GRAPHORUM
CHICAGO, IL | OCTOBER 14-17, 2019

Integrating Property Graphs into the SQL Standard

Keith W. Hare
JCC Consulting, Inc.

Produced by DATAVERSITY®

Abstract

The SQL standard has expanded over the last 30 years to support new technology including XML, temporal data, JSON, Row Pattern Recognition, and Polymorphic Table Functions. This session will present a brief history of the SQL Standard, a high-level overview of the features in the SQL:2016 standard, and introduce current directions including adding support for property graph queries within SQL (SQL/PGQ) and a proposed standard for a declarative Property Graph Query Language that builds on foundational elements from the SQL Standards such as data types, operations, and transactions.

Who Am I?

- JCC Consulting, Inc.
  - President since August 2019
  - Senior Consultant 1985 – 2019
  - Specialize in
    - High performance database systems
    - Data replication and migration
    - Database Administration
- Standards – SQL and GQL
  - Convenor, ISO/IEC JTC1 SC32 WG3 Database Languages
  - Vice Chair, ANSI INCITS DM32, Data Management and Interchange

Introduction

- Brief history of the SQL Standard
- Standards Process, Structure, and Participants
- Database Language Standards Current Work
  - Streaming Data
  - Property Graphs Queries in SQL
  - Property Graph Query Language project
- Summary

SQL Standards – a brief history

- ISO/IEC 9075 Database Language SQL
  - SQL-87 – Transactions, Create, Read, Update, Delete
  - SQL-89 – Referential Integrity
  - SQL-92 – Internationalization, etc.
  - SQL:1999 – User Defined Types
  - SQL:2003 – XML & OLAP
  - SQL:2008 – Expansions and corrections
  - SQL:2011 – Temporal
  - SQL:2016 – JSON, RPR, PTF, MDA (2019)
- 30+ years of integrating new technologies into the SQL standard

SQL:2016 Major Features

- Row Pattern Recognition
  - Regular Expressions across sequences of rows
- Support for Java Script Object Notation (JSON) objects
  - Store, Query, and Retrieve JSON objects
- Polymorphic Table Functions
  - parameters and function return value can be tables whose shape is not known until compile time
- Additional analytics Trigonometric and Logarithm functions
- Multi-dimensional Arrays (2019)

