Exploring Property Graphs with SQL/PGQ in PostgreSQL

Ashutosh Bapat
Major contributor, PostgreSQL
OSS contributor team @ Microsoft

POSETTE 2026

SQL/PGQ

SQL standard (part 16)

Property graph

- Graph (nodes and edges) with properties

Relational data as a property graph

- Logical view

Declarative graph query language

- Query in terms of nodes and edges (not joins and unions)

RDBMS Implementations

- Oracle, DuckPGQ

Committed to PG 19

Commit `2f094e7`

petere committed on Mar 16

SQL Property Graph Queries (SQL/PGQ)

Implementation of SQL property graph queries, according to SQL/PGQ
standard (ISO/IEC 9075-16:2023).

This adds:

- GRAPH_TABLE table function for graph pattern matching
- DDL commands CREATE/ALTER/DROP PROPERTY GRAPH
- several new system catalogs and information schema views
- psql \dG command
- pg_get_propgraphdef() function for pg_dump and psql

A property graph is a relation with a new relkind RELKIND_PROPGRAPH.
It acts like a view in many ways. It is rewritten to a standard
relational query in the rewriter. Access privileges act similar to a
security invoker view. (The security definer variant is not currently
implemented.)

Starting documentation can be found in doc/src/sgml/ddl.sgml and
doc/src/sgml/queries.sgml.

Author: Peter Eisentraut <peter@eisentraut.org>
Author: Ashutosh Bapat <ashutosh.bapat.oss@gmail.com>
Reviewed-by: Junwang Zhao <zhjwpku@gmail.com>
Reviewed-by: Ajay Pal <ajay.pal.k@gmail.com>
Reviewed-by: Henson Choi <assam258@gmail.com>

Ticket to Ride board game

Source: BoardGameGeek (Ticket to Ride: Paris, Days of Wonder, 2024 — image provided by the publisher)

Ticket To Ride

Turn based strategy game

Map like graph

- cities/countries – nodes
- routes – edges

Goal

- Claim paths connecting given cities/countries
- Maximize points

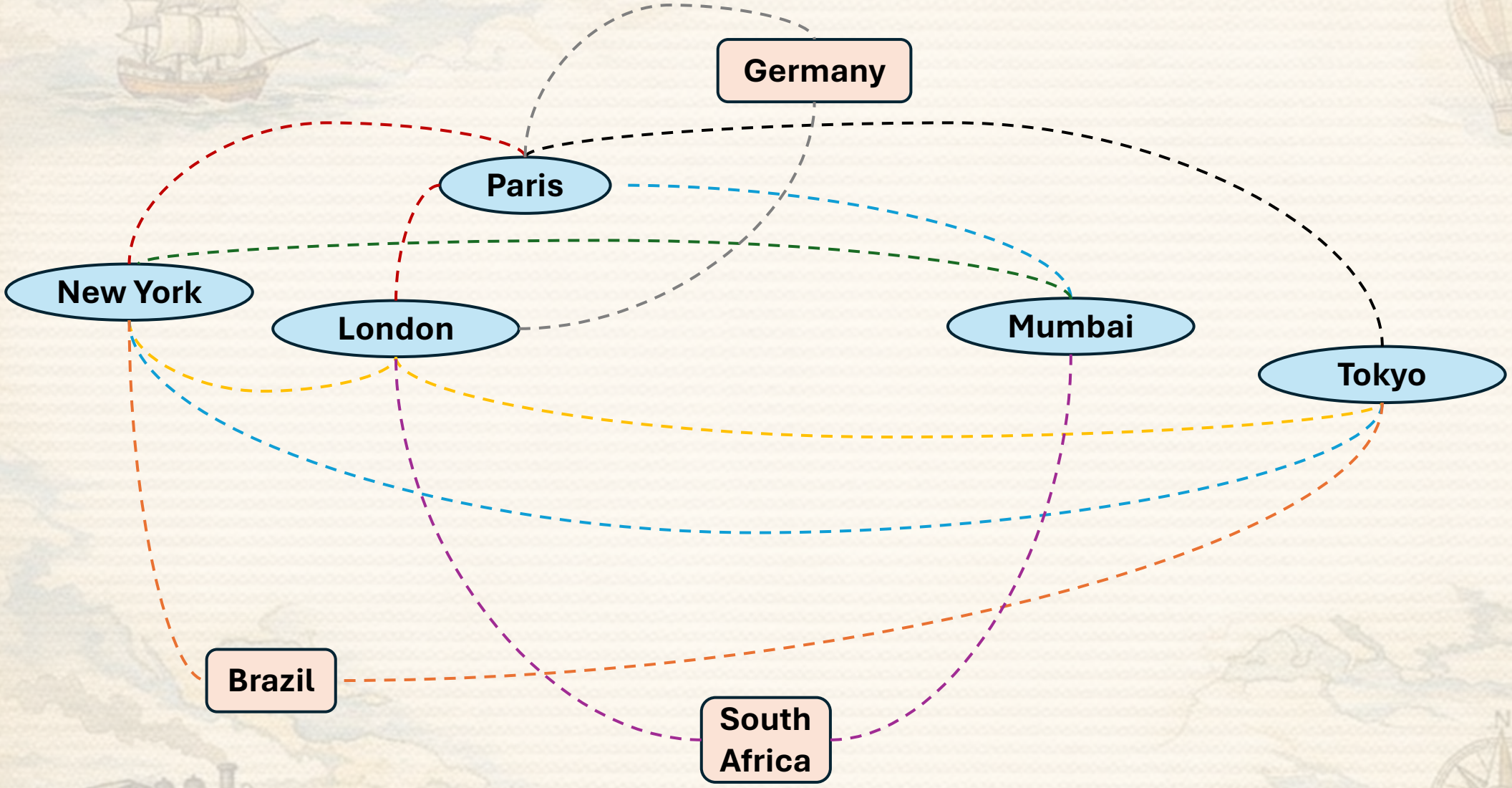

Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Table: cities

```
CREATE TABLE cities(
  city_id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

INSERT INTO cities (city_id, name) VALUES
  (1, 'Mumbai'),
  (2, 'London'),
  (3, 'Paris'),
  (4, 'New York'),
  (5, 'Tokyo');
```

