LEX LDBC Extended GQL Schema

Alastair Green

LDBC 18th TUC, Guangzhou

30 August 2024

gql.today

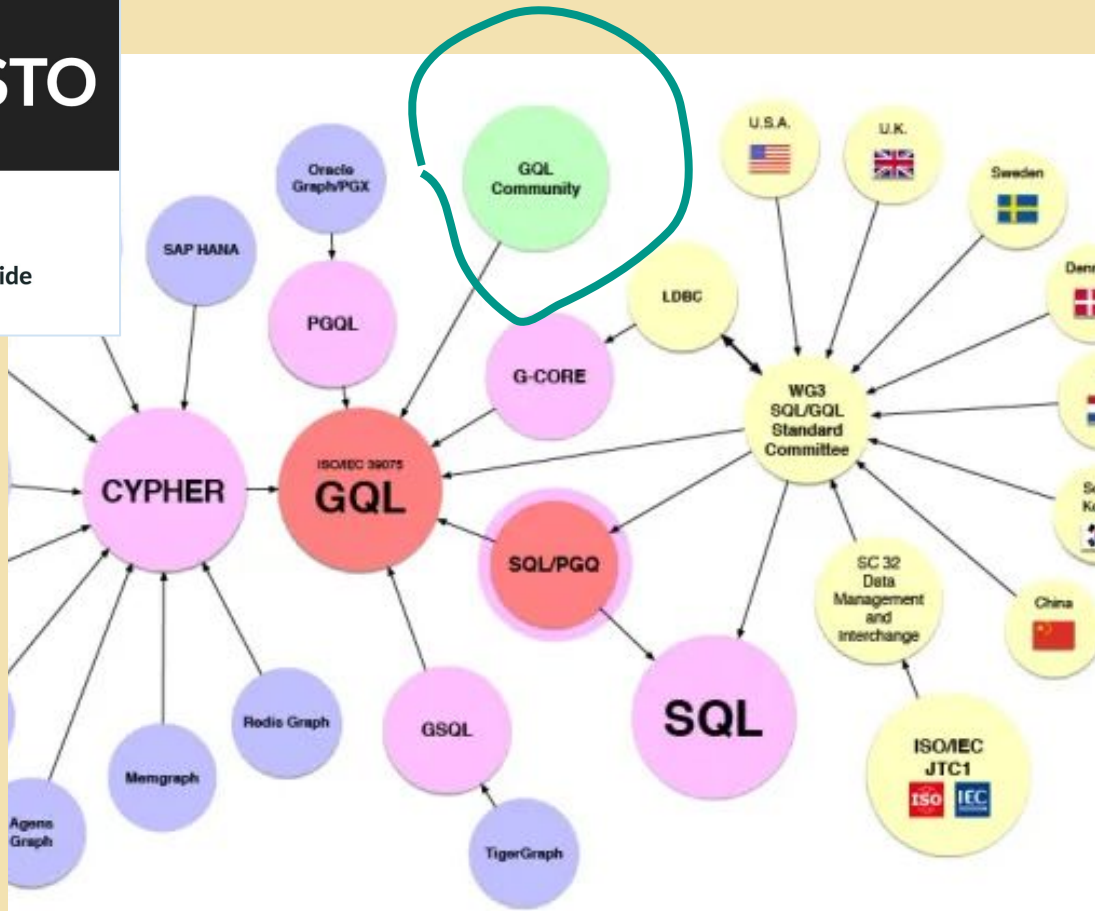
ONE PROPERTY GRAPH QUERY LANGUAGE

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: why we started

“**Schema = types + constraints”:** GQL has graph type, but no constraints
And graph types only support node and edge type subtyping by implication

In late 2022 we restarted work in LDBC on PG schema

LDBC Extended GQL Schema → **LEX**

A proposal for a concrete schema language to feed into GQL++

Design ideas from Neo4j contributions to WG3 in 2018-19
Results of work from the old PGS WG (2020-21) → PG-Schema paper, plus
SQL/JSON schema features, EERM and UML Class Diagrams, SHACL ...

