LEX LDBC Extended GQL Schema

Alastair Green

24 June 2023

My background

I started programming for a living in 1978. Since then, in all the usual capacities:

- Product companies from very large ones to start-ups
- Applications and internal product projects in end-user companies
- Distributed computing, distributed transactions, database management
- Graph data (property graphs) since 2016

Neo4j (2016-20): adoption of **openCypher**; initiating the **GQL database language** project

In March 2019 I co-chaired the W3C Workshop on Web Standardization for Graph Data

- Subtitle was: **Creating Bridges: RDF, Property Graph and SQL**

2018-date Vice-chair of LDBC (part-time). IP policies, professionalization, WG3 Liaison

gql.today

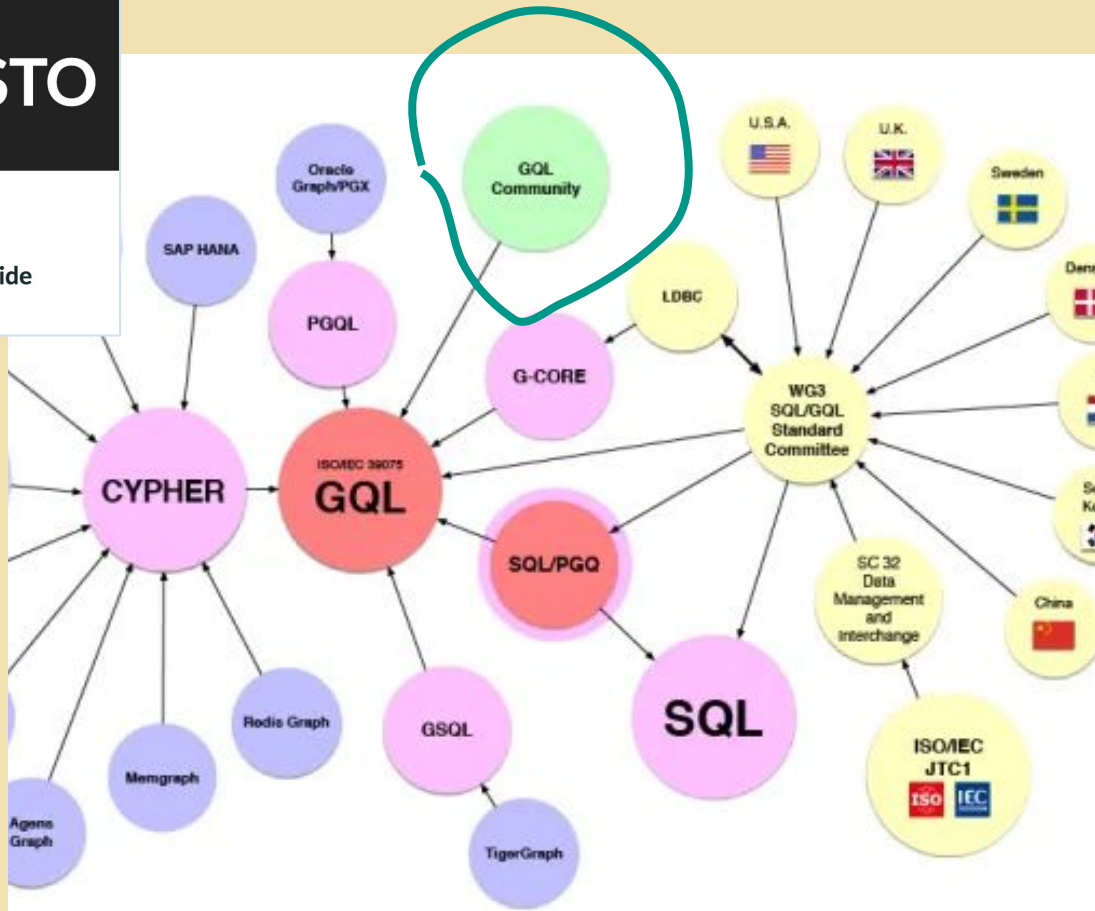
ONE PROPERTY GRAPH QUERY LANGUAGE

THE GQL MANIFESTO

GQL Update – September 16, 2019

GQL Is Now a Global Standards Project alongside SQL

LEX is an example of work of the GQL community in LDBC



LEX project

GQL has schema capability in the form of graph types

There are many aspects of schema which are not addressed by graph types

WG3's focus has been on getting GQL V1.0 out the door

A year ago I floated the idea of restarting work in LDBC on PG schema, with an orientation towards **a proposal for a concrete schema language** to feed into GQL 2.0

> Results of work from the old PGS WG (2020-21) → **PG-Schema** paper, plus
> SQL schema features, EERM and UML Class Diagrams, SHACL ...

