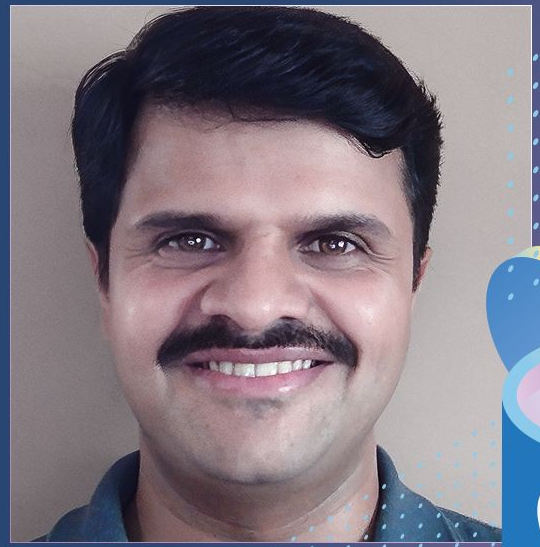
Microsoft | PGConf India

PGConf India, 2025

Graph databases, PostgreSQL and SQL/PGQ



Ashutosh Bapat

Thu 6 Mar | 14:00 IST

Agenda

Graph Databases

SQL/PGQ

PostgreSQL and SQL/PGQ

Introduction to Graph Databases

Graph databases

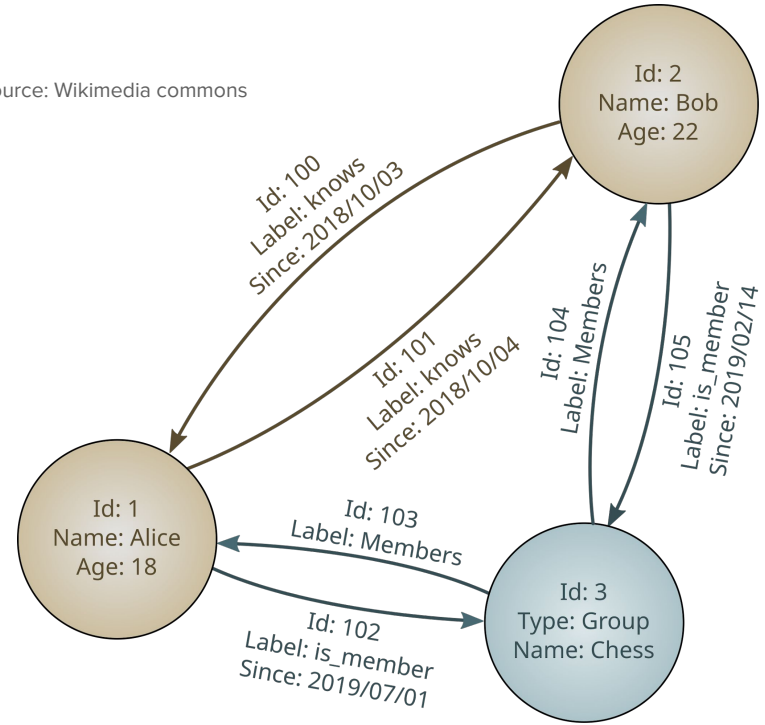
Definition: A graph database uses graph structures with nodes, edges, and properties to represent and store data.

Key Concepts:

- **Nodes**
  - Represent Entities
  - Examples: people, products, locations
- **Edges**
  - Connect nodes
  - Represent the Relationships between them
  - Examples: "knows," "purchased," "located_in"
- **Properties**
  - Attributes of nodes and edges
  - Examples: name, age, price

Optionally nodes and edges may be labelled for classification

Source: Wikimedia commons



Graph database uses

Social Networks

Manage complex relationships within social networks, mine insights

Recommendation Systems

Analyze user preferences and item relationships to suggest relevant content.