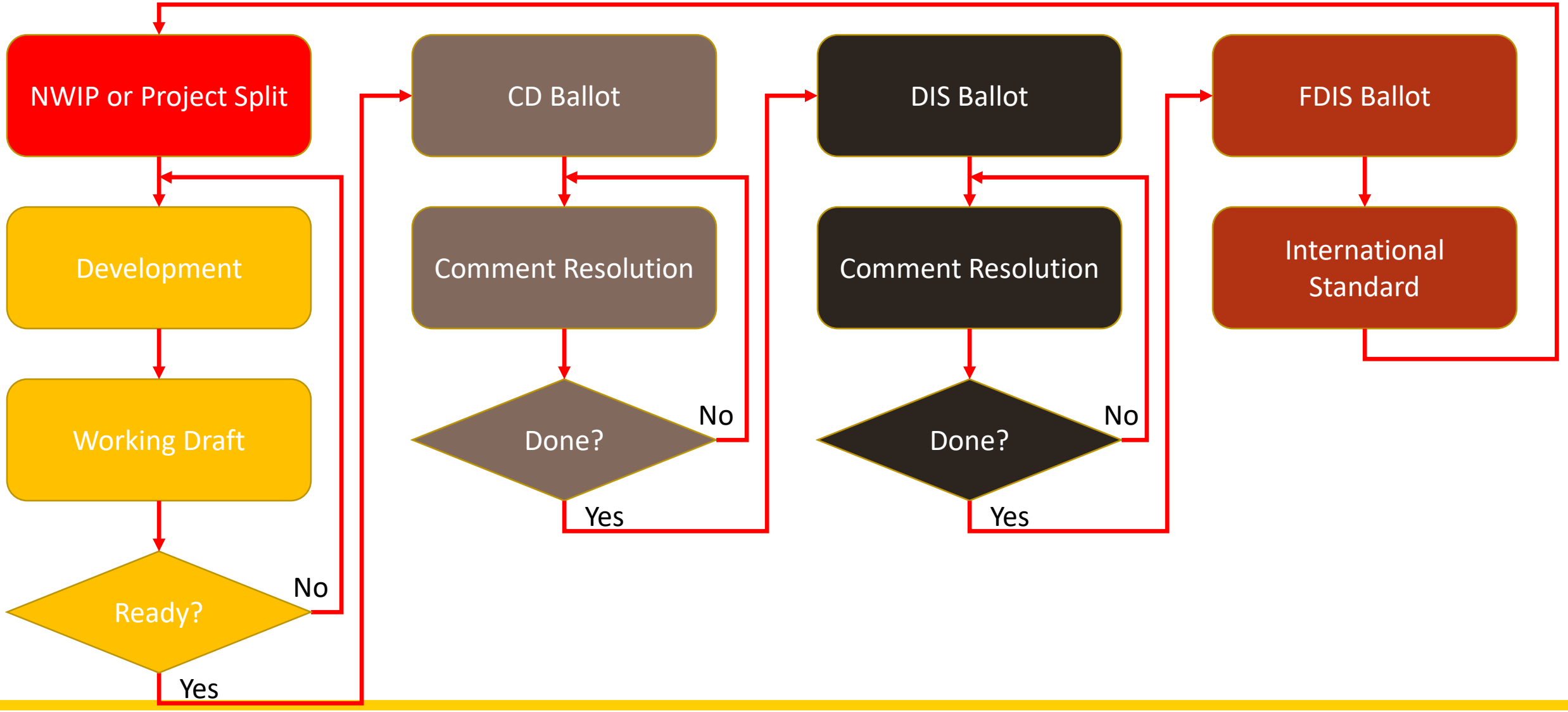
SQL:2016 Parts

Reference	Document title
ISO/IEC 9075-1	Information technology -- Database languages -- SQL -- Part 1: Framework (SQL/Framework)
ISO/IEC 9075-2	Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation)
ISO/IEC 9075-3	Information technology -- Database languages -- SQL -- Part 3: Call-Level Interface (SQL/CLI)
ISO/IEC 9075-4	Information technology -- Database languages -- SQL -- Part 4: Persistent stored modules (SQL/PSM)
ISO/IEC 9075-9	Information technology -- Database languages -- SQL -- Part 9: Management of External Data (SQL/MED)
ISO/IEC 9075-10	Information technology -- Database languages -- SQL -- Part 10: Object language bindings (SQL/OLB)
ISO/IEC 9075-11	Information technology -- Database languages -- SQL -- Part 11: Information and definition schemas (SQL/Schemata)
ISO/IEC 9075-13	Information technology -- Database languages -- SQL -- Part 13: SQL Routines and types using the Java programming language (SQL/JRT)
ISO/IEC 9075-14	Information technology -- Database languages -- SQL -- Part 14: XML-Related Specifications (SQL/XML)
ISO/IEC 9075-15	Information technology -- Database languages -- SQL -- Part 15: Multi-dimensional Arrays (SQL/MDA) (2019)

