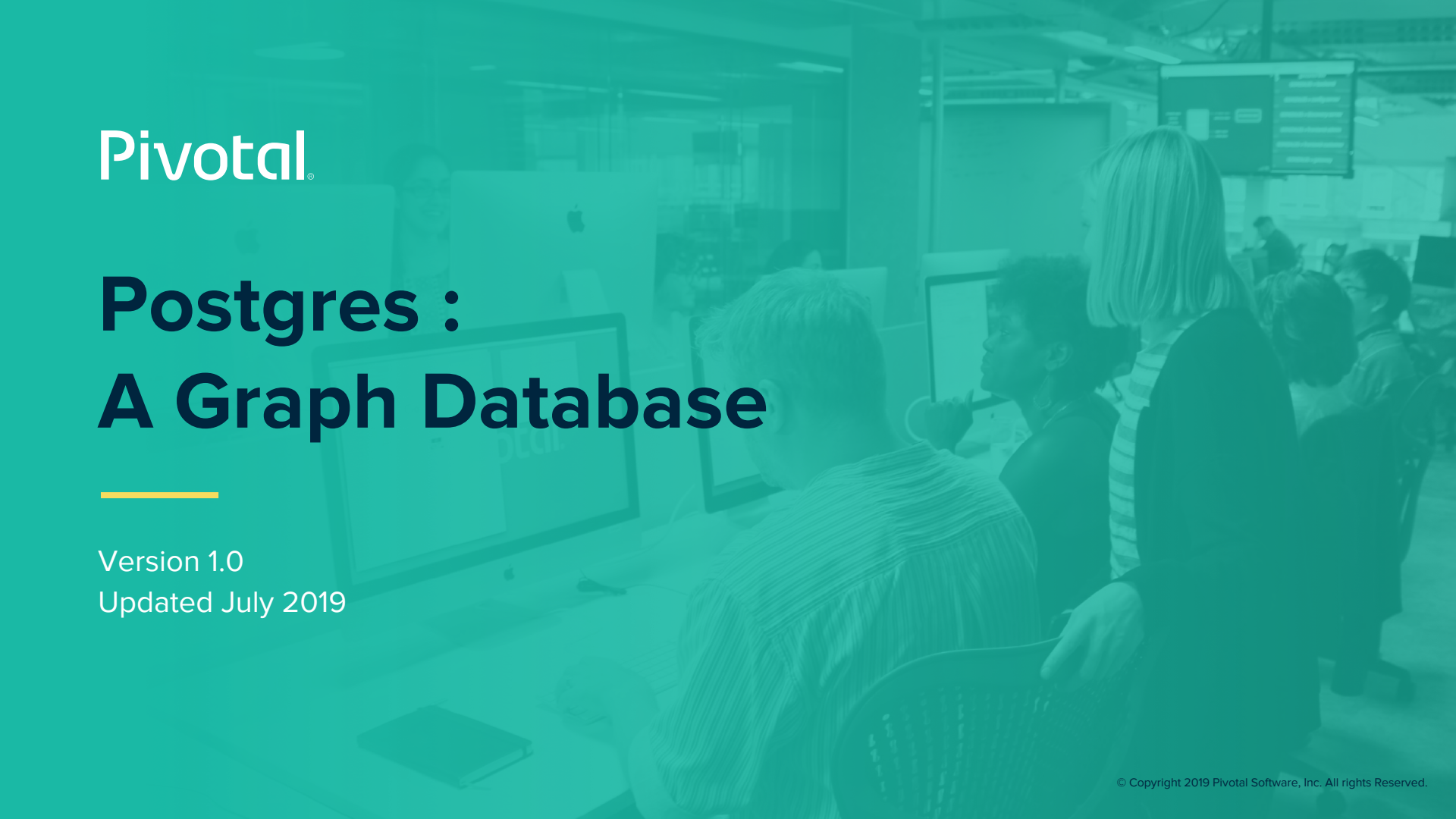


Pivotal

Postgres : A Graph Database

Version 1.0
Updated July 2019

Greg Spiegelberg

Pivotal
Senior Account Data Engineer
(aka sales)

