GraphQL over SPARQL

2025-06-12 LSWT

presented by Claus Stadler

Overview

- Motivation
- GraphQL in a Nutshell
- GraphQL over SPARQL Approach
  - Expressing SPARQL-to-JSON mappings in GraphQL
- Implementation Notes
- Demo: Wikidata Movie Browser
- Demo: OpenData Portal Leipzig GraphQL Demo
  - Generating GraphQL Schema over RDF data
- Limitations (of SPARQL)
- Further Usage Scenarios

Motivation

- SPARQL is the standard query language for RDF graphs.

- SPARQL has 4 query forms
  - SELECT → tabular
  - CONSTRUCT → rdf
  - DESCRIBE → rdf,
  - ASK → boolean

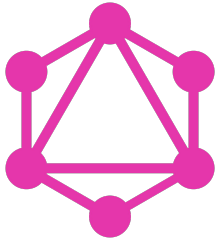
- None produces hierarchical data
  - JSON

- Rendering data in HTML is comparatively easy with JSON

Problem

How to easily bridge SPARQL and JSON?

Our approach leverages GraphQL as the mapping language.



Related Work

Stardog: https://docs.stardog.com/query-stardog/graphql

GraphDB: https://platform.ontotext.com/3.2/tutorials/graphql-query.html?highlight=graphql

Comunica: https://comunica.dev/docs/query/advanced/graphql_ld/

UltraGraphQL: https://github.com/internet-of-production/UltraGraphQL

GraphQL-to-SPARQL: https://www.npmjs.com/package/graphql-to-sparql

Grasp: https://github.com/dbcls/grasp

GraphSPARQL: https://github.com/Meitinger/GraphSPARQL/

graphql-jena: https://github.com/telicent-oss/graphql-jena

Tentris GraphQL: https://github.com/dice-group/tentris/blob/graphql-endpoint/GraphQL_doc.md

(... and more …)

GraphQL in a nutshell

- History (from https://en.wikipedia.org/wiki/GraphQL)
  - 2012 Initial development by Facebook
  - 2015 Draft spec + reference implementation
  - 2018 GraphQL Foundation

- Query and Manipulation Language
  - We focus on the query part.
  - Database agnostic

- In essence: Document format for Queries and Schema.
  Little semantics - many things are up to the implementer.

GraphQL Query Example

```
{
  Movies {
    id
    title
    ageRestriction
  }
}
```

GraphQL Query Example

```
{
  Movies {
    id
    title
    ageRestriction
  }
}
```

The backbone of GraphQL are nested fields.

GraphQL Query Example

```
{
  Movies {
    id
    title
    ageRestriction
  }
}
```

```
{
  “Movies”: [{
    “id”: “tt0095016”
    “title”: “Die Hard”
    “ageRestriction”: 16 # FSK
  }]
}
```

The backbone of GraphQL are nested fields.
You need to configure the server for how to resolve fields.

GraphQL Feature Overview

```
{
  Movies(limit: 10, note: “This is field with an argument”) {
    id @thisIsADirective
    ageRestriction @comment(note: “This is a directive with an argument”)
  }
}
```

GraphQL Feature Overview

```
{
  Movies(limit: 10, note: “This is field with an argument”) {
    id @thisIsADirective
    ageRestriction @comment(note: “This is a directive with an argument”)

    … on Disney { # Inline Fragment
      songs { # Fetch this property for Disney movies
        title
      }
    }
  }
}
```

Expressing SPARQL-to-JSON mappings in GraphQL

- Central Issue: SPARQL has variables, GraphQL doesn’t.
  - How to establish a mapping between GraphQL fields and SPARQL graph patterns

