

QL

or “froGQL: Worst-Case Optimal Joins and Type-Driven Optimization for Lightweight GQL” by Felipe Avendaño, Jean Paul Duchens, **Sebastián Ferrada** and Matías Toro.

ISWC 2026 – Bari, Italy.

froGQL: a graph database in a single file

Graph, schema and indexes in one file, running inside your application



Embeddable

no server: Python, Node.js, Rust, browser (Web Assembly)



Fast

worst-case optimal joins over triples



Typed

proves queries empty before running them

Speaks GQL: enough for 13 of the 14 LDBC SNB Interactive Complex query templates

Embeddable, easy to import and query data

1

```
frogql.import_json_str(path, graph)
```

14k nodes, 69k edges
in about a second

2

```
conn = frogql.open(path)
```

one 7.3 MB file
opened in 0.1 s

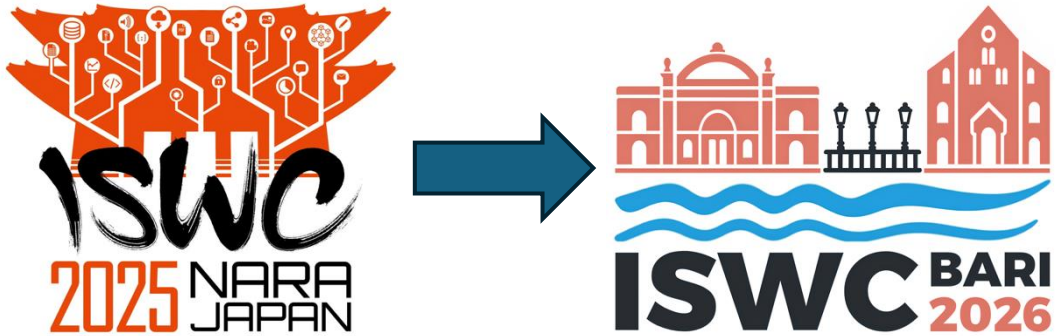
3

```
rows = conn.execute(query)
```

plain Python dicts
ready for pandas

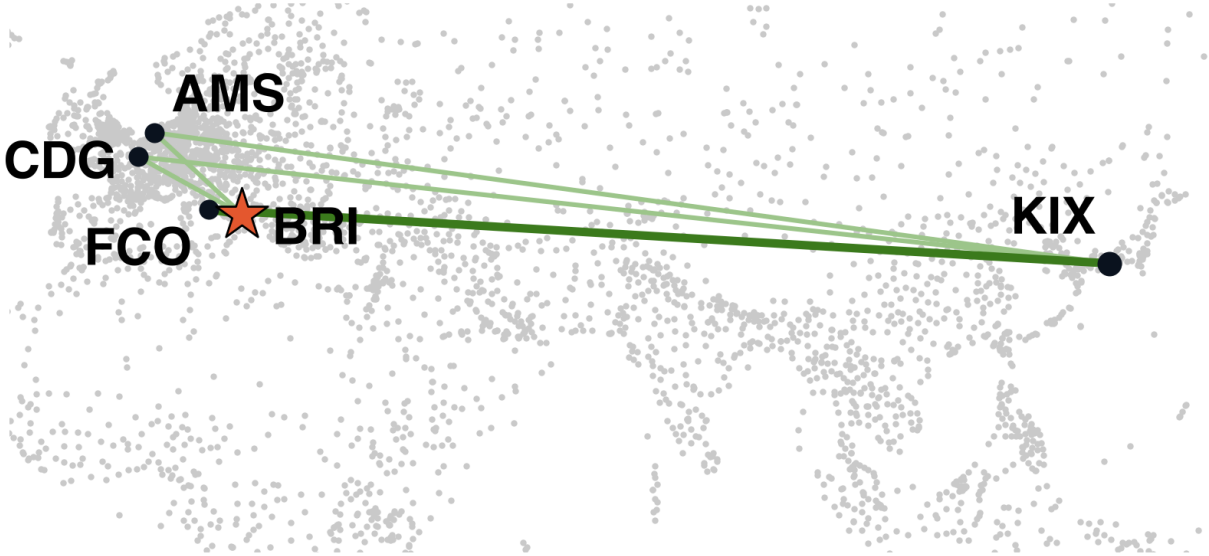
No server, no config, no port: commit the .gdb, email it, or ship it with your paper

How did we get here?




```
MATCH p = ALL SHORTEST
  (:Airport {iata:'KIX'})
  -[:ROUTE]->*
  (:Airport {iata:'BRI'})
RETURN NODES(p)

~70 ms · 3 rows
```