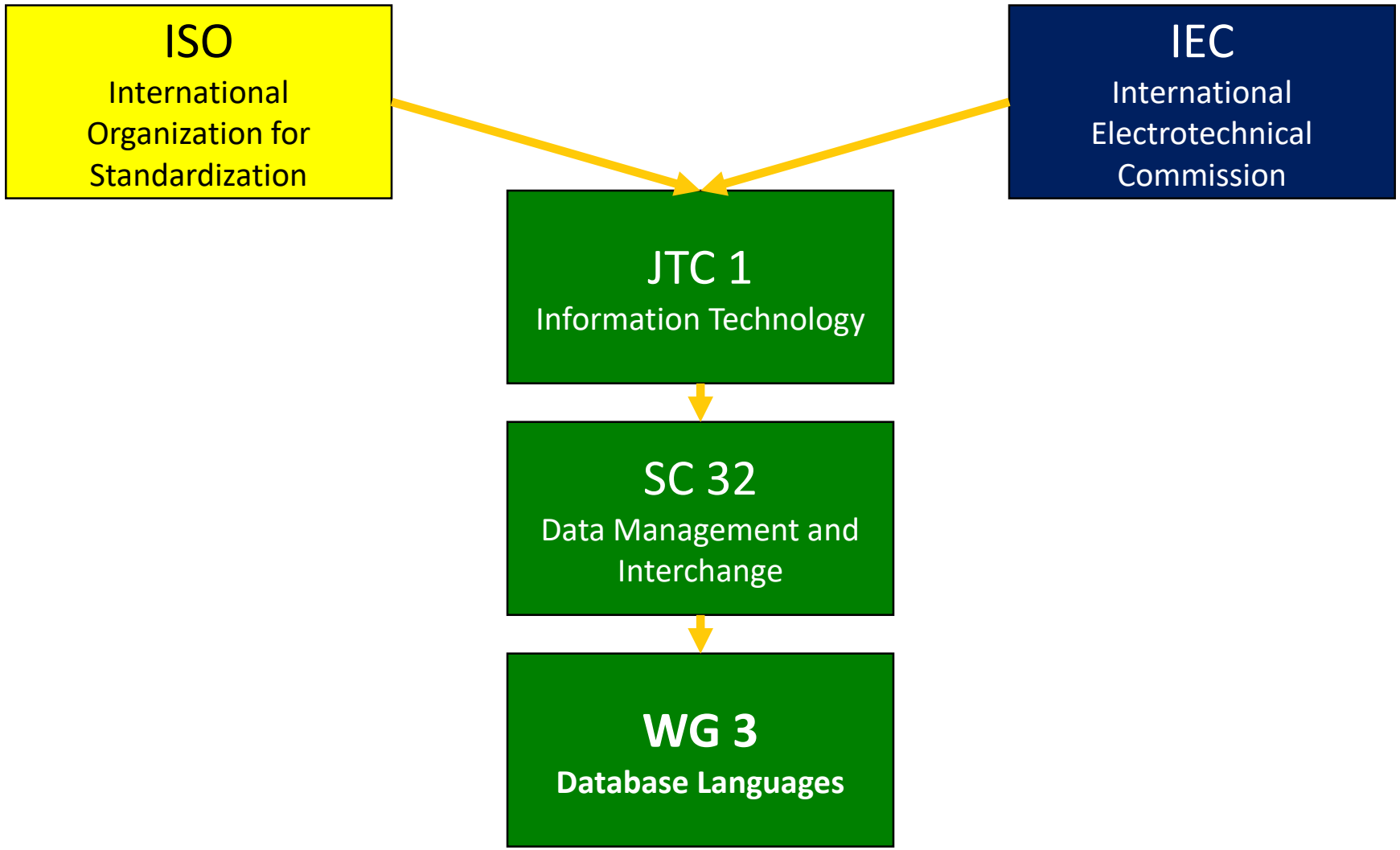
Standardization Process and Structure

- ISO/IEC JTC1 Standardization Process
- International Standards Hierarchy
- USA Standards Structure
- Who is participating?

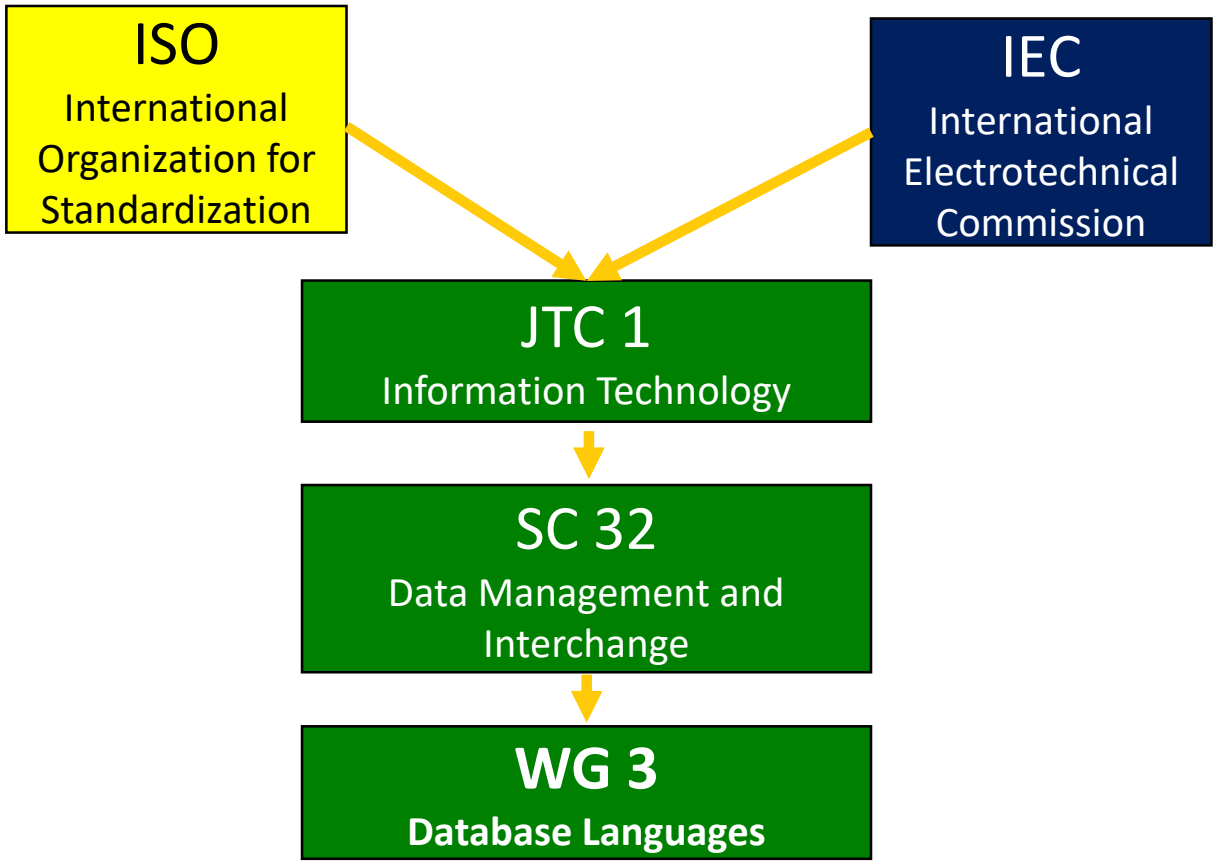
ISO/IEC JTC1 Standardization Process



International Standards Hierarchy



International Hierarchy mirrored in the US



Who participates in Database Languages?