Example: Type keys (identifiers): “key label sets” in GQL graph type ← LEX work

LEX project
direct inputs

PG-Schema: Schemas for Property Graphs

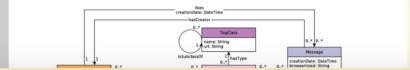
RENZO ANGLES, Faculty of Engineering, Universidad de Talca, Chile
ANGELA BONIFATI, Lyon 1 University & Liris CNRS, France
STEFANIA DUMBRAVA, ENSIIE & SAMOVAR - Institut Polytechnique de Paris, France
GEORGE FLETCHER, Eindhoven University of Technology, Netherlands
ALASTAIR GREEN, LDBC, UK
JAN HIDDERS, Birkbeck, University of London, UK
BEI LI, Google, USA

Jul 2023

Introduction to GQL Schema design

Target: A UML class diagram in ASCII Art "schema patterns"

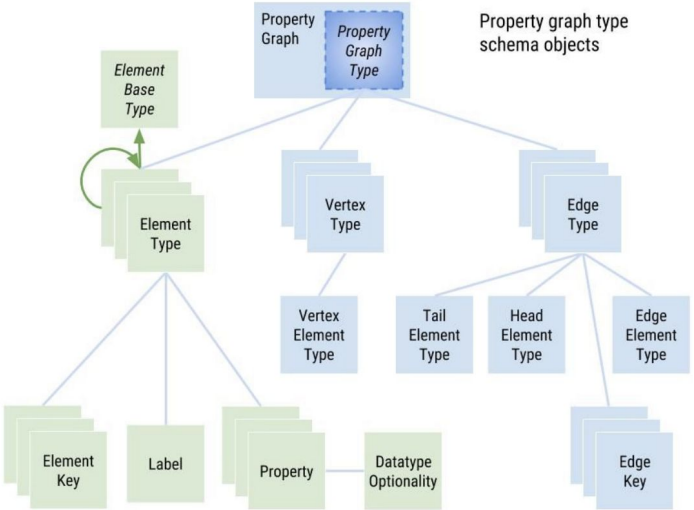
Node and relationship patterns that look like query patterns, gathered into a graph schema or type.
"Draw" the LDBC SNB data model with a keyboard.



```
--    is single (as in both Cypher 9 and the LDBC SNB
--    the use can be inferred from the NENs

(Person), (TagClass), (Tag),
(Message, Post), (Message, Comment),
(Forum),
```

Property Graph Schema

ANSI INCITS sql-pg-2018-0056r1
ANSI INCITS DM32.2-2018-0195r1
ISO/IEC JTC1/SC32 WG3:BNE-022



LEX project
reference points

JSON Schema

Specification Docs Tools Blog

Overview

Getting Started

Reference

Specification

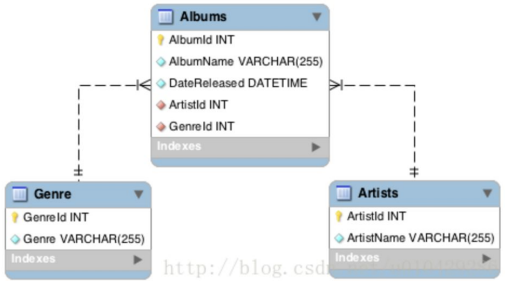
Specification

The current version is *2020-12*! The previous version was 2019-09.

DataBase 专栏收录该内容

参考：http://database.guide/what-is-a-database-schema/

在数据库中， schema（发音 “skee-muh” 或者“skee-mah”，中文叫模式）是数据库的组织
数形式。模式中包含了schema对象，可以是**表**(table)、**列**(column)、**数据类型**(data type)、
关系(relationships)、**主键**(primary key)、**外键**(foreign key)等。数据库模式可以用一个可视化
之间的关系

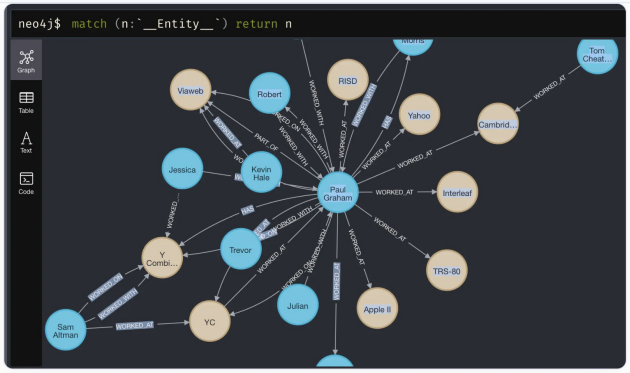


Ontology Modeling with SHACL: Getting Started

Holger Knublauch
Lead Software Developer at TopQuadrant

November 21, 2023

LEX project: why we should keep going



LlamaIndex • May 29, 2024

Introducing the Property Graph Index: A Powerful New Way to Build Knowledge Graphs with LLMs

Knowledge Graphs

All the obvious reasons why we need database and dataset schema ... and newer ones like GraphRAG and graph networks in ML