One of 3: Osaka → Rome → Bari

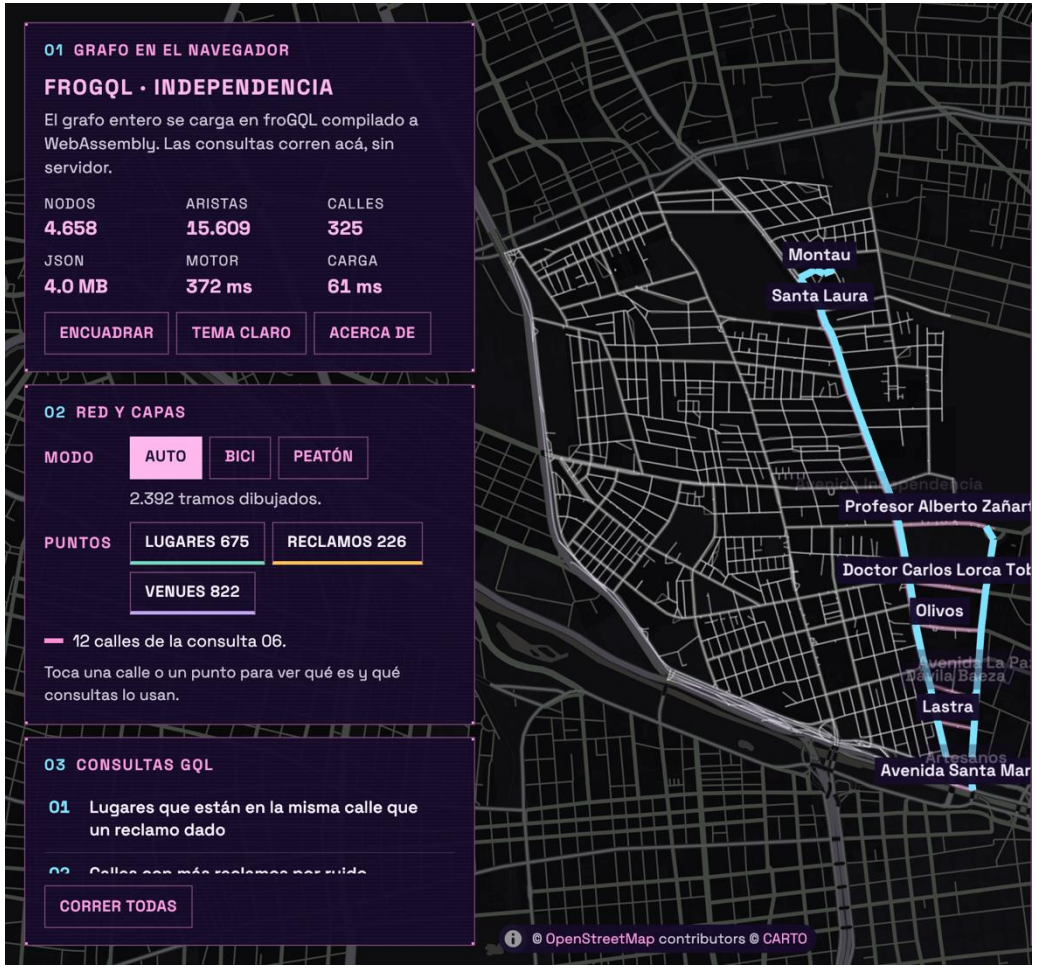
```
pip install frogql
```

```
npm install frogql
```

```
cargo install frogql
```