Fraud Detection

Uncover hidden patterns and relationships in transaction data

Logistics and supply chain management

Map entire chain from suppliers to transportation routes efficiently

find bottlenecks, optimization opportunities

Example usage: Fraudulent transaction detection

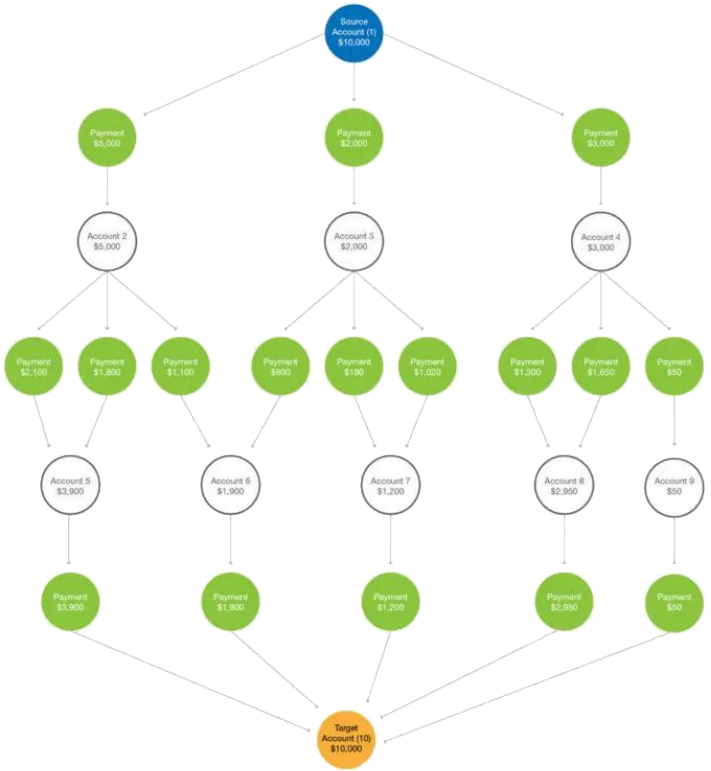
Suspicious transactions

- Across multiple accounts
- Across many banks
- Across different types of instruments
- Held by seemingly unrelated individuals
- Movement of funds

```
MATCH
path=(a:Account)-[:TRANSACTION*3..6]->(b:Account)

RETURN path
```

Source: https://www.graphable.ai/blog/graph-database-fraud-detection/



SQL/PGQ: SQL for Property Graphs

PGQ (Property Graph Queries)

Purpose

- Use SQL to treat relational database as a graph database

Extension to SQL

- PGQ adds graph query capabilities to SQL

Standard

- ISO/IEC 9075-16

Property Graph

Map relational tables to set of nodes or edges in a graph

Classify by labels

Expose columns as properties of nodes or edges

SQL/PGQ DDL constructs

PROPERTY GRAPH

- VERTEX TABLES
  - KEY
  - LABELS
    - PROPERTIES
- EDGE TABLES
  - SOURCE KEY
  - DESTINATION KEY
  - LABELS
    - PROPERTIES

SQL/PGQ query constructs

GRAPH_TABLE - specifies the property graph to use

MATCH - find paths within a graph with a given pattern

- (a: label) - vertexes with a given label
- [b: label] - edges with a given label
- ->, <- - vertex-edge connectors

WHERE

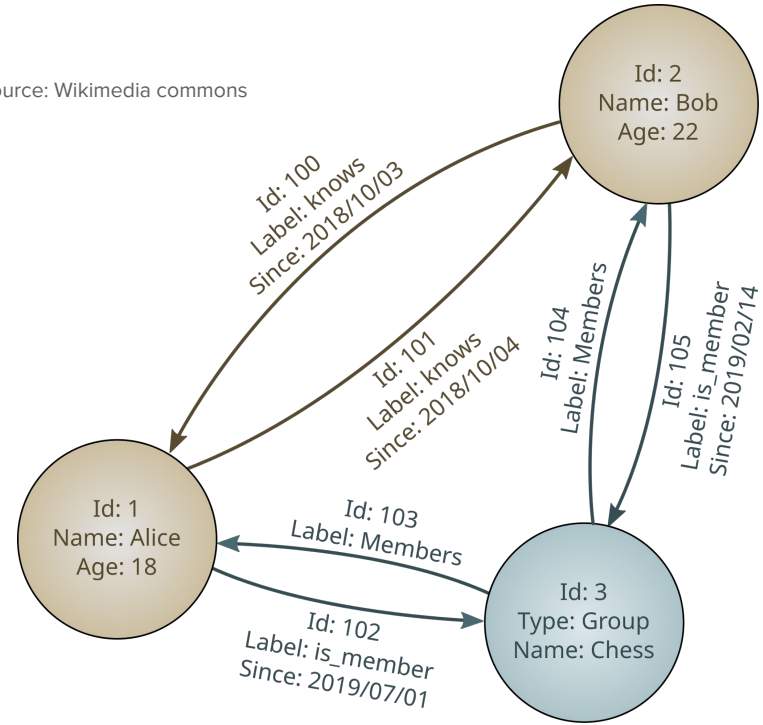
- Within vertex/edge - filters edges or nodes based on properties
- In a pattern: Filter patterns based on properties of elements in the path