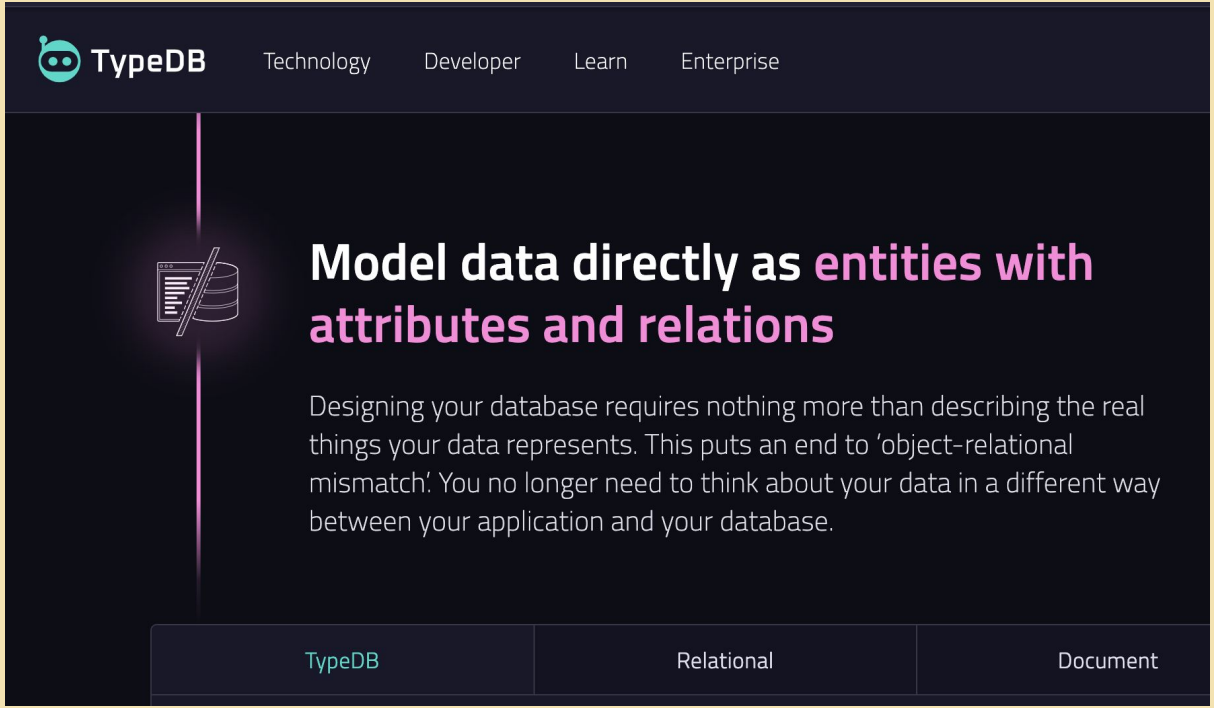
1. Schema-Guided Extraction: Define allowed entity types, relationship types, connections in a schema. The LLM will only extract graph data that conforms t

```python
from llama_index.indices.property_graph import SchemaLLMPathExtractor

entities = Literal["PERSON", "PLACE", "THING"]
relations = Literal["PART_OF", "HAS", "IS_A"]
schema = {
    "PERSON": ["PART_OF", "HAS", "IS_A"],
    "PLACE": ["PART_OF", "HAS"],
    "THING": ["IS_A"],
}

kg_extractor = SchemaLLMPathExtractor(
  llm=llm,
  possible_entities=entities,
  possible_relations=relations,
  kg_validation_schema=schema,
```

Relational inductive biases, deep learning, and graph networks

Peter W. Battaglia[1]*, **Jessica B. Hamrick**[1], **Victor Bapst**[1], **Alvaro Sanchez-Gonzalez**[1], **Vinicius Zambaldi**[1], **Mateusz Malinowski**[1],

LEX project: ... and keep going



Properly unify with standard record-oriented conceptual modelling: EERM/UML...

and other techniques like formal concept analysis, and label and multi-label classification (back to ML)

LEX project **technical themes**

Themes in 2023

- Use cases and requirements
- JSON Schema for property types
- Identifying types using labels

Current work in 2024, heading for 2025

- Refining **JSON Schema** integration (dialect spec, SQL/GQL datatypes)
- **PG-Schema** integration **UML + keys** integration
- “Polymorphic schema”: stating schema using supertypes (covariant schema patterns)
- **Information Schema Graph** there is no Information Schema in GQL
- Experimental Python library: working title is **Grasch**

Possible future work

- schema sub-graphs and transactions
- referential integrity, sub-graph constraints
- SHACL “levelling”

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)

**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 (user-defined types)
- 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 (which doesn't exist yet)?