Expressing SPARQL-to-JSON mappings in GraphQL

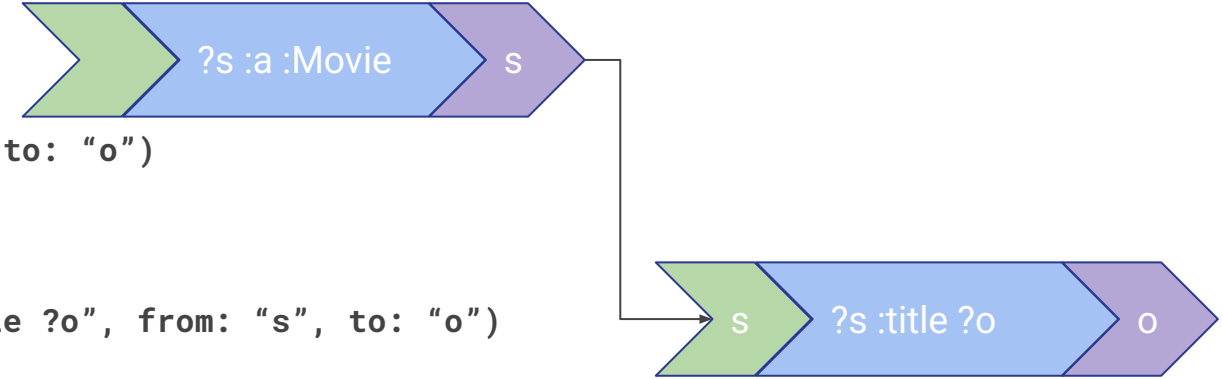
- Each GraphQL field is mapped to a (*SPARQL*) *connective* using **@pattern**:

```
{
  Movies {
    id
    title @pattern(of: “?s :title ?o”, from: “s”, to: “o”)
    ageRestriction
  }
}
```

Expressing SPARQL-to-JSON mappings in GraphQL

- Each GraphQL field is mapped to a (*SPARQL*) *connective*.

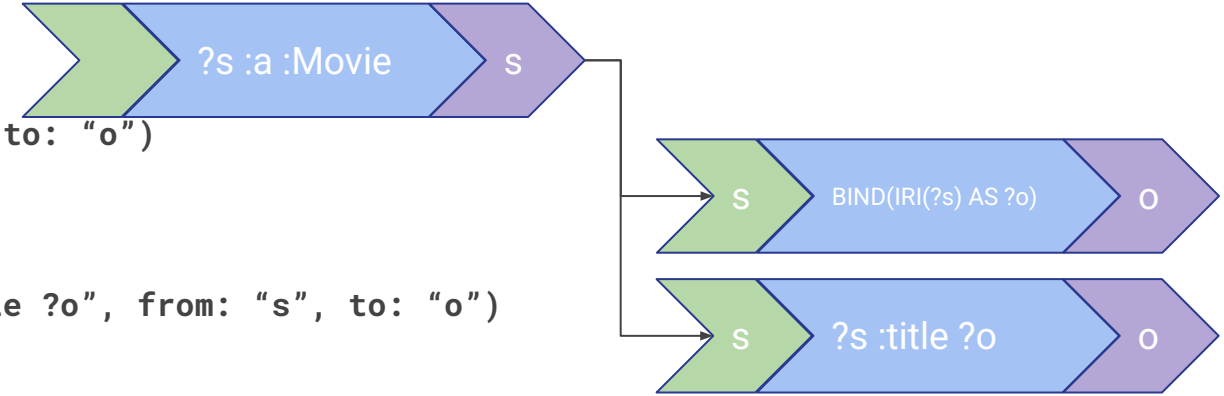
```
{
  Movies
    @pattern(of: “?s a :Movie”, to: “o”)
  {
    id
    title @pattern(of: “?s :title ?o”, from: “s”, to: “o”)
    ageRestriction
  }
}
```



Expressing SPARQL-to-JSON mappings in GraphQL

- Each GraphQL field is mapped to a (*SPARQL*) *connective*.

```
{
  Movies
    @pattern(of: “?s a :Movie”, to: “o”)
  {
    id @bind(of: “IRI(?s)”)
    title @pattern(of: “?s :title ?o”, from: “s”, to: “o”)
    ageRestriction
  }
}
```



