GQL 2.0
A technical manifesto

Alastair Green

15th LDBC TUC, 18 June 2022

The first GQL manifesto was “declarative”: what, not how

It said (in May 2018): we need a standard declarative Graph Query Language

There are some existing languages, let’s draw on them

GQL should be the SQL of graphs (in terms of ecological niche, and reuse of SQL infra)

Graph pattern matching (Cypher, PGQL → GXPath, SQL/PGQ): CRPQ with data tests

Composable queries (like SQL SELECT): graph construction or projection (G-CORE)

Mutate as well as read [unlike PGQ, which is therefore a “gateway drug” for GQL, see PGQL]

Schema, a catalog, multiple named graphs

What's coming in GQL ~~2023~~ 2024?

It said: we need a **standard declarative Graph Query Language**

There are some existing languages, let's draw on them

GQL should be **the SQL of graphs (in terms of ecological niche, and reuse of SQL infra)**

Graph pattern matching (Cypher, PGQL → **GXPath, SQL/PGQ): CRPQ with data tests**

Composable queries (like SQL SELECT): graph projection (G-CORE), views

Mutate as well as read

Simple schema (node and edge type), **no keys, no cardinalities, no path or subgraph types**

A catalog, multiple named graphs

How much is that? **A hell of a lot**

An awful lot of work: largely done in the U.S. INCITS expert bodies and in WG3

By a limited number of companies and participants: this is normal

Oracle and Neo4j require special mention because money to pay for experts and to maintain the community and standards-technical infrastructure is what makes standards happen

The fact that SQL and GQL are being managed by the same technical working group is huge

- ❏ Inheritance of SQL standards technical concepts, terminology, social and build processes
- ❏ Largely shared data model (PGQ is a subset of GQL)
- ❏ GPML shared by PGQ and GQL

How much is that? **Nothing like enough**

Where are the implementations?

TCK, RI ...

And how are we progressing with sufficient schema, “sliding” schema, and graph projecting queries?

Enterprise computing relies on communicable and enforceable schema and views

Data modelling, BI and ETL tooling depend on these two pillars

Efficient and scalable databases use schema

The immense appeal of the graph data model is subverted by these lacks

What next? Possible sub-optimal outcomes

PGQ could be extended to include PGQL-like “graph view mutation” verbs

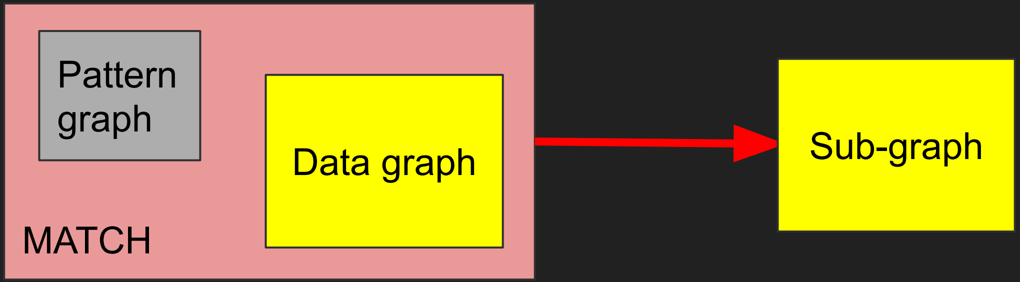
That would be just fine if this was done by allowing the DML of GQL to be “quoted” in a function like GRAPH_ALTER. A fork between GQL DML and SQL graph DML would be unfortunate. The trouble is, this route will not lead easily to graph-projecting queries ...