International Committee

1. China
2. Denmark
3. Germany
4. Japan
5. Korea
6. Netherlands
7. Sweden
8. United Kingdom
9. United States

US Committee

1. Actian Corporation
2. ArangoDB Inc
3. Google
4. IBM Corporation
5. Intersystems Corporation
6. JCC Consulting Inc
7. Microsoft Corporation
8. Neo4j Inc
9. Optum Technology
10. Oracle
11. Redis Labs
12. SAP
13. SQLstream Inc
14. Teradata
15. TigerGraph

Database Language Standards Current Work

- SQL Support for Streaming Data
- Property Graph Queries in SQL
- Property Graph Query Language

SQL Support for Streaming Data

- Process data before or instead of storing it
- Support continuous processing
- Zero or more input streams
- Output to
  - persistent tables
  - zero or more output streams

Map Incoming and Outgoing Streams

- Treat as Tables, apply existing SQL capabilities
- SQL Queries, Insert, Update, Merge
- Compound Statements
- Stored Procedures
- Row Pattern Recognition
- Polymorphic Table Functions
- Datatype support:
  - SQL atomic datatypes
  - JSON, XML
  - Multi-Dimensional Arrays

Additional Analytical Techniques

- Time-based queries
  - Tumbling windows
  - Hopping windows
  - Sliding windows
  - Cascading windows
  - Time based windowing join and aggregation
- Techniques also useful for stored data

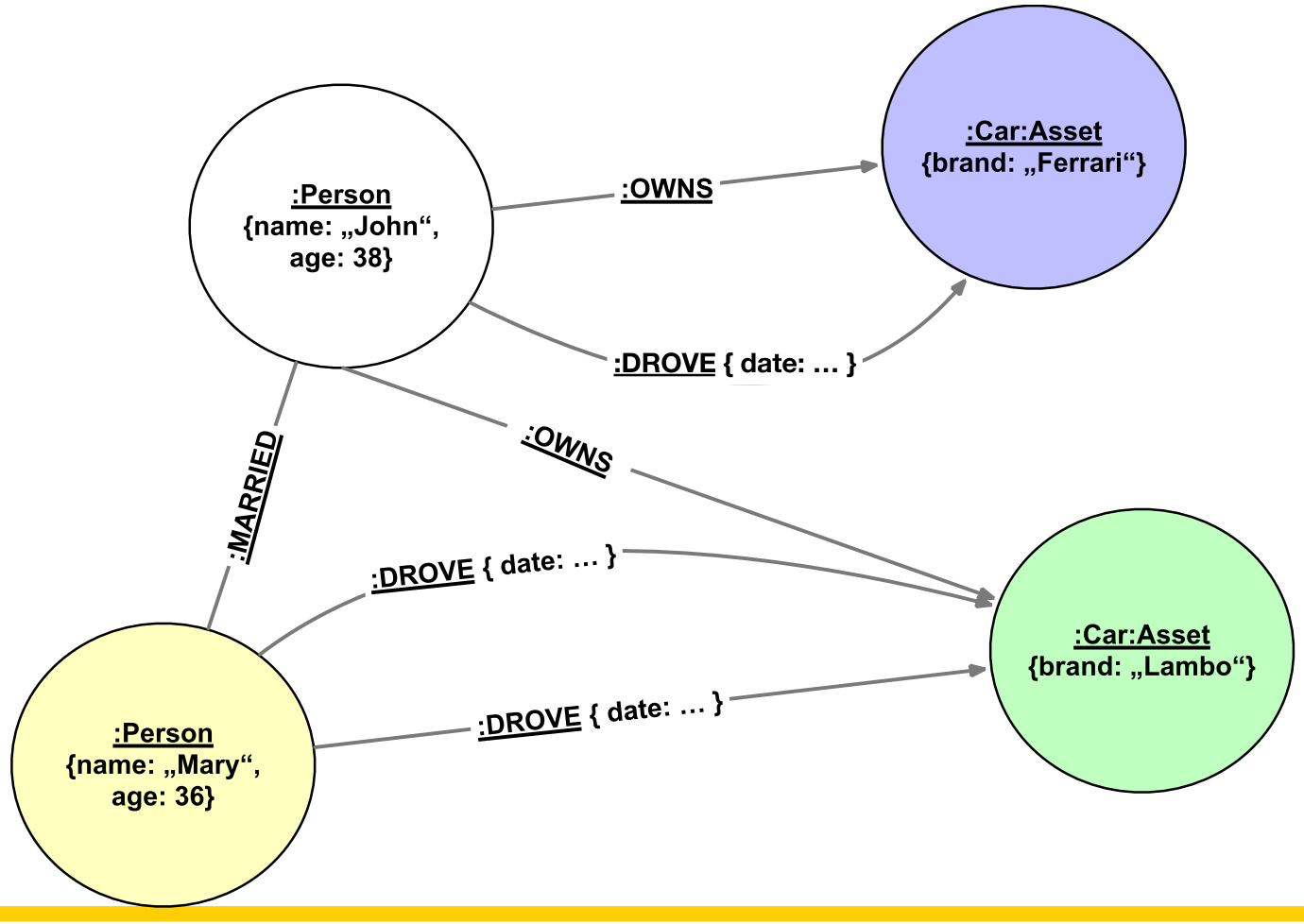
Streaming SQL Status

- Goals and design tradeoff document exists
- Initial working draft in the next year

Property Graphs

- Property Graphs
- Support for Property Graph Queries in SQL
  - SQL/PGQ (9075-16)
- Declarative Property Graph Language
  - GQL