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

Using JSON Schema and JSON Path would increase developer familiarity

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.”

Quote from my 2023 presentation at the 16th TUC on LEX

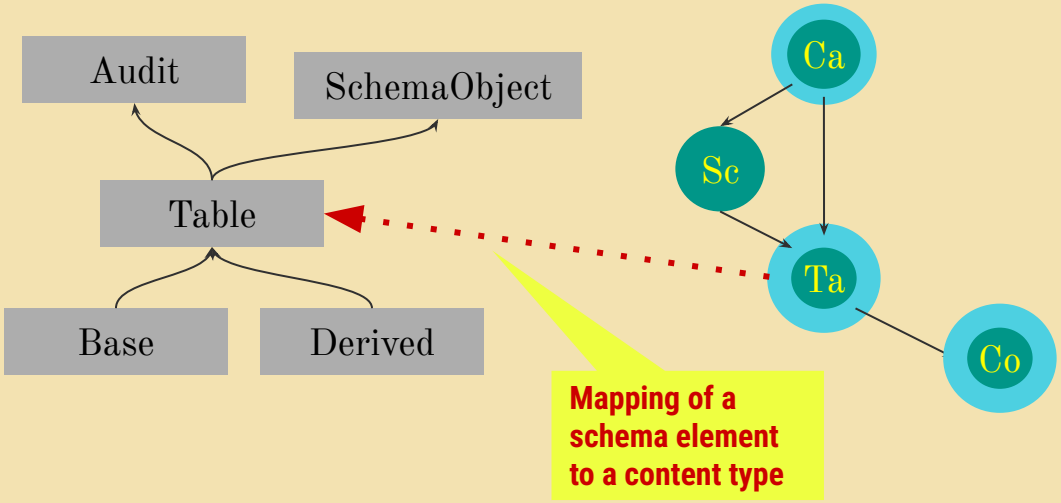
That is what happened. Names induce labels, label sets can be keys in GQL:2024

Information Schema Graph #1 **Schema graph (arbitrary)**

Two parts

Content (attribution) types lattice

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



Information Schema Graph #2 **Catalog (tree)**

It’s like a filesystem: root and path names **/like/this**

The leaf nodes are GQL-schemas

They contain graphs and graph types

It’s an optional part of a GQL implementation

It deals with the problem that the levels in a catalog tend to be influenced by physical architecture (mySQL vs SQLServer to take two extremes)

Information Schema Graph #3 **Type lattice (DAG)**

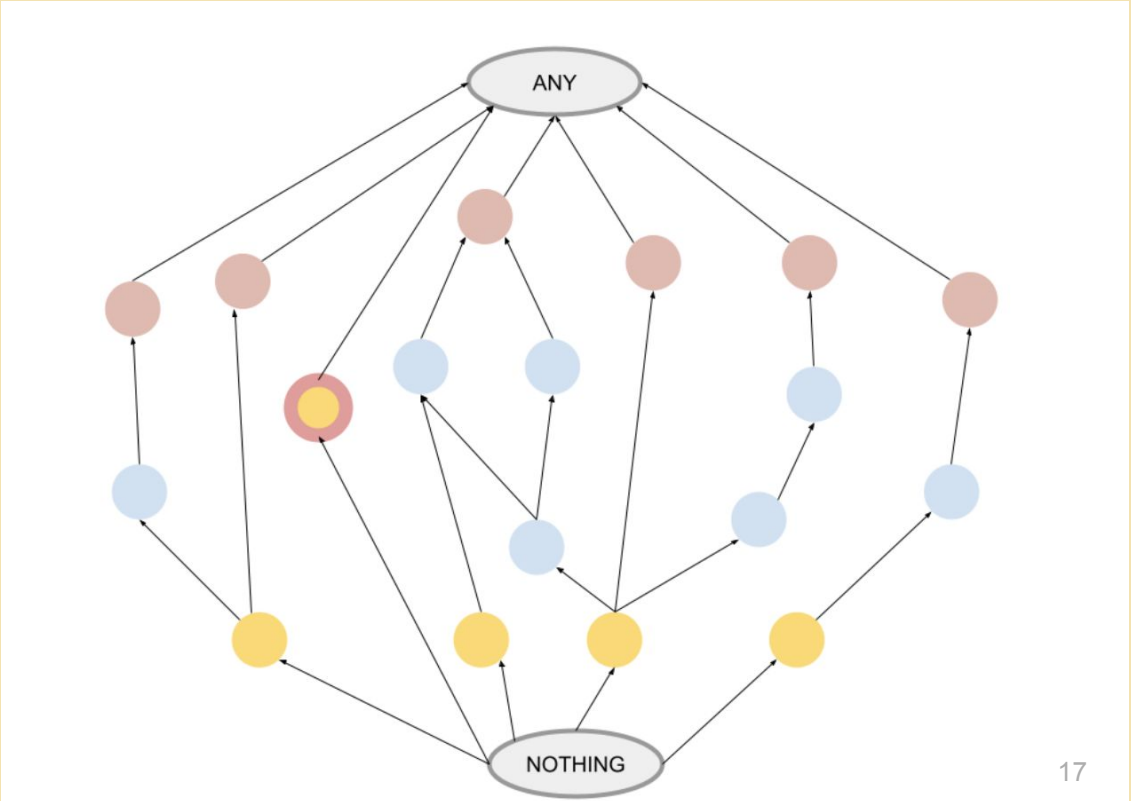
Content Record types
Node Types
Edge Types