LDBC Extended GQL Schema → **LEX**

December 2022 to January 2024: we're half-way through

LEX project participation and activities

About 15 active participants, 35 working group members

Participants from Ant Group, Neo4j, AWS, Tigergraph, Birkbeck, Bloomberg*, Google*, WG3

11 working group meetings (~ fortnightly)

Themes/sub-groups to date

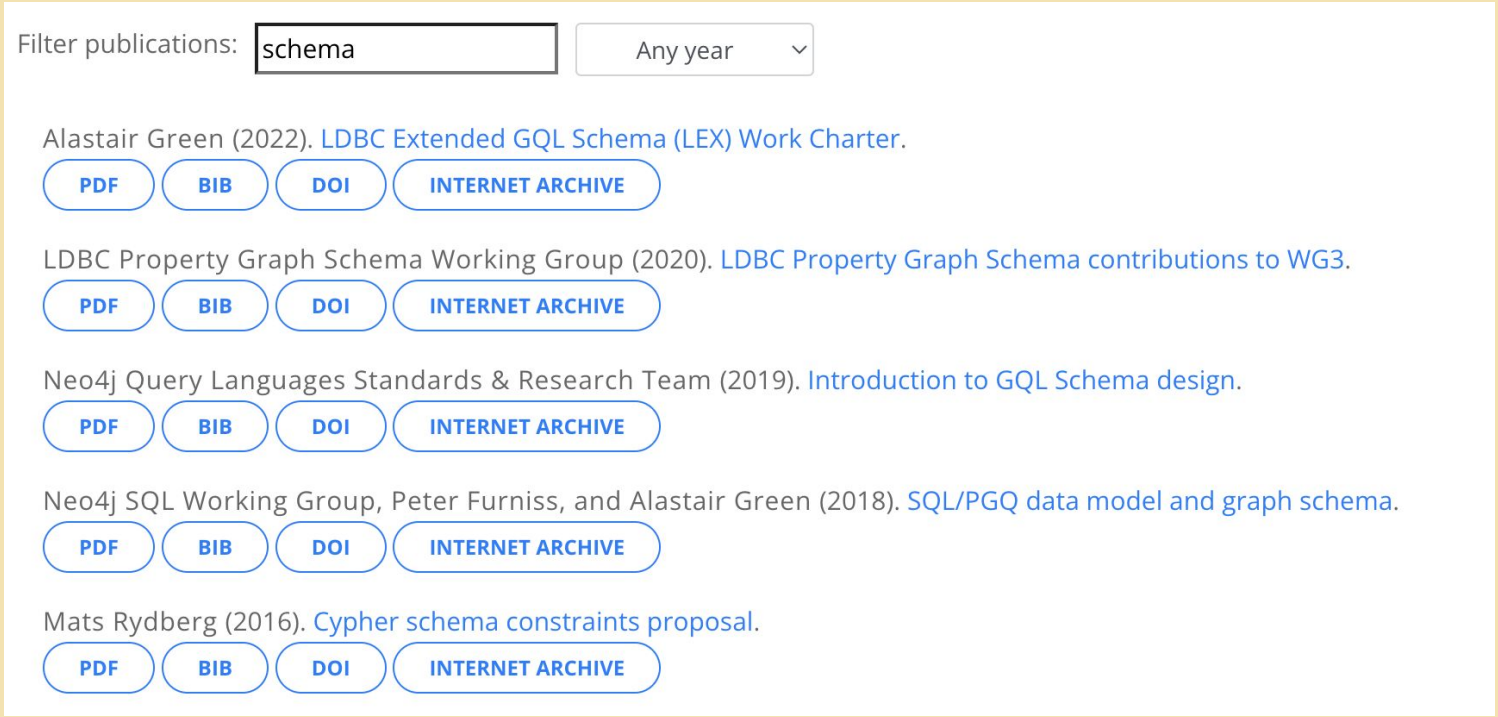
- Use cases and requirements
- JSON Schema
- Named types and labels

Future work: schema graphs, sub-graphs and transactions, referential integrity, sub-graph constraints, common topological patterns within data models

*For participant identification purposes only: listed employer entity has no association with LDBC or any of the work described here

Charter and precursors

https://ldbcouncil.org/publication/



Foundation #1 **GQL graph types**

GQL has schema objects in a catalog directory, called graph types

A **graph type** contains two sets: **node types** and **edge types** ***GT*** = (***NT***, ***ET***)

A **node type** is characterized by a **content type**

A content type is a record type where the fields are **labels** and **property types**: the type is the disjoint union of a set of labels and a set of property types