Table: countries

```
CREATE TABLE countries(
  country_id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

INSERT INTO countries (country_id, name) VALUES
  (101, 'Germany'),
  (102, 'Brazil'),
  (103, 'South Africa');
```

Table: city_routes

```
CREATE TABLE city_routes(
  route_id INT PRIMARY KEY,
  city_id_1 INT REFERENCES cities(city_id),
  city_id_2 INT REFERENCES cities(city_id),
  length INT,
  color VARCHAR(100),
  points INT,
  claimed_by VARCHAR(100));

INSERT INTO city_routes VALUES
  (1001, 1, 3, 7, 'blue', 9, NULL), -- Mumbai ⇔ Paris,
  (1002, 3, 2, 1, 'red', 1, NULL), -- Paris ⇔ London,
  (1003, 1, 4, 13, 'green', 16, NULL), -- Mumbai ⇔ New York,
  (1004, 4, 2, 6, 'yellow', 8, NULL), -- New York ⇔ London,
  (1005, 3, 5, 10, 'black', 12, NULL), -- Paris ⇔ Tokyo,
  (1006, 5, 2, 10, 'white', 12, NULL), -- Tokyo ⇔ London,
  (1007, 3, 4, 6, 'red', 8, NULL), -- Paris ⇔ New York,
  (1008, 4, 5, 11, 'blue', 14, NULL), -- New York ⇔ Tokyo;
```

Table: country_city_routes

```
CREATE TABLE country_city_routes(
  route_id INT PRIMARY KEY,
  country_id INT REFERENCES countries(country_id),
  city_id INT REFERENCES cities(city_id),
  length INT,
  color VARCHAR(100),
  points INT,
  claimed_by VARCHAR(100)
);

INSERT INTO country_city_routes VALUES
  (2001, 101, 3, 1, 'gray', 2, NULL), -- Germany ⇔ Paris,
  (2002, 101, 2, 1, 'gray', 2, NULL), -- Germany ⇔ London,
  (2003, 102, 4, 8, 'orange', 12, NULL), -- Brazil ⇔ New York,
  (2004, 102, 5, 18, 'orange', 24, NULL), -- Brazil ⇔ Tokyo,
  (2005, 103, 1, 7, 'purple', 11, NULL), -- South Africa ⇔ Mumbai,
  (2006, 103, 2, 9, 'purple', 13, NULL), -- South Africa ⇔ London;
```

Property graph: Name

```
CREATE PROPERTY GRAPH ttr_graph
   VERTEX TABLES (
      cities
         KEY (city_id)
         LABEL City PROPERTIES (name),
      countries
         KEY (country_id)
         LABEL Country PROPERTIES (name))
   EDGE TABLES (…);
```

Property graph: Vertex tables

```
CREATE PROPERTY GRAPH ttr_graph
    VERTEX TABLES (
        cities
            KEY (city_id)
            LABEL City PROPERTIES (name),
        countries
            KEY (country_id)
            LABEL Country PROPERTIES (name))
    EDGE TABLES (…);
```

Property graph: Keys

```
CREATE PROPERTY GRAPH ttr_graph
   VERTEX TABLES (
      cities
         KEY (city_id)
         LABEL City PROPERTIES (name),
      countries
         KEY (country_id)
         LABEL Country PROPERTIES (name))
   EDGE TABLES (…);
```

Property graph: Labels

```
CREATE PROPERTY GRAPH ttr_graph
   VERTEX TABLES (
      cities
         KEY (city_id)
         LABEL City PROPERTIES (name),
      countries
         KEY (country_id)
         LABEL Country PROPERTIES (name))
   EDGE TABLES (…);
```