≤ = <:

record subtypes (width and depth) → node subtypes → edge subtypes

Optional attributes

Type keys (key label sets)



Experimental **Python** library **Grasch** (early, early days)

Create Catalog, Lattices, Schema Graphs

Store theser graphs in e.g. Kùzu

Integrate JSON Schema

Consumer of the TCK

GraphSchema.py

```
class TypeLatticeNodeIncidentEdges[ORDERABLE_TYPE]:...

class TypePoset(Generic[ORDERABLE_TYPE]):...

class TypeLatticeNodeIncidentEdges('TypeLattice'[ORDERABLE_TYPE]):...

class JoinSemiTypeLattice(Generic[ORDERABLE_TYPE]):...

class MeetSemiTypeLattice(Generic[ORDERABLE_TYPE]):...

class TopHalfTypeLattice(JoinSemiTypeLattice[ORDERABLE_TYPE]):...

class BottomHalfTypeLattice(MeetSemiTypeLattice[ORDERABLE_TYPE]):...

class TypeLattice(JoinSemiTypeLattice[ORDERABLE_TYPE], MeetSemiTypeLattice[ORDERABLE_TYPE]):...

class AnyNothingTypeLattice(TypeLattice[ORDERABLE_TYPE]):...

class ContentTypeInterface(Protocol):...

@total_ordering  11 usages  Alastai
class ContentType(Orderable, ContentTypeInterface):...

class ElementType(ContentTypeInterface):...

class NodeType(ElementType):...

class EdgeType(ElementType):...

class GraphTypeInterface(Protocol):...
```

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.