An **edge type** is characterized by a content type *and*

- **orientation** (directed or undirected, and if directed, direction)
- **node type of the endpoints** of the edge

Graphs can be untyped, or in a graph type

A graph is conformant to a graph type if its elements are in a node type or an edge type

Foundation #2 **PG-Schema**

PG-Schema adds

- content types* independent of element types (“abstract” types)
- union and intersection typing: intersection types are extensions* (inheritance)
- strict and lax graph types
- extensible content types (open for labels, open for property types)
- type names

**content types are data record types; intersection is undefined if property type fields of the same name have different data types, if defined, then intersection of content types is record “width subtyping”*

Extension #1 **JSON Schema**

Allow database (GQL and SQL) types to be used as well as primitive JSON types

Plus

- Nested property types
- Typed structures
- Union types (e.g. NaN, +Inf, -Inf as well as numeric strings for floats)
- Constraints on values of leaf nodes (ranges, string picture regexes)
- Cross-field dependencies
- Allows definition of domains (refinement types)

Oracle 23c allows this feature (proprietary extension to SQL)

Extension #1 JSON Schema (cont.)

A JSON database data type schema definition looks like this:

```
{
  "$comment": "We pretend here that // and /* */ comments are allowed in JSON
  "$comment": “schema: if we followed JSONC then they could be”
  "$id": "tag:iso.org,2023:JTC1.SC32.WG3:JSONSchemaDatabaseDialect:databaseTypes"
  "$defs": {
      "GQL.UNSIGNED_INTEGER_64" : { // database data type schema (DDS) name
          "databaseType": "ISO/IEC DIS 39075 -- GQL unsigned 64-bit integer"
                    // prefix is external specification identifier/name,
                    // then the external spec’s name for the data type
          "type": "integer", // JSON primitive type
          "minimum": 0, // optionally, JSON schema constraints which are permitted
                        // for the primitive type in question
          "maximum": 18446744073709551615
    }
    // there will be many more small schemas defined under $defs,
    // one for each predefined SQL datatype and primitive GQL type
  }
}
```

Extension #1 JSON Schema (cont.)

Consensus: PG features like labels, content to element type mappings, keys on element types etc belong in GQL native schema DDL

Is this the way of dealing with nested data?

Or is it only applicable to a GQL JSON datatype?

Could this be shared across SQL and GQL?

Could facilitate use of JSON Path for hierarchical data (mixed path languages)

This already exists for SQL/PGQ

Extension #2 **Type keys and aliases**

GQL is a structurally-typed language

Same set of labels, same set of property types => types have the same semantic

If we allow a subset of labels to functionally determine the whole content type then we have a type key, which for a set of content types (graph type) identifies each type

If we have one label in the set then the type key is a type name

If we have syntactic sugar for the simple case of a named type, then we can induce a canonical form where the name induces a label. The type name is then in the structural type.

Extension #2 Type keys and aliases (cont.)

```
// CONTENT types with name (no additional labels)
Person {name STRING, dob DATE} // the 80% case: least code // the blue, bold code is sugared syntax
: Person => Audit {name STRING, dob DATE} // the red, roman code is the canonical form
KNOWS {since DATE}
: KNOWS => {since dob DATE}

// NODE type declared using content type name
(*Person) // the “*” marking that Person is a type label which determines a whole content type
(: Person => {name STRING, dob DATE})

// EDGE type declared using named content types that are already declared
(*Person)-[*KNOWS]-(*Person) // the 80% case: least code

(: Person => {name STRING, dob DATE})
    -[KNOWS => {since DATE}]->()
(: Person => {name STRING, dob DATE})
```

Extension #2 Type keys and aliases (cont.)

This gives us the effect of **nominal typing** (use option)

And you can MATCH by name, which is a label => no change to GPML

This allows the 80% case of one label = a type marker, plus property types to be achieved very simply

It allows all of the power of the GQL PGDM to be exploited if desired

Type keys can be used as proxies for their determined content types in DDL constructs like union, intersection/extension, and when defining node and edge types

Also add aliases that are identifiers that do NOT participate in the structural type

