Formalizing GQL

presentation by Leonid Libkin

University of Edinburgh and RelationalAI

LDBC TUC 2023
Standards are great but not for academics
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics
You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics.

You want to write a paper about pattern matching and start with the syntax.
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax
And then you want to work with it but it’s like “find the rabbit”
Our Goal

GQL to the (academic) masses
Our Goal

GQL to the (academic) masses
- Distill
Our Goal

GQL to the (academic) masses
- Distill
- Formalize, provide the semantics
Our Goal

<table>
<thead>
<tr>
<th>GQL to the (academic) masses</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Distill</td>
</tr>
<tr>
<td>- Formalize, provide the semantics</td>
</tr>
<tr>
<td>- Plus initial results</td>
</tr>
</tbody>
</table>
Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings
Our Goal

GQL to the (academic) masses
- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Word of caution
GQL is a moving target
We do our best.....
Our Goal

GQL to the (academic) masses
- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Word of caution
GQL is a moving target
We do our best.....
## Our Goal

<table>
<thead>
<tr>
<th>GQL to the (academic) masses</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Distill</td>
</tr>
<tr>
<td>- Formalize, provide the semantics</td>
</tr>
<tr>
<td>- Plus initial results</td>
</tr>
<tr>
<td>- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc</td>
</tr>
<tr>
<td>- Outline research challenges that GQL brings</td>
</tr>
</tbody>
</table>

### Papers/talks

- Last year: **SIGMOD’22** on pattern matching (WG3+FSWG)
- Then **PODS’23** paper: formalization of pattern matching
  - also subject of KR 2023 keynote
- **EDBT/ICDT 2023** keynote: core GQL
  - talk + paper

---

**Word of caution**

GQL is a moving target
We do our best.....
GQL in a Nutshell

Core of the language

Graph Pattern Matching

⇝

graph

relation
GQL in a Nutshell

Core of the language

Graph Pattern Matching

Relational Querying

Graph ⇝ relation
GQL in a Nutshell

Graph Pattern Matching

Core of the language

Relational Querying

⇝

∪

∩

θ
GQL in a Nutshell

- Core of the language
- Graph Pattern Matching
- Relational Querying
- extras (e.g., combining graph and table)
GQL in a Nutshell

- Core of the language
- Graph Pattern Matching
- Relational Querying
- Updates, etc.

- Graph
- Relation

⇝

∪
∩
−
θ

+ extras
(e.g., combining graph and table)
The Core:
Graph Pattern Matching

PODS 2023
Pattern calculus in a nutshell: PODS 23
Pattern calculus in a nutshell: PODS 23

Node pattern \( \nu := (x : \ell) \)
Node pattern $\nu := (x : \ell)$ match an $\ell$-labeled node, assign to a variable
Pattern calculus in a nutshell: PODS 23

Node pattern  \( \nu := (x : \ell) \)  match an \( \ell \)-labeled node, assign to a variable

Edge pattern  \( \alpha := \xrightarrow{x : \ell} | \xleftarrow{x : \ell} | \xrightarrow{x : \ell} \)
Pattern calculus in a nutshell: PODS 23

Node pattern \( \nu := (x : \ell) \)  
match an \( \ell \)-labeled node, assign to a variable

Edge pattern \( \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \xrightarrow{-} \)  
\( \ell \)-labeled edge directed left/right/any-directed, assign to a variable
Pattern calculus in a nutshell: PODS 23

Node pattern \( \nu := (x : \ell) \) match an \( \ell \)-labeled node, assign to a variable

Edge pattern \( \alpha := \longrightarrow | \longleftarrow | \overset{\ell}{-} \) \( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Both \( x \) and \( \ell \) are optional
Pattern calculus in a nutshell: PODS 23

Node pattern \( \nu := (x : \ell) \) match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern \( \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \underbrace{--------}_{\ell} \) \( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns \( \pi := \nu | \alpha | \pi \pi | \pi + \pi | \pi^{n,m} | \pi^{\langle \theta \rangle} \quad 0 \leq n \leq m \leq \infty \)
Pattern calculus in a nutshell: PODS 23