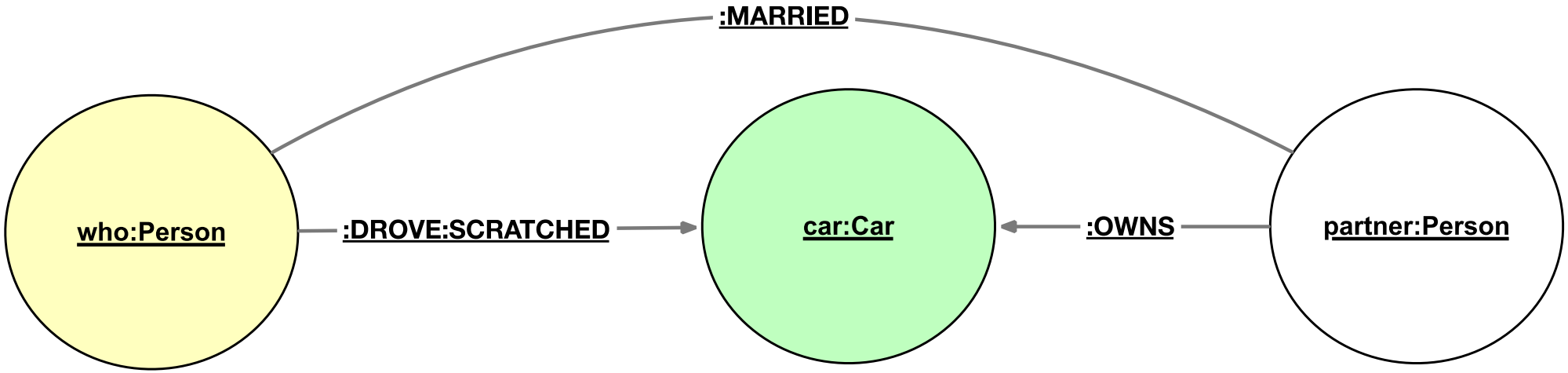
Property Graphs



- Nodes/Vertices
- Relationships/Edges
- 1..* Labels
- 0..* Key-Value Properties
- Intrinsic Identity
- Schema:
  Each label defines its allowed properties

Thanks to Stefan Plantikow

Property Graph Pattern Matching




```
SELECT * FROM MyGraph GRAPH_TABLE (
   MATCH (who:Person)-[:DROVE&SCRATCHED]->(car:Car),
         (car)<-[:OWNS]-(partner:Person)
   WHERE EXISTS (who)-[:MARRIED]-(partner)
   COLUMNS ( who.name AS driver, partner.name AS owner )
)
```


Thanks to Stefan Plantikow

Why Property Graphs?

Use Cases

- Product Recommendation
- Fraud Detection & Analytics
- Money Laundering detection
- Shortest Path
- Supply Chain Management
- Source Code and Document Analysis
- Etc.

SQL Extensions for Property Graphs (PGs)

- Goal: define extensions to query property graphs
- Agree on one (or possibly more) representation of PGs in SQL
  - Most obvious, in tables or views of tables
  - Maybe later, some “native” storage format
- Agree on the way to query PGs in SQL
  - Query PGs “natively” (use the power of pattern matching)
  - Represent result as a table (unleash the power of SQL on the result)
  - Maybe later DML operations on a property graph directly, and graph (view) construction
- Targeted for the next version of SQL Standard (~2020/21)

Why Property Graphs with SQL?

- Users are using both SQL data and Property Graph data
- Application development is easier, better, quicker, faster if only one interface
- “Ascii Art” path expressions provide powerful query capabilities
  - Simpler to write and understand than SQL WITH and Recursive Queries
- Support analytical techniques that are difficult in SQL
  - For example, Shortest Path, Cheapest Path

Brief SQL/PGQ Tutorial

The following slides provide a short tutorial on SQL/PGQ

- Property Graph Definition (DDL)
- Querying Property Graphs

Property Graph Definition (DDL) – Example

- Example:

```
CREATE PROPERTY GRAPH myGraph
  VERTEX TABLES (Person, Message)
  EDGE TABLES (
     Created SOURCE Person DESTINATION Message,
     Commented SOURCE Person DESTINATION Message )
```

Create a PG w/ two vertex tables and two edge tables.

- Existing tables (or views): Person, Message, Created, Commented
- Infer keys & connections from primary/foreign keys of underlying tables
  - PK-FK determines connection between vertices via edges (e.g., `person -[created]-> message`)
- All columns of each table are exposed as properties of the corresponding vertex/edge (tables)

DDL – Example (cont.)

Same PG as before –
but fine-grained control over
labels, properties, etc.

Example for optional clauses:

```
CREATE PROPERTY GRAPH myGraph
    VERTEX TABLES (
        People KEY ( id )
               LABEL Person
               PROPERTIES ( emailAddress AS email ),
        Messages KEY ( id )
                 LABEL Message
                 PROPERTIES ( created AS creationDate, content ) )
    EDGE TABLES (
        CreatedMessage KEY ( id )
                       SOURCE KEY ( creator ) REFERENCES People
                       DESTINATION KEY ( message ) REFERENCES Messages
                       LABEL Created NO PROPERTIES,
        CommentedOnMessage KEY ( id )
                           SOURCE KEY ( commenter ) REFERENCES People
                           DESTINATION KEY ( message ) REFERENCES Messages
                           LABEL Commented NO PROPERTIES )
```

Querying PGs – Example

Get the creationDate and content of the messages created by one person ("email1") and commented on by another person ("email2").