Property graph: Properties

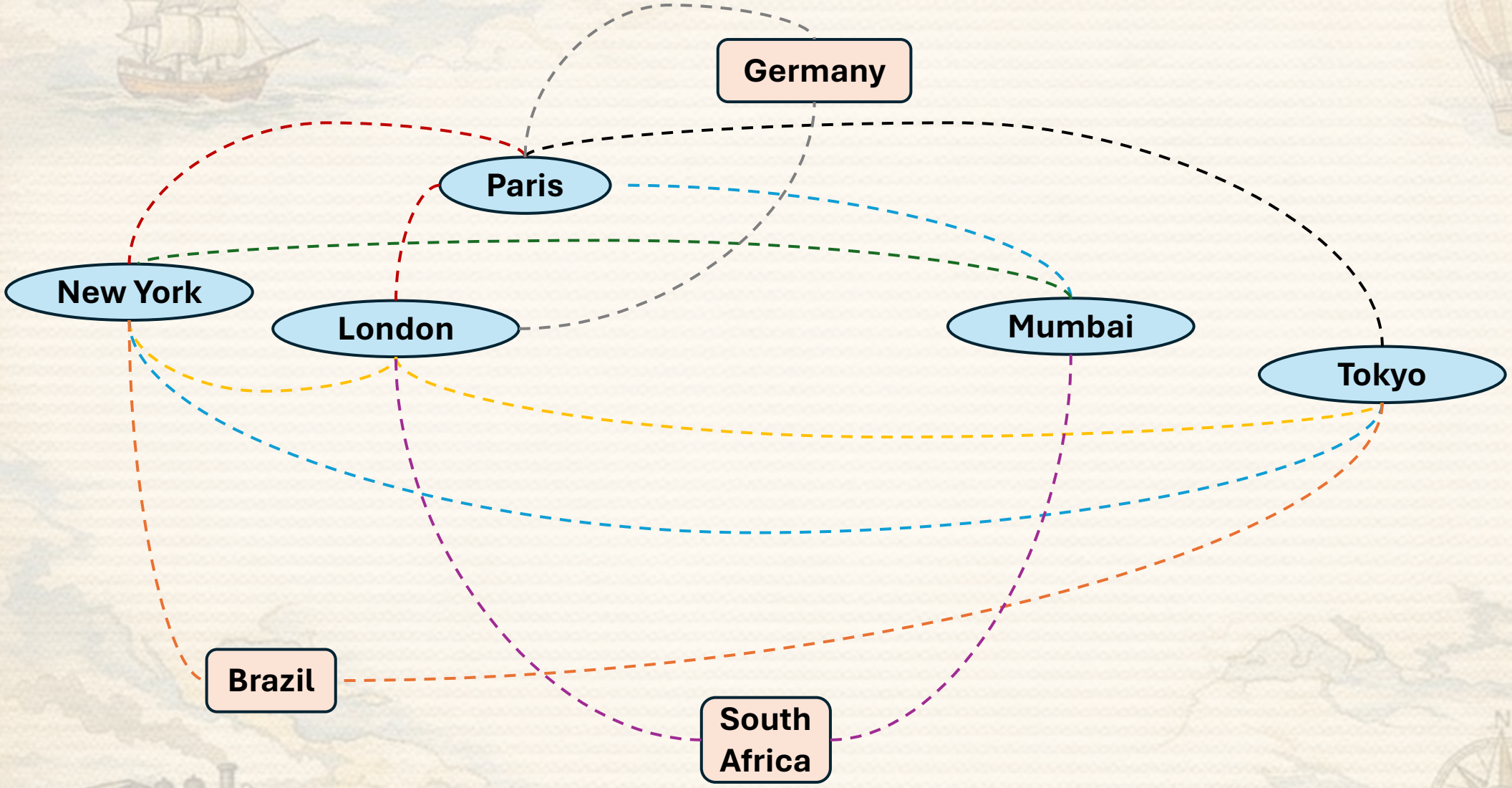
```
CREATE PROPERTY GRAPH ttr_graph
   VERTEX TABLES (
      cities
         KEY (city_id)
         LABEL City PROPERTIES (name),
      countries
         KEY (country_id)
         LABEL Country PROPERTIES (name))
   EDGE TABLES (…);
```

Property graph: End points of Edges

```
CREATE PROPERTY GRAPH ttr_graph
    VERTEX TABLES (…)
    EDGE TABLES (
        city_routes
        KEY (route_id)
        SOURCE KEY (city_id_1) REFERENCES cities(city_id)
        DESTINATION KEY (city_id_2) REFERENCES cities (city_id)
        LABEL Route PROPERTIES (route_id,length,color,points,claimed_by),
        country_city_routes
        KEY (route_id)
        SOURCE countries
        DESTINATION cities
        LABEL Route PROPERTIES (route_id,length,color,points,claimed_by));
```

Property graph: Shared labels

```
CREATE PROPERTY GRAPH ttr_graph
    VERTEX TABLES (…)
    EDGE TABLES (
        city_routes
        KEY (route_id)
        SOURCE KEY (city_id_1)
        DESTINATION KEY (city_id_2)
        LABEL Route PROPERTIES (route_id,length,color,points,claimed_by),
        country_city_routes
        KEY (route_id)
        SOURCE countries
        DESTINATION cities
        LABEL Route PROPERTIES (route_id,length,color,points,claimed_by));
```


Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Reading colors

Implemented	green
Patch proposed	blue
Under design	brown
Standard, not started	red

Simple one-stop route

```
SELECT *
FROM GRAPH_TABLE (
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[r1 IS Route]-(conn IS City)-[r2 IS Route]-
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    a.name AS city1,
    conn.name AS connection,
    b.name AS city2,
    r1.length + r2.length AS total_length,
    r1.points + r2.points AS total_points
  )
);
```

Simple one-stop route: Vertex patterns

```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[r1 IS Route]-(conn IS City)-[r2 IS Route]-
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    a.name AS city1,
    conn.name AS connection,
    b.name AS city2,
    r1.length + r2.length AS total_length,
    r1.points + r2.points AS total_points
  )
);
```

Simple one-stop route: Edge patterns

```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[r1 IS Route]-(conn IS City)-[r2 IS Route]-
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    a.name AS city1,
    conn.name AS connection,
    b.name AS city2,
    r1.length + r2.length AS total_length,
    r1.points + r2.points AS total_points
  )
);
```