Node pattern
\[ \nu := (x : \ell) \]
match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern
\[ \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \xrightleftharpoons{\ell} \]
\( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns
\[ \pi := \nu | \alpha | \pi \pi | \pi + \pi | \pi^{n..m} | \pi(\theta) \quad 0 \leq n \leq m \leq \infty \]

node edge concatenation union repetition selection with condition n-to-m times
Pattern calculus in a nutshell: PODS 23

Node pattern
\[ \nu := (x : \ell) \]
match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern
\[ \alpha := x : \ell \quad x : \ell \quad x : \ell \]
\( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns
\[ \pi := \nu \mid \alpha \mid \pi \pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi(\theta) \quad 0 \leq n \leq m \leq \infty \]
node edge concatenation union repetition selection with condition n-to-m times

Conditions
\[ \theta := x . a = c \mid x . a = y . b \mid \theta \lor \theta \mid \theta \land \theta \mid \neg \theta \]
Pattern calculus in a nutshell: PODS 23

Node pattern \[ \nu := (x : \ell) \] match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern \[ \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \xrightarrow{\ell} \] \( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns \[ \pi := \nu \mid \alpha \mid \pi \pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle \quad 0 \leq n \leq m \leq \infty \]

- node
- edge
- concatenation
- union
- repetition
- selection with condition
- n-to-m times

Conditions \[ \theta := x . a = c \mid x . a = y . b \mid \theta \lor \theta \mid \theta \land \theta \mid \neg \theta \]

- key-value comparisons
- Boolean combinations
### Pattern calculus in a nutshell: PODS 23

#### Node pattern

\[ \nu := (x : \ell) \]

match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

#### Edge pattern

\[ \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \longrightarrow \]

\( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

#### Patterns

\[ \pi := \nu | \alpha | \pi \pi | \pi + \pi | \pi^{n..m} | \pi(\theta) \quad 0 \leq n \leq m \leq \infty \]

<table>
<thead>
<tr>
<th>node</th>
<th>edge</th>
<th>concatenation</th>
<th>union</th>
<th>repetition</th>
<th>selection with condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )-to-( m ) times</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Conditions

\[ \theta := x \cdot a = c | x \cdot a = y \cdot b | \theta \lor \theta | \theta \land \theta | \neg \theta \]

key-value comparisons | Boolean combinations

#### Queries

\[ Q := \sigma \pi | p = \sigma \pi | Q, Q \]
Pattern calculus in a nutshell: PODS 23

Node pattern \[ \nu := (x : \ell) \] match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern \[ \alpha := x : \ell \mid x : \ell \mid x : \ell \] \( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns \[ \pi := \nu \mid \alpha \mid \pi \pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi(\theta) \quad 0 \leq n \leq m \leq \infty \]

node edge concatenation union repetition selection with condition n-to-m times

Conditions \[ \theta := x . a = c \mid x . a = y . b \mid \theta \lor \theta \mid \theta \land \theta \mid \neg \theta \]

key-value comparisons Boolean combinations

Queries \[ Q := \sigma \pi \mid p = \sigma \pi \mid Q, Q \]

ensure finitely many paths
Pattern calculus in a nutshell: PODS 23

Node pattern
\[ \nu := (x : \ell) \]
match an \( \ell \)-labeled node, assign to a variable

Both \( x \) and \( \ell \) are optional

Edge pattern
\[ \alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \underrightarrow{\ell} \]
\( \ell \)-labeled edge directed left/right/any-directed, assign to a variable

Patterns
\[ \pi := \nu \mid \alpha \mid \pi \pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi(\theta) \quad 0 \leq n \leq m \leq \infty \]

node edge concatenation union repetition selection with condition n-to-m times

Conditions
\[ \theta := x . a = c \mid x . a = y . b \mid \theta \lor \theta \mid \theta \land \theta \mid \neg \theta \]

key-value comparisons Boolean combinations

Queries
\[ Q := \sigma \pi \mid p = \sigma \pi \mid Q, Q \]
ensure finitely many paths name matched path
Pattern calculus in a nutshell: PODS 23

Node pattern
$$\nu := (x : \ell)$$
match an $\ell$-labeled node, assign to a variable

Both $x$ and $\ell$ are optional

Edge pattern
$$\alpha := \xrightarrow{\ell} | \xleftarrow{\ell} | \xrightarrow{x}$$
$\ell$-labeled edge directed left/right/any-directed, assign to a variable