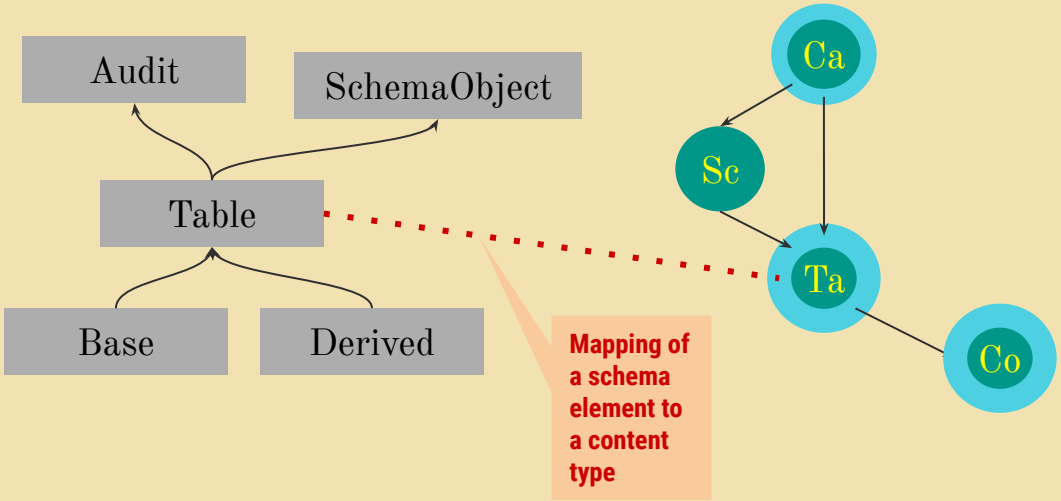
Just for DDL convenience

Possible Future #1 **Schema graphs**

Two parts

Content (attribution) types lattice

Schema graphs, (a representation of a GQL graph type)

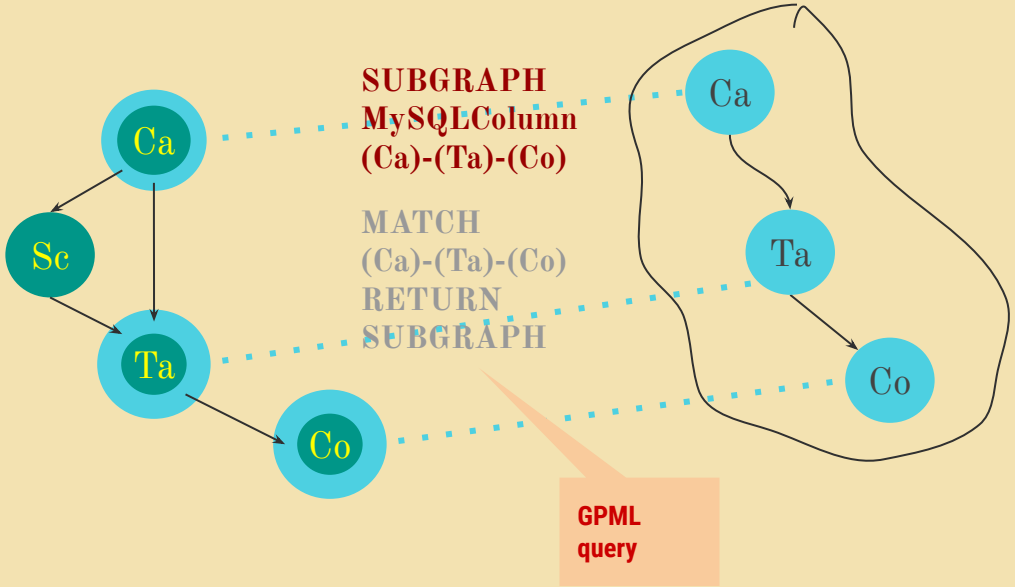


Possible Future #2 **Schema sub-graphs**

Third parts

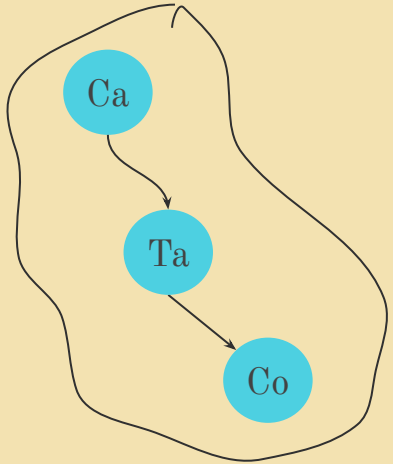
Schema graphs, the ground for schema subgraph (a representation of a GQL graph type)

Schema subgraphs, defined by GPML queries: the **units of mutation and constraint**



Possible Future #3 **Naming sub-graphs in DML ops**

```
SUBGRAPH
MySQLColumn
(Ca)-(Ta)-(Co)
```