Simple one-stop route: Shape of result

```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[r1 IS Route]-(conn IS City)-[r2 IS Route]-
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    a.name AS city1,
    conn.name AS connection,
    b.name AS city2,
    r1.length + r2.length AS total_length,
    r1.points + r2.points AS total_points
  )
);
```

Behind the scenes

Element patterns -> element tables

> Label expression -> Set of labels
>
> Set of labels -> {set of properties}, {set of vertex/edge tables}

Path pattern -> JOIN + UNION

Property references -> table column references

Behind the scenes

Element patterns -> element tables

Path pattern -> JOIN + UNION

- Path pattern -> sequence of element patterns
- Element pattern -> set of vertex/edge tables
- Depth first-search through K-partite graph, where K = no of vertex/edge tables
- Each path -> sequence of joins between vertex/edge tables
- Set of paths -> UNION of JOINs

Property references -> table column references

Behind the scenes

Element patterns -> element tables

Path pattern -> UNION + JOIN

Path pattern -> sequence of element patterns

Element pattern -> set of vertex/edge tables

Depth first-search through K-partite graph

Each path -> sequence of joins between vertex/edge tables

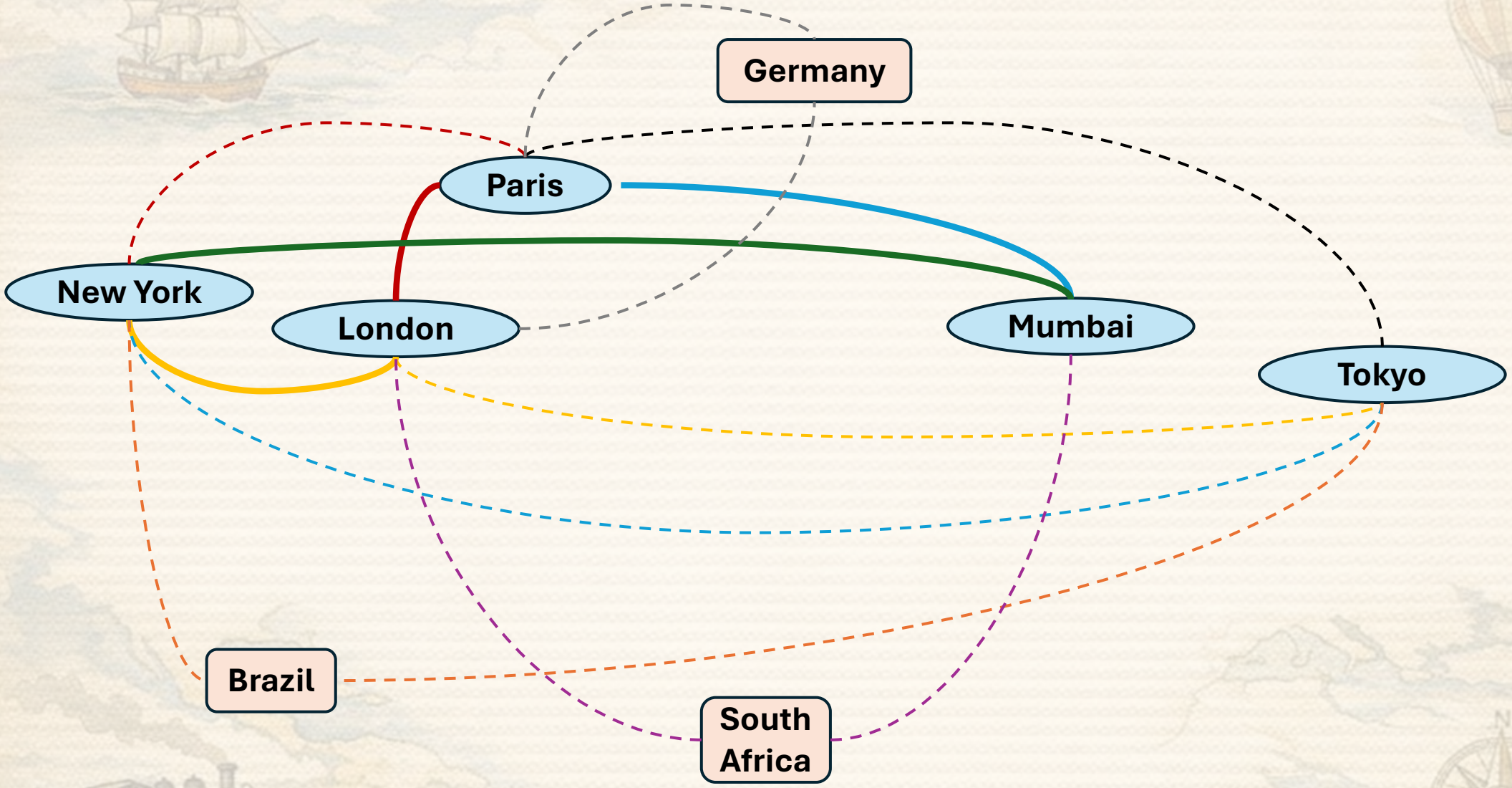
Set of paths -> UNION of JOINs

Property references -> table column references

Simple one-stop route: SQL

```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[r1 IS Route]-
    (conn IS City)
    -[r2 IS Route]-
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    a.name AS city1,
    conn.name AS connection,
    b.name AS city2,
    r1.length + r2.length AS total_length,
    r1.points + r2.points AS total_points
  )
);
```

```
SELECT DISTINCT
  a.name AS city1,
  b.name AS connection,
  c.name AS city2,
  r1.length + r2.length AS total_length,
  r1.points + r2.points AS total_points
FROM cities a
JOIN city_routes r1
  ON a.city_id IN (r1.city_id_1, r1.city_id_2)
JOIN cities conn
  ON conn.city_id IN (r1.city_id_1, r1.city_id_2)
 AND conn.city_id <> a.city_id
JOIN city_routes r2
  ON conn.city_id IN (r2.city_id_1, r2.city_id_2)
JOIN cities b
  ON b.city_id IN (r2.city_id_1, r2.city_id_2)
 AND b.city_id <> conn.city_id
WHERE a.name = 'Mumbai'
  AND b.name = 'London';
```


Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Label disjunction

```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[ IS Route]-
    (b IS City | Country)
    -[ IS Route]-
    (c IS City WHERE c.name = 'London')
  COLUMNS (a.name AS src, b.name AS via, c.name AS dst)
);
```

Shared Route label

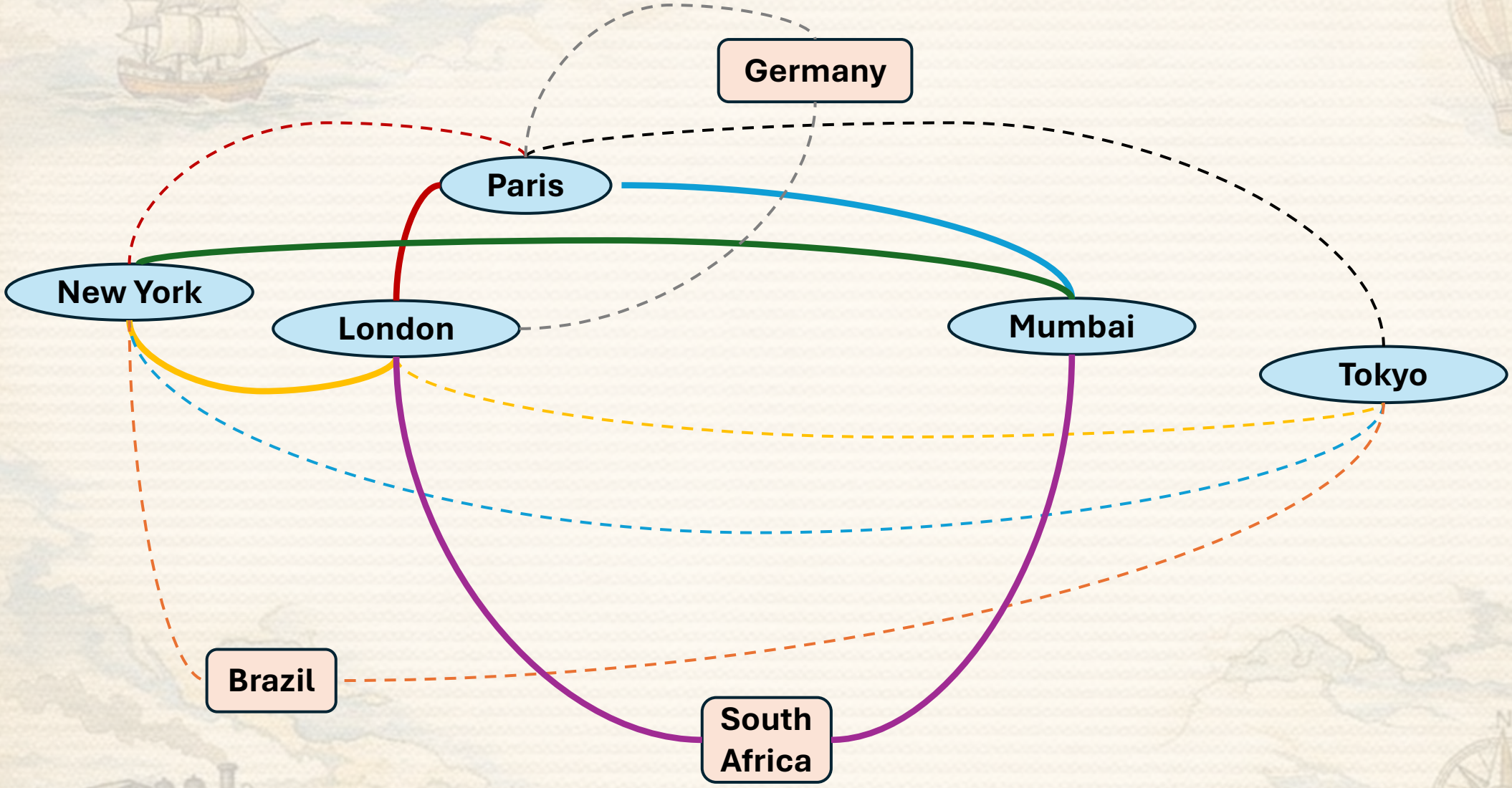
```
SELECT *
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    (a IS City WHERE a.name = 'Mumbai')
    -[ IS Route]-
    (b IS City | Country)
    -[ IS Route]-
    (c IS City WHERE c.name = 'London')
  COLUMNS (a.name AS src, b.name AS via, c.name AS dst)
);
```