People could start to reach for counter-cultural alternatives to the missing parts of GQL: SHACL, Legend, dbt have all been seen recently around the edges of the abhorrent vacuum

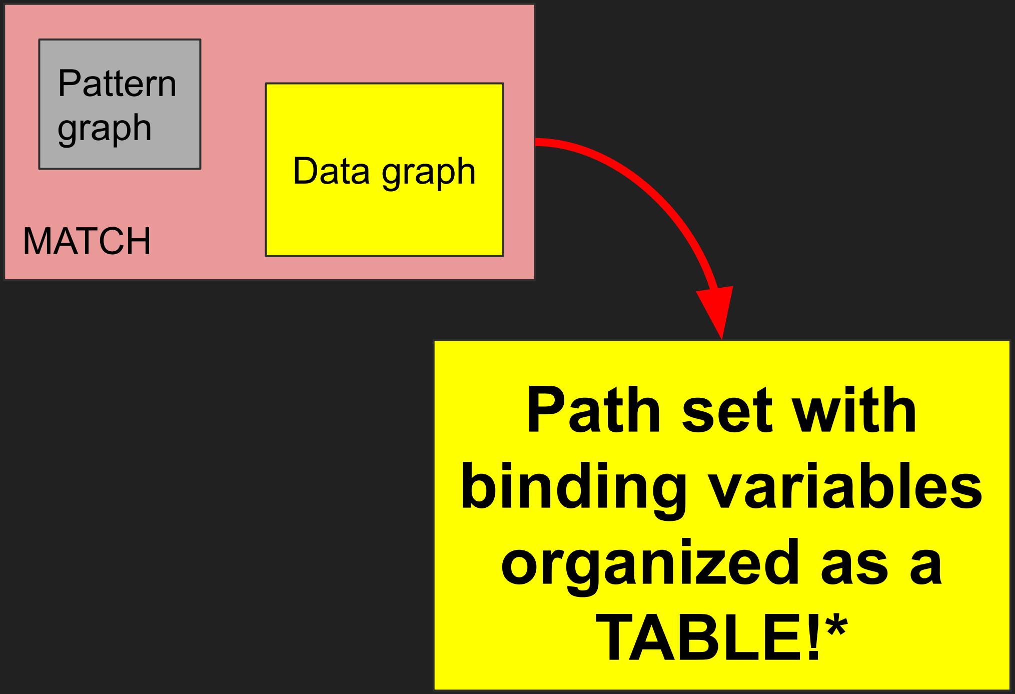
The lack of full-power schema and composable queries (therefore views) is enough to condemn GQL (and graph data management) to niche status, when in fact it could develop into a way of doing everything you can do in SQL, but with a property graph mindset