- Noodling around with *gres since Ingres
- SysAdmin, DBA, Engineer
- System / Software / Storage / Data Architect
- Husband & Dad
- Lego enthusiast
- Ski bum

Agenda

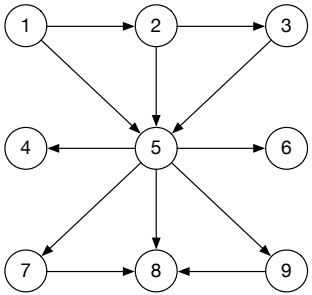
- Graphs Overview (why & why not)
- Postgres Solution
- Application Highlight

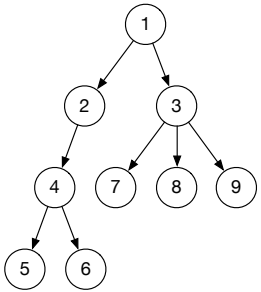
Overview

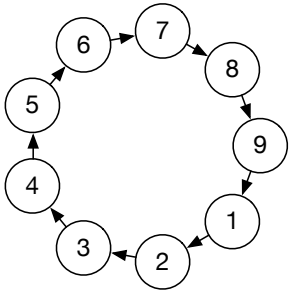
Graphs

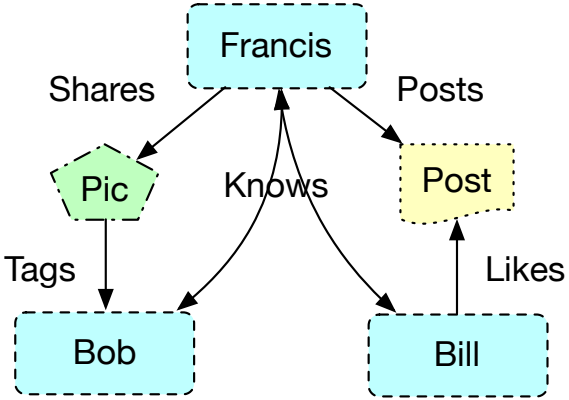
One or many of same *entity type* are related

- Person has friend(s)
- Person shares pictures & posts
- Person likes picture / posts
- Person tagged in picture









Schema-free, loose typing and all entities treated equally

Technologies

NoSQL

- Neo4j
- ArangoDB
- AgensGraph (Postgres based)
- OrientDB
- RedisGraph
- Amazon Neptune
- Azure Cosmos DB

Be aware of:

- Scaling limited or unproven
- Some mixed reviews and stability questionable
- API's are not standard
- SQL-like, Cypher & custom languages
- Limitations of secondary indexes
- Consistency : Eventual
- Many better used as a document database
- Don't be pulled in by data UX

Understand the problem you are trying to solve.

When To Use Graph Databases

- Data is not well defined or at all
- Relationships and attributes are evolving
- Closed loops and identification of are required

When To Not Use Graph Databases

- Data and relationships are well defined and understood
- Relationships do not exist (is it analytic?)
- High value use case involves:
  - Bulk scans
  - Key-value store
  - No start or end point in queries
- Text or BLOBS are used as edge attributes
- You were won over by a wiz-bang visualization

Postgres Solution

Relational Database

Different entities connected via foreign keys

- One to One – Organizations has Parent

```
orgs o JOIN orgs p ON o.id = p.parent_id
```

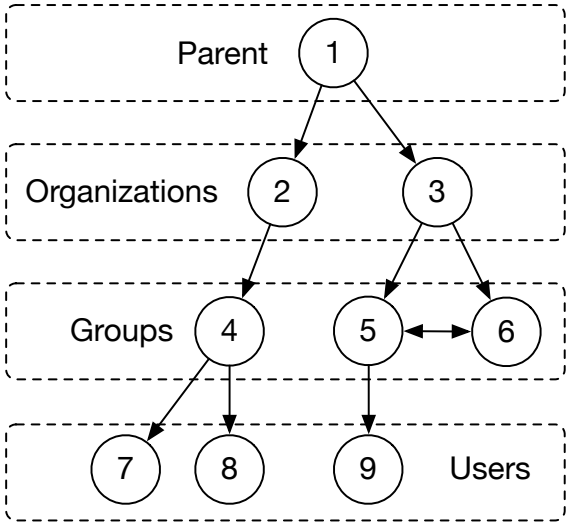
- One to Many – User belongs to Groups

```
users u JOIN groups_users g ON u.id = g.user_id
```