Implementations of our GraphQL Engine

- RDF Processing Toolkit
  - Our standalone RDF CLI + Server

- Apache Jena Fuseki Plugin

- Summary: https://github.com/AKSW/graphql-over-sparql

GraphQL over SPARQL Engine Design Goals

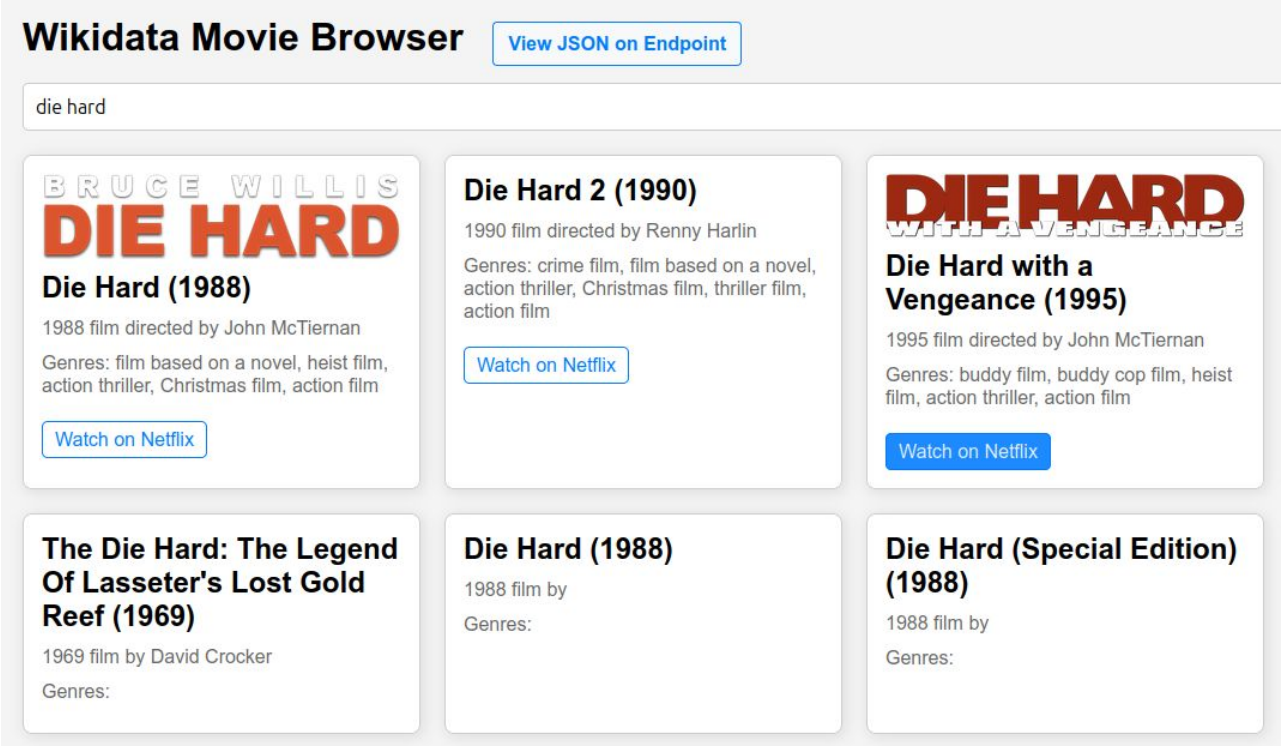
- SPARQL-based
  - Avoid need for additional server-side APIs for RDF data access.
  - Allow for proxy architectures (decouple GraphQL API from SPARQL backend)

- Performance: 1 GraphQL query → 1 SPARQL query
  - Avoid multiple queries for performance → Limitations!
    - Minimize client-server communication

- Support directives on GraphQL query level (ad-hoc annotations) and GraphQL schema level.