One engine, one .gdb file: Python, Node.js, Rust… and your browser

Same engine, in your browser

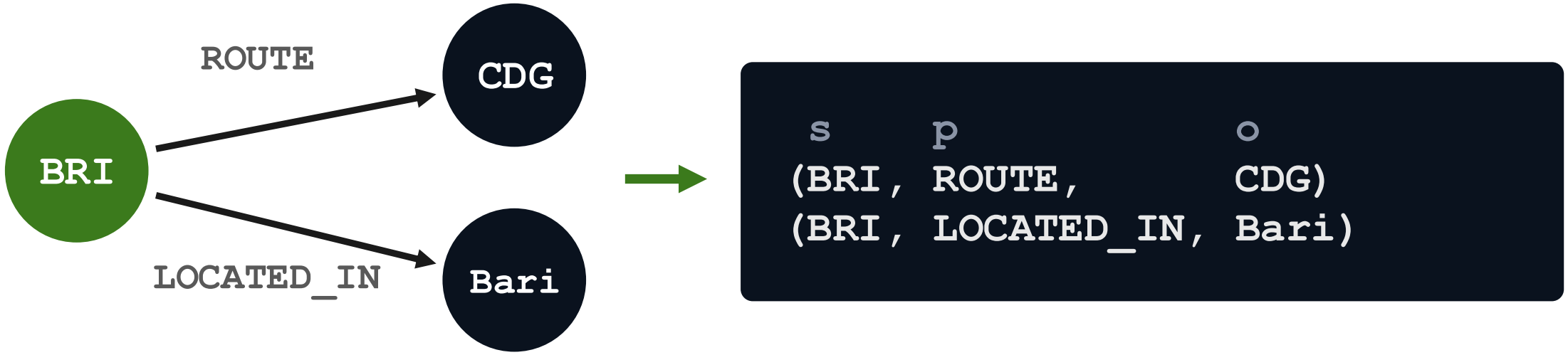


0 servers: froGQL runs as WebAssembly inside the page

A district of Santiago as one property graph: streets, places, noise complaints, venues

Try it live
on your phone, right now

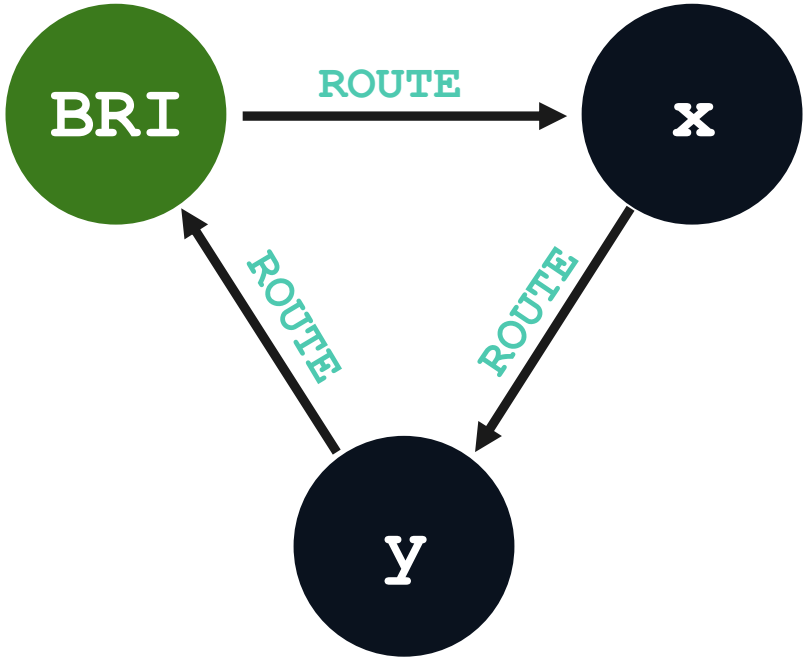
Under the hood it's all triples



SPO SOP PSO POS OSP OPS

Six sorted orderings: any bound prefix is one binary search away. The join machinery of RDF engines, reused for property graphs.

Worst-case optimal joins




```
MATCH (bri:Airport {iata:'BRI'})
  -[:ROUTE]->(x:Airport)
  -[:ROUTE]->(y:Airport)
  -[:ROUTE]->(bri)
RETURN x.iata, y.iata

20 ms · 743 round trips
```