Label disjunction, shared labels: SQL

```
-- Via City
SELECT DISTINCT a.name AS src, b.name AS via, c.name AS dst
FROM cities a
JOIN city_routes r1 ON a.city_id IN (r1.city_id_1, r1.city_id_2)
JOIN cities b       ON b.city_id IN (r1.city_id_1, r1.city_id_2) AND b.city_id <> a.city_id
JOIN city_routes r2 ON b.city_id IN (r2.city_id_1, r2.city_id_2)
JOIN cities c       ON c.city_id IN (r2.city_id_1, r2.city_id_2) AND c.city_id <> b.city_id
WHERE a.name='Mumbai' AND c.name='London'

UNION

-- Via Country
SELECT DISTINCT a.name AS src, k.name AS via, c.name AS dst
FROM cities a
JOIN country_city_routes r1 ON a.city_id = r1.city_id
JOIN countries b            ON b.country_id = r1.country_id
JOIN country_city_routes r2 ON b.country_id = r2.country_id
JOIN cities c               ON c.city_id = r2.city_id
WHERE a.name='Mumbai' AND c.name='London';
```


Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Variable length paths: quantifier

```
SELECT path, total_length, total_points
FROM GRAPH_TABLE(
  ttr_graph
  MATCH ACYCLIC
    (a IS City WHERE a.name = 'Mumbai')
    -[r IS Route]-{1,5}
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    array_agg(r.route_id) AS path,
    sum(r.length) AS total_length,
    sum(r.points) AS total_points
  )
);
```

Variable length paths: path mode

```
SELECT path, total_length, total_points
FROM GRAPH_TABLE(
  ttr_graph
  MATCH ACYCLIC
    (a IS City WHERE a.name = 'Mumbai')
    -[r IS Route]-{1,5}
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    array_agg(r.route_id) AS path,
    sum(r.length) AS total_length,
    sum(r.points) AS total_points
  )
);
```

Variable length paths: Aggregates

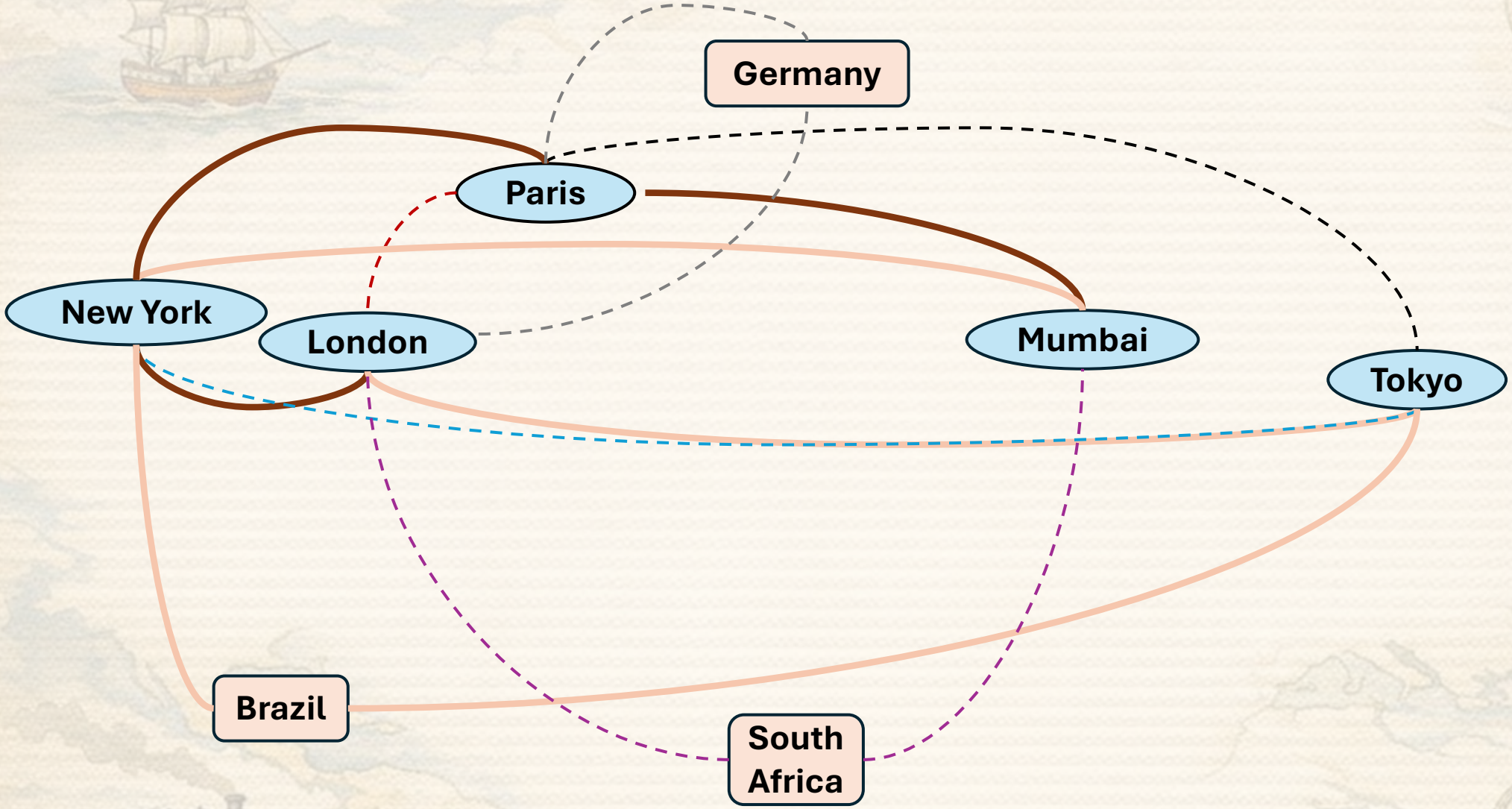
```
SELECT path, total_length, total_points
FROM GRAPH_TABLE(
  ttr_graph
  MATCH ACYCLIC
    (a IS City WHERE a.name = 'Mumbai')
    -[r IS Route]-{1,5}
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    array_agg(r.route_id) AS path,
    sum(r.length) AS total_length,
    sum(r.points) AS total_points
  )
);
```