Patterns
$$\pi := \nu | \alpha | \pi \pi | \pi + \pi | \pi^{n..m} | \pi(\theta) \quad 0 \leq n \leq m \leq \infty$$

node edge concatenation union repetition selection with condition n-to-m times

Conditions
$$\theta := x.a = c | x.a = y.b | \theta \lor \theta | \theta \land \theta | \neg \theta$$
key-value comparisons Boolean combinations

Queries
$$Q := \sigma \pi | p = \sigma \pi | Q, Q$$
ensure finitely many paths name matched path join
Semantics — Idea

\[ Q = \pi_1, \pi_2, \ldots, \pi_n \quad \text{with variables} \quad x_1, x_2, \ldots, x_m \]

MATCH result: a tuple of paths + a table

GQL and SQL/PGQ only keep the table
What’s in the table?

- Graph elements
  - Nodes
  - Edges
- Paths (when named: $x = \pi$)
- Lists of graph elements
What’s in the table?

- Graph elements
  - Nodes
  - Edges
- Paths (when named: $x = \pi$)
- Lists of graph elements

\[
\pi = \ldots(x)\ldots \xrightarrow{y} \ldots(\ldots(u)\ldots)^{n,m} \ldots(\ldots \xrightarrow{W} \ldots)^{\varepsilon,k} \ldots
\]

\[
n \quad e \quad [n_1, n_2, \ldots] \quad [e_1, e_2, \ldots]
\]
What’s in the table?

- Graph elements
  - Nodes
  - Edges
- Paths (when named: $x = \pi$)
- Lists of graph elements

$$\pi = \ldots(x)\ldots \xrightarrow{y} \ldots \ldots (\ldots(u)\ldots)^{n,m} \ldots (\ldots \xrightarrow{W} \ldots)^{\ell,k} \ldots$$

- $n$  
- $e$  
- $[n_1, n_2, \ldots]$  
- $[e_1, e_2, \ldots]$  

Tables may have nulls: $(x) + \xrightarrow{y}$

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>e</td>
</tr>
</tbody>
</table>
What we have done in the PODS paper

- **Formal semantics** for well-typed expressions
- **Type system**: when a variable is assigned:
  - a graph element, or a list, or could be assigned NULL
- **Complexity**
  - **PSPACE** data complexity of enumeration
  - Not surprising: there are many paths
  - Note: Cypher is **NP-hard**. Things may work in practice, but not in theory!
- **Expressivity**
  - Subsumes CRPQs, inverses, unions, nested regular expressions, regular queries
Relational Querying in GQL (streamlined)

Basic Operations on Tables
- **RETURN** (projection)
- **LET** (add columns)
- **FILTER** (selection)
- **FOR** (unnest for lists)
- another **MATCH** (join with the current working table)

Union, Intersection, Difference
If $Q_1$ and $Q_2$ are GQL queries, then so are
- $Q_1$ **UNION** $Q_2$
- $Q_1$ **INTERSECT** $Q_2$
- $Q_1$ **EXCEPT** $Q_2$
- $Q_1$ **OTHERWISE** $Q_2$
Multiple Graphs

**USE** G1
  **MATCH** $\pi_1$
  **WHERE** $\theta_1$
  **RETURN** $L_1$

**NEXT USE** G2
  **MATCH** $\pi_2$
  **WHERE** $\theta_2$
  **RETURN** $L_2$

..........................

**NEXT USE** G$n$
  **MATCH** $\pi_n$
  **WHERE** $\theta_n$
  **RETURN** $L_n$
ICDT ’23 Paper:
“A Researcher’s Digest of GQL”

Idea:

A syntax closer to actual GQL
But still OK for academics to use for research
**Syntax: PATTERNS**