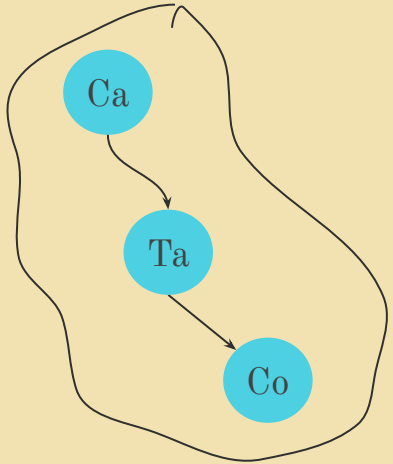


Possible Future #4 Using sub-graphs to define **RI graphs**

Referential integrity can manifest itself in edges in a graph

We need to be able to define RI sub-graphs

SUBGRAPH
MySQLColumn
(Ca)-(Ta)-(Co)



Possible Future #5 **Complex constraints** on sub-graphs

Any constraint on a query over the schema graph is a use of the schema sub-graph concept

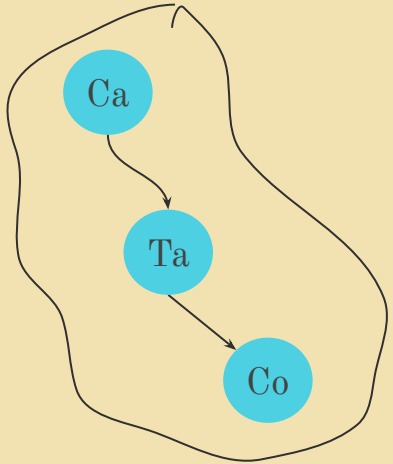
PG-Keys shows this for node types, edge types and discusses path types

But we can generalize to sub-graph types

Cardinality constraints are an example

Non-local keys

SUBGRAPH
MySQLColumn
(Ca)-(Ta)-(Co)



Possible Future #6 **Topology patterns**

There are patterns (like trees) which are commonly used, and laboriously defining them through node and edge types is a waste of time

N-ary relations (as in ERM) are a second case: a standard mapping to a node the reifies the relation could be defined

Much more to think about

A pattern-defined schema graph might not just be for PGs

Schema graphs can be queried, so they are a form of “information schema”

New participants are very welcome

LEX project papers presented to WG3 on 14 June 2023

Ideas for GQL Expansions (WG3:DCA-031) (LEX-036)*

LDBC Extended Schema (LEX) Overview (WG3:DCA-036) (LEX-035)

LEX Working Group - Use Case Work Read-out
(WG3:DCA030r1) (LEX-031)

Introducing PG-Schema (WG3:DCA-037) (LEX-034)

Types, subtypes, labels, names and aliases in a Graph Schema Language (GSL).
(WG3:DCA-038r2) (LEX-027r3)

A Database Dialect of JSON Schema (WG3:DCA-039r1) (LEX-030r1)

Schema sub-graphs and incremental transactional updates of graph databases
(DCA045r1) (LEX-033r1) **These links work for LDBC members, see the next page*

Joining Linked Data Benchmark Council

Individuals can join LDBC and gain access to this work and to the draft specifications of GQL and SQL/PGQ without charge. Send e-mail to info@ldbcouncil.org.