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

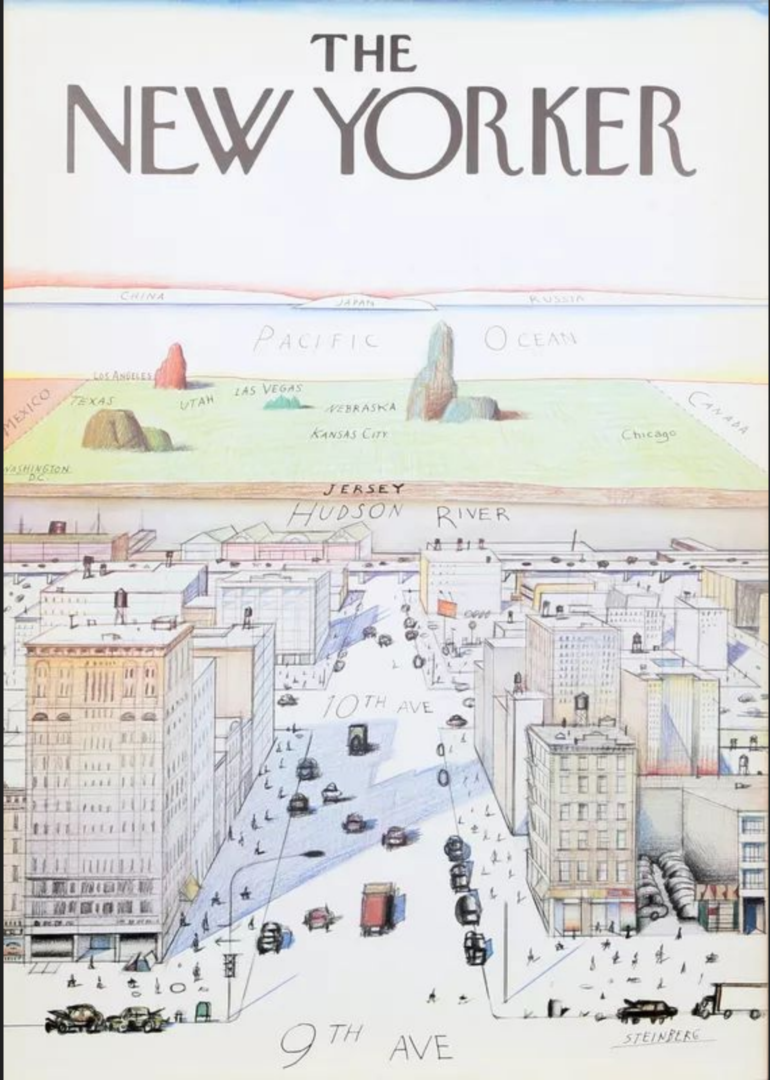


Or does it produce tables?