<table>
<thead>
<tr>
<th><strong>PATH PATTERN</strong></th>
<th>For $x \in \text{Vars}$, $\ell \in \mathcal{L}$, $0 \leq n \leq m \in \mathbb{N}$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(descriptor)</td>
<td>$\delta := x : \ell \ \text{WHERE} \ \theta$ $x$, $\ell$, and \text{WHERE} $\theta$ are optional</td>
</tr>
<tr>
<td>(path pattern)</td>
<td>$\pi := (\delta)$ $\text{WHERE} \ \theta$ $\text{WHERE} \ \theta$ are optional</td>
</tr>
<tr>
<td></td>
<td>$[-\delta] \rightarrow \leftarrow[-\delta] \rightarrow[-\delta]$ $\pi \pi$ $\pi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EXPRESSION and CONDITION</strong></th>
<th>For $x \in \text{Vars}$, $\ell \in \mathcal{L}$, $a \in \mathcal{K}$, $c \in \text{Const}$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(expression)</td>
<td>$\chi := x \mid x.a \mid c$ $\chi = \chi \mid \chi \chi \mid \chi \text{ IS NULL}$</td>
</tr>
<tr>
<td>(condition)</td>
<td>$\theta := x : \ell \mid \exists {Q}$ $\theta \text{ OR} \theta \mid \theta \text{ AND} \theta \mid \text{ NOT} \theta$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>GRAPH PATTERN</strong></th>
<th>For $x \in \text{Vars}$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(path mode)</td>
<td>$\mu := (\text{ALL} \mid \text{ANY}) [\text{SHORTEST}] [\text{TRAIL} \mid \text{ACYCLIC}]$</td>
</tr>
<tr>
<td>(graph pattern)</td>
<td>$\Pi := \mu [x =] \pi \mid \Pi, \Pi$</td>
</tr>
</tbody>
</table>

**Figure 2** Syntax of GQL
### Syntax: QUERIES

**Clause and Query** For \( k \geq 0, \ell \geq 1, \) and \( x, y, x_1, \ldots, x_k \in \text{Vars, and } G \in \mathbb{G} : \)

<table>
<thead>
<tr>
<th>Clause</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C := )</td>
<td>MATCH II</td>
</tr>
<tr>
<td>|</td>
<td>LET ( x = \chi )</td>
</tr>
<tr>
<td>|</td>
<td>FOR ( x ) IN ( y )</td>
</tr>
<tr>
<td>|</td>
<td>FILTER ( \theta )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linear Query</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L := )</td>
<td>USE ( G ) ( L )</td>
</tr>
<tr>
<td>| |</td>
<td>( C ) ( L )</td>
</tr>
<tr>
<td>| |</td>
<td>RETURN ( \chi_1 ) AS ( x_1 ), \ldots, ( \chi_k ) AS ( x_k )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q := )</td>
<td>( L )</td>
</tr>
<tr>
<td>| |</td>
<td>USE ( G ) { ( Q_1 ) THEN ( Q_2 ) \ldots ) THEN ( Q_\ell ) }</td>
</tr>
<tr>
<td>| |</td>
<td>( Q ) INTERSECT ( Q )</td>
</tr>
</tbody>
</table>
A Researcher's Digest of GQL

Above, for a pattern where defined in Section 3.

Semantics of Repetition

Other cases of the forward edge patterns are treated by moving the label and conditions outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Repetition

Above, for a pattern where defined in Section 3.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

Semantics of Concatenation, Union, and Conditioning

Note that since \( n_2 \neq 0 \) is assumed to be well-formed, all variables shared by \( n_1 \) and \( n_2 \) are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.
1:14 A Researcher’s Digest of GQL

- Remark 13. Recall that different graphs may share nodes and edges. Hence the condition $(\mu(x) \neq \text{Dom}(x^T))$ above, does not imply that $\mu(x)$ is a node or an edge in $G$, but does not imply that it was matched in $G$.

The semantics $\text{Sem}_G$ of a condition $\theta$ is an element in $\{-\text{true}, \text{false}, \text{null}\}$ that is evaluated with respect to a binding $\mu$ and a graph $G$, and is defined as follows:

$$\text{Sem}_G(\theta) = \begin{cases} \text{true} & \text{if } \mu(x) \in N \land \exists e \in E \land \theta \\ \text{false} & \text{if } \mu(x) \in N \land \exists e \in E \land \theta \\ \text{null} & \text{otherwise} \end{cases}$$

4.5 Semantics of Queries

Clauses and queries are interpreted as functions that operate on tables. These tables are our abstraction of SQL's working tables.

**Definition 12.** A table $T$ is a set of bindings that have the same domains, referred to as Dom$(T)$.

Note that tables do not have hersones: two different bindings in a table might associate a variable to values of incompatible types.