- Many to Many – Groups belong to Groups

```
groups_groups g1 JOIN groups_groups g2 ON g1.g1_id = g2.g2_id
```

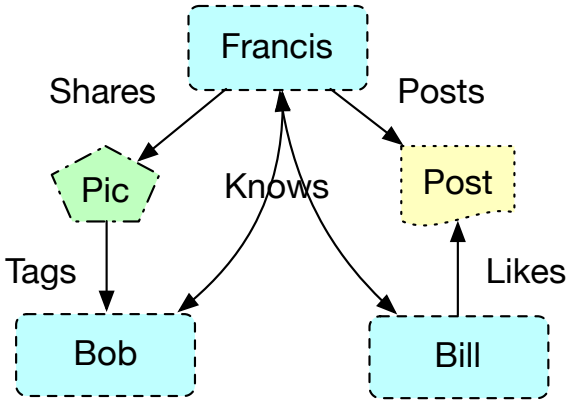
Is not Many:Many the requirement leading to a graph db?



Graph in SQL

```
CREATE SEQUENCE entities_id_seq;

CREATE TABLE entities(
  id       int8 PRIMARY KEY DEFAULT nextval('entities_id_seq'),
  created  timestamp DEFAULT now(),
  name     varchar(32) NOT NULL,
  stuff    varchar(32)
);

CREATE SEQUENCE relationships_id_seq;

CREATE TABLE relationships(
  id        int8 PRIMARY KEY DEFAULT nextval('relationships_id_seq'),
  created   timestamp DEFAULT now(),
  entity_a  int8 REFERENCES entities(id)
                ON UPDATE CASCADE ON DELETE CASCADE,
  entity_b  int8 REFERENCES entities(id)
                ON UPDATE CASCADE ON DELETE CASCADE,
  reltype   varchar(32) NOT NULL
);
```

```
CREATE VIEW nodes AS
SELECT * FROM entities;

CREATE VIEW edges AS
SELECT * FROM relationships;
```



Recursion

WITH RECURSIVE

- Introduced in PostgreSQL 8.4 circa 2009
- Available in Greenplum 5 (beta) & 6 (prod)
- Provides ability to
  - Implement trees or other hierarchical forms
  - Path enumeration
  - Find shortest path (Traveling Salesman)
  - Find paths based upon other criteria
  - And many others

```
WITH RECURSIVE path
(id, name, parent_id, depth) AS (
-- Seed / root / parent
SELECT t.id, t.name, t.parent_id,
       1 AS depth
  FROM tree t
 WHERE t.id = START

  UNION ALL

-- Children of parent
SELECT p.id, p.name, p.parent_id,
       p.depth + 1 AS depth
  FROM path p, tree t
 WHERE p.id = t.parent_id
)
SELECT * FROM path
 ORDER BY id
```

Implementation Considerations

- Graph type
  - Tree / directional
  - Are cycles permissible
- Effective dating giving a historical perspective
  - Imagine a 3D tree where a slice is defined by a time range and include links between slices
- Use weights, measures and other attributes to describe edges and use in queries
  - Boston to New York is 215 miles by I-90
  - Bike path is paved, packed dirt, or gravel

Common Graph Traversals & Tools

- Very general traversal can be time consuming, implement safeguards
- Shortest Path (SP)
- Single Source Shortest Path (SSSP)
- Hyperlink Induced Topic Search (HITS)
- Page Rank
- Weakly Connected Components
- Measures & weights of relationship

- Apache MADlib, PostGIS, pgRouting, Boundless Desktop
    - MADlib great but can be expensive (All-Pairs-Shortest-Path)