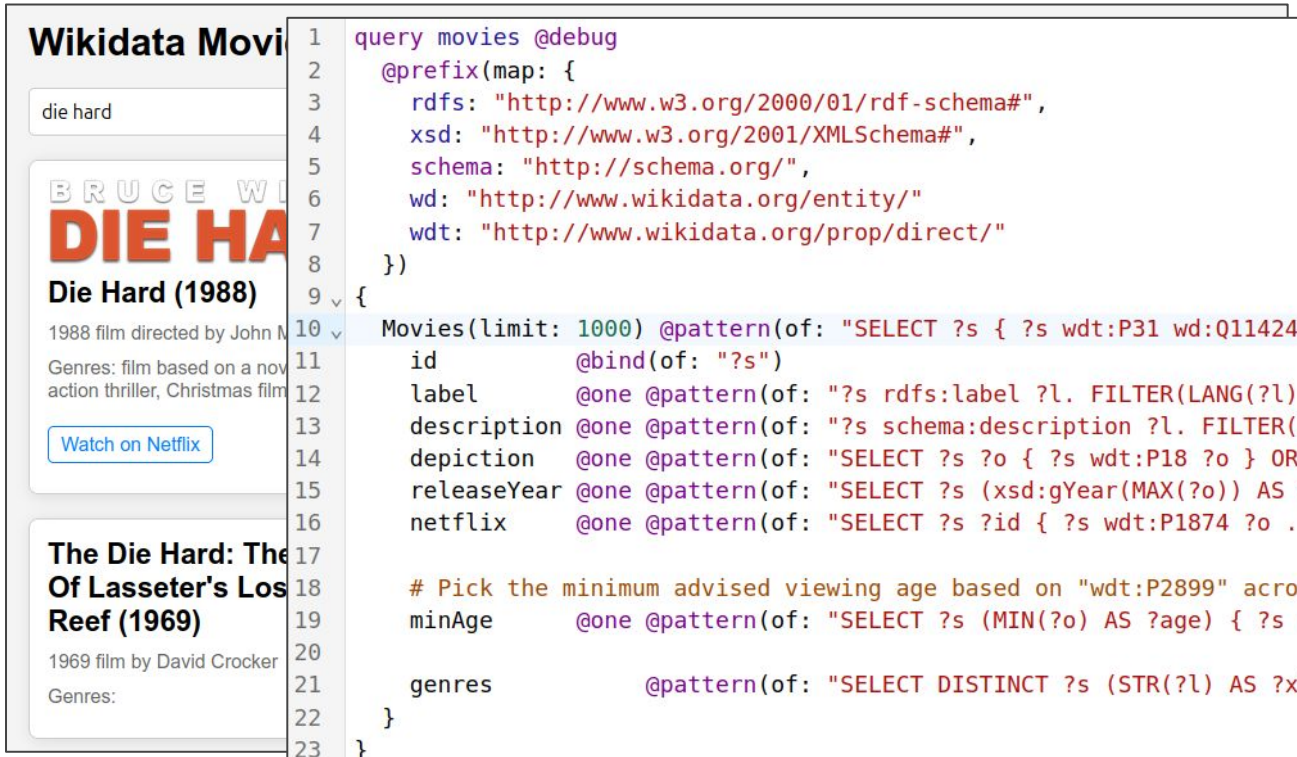
- Fully Streaming: Each binding of the SPARQL result set immediately drives the JSON output.
  - No in-memory copy.

WikiData Movie Browser Demo



WikiData Movie Browser Demo

Online Demo




```
query movies @debug
  @prefix(map: {
    rdfs: "http://www.w3.org/2000/01/rdf-schema#",
    xsd: "http://www.w3.org/2001/XMLSchema#",
    schema: "http://schema.org/",
    wd: "http://www.wikidata.org/entity/"
    wdt: "http://www.wikidata.org/prop/direct/"
  })
{
  Movies(limit: 1000) @pattern(of: "SELECT ?s { ?s wdt:P31 wd:Q11424 . FILTER (exists { ?s rdfs:label ?l .
    id          @bind(of: "?s")
    label       @one @pattern(of: "?s rdfs:label ?l. FILTER(LANG(?l) = 'en')")
    description @one @pattern(of: "?s schema:description ?l. FILTER(LANG(?l) = 'en')")
    depiction   @one @pattern(of: "SELECT ?s ?o { ?s wdt:P18 ?o } ORDER BY ?o LIMIT 1")
    releaseYear @one @pattern(of: "SELECT ?s (xsd:gYear(MAX(?o)) AS ?date) { ?s wdt:P577 ?o } GROUP BY ?s"
    netflix     @one @pattern(of: "SELECT ?s ?id { ?s wdt:P1874 ?o . BIND(IRI(CONCAT('https://www.netflix.

    # Pick the minimum advised viewing age based on "wdt:P2899" across any rating scheme
    minAge      @one @pattern(of: "SELECT ?s (MIN(?o) AS ?age) { ?s (!<p>)/wdt:P2899 ?o } GROUP BY ?s") @s

    genres          @pattern(of: "SELECT DISTINCT ?s (STR(?l) AS ?x) { ?s wdt:P136/rdfs:label ?l . FILTER
  }
}
```