**Semantics of Clauses**

The semantics $\text{Sem}_G(Q)$ of a clause $Q$ is a function that maps tables into tables, and is parameterized by a graph $G$. Patterns, conditions and expressions in a clause are evaluated with respect to $G$.

$$\text{Sem}_G(Q) = \bigcup_{\mu} \{ (x, \mu(x)) \mid \mu(x) \in \text{Dom}(Q) \land Q \}

Note that if $G$ has a variable that already occurs in Dom$(T)$, a join is performed. Unlike in the case of path patterns and graph patterns, this join can involve variables bound to lists or paths. While this is not problematic mathematically, it might be disallowed in future iterations of GQL.

N. Francis et al.

If $\alpha \notin \text{Dom}(Q)$, then

$$\text{LET } x \mapsto \text{Sem}_G(Q) \text{ in } \mu \rightarrow \mu \text{ if } \mu(x) \neq \text{null} \text{ and } \mu(x) \neq \text{null} \text{ then}$$

$$\text{FILTER } \text{Sem}_G(Q) \text{ in } \mu \rightarrow \mu \text{ if } \mu(x) \neq \text{null} \text{ and } \mu(x) \neq \text{null} \text{ then}$$

$$\text{IF } x \text{ in } \text{Sem}_G(Q) \text{ and } \forall y \in \text{Sem}_G(Q) \text{, then } \text{LET } x \mapsto \text{Sem}_G(Q) \text{ in } \mu \rightarrow \mu \text{ if } \mu(x) \neq \text{null} \text{ and } \mu(x) \neq \text{null} \text{ then}$$

**Semantics of Linear Queries**

$$\text{LET } Q \text{ in } \text{Sem}_G(Q) \text{ in } \mu \rightarrow \mu \text{ if } \mu(x) \neq \text{null} \text{ and } \mu(x) \neq \text{null} \text{ then}$$

**Semantics of Queries**

The output of a query $Q$ is defined as

$$\text{Output}(Q) = \text{Sem}_G(Q) \text{ in } \mu \rightarrow \mu \text{ if } \mu(x) \neq \text{null} \text{ and } \mu(x) \neq \text{null} \text{ then}$$

5 A Few Known Discrepancies with the GQL Standard

In pursuing the goal of introducing the key features of GQL to the research community, we inevitably had to make decisions that resulted in discrepancies between our presentation and the 50+ pages of the forthcoming Standard. In this section, we describe a non-exhaustive list of differences between the actual GQL Standard and our digest. To start with, in all our formal development we assumed that queries are given by their syntax trees, which results from parsing them. Since we completely omitted such parsing-related aspects as parentheses, operator precedence etc. Also we note that many GQL features, even those described here, are optional, and not every implementation is obliged to have them all.

3 Note that null is treated as just an idc
What Do the Papers Omit?
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities

Aggregation
- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
  - e.g. $\text{SUM}(e.\text{weight}) < 100$
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities

Aggregation
- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
  - e.g. $\text{SUM(e.weight)} < 100$

Procedure calls
- inlined: `CALL { ... }
- named: `CALL <proc-name> (<params>)`
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities

Aggregation
- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
  - e.g. \( \text{SUM(e.weight)} < 100 \)

Procedure calls
- inlined: \texttt{CALL \{\ldots\}}
- named: \texttt{CALL <proc-name> (<params>)}

Updates
insert, set, delete
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities

Aggregation
- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
  - e.g. \( \text{SUM}(e.\text{weight}) < 100 \)

Procedure calls
- inlined: \texttt{CALL \{ ... \}}
- named: \texttt{CALL <proc-name>(<params>)}

Updates
insert, set, delete

Extra path modes
e.g. \texttt{TRAIL} along several paths \( \pi_1, ..., \pi_n \)
What Do the Papers Omit?