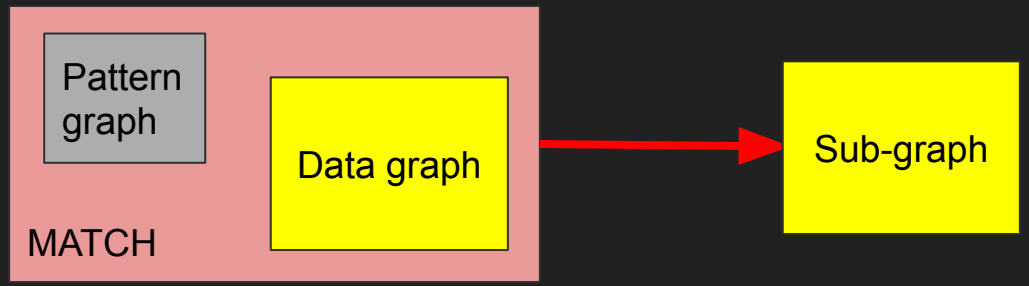
Appendix

“GQL 2.0: A Technical Manifesto”, June 2022, LDBC TUC

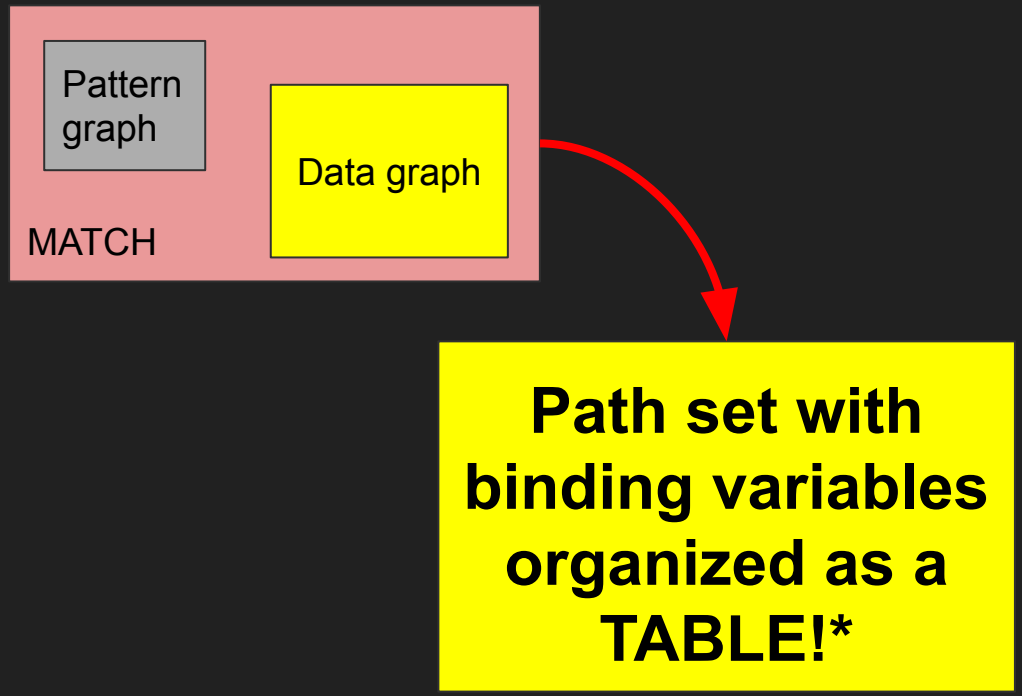
Some of the slides from that presentation follow talking about sub-graph extraction (and compositional graph-projecting queries)

GPM: does it produce sub-graphs?

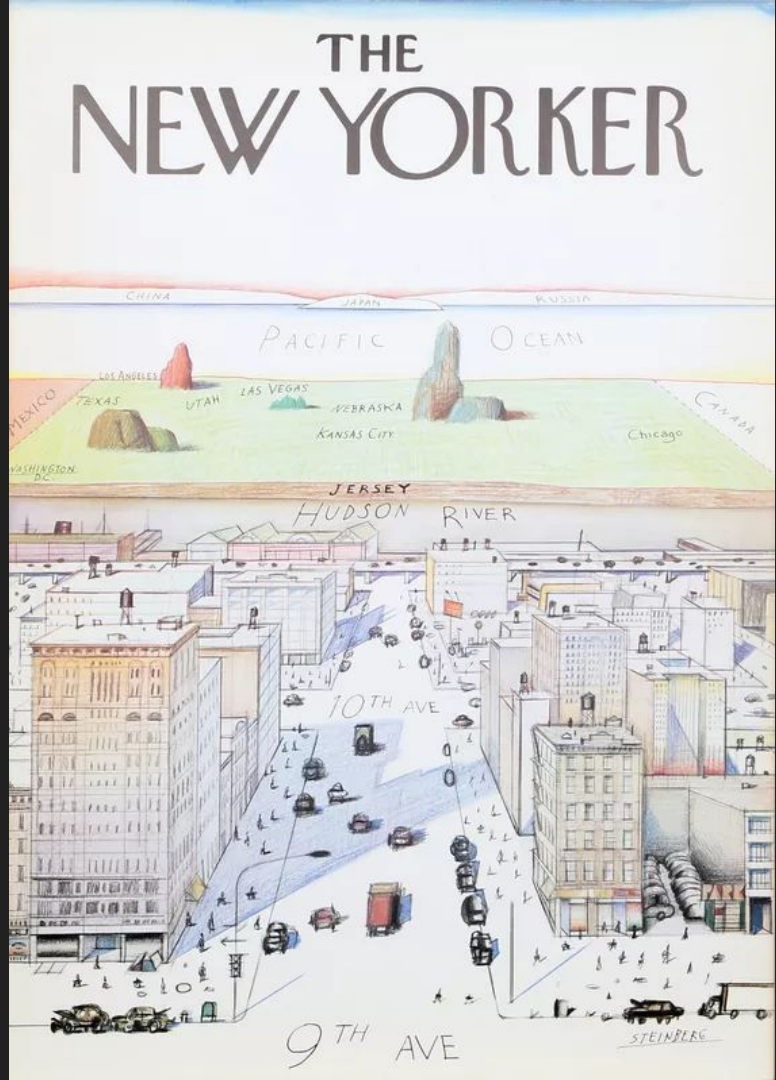
For semi-political and semi-technical reasons people don’t like talking about graph pattern matching in PGQ/GQL in the following (classical) way