Variable length paths: SQL

```
WITH walk(city_id, route_ids, total_length, total_points, depth) AS (
  SELECT c.city_id, ARRAY[]::INT[] AS route_ids, 0, 0, 0
  FROM cities c
  WHERE c.name='Mumbai'

  UNION ALL

  SELECT
    CASE WHEN r.id1 = w.city_id THEN r.id2 ELSE r.id1 END,
    array_append(w.route_ids, r.route_id),
    w.total_length + r.length,
    w.total_points + r.points,
    w.depth + 1
  FROM walk w
  JOIN (
    SELECT city_id_1 AS id1, city_id_2 AS id2, route_id, length, points FROM city_routes
    UNION ALL
    SELECT country_id AS id1, city_id AS id2, route_id, length, points FROM country_city_routes
  ) r ON w.city_id IN (r.id1, r.id2)
  WHERE w.depth < 5
)
CYCLE city_id SET is_cycle TO 1 DEFAULT 0
SELECT w.route_ids AS path, w.total_length, w.total_points
FROM walk w
JOIN cities c ON c.city_id = w.city_id
WHERE c.name='London' AND w.is_cycle=0
ORDER BY total_points DESC, total_length DESC
FETCH FIRST 5 ROWS ONLY;
```


Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Multiple tickets: Multiple path patterns

```
SELECT path1, path2
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    ACYCLIC (a1 IS City WHERE a1.name = 'Mumbai')-[r1 IS Route]-{1,4}(b1
IS City WHERE b1.name = 'London'),
    ACYCLIC (a2 IS City WHERE a2.name = 'Paris') -[r2 IS Route]-{1,4}(b2
IS City WHERE b2.name = 'Tokyo')
  COLUMNS (
    array_agg(r1.route_id) AS path1,
    array_agg(r2.route_id) AS path2
  )
);
```

Multiple tickets: Multiple path patterns

```
SELECT path1, path2
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    ACYCLIC (a1 IS City WHERE a1.name = 'Mumbai')-[r1 IS Route]-{1,4}(b1
IS City WHERE b1.name = 'London'),
    ACYCLIC (a2 IS City WHERE a2.name = 'Paris') -[r2 IS Route]-{1,4}(b2
IS City WHERE b2.name = 'Tokyo')
  COLUMNS (
    array_agg(r1.route_id) AS path1,
    array_agg(r2.route_id) AS path2
  )
);
```

Multiple path patterns: path mode

```
SELECT path1, path2
FROM GRAPH_TABLE(
  ttr_graph
  MATCH
    ACYCLIC (a1 IS City WHERE a1.name = 'Mumbai')-[r1 IS Route]-{1,4}(b1
IS City WHERE b1.name = 'London'),
    ACYCLIC (a2 IS City WHERE a2.name = 'Paris') -[r2 IS Route]-{1,4}(b2
IS City WHERE b2.name = 'Tokyo')
  COLUMNS (
    array_agg(r1.route_id) AS path1,
    array_agg(r2.route_id) AS path2
  )
);
```