Postfix operator applied to graph, returns table

```
SELECT GT.creationDate, GT.content
FROM myGraph GRAPH_TABLE (
  MATCH
   (Creator IS Person WHERE Creator.email = :email1)
      -[ IS Created ]->
   (M IS Message)
      <-[ IS Commented ]-
   (Commenter IS Person WHERE Commenter.email = :email2)
      WHERE ALL_DIFFERENT (Creator, Commenter)
  ONE ROW PER MATCH
  COLUMNS (
      M.creationDate,
      M.content )
 ) AS GT
```

Vertex pattern enclosed in ( )

Edge pattern enclosed in -[]->

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

Querying PGs – Example (cont.)

```
SELECT L.Here, GT.GasID, L.There, GT.TotalCost, GT.Eno, GT.Vid GT.Eid
FROM List AS L LEFT OUTER JOIN MyGraph GRAPH_TABLE (
  MATCH CHEAPEST (
      (H IS Place WHERE H.ID = L.Here)
          ( -[R1 IS Route COST R1.Traveltime]-> )*
              (G IS Place WHERE G.HasGas = 1)
           ( -[R2 IS Route COST R2.Traveltime]-> )*
      (T IS Place WHERE T.ID = L.There) )
  ONE ROW PER STEP (V, E)
  COLUMNS ( H.ID AS HID, G.ID AS GasID, T.ID AS TID, TOTAL_COST() AS
totalCost,
            ELEMENT_NUMBER (V) AS Eno, V.ID AS Vid, E.ID AS Eid )
) AS GT ON (GT.HID = L.Here AND GT.TID = L.There)
ORDER BY L.Here, L.There, Eno
```

HERE	THERE
Home	HQ
Downtown	Uptown

Given a table with a list of pairs of places called Here and There, for each row in the list, find the cheapest path from Here (H) to There (T), with a stop at a gas station (G) along the way.