Leapfrog Triejoin over the six orderings, one variable at a time

wco: never more work than the largest possible answer ☺

Up to 162× faster than pairwise joins

Same engine, same data, same query: only the join algorithm changes

Triangles through BRI

94 ms

vs 15.2 s pairwise

162× faster

4-cliques through BRI

1.3 s

vs 29.8 s pairwise

23× faster

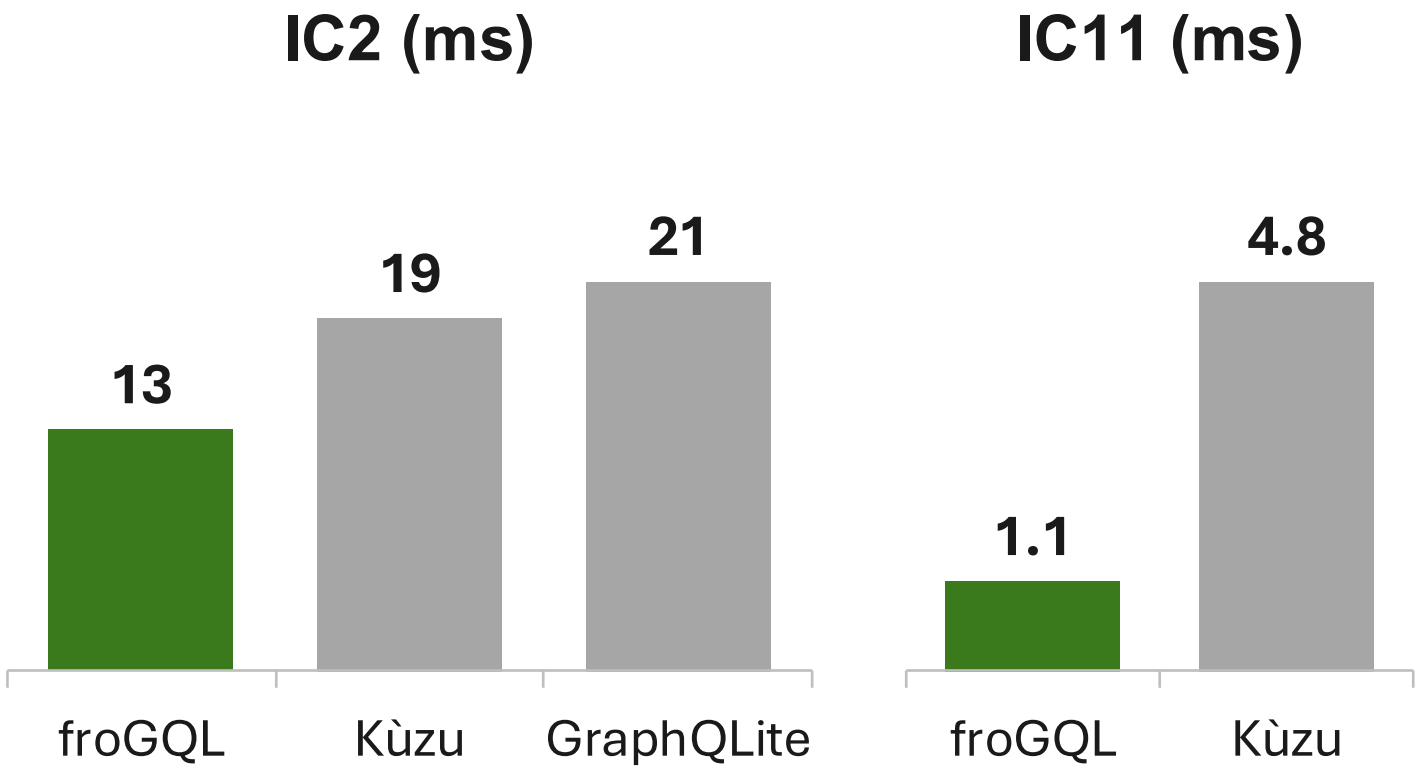
Domestic triangles in Italy

96 ms

vs > 120 s pairwise

> 1,250× faster

LDBC SNB: fastest on anchored queries



9.5 s

to load SF0.1 into one 142 MB file

1.2 s

to reopen it, every later session

LDBC SNB SF0.1 (327k nodes, 1.5M edges) · median latency · Ryzen 9, 128 GB RAM
Kùzu leads on aggregation-heavy IC5, IC6, IC9: our next target

How we prove a query empty

1 The graph type

```
(:Airport)
-[:LOCATED_IN]->
(:Country)
```

inferred at import, or declared

2 The query

```
(c)
-[:LOCATED_IN]->
(k:Airline)
```

what it says about each variable

3 Meet them

```
k : Country
  & Airline
  = ⊥
```

no node can be both: empty

⊥ spreads to the whole pattern: no rows, in 0.2 ms, without reading the graph

Type system: “Flexible and Expressive Typed Path Patterns for GQL” (Ye et al., 2025). It runs before, and independently of, the engine.

Pre-check of empty queries in microseconds

67 µs

to reject a 4-hop query the engine needs 18 s to enumerate

132 s

saved on a single query at LDBC SNB (SF0.3)

< 1%

type-check overhead on valid queries

```
... (c)-[:LOCATED_IN]->(k:Country) WHERE k.name = 42
refuted by the type checker in 0.2 ms (all round trips: 2.8 s)
```

Catching mistakes before they run

```
MATCH (a:Airpot)-[:ROUTE]->(b:Airport)
⚠ (:Airpot) does not match any node type in the schema

MATCH (a:Airport) WHERE a.altitude > '2000'
⚠ Binop Gt between types int and str is not defined

MATCH (a:Airport) RETURN a.altiude
⚠ Attribute altiude not found in Airport {altitude:int, …}
```