SPARQL Query mainly based on LATERAL and UNION

```
SELECT  ?state ?v_0 ?v_1
WHERE
  {   { BIND("state_0" AS ?state)}
    UNION
      { { SELECT  *
          WHERE
            { { SELECT  ?field1_s
                WHERE
                  { ?field1_s  <http://www.wikidata.org/prop/direct/P31>  <http://www.wikidata.org/entity/Q11424>
                    FILTER EXISTS { ?field1_s  <http://www.w3.org/2000/01/rdf-schema#label>  ?l
                                    FILTER langMatches(lang(?l), "en")
                                    FILTER contains(lcase(str(?l)), lcase(""))
                                  }
                  }
              }
            }
          LIMIT   1
        }
        LATERAL
          {   { BIND("state_1" AS ?state)
                BIND(?field1_s AS ?v_0)
              }
            UNION
              { BIND("state_2" AS ?state)
                BIND(?field1_s AS ?field1_field1_bindvar_1)
                BIND(?field1_field1_bindvar_1 AS ?v_0)
              }
            UNION
              { BIND("state_3" AS ?state)
                ?field1_s  <http://www.w3.org/2000/01/rdf-schema#label>  ?field1_field2_l
                FILTER ( lang(?field1_field2_l) = "en" )
                BIND(?field1_s AS ?v_0)
                BIND(?field1_field2_l AS ?v_1)
              }}
      }}
```

	state	v_0	v_1
1	state_0		
2	state_1	<http://www.wikidata.org/entity/Q1000094>	
3	state_2	<http://www.wikidata.org/entity/Q1000094>	
4	state_3	<http://www.wikidata.org/entity/Q1000094>	"You're Dead"@en
5	state_1	<http://www.wikidata.org/entity/Q1000174>	
6	state_2	<http://www.wikidata.org/entity/Q1000174>	
7	state_3	<http://www.wikidata.org/entity/Q1000174>	"Tinko"@en

From RDF to a GraphQL Server

Representative sub set of the original RDF data

Analyzes which source classes are linked to which target classes by which properties based on the instances. Generates GraphQL types and properties.

Open Data Portal Demo

- https://github.com/AKSW/graphql-over-sparql/tree/main/odp-leipzig
- Download RDF Catalog from https://opendata.leipzig.de/catalog.ttl
- Invoke GraphQL Schema generator to generate GraphQL-RDF mapping
- Start a server with the mapping

Online Demo

Limitation: SPARQL spec does not mandate order

- `SELECT * { VALUES ?s { <a> <b> } }`
  - Engines may return <b> <a>

- `SELECT * { { BIND(<a> AS ?s) } UNION { BIND(<b> AS ?s) } }`
  - Engines may return <b> <a>

- `SELECT * { SELECT ?s { … } ORDER BY ?s }`
  - Engines may ignore order on sub-select

- SELECT (rowNum() AS ?rowNum) ?s { … }
  - SPARQL does not feature rowNum known from SQL - e.g. PostgreSQL, Oracle, …

Complex mappings with multiple variables

```
query @pretty
{
  Locations @pattern(of: "SELECT ?city ?country { ?s a :Location ; :city ?city ; :country ?country } ORDER BY ?city ?country",
                     from: ["country", "city"], to: ["country", "city"])
            @prefix(name: "", iri: "http://www.example.org/")
  {
    city    @via(of: "city")    @one
    country @via(of: "country") @one
    avgTemperature @one @pattern(of: """SELECT ?city ?country (AVG(?temp) AS ?avg) {
                                        ?x :city ?city ; :country ?country ; :measuredTemperature ?temp
                                      } GROUP BY ?city ?country
                                    """, from: ["country", "city"], to: "avg")
  }
}
```

Send | Abort | Curl | Link | Sparql

```
{
  "data": {
    "Locations": [
      {
        "city": "http://www.example.org/Leipzig",
        "country": "http://www.example.org/Germany",
        "avgTemperature": 15
      }
    ]
  },
  "errors": []
}
```

Data-Catalog+Void Example

For each class list me the number of datasets that contain it as JSON.