Bag semantics
Our semantics is correct up to multiplicities

Aggregation
- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
  - e.g. \( \text{SUM}(e.\text{weight}) < 100 \)

Procedure calls
- inlined: `CALL { ... }`
- named: `CALL <proc-name> (<params>)`

Updates
insert, set, delete

Extra path modes
- e.g. `TRAIL` along several paths \( \pi_1, \ldots, \pi_n \)

Catalog operations

Data types and value expressions

Predicates (including handling nulls)
Open Questions
Open Questions

The first version of the standard will be published in 2024.
It all makes sense if you take a quick look....
Open Questions

The first version of the standard will be published in 2024.
It all makes sense if you take a quick look....
Open Questions

The first version of the standard will be published in 2024.
It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC.
We need to do research on GQL now!
Our work gives you the platform.
Open Questions

The first version of the standard will be published in 2024. It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC. We need to do research on GQL now! Our work gives you the platform.

Questions:
Is it good? Is it usable? Or the best simply because everything else is average?
Open Questions

The first version of the standard will be published in 2024. It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC. We need to do research on GQL now! Our work gives you the platform...

Questions:
Is it good? Is it usable? Or the best simply because everything else is average?
Directions for Research:
Theoretician’s Comfort zone

Expressive Power and Complexity

- Clean Language Fragments and Extensions
  - Think of First-Order Logic and everything we know about its power, complexity, and that of CQs, UCQs, Datalog, etc etc
- DB theory folks are really good at this
Directions for Research: Theoretician’s Comfort zone

Expressive Power and Complexity
- Clean Language Fragments and Extensions
  - Think of First-Order Logic and everything we know about its power, complexity, and that of CQs, UCQs, Datalog, etc etc
- DB theory folks are really good at this

Query processing and Optimization
- Containment, Equivalence, ...
  - GQL goes much beyond CRPQs
- Practical algorithms, data structures
Directions for Research: Extra features
Directions for Research: Extra features
Directions for Research: Extra features

Updates

- Updating graphs is not a trivial matter
- Many alternative semantics need to be explored
  (even the case of Cypher was highly problematic)
Directions for Research: Extra features

Updates

- Updating graphs is not a trivial matter
- Many alternative semantics need to be explored (even the case of Cypher was highly problematic)

Design analysis: alternatives, suggestions, holes

- Many examples: e.g., dealing with group variables. Are the current restrictions (e.g., no comparisons) necessary?
- Can variables be used non-locally?
  - E.g. `MATCH (x) ( -[y:a]-> WHERE x.k+y.k=10 )* (z)`
- Implications for complexity?
Directions for Research: What is missing
Directions for Research: What is missing

- Graph-to-Graph Queries
  - Like Cypher, GQL is an engine for turning graphs into relations
  - This has many limitations: how to do views? subqueries?
  - Need design principles for graph-to-graph languages.
Directions for Research: What is missing

- **Graph-to-Graph Queries**
  - Like Cypher, GQL is an engine for **turning graphs into relations**
  - This has many limitations: how to do views? subqueries?
  - Need design principles for graph-to-graph languages.

**Schemas and Constraints**
- Taken for granted for relational databases
- Much less work on property graphs but it’s coming
- **PG-KEYS** (SIGMOD’21), **PG-SCHEMA** (SIGMOD’23)
Final Thoughts
Final Thoughts

After 6 years of work, GQL will become an ISO Standard
Final Thoughts

After 6 years of work, GQL will become an ISO Standard

There is a reason to be happy but:

- *When you jump for joy, beware that no one moves ground under your feet*
- NDL was a standard too but lost to relational
Final Thoughts

After 6 years of work, GQL will become an ISO Standard

There is a reason to be happy but:

- *When you jump for joy, beware that no one moves ground under your feet*
- NDL was a standard too but lost to relational

Relational querying of graphs is very active too

- SQL/PGQ in relational systems
- Relational languages for KG, e.g. RelationalAI
- Native Graph Querying vs Relational Graph Querying will be playing out in the next N years
  - *The Register, 6 March 2023*: “The Great Graph Debate: Revolutionary concept in databases or niche curiosity”
Final Thoughts

There is a reason to be happy but:
- When you jump for joy, beware that no one moves ground under your feet
- NDL was a standard too but lost to relational

Relational querying of graphs is very active too
- SQL/PGQ in relational systems
- Relational languages for KG, e.g. RelationalAI
- Native Graph Querying vs Relational Graph Querying will be playing out in the next N years
  - The Register, 6 March 2023: "The Great Graph Debate: Revolutionary concept in databases or niche curiosity"

Our community has a lot to offer in this debate — on both fronts
It takes a (cat) team