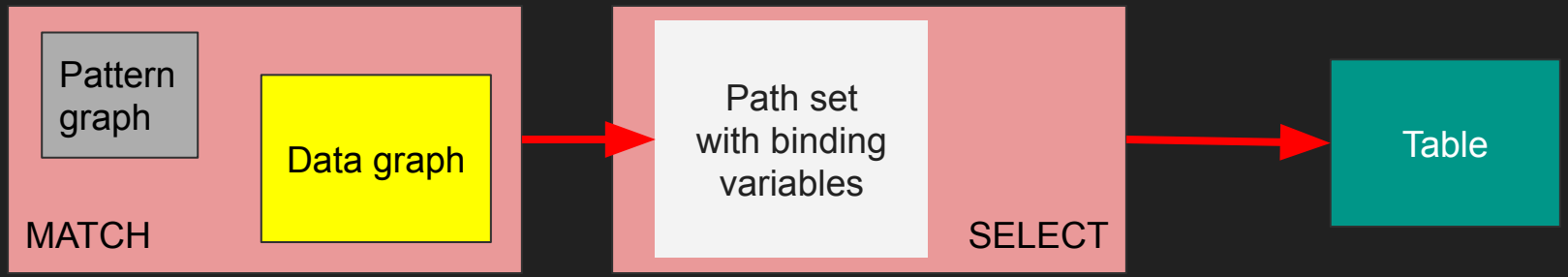
Or does it produce tables?