Implementation Considerations - Relationships

- Edge / Relationship Types (likes, friends, tagged, connection, ...)
  - Embed as text
  - Create an ENUM
  - Separate lookup table for relationship type (FOREIGN KEY)
  - Separate table / partition for each type of relationship

```
CREATE TABLE relationships(
  id        int8 PRIMARY KEY DEFAULT nextval('relationships_id_seq'),
  created   timestamp DEFAULT now(),
  entity_a  int8 REFERENCES entities(id)
                ON UPDATE CASCADE ON DELETE CASCADE,
  entity_b  int8 REFERENCES entities(id)
                ON UPDATE CASCADE ON DELETE CASCADE,
  reltype   varchar(32) NOT NULL
);
```

Query Considerations

- Multi-directional edges
  - May be tempted to (entity_a OR entity_b)
  - Recommend directional over bi- or multi-
- Watch for cycles (endless recursion)
- Very deep graphs (many, many 100,000's of edges in possible results), YMMV
- Always include starting point and destination in query
- Duplicate relationships, use UNIQUE(id1, id2)
- Implement safeguards in data creation and queries!

```
CREATE OR REPLACE FUNCTION find_path_depth_distance(
          start int, destination int, max_depth int
) RETURNS TABLE(depth int, distance int, path integer[]) AS $$
BEGIN
RETURN QUERY
WITH RECURSIVE search_path(id, link, depth, distance, route, cycle) AS (
          SELECT p.loc1_id AS id, p.loc2_id AS link, 1 AS depth,
                 p.distance AS distance, ARRAY[p.loc1_id] AS route, false AS cycle
            FROM paths p
           WHERE p.loc1_id = start AND p.active
UNION ALL
          SELECT p.loc1_id AS id, p.loc2_id AS link, sp.depth + 1 AS depth,
                 p.distance + sp.distance AS distance, route || p.loc1_id AS route,
                 p.loc1_id = ANY(route) AS cycle
            FROM paths p, search_path sp
           WHERE p.loc1_id = sp.link AND p.active AND NOT cycle
             AND sp.depth + 1 <= max_depth
)
SELECT sp.depth, sp.distance, (sp.route || sp.link) AS route
            FROM search_path AS sp
           WHERE link = destination
             AND NOT cycle
           ORDER BY depth ASC, distance ASC;
END;
$$ LANGUAGE 'plpgsql';
```

Query Optimizations

- Index on edge / relationship id columns
- Limit number of relationship (edge) types and index if necessary
    - If types stored in static lookup table, CLUSTER
- Index on primary filter predicates (duh)
    - Spatial index (GIST), range limiting (BRIN), pg_trgm (USING GIST(txt gist_trgm_ops))
        - pg_trgm GUCs similarity_threshold, word_similarity_threshold, strict_word_similarity_threshold
- PREPARE the statement if pattern used repeatedly
- Use functions for well known traversal patterns
- Consider a materialized view if query is often repeated on subset of data

CATMAID

https://catmaid.readthedocs.io/en/stable/

Application Highlight

HHMI Janelia Research Campus

- PostgreSQL journey started in 2009 with smaller data sets
- PostGIS added as data grew and faster and more correct intersections in 3D from different orthogonal perspectives
- Today, heavily reliant on PostgreSQL 11 and PostGIS 2.5
- Spatial (geom), z-Range (gist) and pg_trm indexes modeling neurons by directed interconnected nodes where each nodes references its parent (trees)
- Full adult fruit fly brain resulting in ~780 GB database with ~800,000,000 edges
    - Sum of all edges amount to 190 meters... remember they're dealing in nanometers

Sources:
Tom Kazimiers, Senior Software Engineer, HHMI Janelia Research Campus
https://ai.googleblog.com/2019/08/an-interactive-automated-3d.html

HHMI : Neuron Reconstruction Over Time

Source:
https://twitter.com/tomkazimiers/media

HHMI : Adult Fruit Fly

Source:
https://catmaid.virtualflybrain.org/

Pivotal

Transforming How The World Builds Software And Software Needs Data