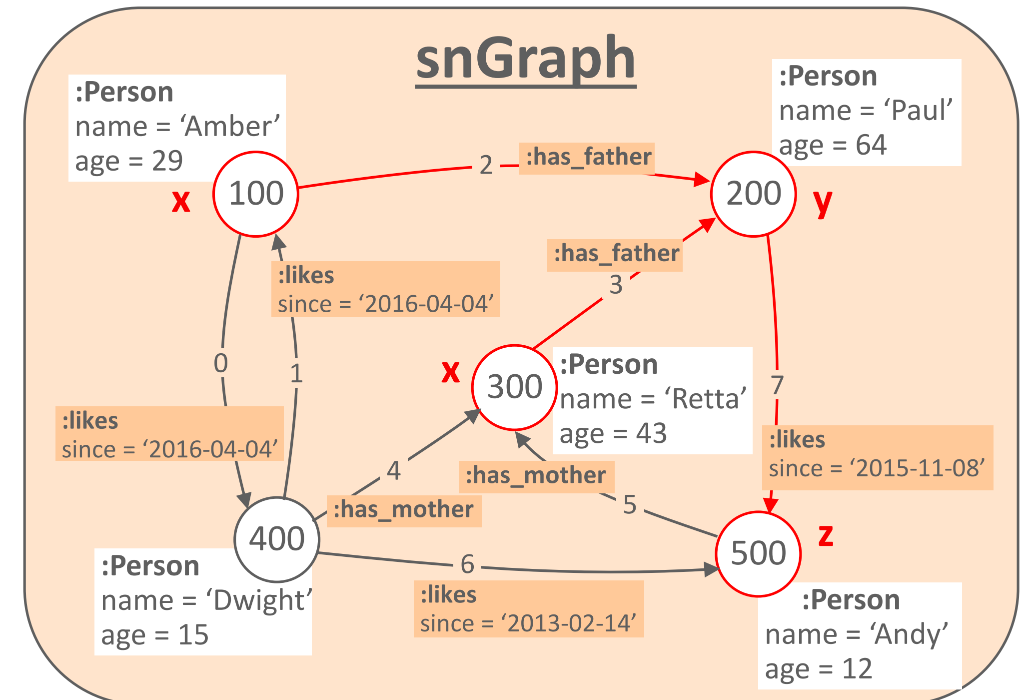
# The UNION Aggregate

Can be used in combination with GROUP BY too: construct a graph for each group (not shown)

- The UNION aggregate merges a set of graphs into a single (large) graph

```
SELECT UNION(GRAPH{ (x) -[e]-> (y) }) AS myGraph
FROM snGraph
WHERE
  (x) -[e:has_father|has_mother]-> (y),
  (y) -[:likes]-> (z)
```




ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

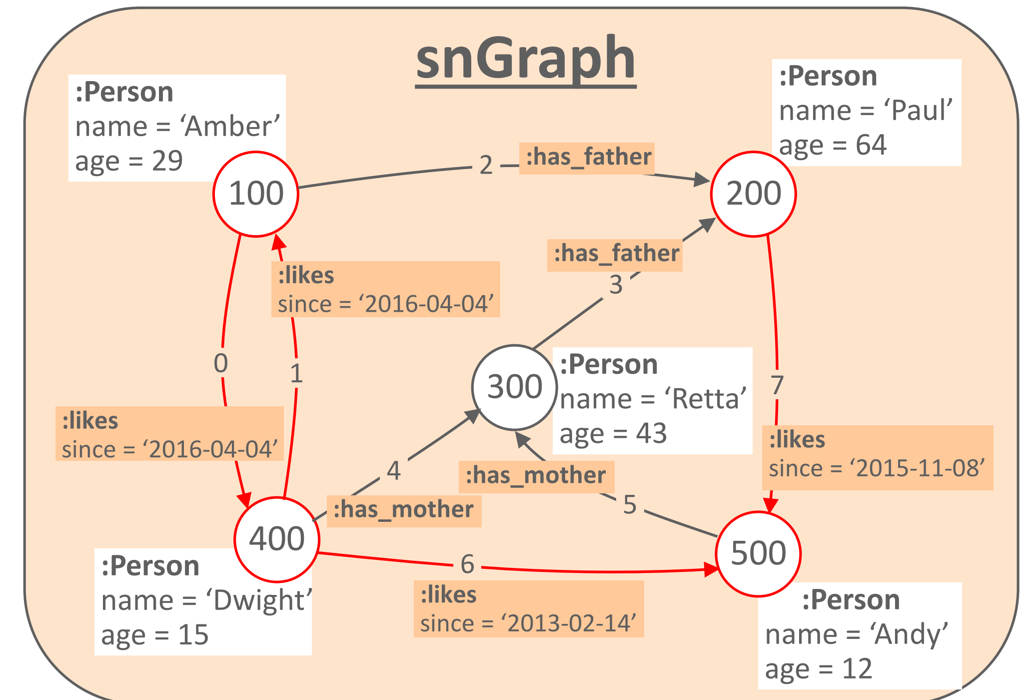
# Query composition with queries that return graphs

- Composition of queries that return tables is similar to SQL (not shown)
- But PGQL can also compose queries that return graphs:

```
SELECT COUNT(*)
FROM likesGraph IN {
  SELECT UNION(GRAPH{*}) AS likesGraph
  FROM snGraph
  WHERE () -[:likes]-> ()
}
WHERE
  (n)
```

Result:

| COUNT(*) |
|---|
| 4 |




ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

https://github.com/oracle/pgql-lang

# Conclusion

- PGQL: a Property Graph Query Language
  - Close to SQL and aligned with existing database management systems
  - Graph pattern matching + SQL-like functionality
  - Path Queries
    - RPQs on Property Graphs
    - Reachability + Shortest path finding
  - Graph Construction
    - GRAPH: specify a production pattern
    - UNION: take the union of a set of graphs
    - Can compose queries that return a table / graph

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

# Safe Harbor Statement

The preceding 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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

ORACLE®