QueryBridge: GraphQL-to-Datalog Optimizer

Translate GraphQL queries into Datalog programs with demand-transformation optimisation

Abishek Aditya
Stony Brook University
116551308

May 14, 2025

1 Problem and Plan

1.1 Problem Description

Objective. QueryBridge converts *GraphQL* queries into *XSB Datalog* and rewrites the resulting program with demand transformation—a magic-sets–style optimisation that aggressively pushes bindings downward, eliminating unnecessary intermediate relations. The tool accepts:

1. a GraphQL schema file,
2. one or more GraphQL query files.

It emits an `.P` file whose top-level predicate is `ans/N`. Demand transformation follows Liu *et al.* [1].

Interface.

- **CLI** `python -m querybridge schema.gql query.gql [--demand]`
- **API** `translate_graphql_to_xsb(schema, query, apply_demand)`→string

Requirements.

1. Parse GraphQL schemas and queries.
2. Generate semantically equivalent Datalog.
3. Apply demand transformation on request.
4. Support nested selections, arguments and fragments.
5. Translate typical queries in under two seconds.

2 Overview of the Tool

QueryBridge is organised as a three-stage pipeline:

1. GraphQL front-end builds typed ASTs.
2. Rule generator emits naïve Datalog.
3. Optimiser rewrites with demand transformation, then pretty-prints.

3 Features

- GraphQL schema and query parsing.
- Generation of XSB-compatible Datalog.
- Demand transformation optimiser [1].
- Support for deeply nested queries and argument filters.
- Clean, extensible Python code base.

4 Methodology: Demand Transformation and SIP Strategies

- **Demand Transformation** [1]: Refines classic magic-set rewriting by dynamically pruning irrelevant facts during evaluation, slashing intermediate materialisation cost.
- **SIP (Sideways Information Passing) Strategies** [2]: Control how variable bindings propagate between sub-goals, letting the optimiser choose an evaluation order that minimises search space in Datalog.

These two techniques steer both translation and optimisation, enabling QueryBridge to handle deeply nested and recursive GraphQL queries efficiently.

5 Prior Research

- **Hasura GraphQL Engine** [3]: Generates optimised SQL from GraphQL but hides the translation inside a server process.
- **XSB Datalog Engine** [4]: Mature logic-programming runtime with tabling and indexing; used here as the execution back-end.
- **Magic-Set Optimisation Slides** [2]: Overview of magic sets, semi-naïve evaluation and SIP strategy selection.
- **Deductive DBMS CORAL** [5]: Demonstrates magic-set style optimisations in a real database setting.

6 Installation

1. Clone the repository:

```
git clone <repo-url>
cd QueryBridge
```

2. Create a virtual environment and install:

```
python -m venv venv
source venv/bin/activate          # Windows: venv\Scripts\activate
pip install -e .
```

7 Usage

7.1 Command-Line

```
python -m querybridge                    # default demo
python -m querybridge schema.gql query.gql --demand
```

7.2 Library

```
from querybridge import translate_graphql_to_xsb

code = translate_graphql_to_xsb("schema.gql", "query.gql",
                                apply_demand=True)
print(code)
```

8 Complex Path Searching

GraphQL can traverse arbitrarily long edge chains—e.g. "all repositories reachable from a user via `starredBy` / `forkedFrom` up to depth 3." These recursive patterns map neatly to Datalog, and QueryBridge can *generate* the rules, but efficient execution still requires a recursion-aware SQL engine—something the ecosystem currently lacks.

9 State of the Art

Project	**Capabilities**
Hasura/PostGraphile	Full GraphQL servers in front of Postgres—planning, auth, caching, live queries—yet the SQL generator is buried inside the server; no "give me SQL" API.
graphene,strawberry-sqlalchemy	Maps SQLAlchemy models ⇒ GraphQL types and runs through the ORM; never produces raw SQL text.
sgqlc-build-sqla (prototype)	Creates SQLAlchemy models from a schema and helps construct queries, but still demands manual AST walking; not production-ready.
joernio/graphql-to-sql (2019 demo)	Translator for a tiny GraphQL subset (no fragments, variables, nesting); unmaintained and outputs only naïve `SELECT`s.

Table 1: Why existing work cannot serve as a direct GraphQL-to-SQL translator.

10 Implementation Status

QueryBridge v 0.3 can already take a GraphQL schema + query pair, compile it to naïve XSB Datalog, and then rewrite the program with classic Magic-Set demand transformation so that only facts relevant to the bound arguments reach the join—yielding visibly smaller rule sets and faster XSB execution in our unit tests. Each test directory under tests/ contains a schema,

query, and ground-truth fact file; the runner generates two Datalog files (with and without demand), executes both under XSB, and asserts that the answer predicates idential, confirming that demand transformation preserves semantics even as it prunes intermediate relations. A separate "benchmark" target meant to time queries and count rule sizes is stubbed out because of the state-of-the-art tools available in python not returning proper SQL queries for complex graphql queries to benchmark against.

11 Future Work

- Integrate a cost model to decide when demand transformation is profitable.
- Emit SQL for non-recursive fragments to enable hybrid SQL + Datalog back-ends.
- Extend recursion support once a suitable GraphQL-to-SQL translator emerges.

Acknowledgements

Thanks to Prof. Yanhong Annie Liu for guidance and the GraphQL Foundation for `graphql-core`.

References

[1] Y. A. Liu, S. D. Stoller, and T. Teitelbaum. Graph queries through Datalog optimizations. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010. `https://doi.org/10.1145/1836089.1836093`.

[2] P. Brass. *Magic Sets, SIP Strategies and Optimising Logic Programs.* Lecture slides, 2022. `https://users.informatik.uni-halle.de/~brass/lp22/print/cd_magic.pdf`.

[3] Hasura Inc. Hasura GraphQL engine. `https://hasura.io/graphql/`. Accessed 2025-04-18.

[4] The XSB Project. The XSB logic programming and deductive database system. `https://xsb.sourceforge.io`. Accessed 2025-04-18.

[5] R. Ramakrishnan, A. Silberschatz, and D. Stuckey. The CORAL deductive database system. *The VLDB Journal*, 1993.