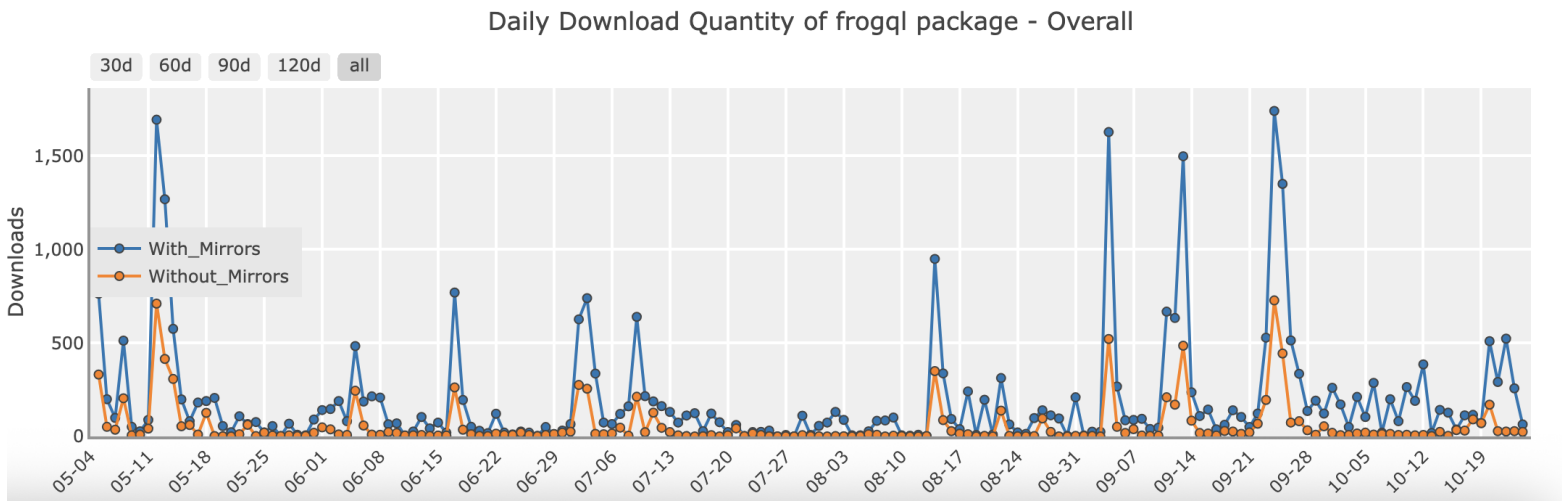
🐸QL raises Python warnings for impossibly typed queries

People are already using it

2,900
PyPI downloads last month

1,085
npm downloads, peak day

26
releases since May



Daily PyPI downloads, May–October (pypistats.org)

What's next

SPARQL front-end

RDF and SPARQL over the same six-ordering triple core

Faster post-join pipeline

sorting, grouping and deduplication beyond LTJ

Bigger graphs

LDBC SF1 and beyond, compact indexes, concurrent readers

Types for any engine

the type checker as a pre-pass for other GQL engines

pip install frogql

🧩 Embeddable

One .gdb file, no server: Python, Node.js, Rust and your browser

⚡ Fast

Worst-case optimal joins over triples: up to 162× faster than pairwise joins

🔎 Typed

Proves impossible queries empty in microseconds, before touching the data

MIT licensed · PyPI, npm, crates.io · benchmarks in the repo

Scan to try it
github.com/pleiad/frogql

IDIA

dcc CIENCIAS DE LA COMPUTACIÓN UNIVERSIDAD DE CHILE

Instituto Milenio Fundamentos de los datos

Backup: LDBC SNB SF0.1, all results

	froGQL	Kùzu	GraphQLite
IC2	**13 ms**	19 ms	21 ms
IC5	164 ms	**38 ms**	*
IC6	123 ms	**14 ms**	*
IC8	27 ms	21 ms	**13 ms**
IC9	1,543 ms	**61 ms**	—
IC11	**1.1 ms**	4.8 ms	*
Bulk load	9.5 s	**1.1 s**	10.7 s
File size	142 MB	**94 MB**	335 MB
Peak RSS	519–730 MiB	380–480 MiB	27–33 MiB †

* fewer rows, not comparable · — only in the paper's Fig. 6 · † excludes memory-mapped files
Median of 3 runs after warm-up · Ryzen 9 9900X3D, 128 GB · lower is better, best in green

Backup: large GQL coverage

13 of 14

LDBC SNB Interactive
Complex query templates

ISO GQL

the 2024 standard query
language for property graphs

- **ANY SHORTEST** paths
- **->*** and **->+** repetition
- **{n,m}** quantifiers
- **TRAIL · ACYCLIC**
- path functions
- **GROUP BY**, aggregates
- **VALUE** subqueries
- multi-pattern joins
- **INSERT**, `save()`
- **CREATE GRAPH TYPE**
- **VALIDATE**
- secondary indexes