COLUMNS

- Project properties of nodes or edges in the paths

Property graph and relational data

Source: Wikimedia commons



Person

Id	Name	Age
1	Alice	18
2	Bob	22

Knows

Id	From	To	Since
100	1	2	2018/10/03
101	2	1	2018/10/04

Group

Id	Name
1	Chess

MemberOf

Id	Member ID	Group ID	Since
102	1	1	2019/07/01
105	2	1	2019/02/14

SQL/PGQ property graph example: Vertex

PROPERTY GRAPH Gymkhana

VERTEX TABLES

Person

Label Person

Properties id, name, age

Group

Label SportsGroup

Properties group_id, name

SQL/PGQ property graph example: Edges

EDGE TABLES

Knows

Source: Person(id)

Destination: Person(id)

Label Relations

Properties id, since

Label Knows

Properties id, since, how

MemberOf

Source: Person(id)

Destination: Group(id)

Label Relations

Properties id, since

Label MemberOf

Properties id, type, since

SQL/PGQ query example

People who don’t know each other but are members of same group

```
(b:person)->(c:Group)<-(a:Person)
```

Introduce them

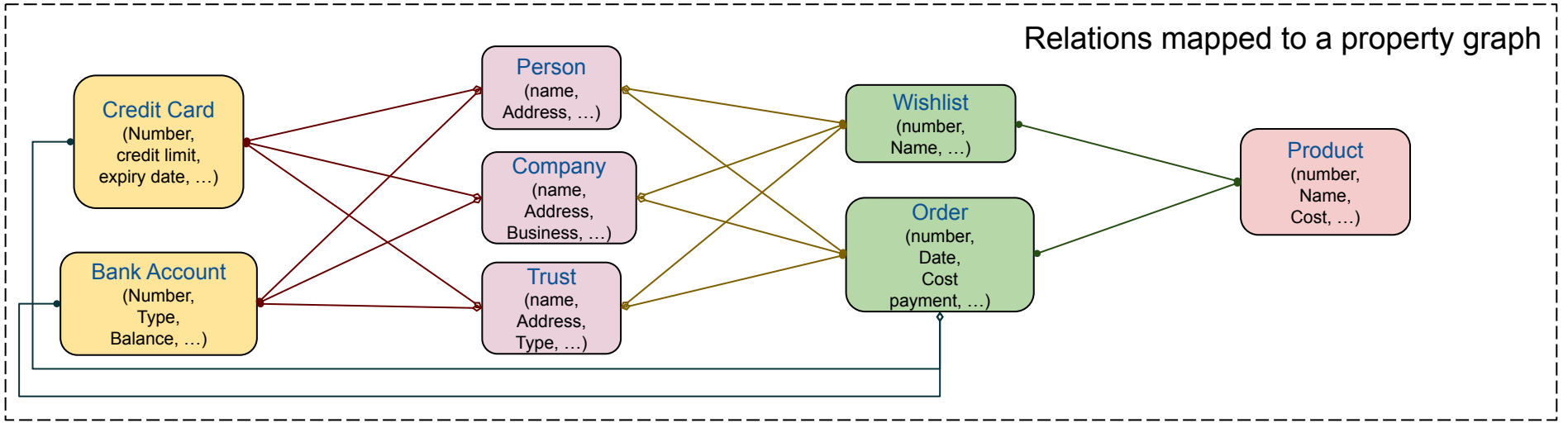
People who made acquaintances because of Gymkhana

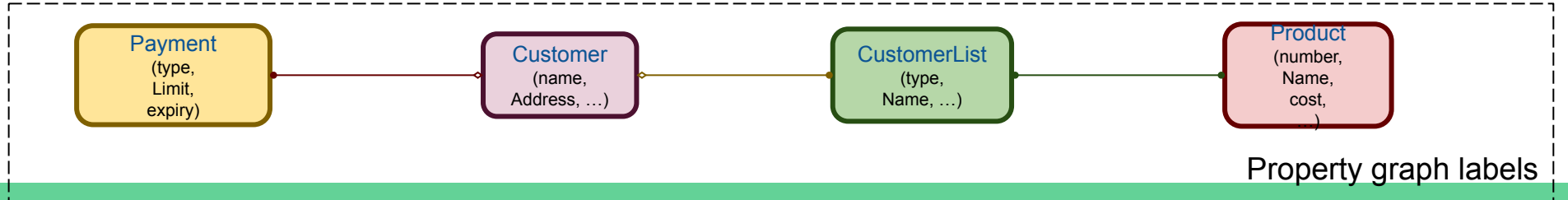
```
(c:Group)<-[ma:MemberOf]-(a:Person)-[k:Knows]->(b:person)-[mb:MemberOf]->(c
)

WHERE (k.since  > ma.since and k.since > mb.since)
```