Multiple path patterns: SQL

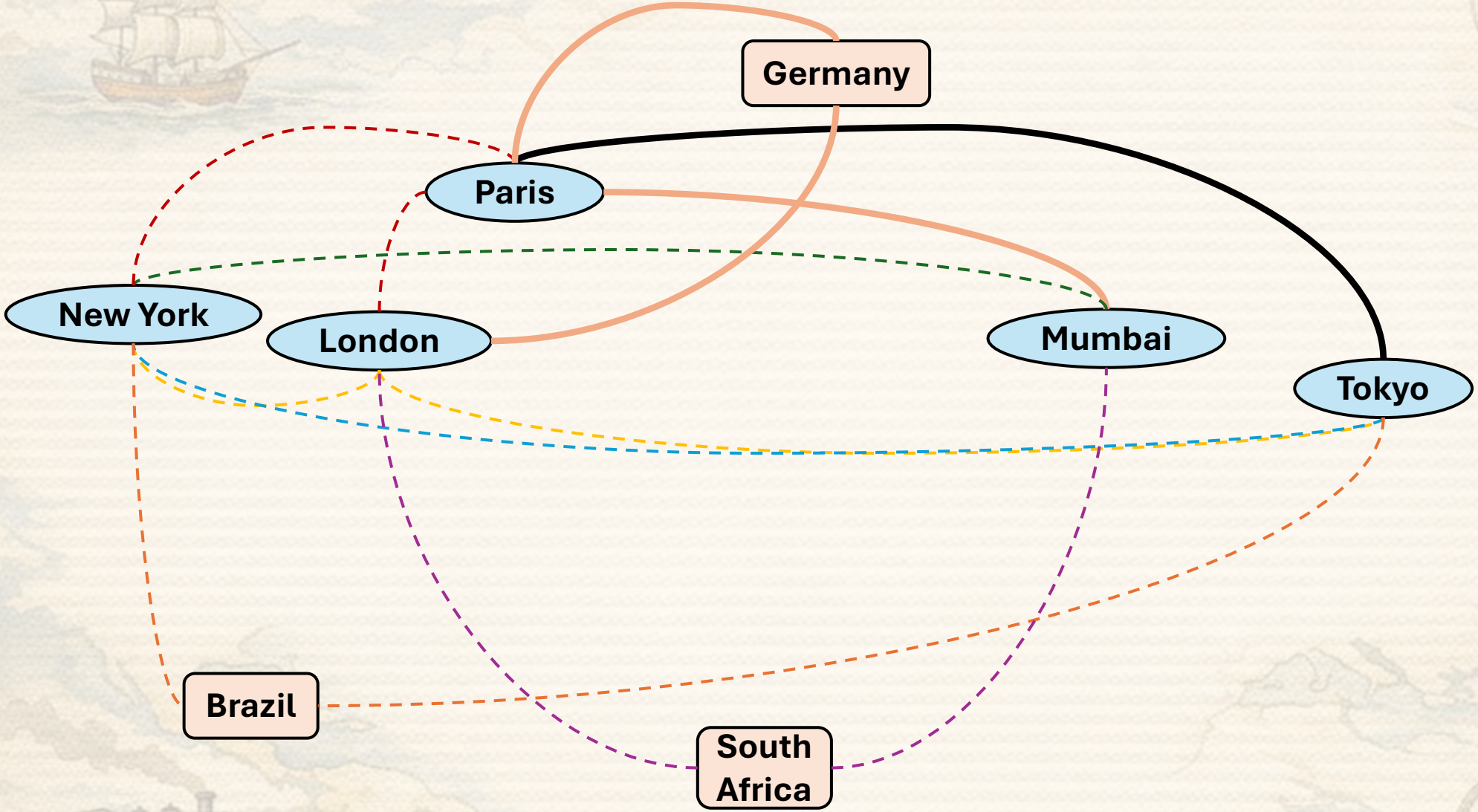
```
SELECT routes1.route_ids AS path1, routes2.route_ids AS path2
FROM
(WITH walk(city_id, route_ids) AS (
  SELECT city_id, NULL
  FROM cities
  WHERE name = 'Mumbai'

  UNION ALL

  SELECT
    CASE WHEN r.id1 = w.city_id THEN r.id2 ELSE r.id1 END,
    w.route_ids || array[r.route_id]
  FROM walk w
  JOIN (SELECT city_id_1 AS id1, city_id_2 AS id2, route_id, length, points FROM city_routes
        UNION
        SELECT country_id AS id1, city_id AS id2, route_id, length, points FROM country_city_routes) r ON w.city_id IN (r.id1, r.id2)
  WHERE array_length(w.route_ids, 1) < 4
)
CYCLE city_id SET is_cycle TO 1 DEFAULT 0
SELECT w.route_ids AS route_ids
FROM walk w
JOIN cities c ON c.city_id = w.city_id
WHERE c.name = 'London'
  AND w.is_cycle = 0) routes1,
(WITH walk(city_id, route_ids) AS (
  SELECT city_id, NULL
  FROM cities
  WHERE name = 'Paris'

  UNION ALL

  SELECT
    CASE WHEN r.id1 = w.city_id THEN r.id2 ELSE r.id1 END,
    w.route_ids || array[r.route_id]
  FROM walk w
  JOIN (SELECT city_id_1 AS id1, city_id_2 AS id2, route_id, length, points FROM city_routes
        UNION
        SELECT country_id AS id1, city_id AS id2, route_id, length, points FROM country_city_routes) r ON w.city_id IN (r.id1, r.id2)
  WHERE array_length(w.route_ids, 1) < 4
)
CYCLE city_id SET is_cycle TO 1 DEFAULT 0
SELECT w.route_ids AS route_ids
FROM walk w
JOIN cities c ON c.city_id = w.city_id
WHERE c.name = 'Tokyo'
  AND w.is_cycle = 0) routes2
```


Germany
Paris
New York
London
Mumbai
Tokyo
Brazil
South Africa

Quiz

Find whether Team-A has claimed path connecting Mumbai and London

```
SELECT path, total_length, total_points
FROM GRAPH_TABLE(
  ttr_graph
  MATCH ACYCLIC
    (a IS City WHERE a.name = 'Mumbai')
    -[r IS Route]-{1,5}
    (b IS City WHERE b.name = 'London')
  COLUMNS (
    array_agg(r.route_id) AS path,
    sum(r.length) AS total_length,
    sum(r.points) AS total_points
  )
);
```

SQL/PGQ is not

A native graph database

Graph database storage

SQL/PGQ and Apache AGE

SQL/PGQ

In-core graph query capabilities

Focus
- SQL/PGQ Features
- Optimizations

Precedence: Partitioning

Apache/AGE

Cypher-like syntax

Focus
- Portability, migrations
- Cypher -> SQL/PGQ, SQL

Precendence: pg_partman

Quiz: Answer

```
SELECT count(claimed) > 0 AS has_claimed_path
FROM GRAPH_TABLE(
  ttr_graph
  MATCH ACYCLIC
    (a IS City WHERE a.name = 'Mumbai')
    -[r IS Route WHERE r.claimed_by = 'Team-A']-{1,5}
    (b IS City WHERE b.name = 'London')
  COLUMNS (1 AS claimed)
);
```

Thank you

Questions