Remaining slides from 2022 provide some additional context on GQL gaps/futures

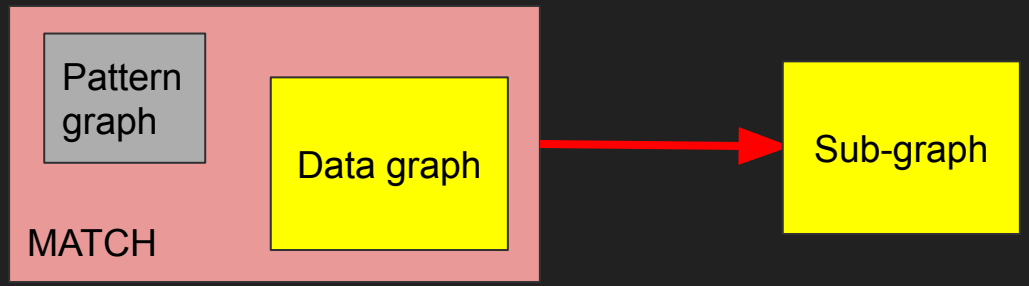
Appendix

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

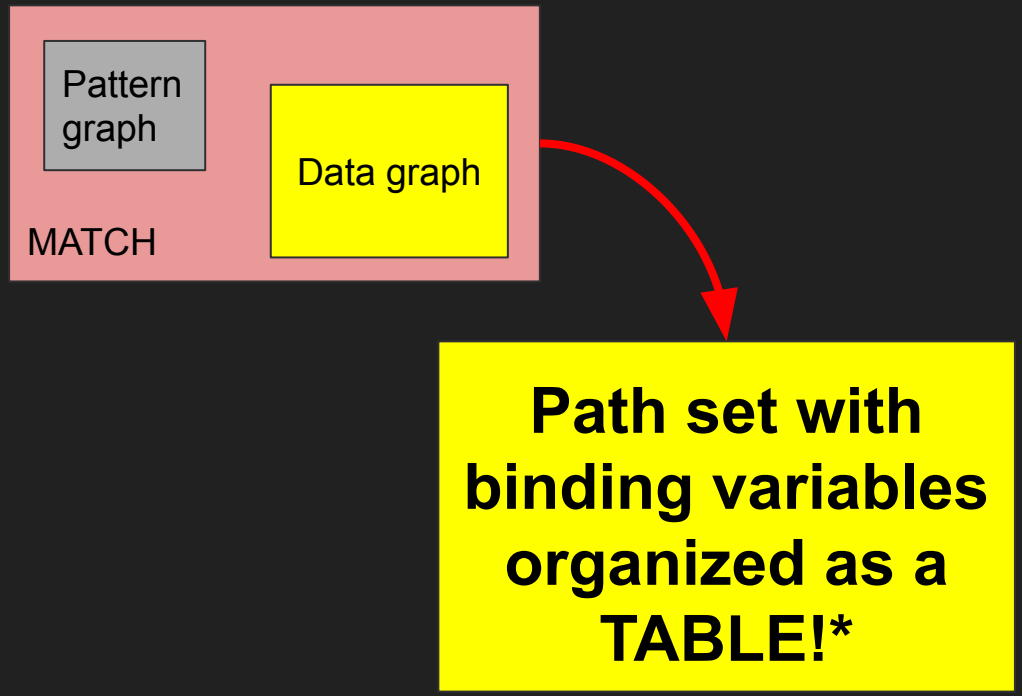
Some of the slides from that presentation follow talking about sub-graph extraction (and compositional graph-projecting queries)

GPM: does it produce sub-graphs?

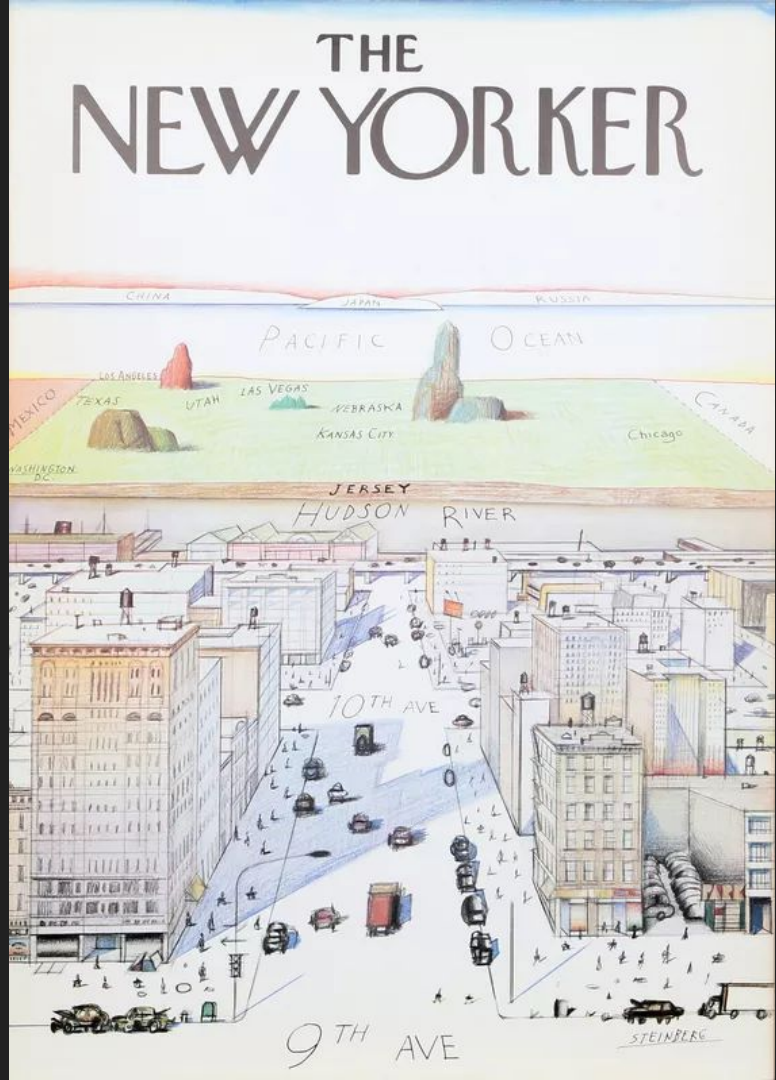
For semi-political and semi-technical reasons people don’t like talking about graph pattern matching in PGQ/GQL in the following (classical) way