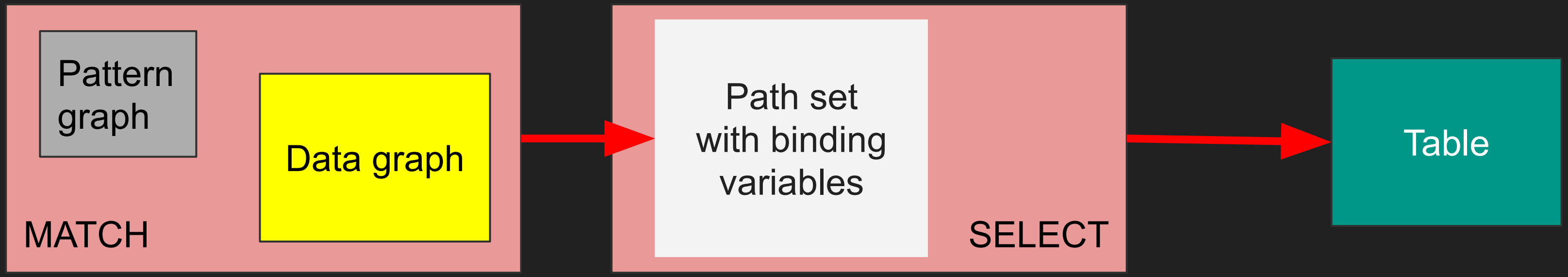


Path set with binding variables organized as a TABLE!*

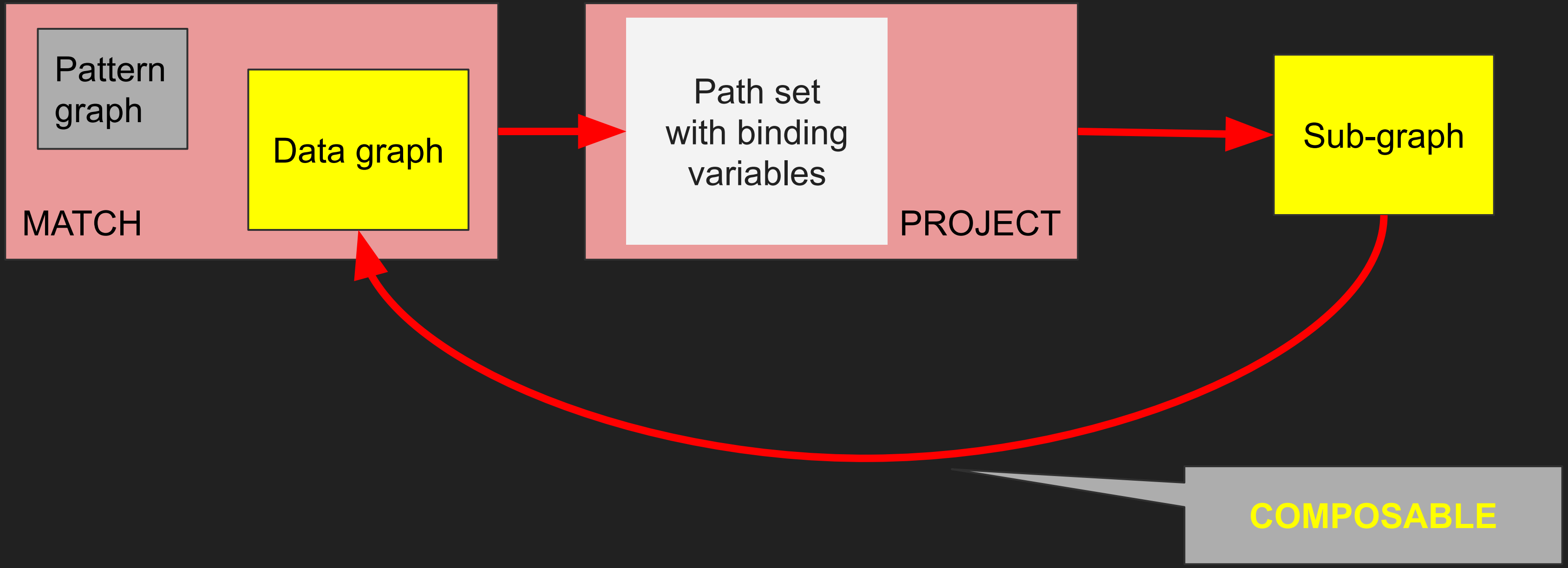
*Can be viewed as a sub-graph



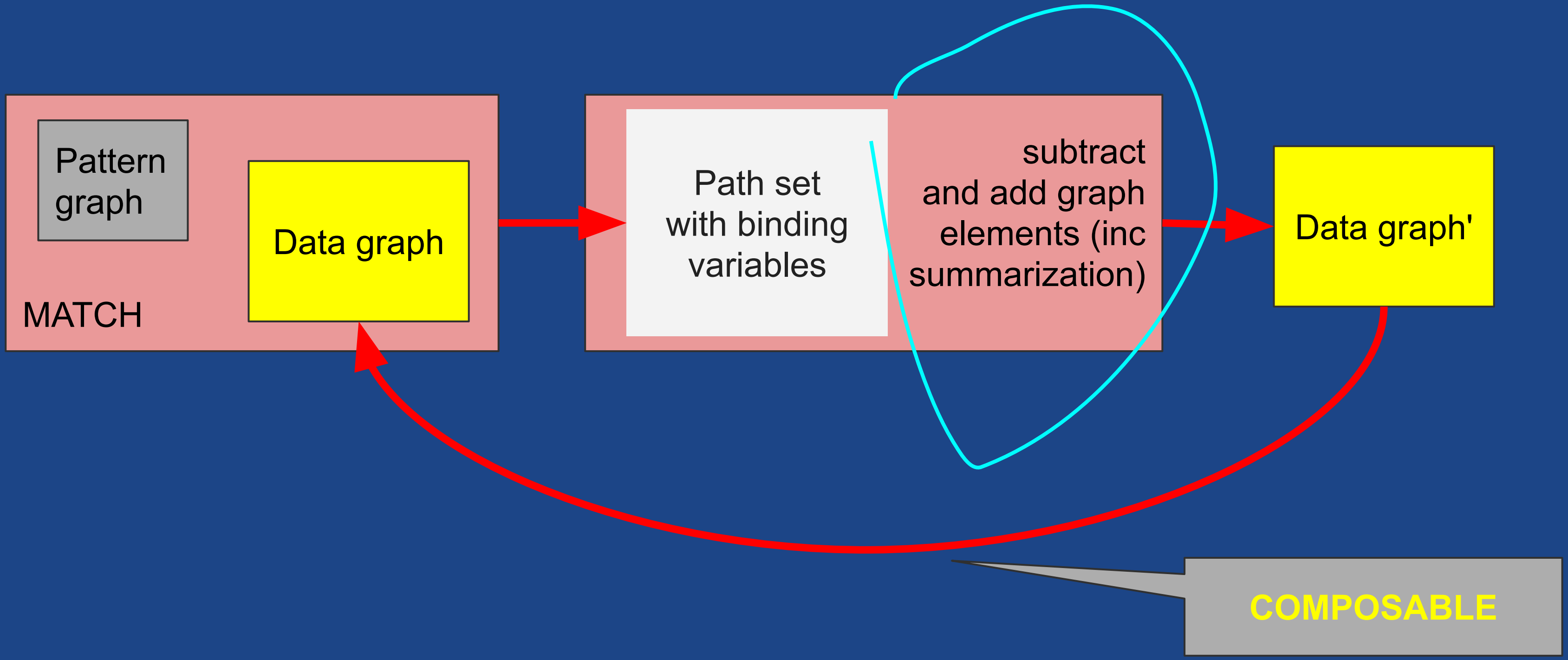
What happens in PGQ is this



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



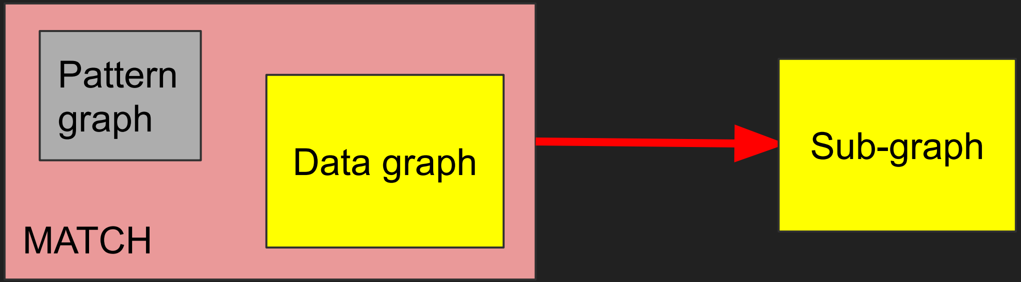
And what really needs to happen in GQL is this → views


Pattern graph
Data graph
MATCH
Path set with binding variables
subtract and add graph elements (inc summarization)
Data graph'
COMPOSABLE

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

Schema: a union of sub-graphs

GQL graph types are the union of a set of node types and edge types. This can be thought of as a **schema graph**, which maps by a graph homomorphism to all conformant data graphs.

If we generalize that, we can say that a graph type is a union of sub-graph types

This allows us to state minimal units of existence

Sub-graphs can also be the subject of constraints (e.g.) cardinalities etc

A sub-graph T of a sub-graph S of a schema graph is a supertype of S

A sub-graph of graph can be a determinant of the graph (can constitute a key)

Raising schema to the level (generality) of sub-graphs is a practical requirement

Schema: a union of sub-graphs

GQL graph types are the union of a set of node types and edge types. This can be thought of as a **schema graph**, which maps by a graph homomorphism to all conformant data graphs.

If we generalize that, we can say that a graph type is a union of sub-graph types

This allows us to state minimal units of existence

Sub-graphs can also be the subject of constraints (e.g.) cardinalities etc

A sub-graph T of a sub-graph S of a schema graph is a supertype of S

A sub-graph of graph can be a determinant of the graph (can constitute a key)