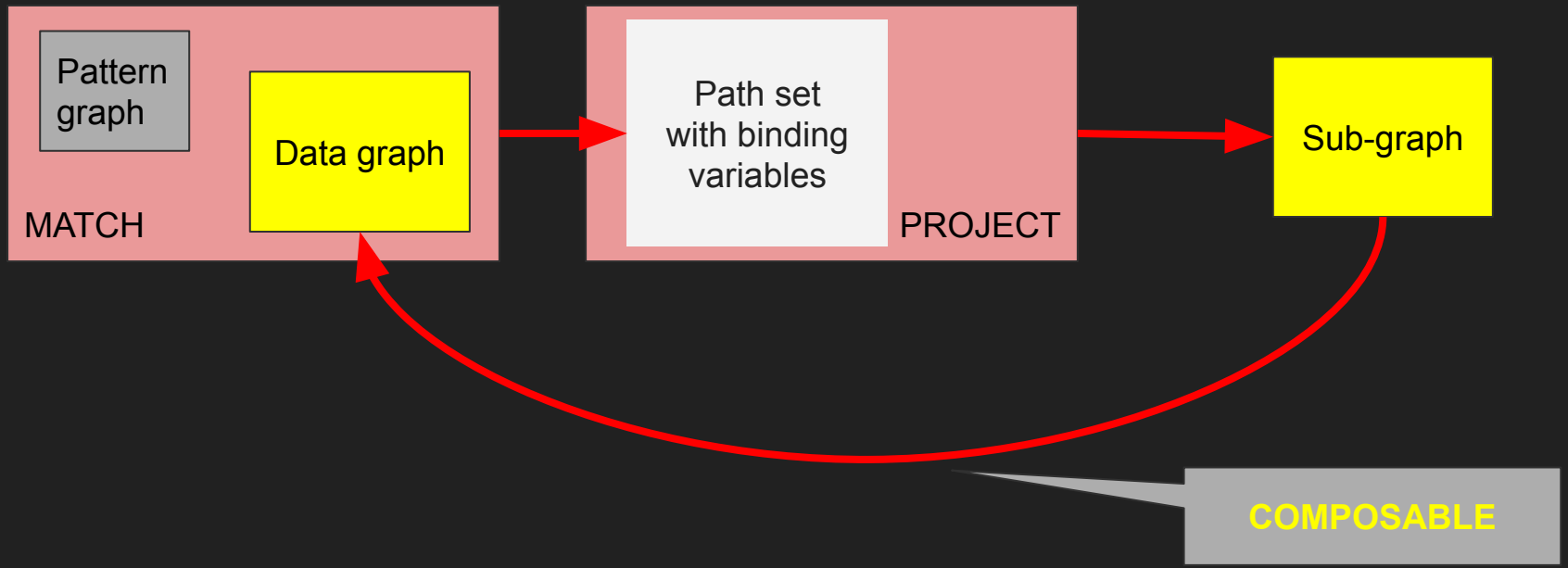
*Can be viewed as a sub-graph



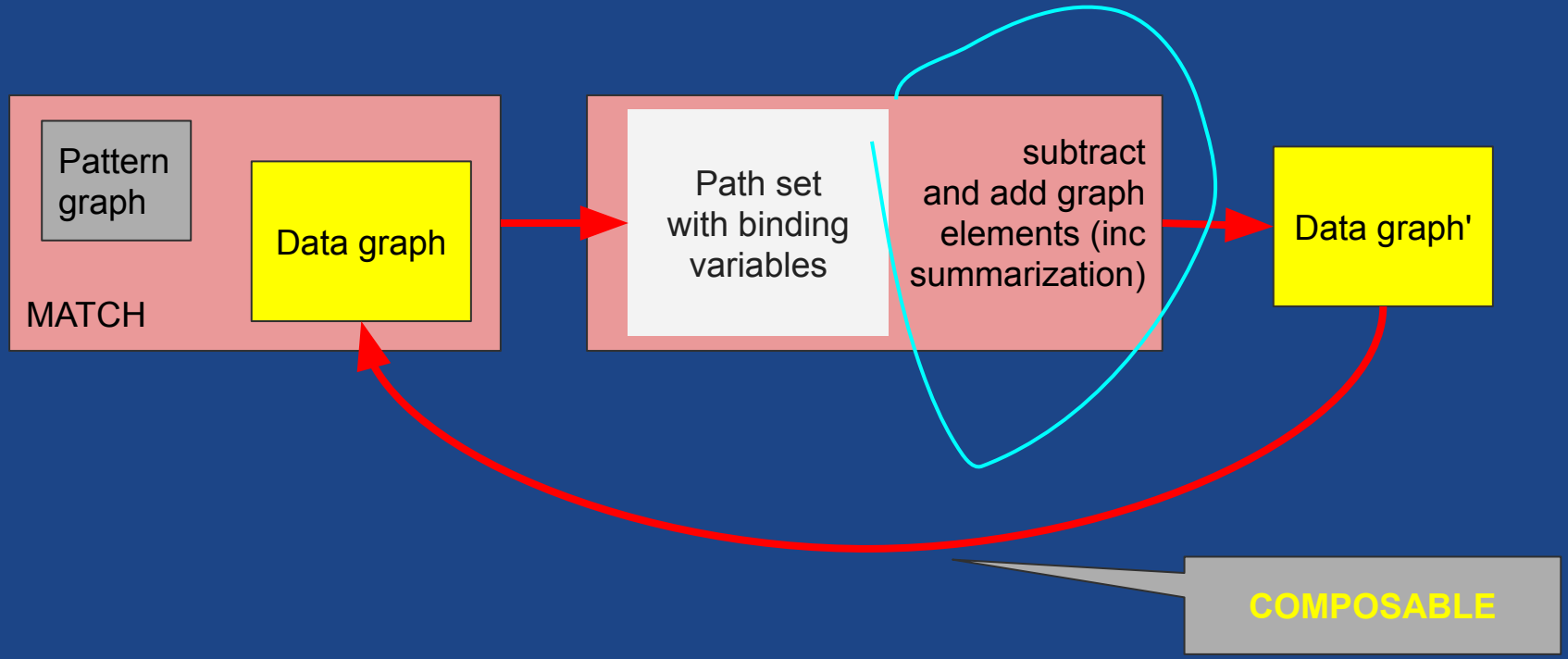
What happens in PGQ is this



What will also happen in GQL (ask Keith when) is this



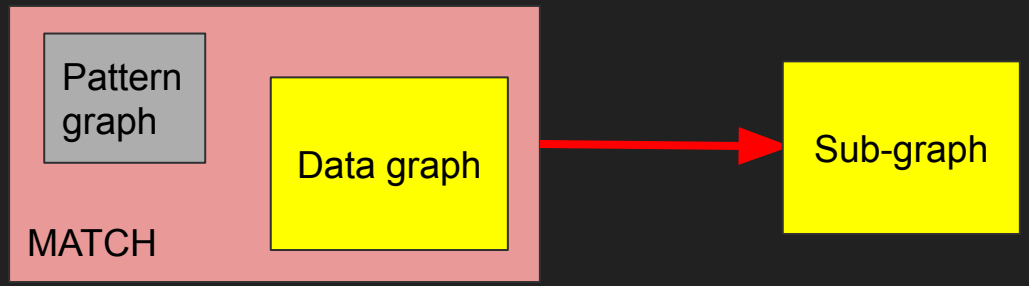
And what really needs to happen in GQL is this ⟶ views



Back to pattern graphs and sub-graphs

Pattern graphs (graph patterns in GPML-speak) use variables to join paths to encode a graph

And in querying that means: encode or express a matched sub-graph



Of course paths are degenerate graphs, and edges are degenerate paths and nodes are degenerate edges