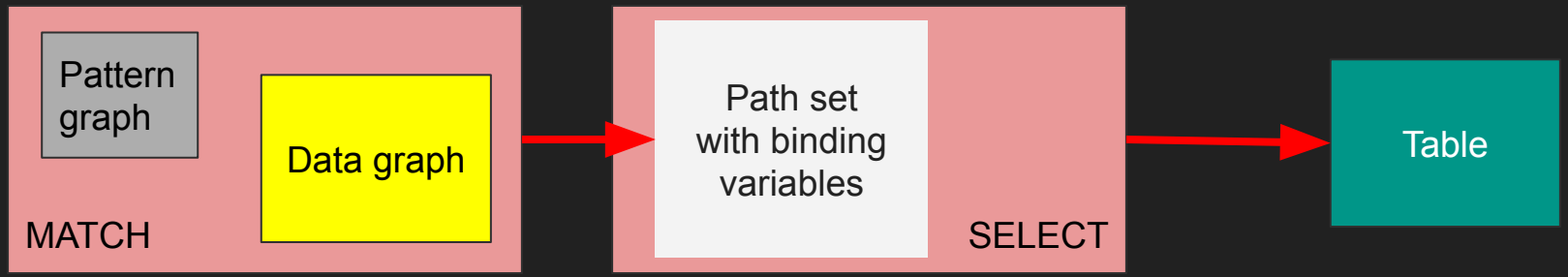
Or does it produce tables?