Raising schema to the level (generality) of sub-graphs is a practical requirement

Schema is needed, but people also want no schema

A lot of work in LDBC PGS WG on the subject of extensible or partial schema

At one end of the spectrum that means that you can add anything to a non-existent schema

At the other end, you can only add anything that conforms to a total, non-extensible schema

You cannot do serious enterprise data management without schema

You can’t optimize to the extent that SQL implementations can, without schema

ETL, data modelling, vizualization, business intelligence tooling all need schema

Nature abhors a vacuum ...

Updates are frequently merges because of connectedness

So you can say that mutation of the graph is bringing a part of it into line with a sub-graph type, with certain values

This is the only major part of Cypher that caused confusion and dissatisfaction

Again, GQL has taken this on board, to a degree

But this sub-graph oriented concept of mutation (which lines up with schema) is not fully developed, to my mind

Schemas, views, updates*

Using a query-consistent sub-graph metaphor or design axis
To achieve a consistent database language that spans the fundamental capabilities of SQL

LDBC should reactivate its community work, wider than WG3 in terms of participation, to propose concrete suggestions for these areas to pre-load the next edition of GQL

Will the openCypher users/implementers create a GQL TCK by forking the Cypher one?

This is a challenge, and an opportunity, for all graph data management vendors

With these features, the level of capability of a product will be higher than the current prevailing cultural level

** temporal, spatial etc are all good things but they are not critical, central lacks in GQL IMO*