Thanks to Oskar Van Rest & Jan Michels

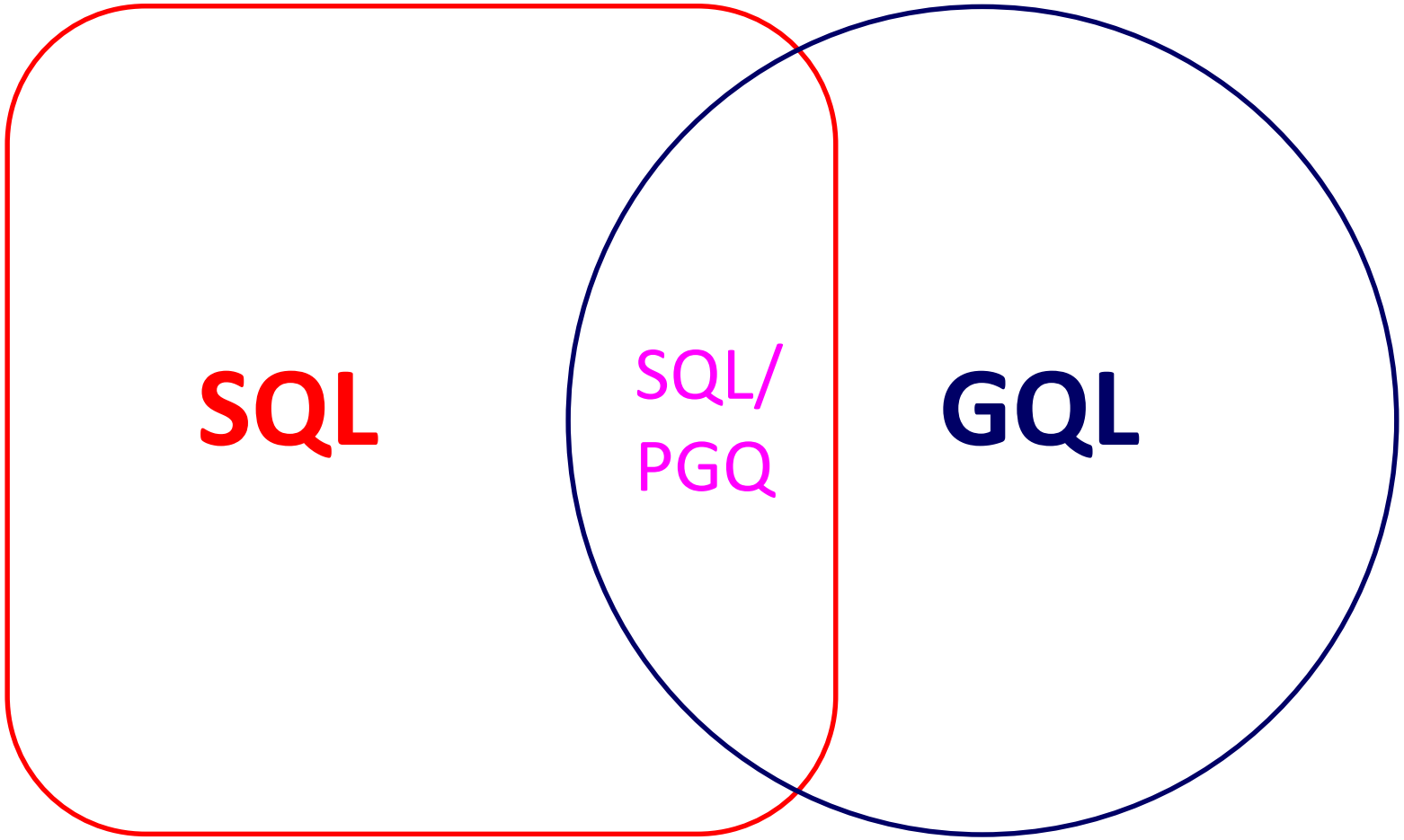
SQL/PGQ Status

- Project Split exists – 9075-16 SQL/PGQ
  - 48 month project (maximum)
- Informal Working Draft exists – developed over last 18 months
- Some detailed content exists
- More detailed contented needed

What about a Graph Query Language Standard?

- Declarative property graph query language
- Parallel to SQL standard
- Take advantage of existing SQL definitions
  - Datatypes
  - Operations
  - Transactions
  - Etc.
- Composable – property graph queries return property graphs
  - Support nested queries & views
- Compatible with SQL/PGQ
  - SQL/PGQ will be completed first
  - Foundational work moved into GQL in later SQL/PGQ revision

SQL, SQL/PGQ, and GQL



Graph Query Language Design Questions

- Schema-less versus Defined Schema
- Data Types
- Internationalization
- Transactions
- Existing Languages
- Ideas for GQL Standard V1+n

Schema-less versus Defined Schema

Schema-less

- Flexible, Fast startup
- Need schema discovery capabilities
- Potential for uncontrolled garbage

Defined Schema

- Required to define and enforce access control
- Useful for query optimization
- Potential for death by design

Valid use cases for both approaches

Data Types

Atomic Data Types

- Boolean
- Character String
- Binary String
- Exact Numeric
- Approximate Numeric
- Time, date, timestamp, interval
- NULLs?

Complex Data Types

- User Defined Types
- Multi-dimensional Arrays
- JSON
- XML

Might be GQL V1+n

Internationalization

- Internationalization support added to SQL standard before Unicode standard was created
- For GQL, use Unicode