Advertise them

ER diagram as a property graph





DDL

```
CREATE PROPERTY GRAPH shop

VERTEX TABLES (
    CreditCard label Payment,
    BankAccount label Payment,
    Person label Customer,
    Company label Customer,
    Trust label Customer,
    Wishlist label ProdLink,
    Order label ProdLink,
    Product)

EDGE TABLES (
    CCOwns label Owns
    BAHolds lable Owns,
    CustOrders label CustLink,
    CustWishlist label CustLink,
    CompanyOrders label CustLink,
    CompanyWishlist label CustLink,
    TrustOrders label CustLink,
    TrustWishlist label CustLink,
    OrderCCPayment label OrderPayment,
    OrderBAPayment label OrderPayment,
    OrderItems label ItemLink,
    WishlistItems label ItemLink);
```

Complex query made simple

Find all products paid by credit card

```
(o)->(py:Payment WHERE py.type = 'CC')<-()->(o:Order)->(p:Product) COLUMNS (p.name)
```

Break down

`(o:Order)->(p:Product)` - *all products across orders*

`(o)->(py:Payment WHERE py.type = 'CC')` - *all orders paid by credit card*

`O` - *Links the two orders*

`(py:...)<-()->(o:Order)` - *links payments and orders by the customer/owner of payment method*

`COLUMNS (p.name)` - *projection*

Advantages of relational database with SQL/PGQ

Integration: Leverage existing SQL infrastructure and expertise

Standardization: ISO standard ensures portability and interoperability.

Unified Data Management: Query both relational and graph data within a single system.

Performance: Optimized implementations of PGQ can improve performance compared to pure SQL approaches for graph queries.

ACID guarantees

Query graphs already in relational form

Advantages of native graph databases

Flexibility: graph databases have flexible schema

Storage: graph databases have optimal storage

Performance: graph databases may have better performance

Graph databases and PostgreSQL

PostgreSQL and graph databases

- Apache AGE
  - Cypher like graph query language
    - Wrapped in a function call
  - PostgreSQL is used as a storage
- pgRouting
  - extends the PostGIS / PostgreSQL
  - provides geospatial routing functionality
  - Not for generic graphs

Native SQL/PGQ support: WIP

Most of the DDL: CREATE, ALTER, DROP property graph

All tools supported - dump/restore, upgrade, ecpg etc.

Basic query constructs

- Basic path pattern specification: no quantifiers, embedded patterns yet
- Label disjunction
- Well integrated with rest of the SQL

Current status

Patch authored by Peter Eisentraut and me

Proposed on hackers,

Code complete

- But it’s late for PG18, hopefully PG19

Reviews: functional, documentation, code - welcome

Testing: welcome

PostgreSQL extensibility

Storage optimized for graph databases

- Pluggable storage method
- Query rewrite rules
- Custom plan nodes
- Planner hooks

GQL support

- Pluggable parser 😉
- UDF - Apache AGE

Microsoft

Talks by our Microsoft team

Training: Developing RAG Apps with Azure Database for PostgreSQL & GraphRAG

Varun Dhawan

Wed 5 Mar | 9:00

Hacking Postgres Executor For Performance

Amit Langote

Thu 6 Mar | 11:30

Graph databases, PostgreSQL and SQL/PGQ

Ashutosh Bapat

Thu 6 Mar | 14:00

Unleashing the Power of Azure Database for PostgreSQL Flexible Server

Shriram Muthukrishnan

Thu 6 Mar | 14:00

Keynote: All the Postgres Things at Microsoft

Sujit Kuruvilla

Thu 6 Mar | 16:45

Using Postgres to locate the best coffee near you

Varun Dhawan

Fri 7 Mar | 10:45

Postgres: ServerLESS is more?

Nikhil Sontakke

Fri 7 Mar | 11:30

Beginner's Guide to Partitioning vs. Sharding in Postgres

Claire Giordano

Fri 7 Mar | 14:45

Thank you

Types of graph databases (TBD - do we need this slide?)

(Start from here)

Property Graph Databases

- Most common type
- properties on nodes and edges (e.g., Neo4j, Amazon Neptune)
- Language: Cypher

RDF (Resource Description Framework) Databases

- Used for semantic web and linked data (e.g., Stardog, AllegroGraph)
- SPARQL