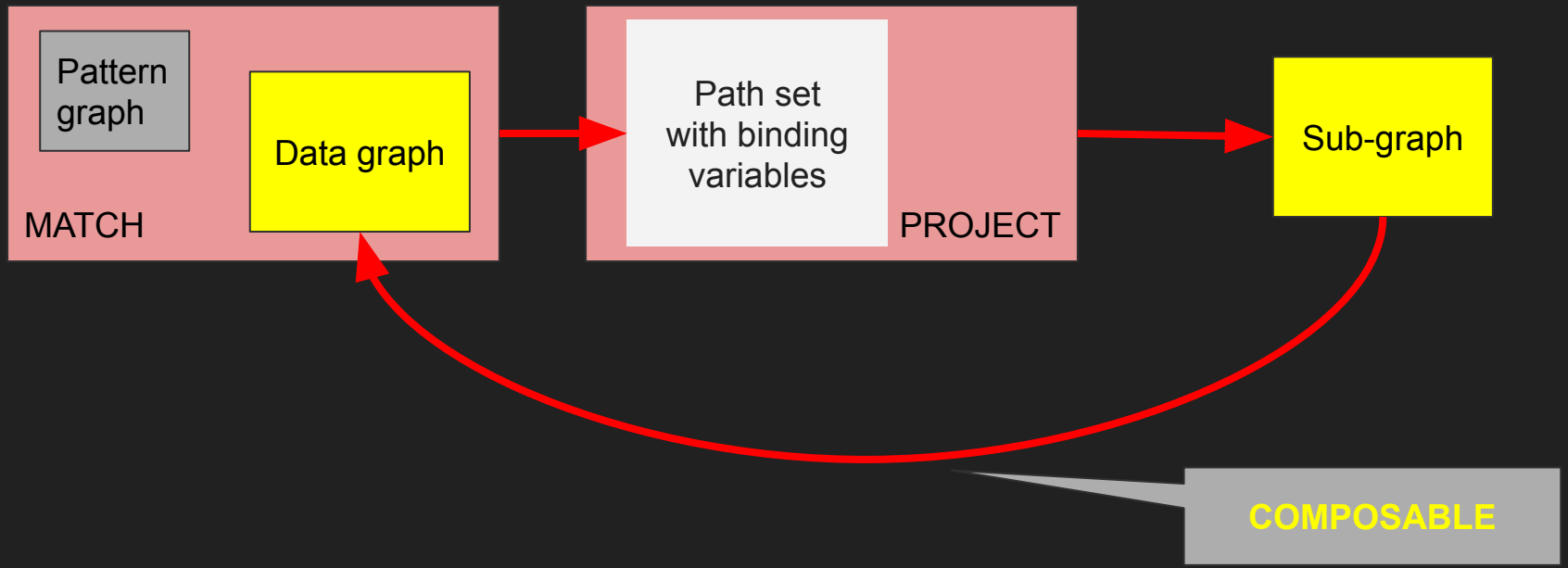
*Can be viewed as a sub-graph



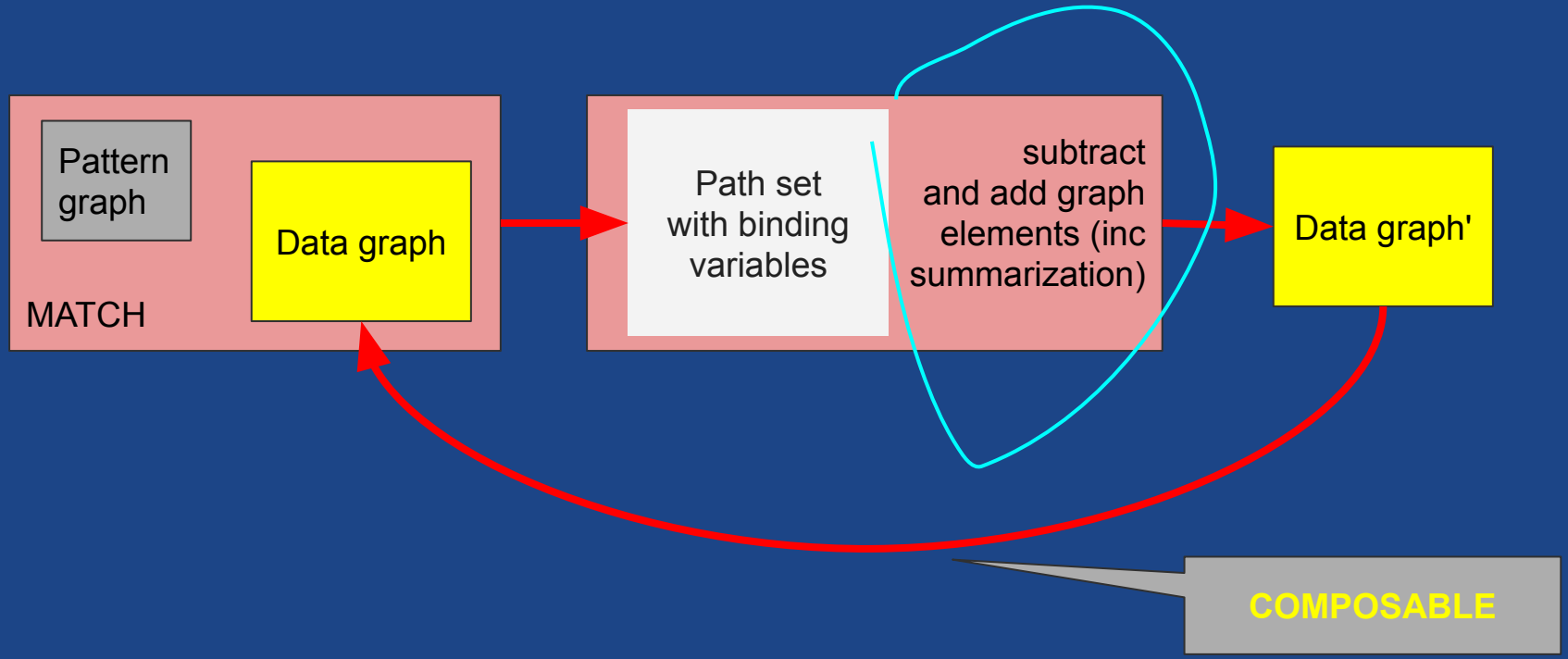
What happens in PGQ is this



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



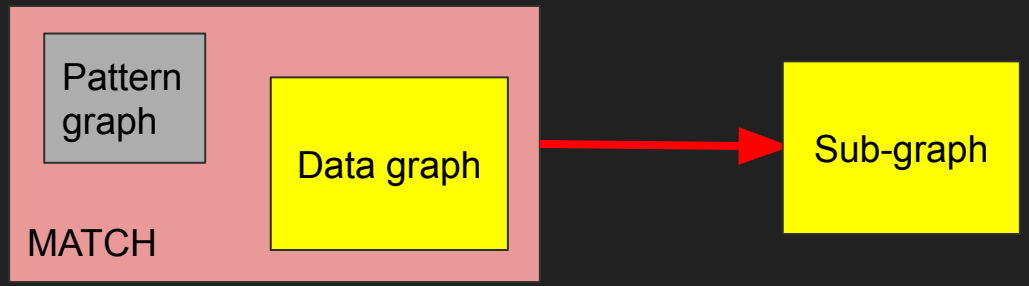
And what really needs to happen in GQL is this ⟶ views



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