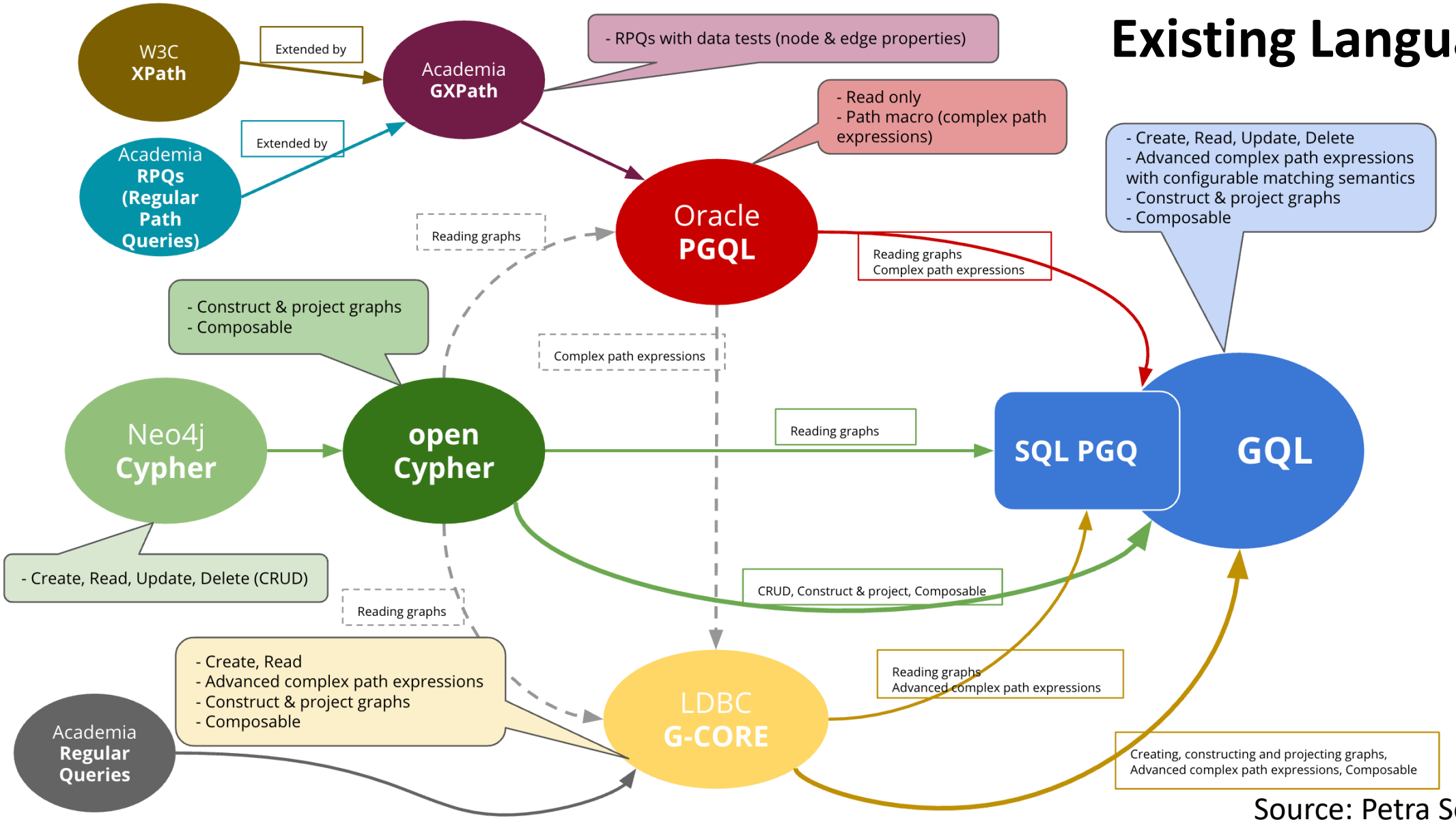
Transactions

- Begin Transaction
- Commit or Rollback
- Isolation Levels – still under discussion
  - Serializable
  - Repeatable Read
  - Read Committed
  - Read regardless?
- Transaction Savepoints?

Existing Languages

- Informal “Existing Languages Working Group”
- Comparative analysis
- Reference document of fine-grained graph query features
- Help drive requirements for a Graph Query Language standard

Existing Languages



Source: Petra Selmer

Ideas for GQL Standard V1+n

- Streaming Property Graphs
- Temporal Support
  - System Versioned Graphs
  - Application Time Period Graphs
- Distribution and replication
  - BASE transactions

GQL Status

- New Work Item Proposal (NWIP)
  - International ballot closed September 8, 2019
  - Requirements for approval are:
    - Majority of countries who vote to approve or disapprove – 10 out of 11 approved
    - At least five countries must name experts for the project – 7 countries named experts
- Work has started
  - Outline of Working Draft
  - List of potential Content
  - Initial early informal editor's draft

Why a Property Graph Query Language Standard?

- Multiple vendor and open source dialects exist today
- Build consensus on requirements, syntax, and semantics
- Easier to get started with property graph databases
- Vendors compete on
  - Performance
  - Analytical capabilities
  - Robustness

What about Semantic Graphs?

- W3C RDF (Resource Description Framework)
  - https://www.w3.org/TR/rdf11-concepts/
  - Triples specify the edges – subject, predicate, object
  - Nodes are inferred from the subject and object
  - OWL Ontologies support inference engines
  - SPARQL https://www.w3.org/TR/sparql11-overview/
- Complementary approaches
- WG3 has opened informal lines of communication with W3C RDF and SPARQL communities
  - W3C “Workshop on Web Standardization for Graph Data: Creating Bridges: RDF, Property Graph and SQL”, March 4-6 2019, Berlin, Germany
  - Report: https://www.w3.org/Data/events/data-ws-2019/report.html

Summary

- Expanding SQL Standard to support market requirements
- Recently published support for Multidimensional arrays
- New work
  - Next generation of 9075 Database Language SQL
  - Streaming SQL
  - 9075-16 SQL/PGQ – SQL support for Property Graph Queries
  - New Project – declarative Graph Query Language – GQL
- Momentum building for both SQL/PGQ and GQL

Questions?

```
SELECT * FROM Graph
  GRAPH_TABLE (
    MATCH(who:AudienceMember)
          -[has:Questions]
          ->(for:Speaker)
    COLUMNS who.name AS audience,
            who.question AS question,
           for.name as speaker );
```

Related Web Sites

- Linked Data Benchmark Council (LDBC)
  - http://ldbcouncil.org/
  - Publications include:
    - “G-CORE: A Core for Future Graph Query Languages”
    - “Towards a property graph generator for benchmarking”
- GQL Standards web page
  - Information about the Graph Query Language standards development
  - https://www.gqlstandards.org

Existing Property Graph Languages

- AQL https://www.arangodb.com/docs/stable/aql/
- G-CORE https://arxiv.org/pdf/1712.01550.pdf
- GraphQL https://graphql.github.io/graphql-spec/draft/
- GRAQL https://dev.grakn.ai/docs/query/overview
- GSQL https://docs.tigergraph.com/dev/gsql-ref
- Neptune https://aws.amazon.com/neptune/
- openCypher https://s3.amazonaws.com/artifacts.opencypher.org/openCypher9.pdf
- OrientDB https://orientdb.com/
- PGQL http://pgql-lang.org/spec/1.2/
- Tinkerpop & Gremlin http://tinkerpop.apache.org/docs/current/reference/