Auto-Completion in UI based on GraphQL Schema

```
query @pretty @debug {
  ClassToDatasets {
    class
    dcatDatasetCou
      dcatDatasetCount Scalar
      uri
      creator
    }
  }
}
```

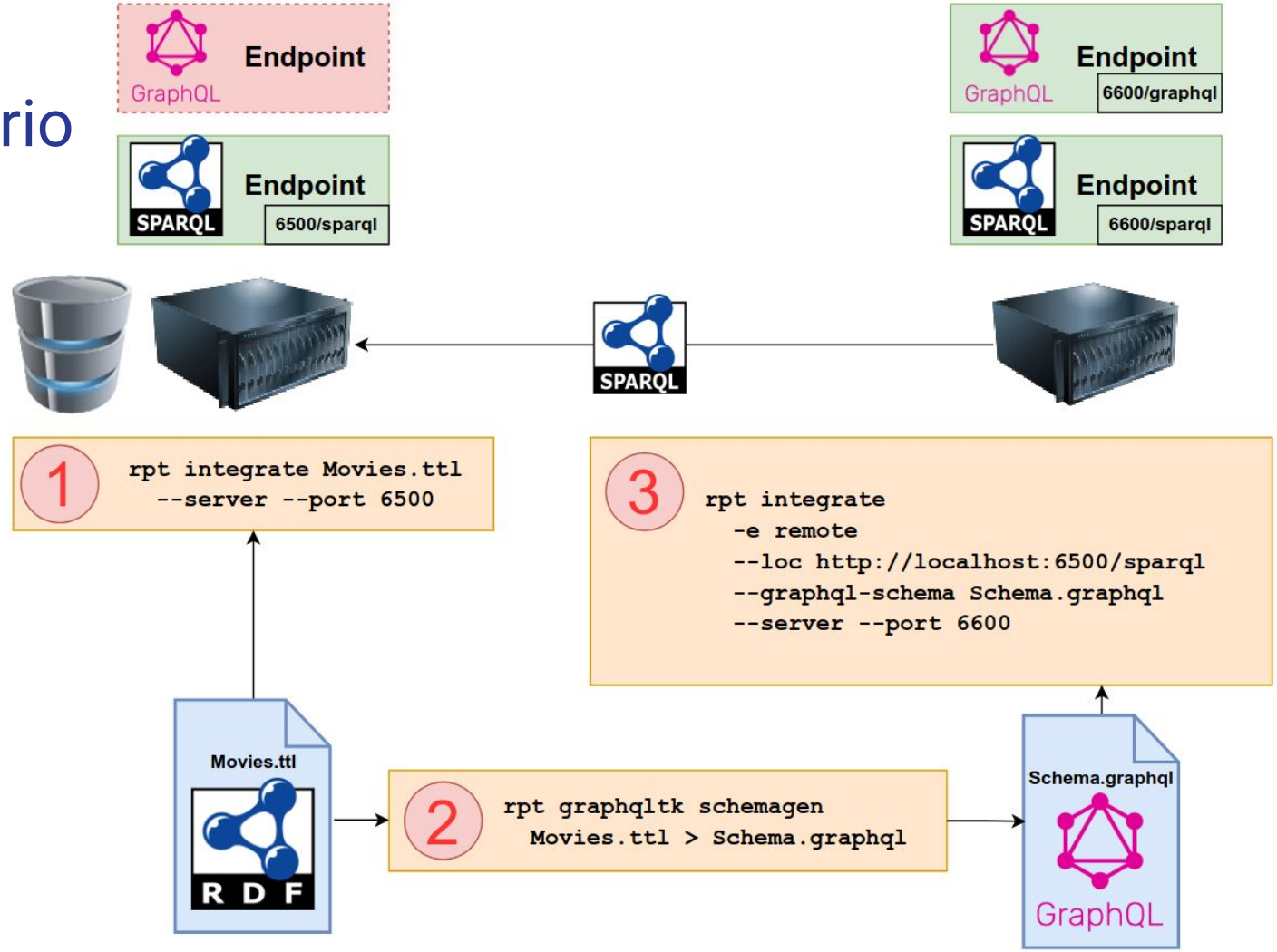
Send | Abort | Curl | Link | Sparql


```
{
  "data": {
    "ClassToDatasets": [
      {
        "class": "http://www.opengis.net/ont/geosparql#Geometry",
        "dcatDatasetCount": 27,
        "dcatDatasets": [
          {
            "uri": "urn:mvn:org.coypu.data.disasters:disasters:0.20240126.1000#dataset",
            "creator": ""
          },
          {
            "uri": "urn:mvn:org.coypu.data.disasters:disasters:0.20231218.0932#dataset",
            "creator": ""
          },
          {
            "uri": "urn:mvn:org.coypu.data.disasters:disasters:0.20240108.1501#dataset",
            "creator": ""
          },
          {
            "uri": "urn:mvn:org.coypu.data.disasters:disasters:0.20240203.1244#dataset",
            "creator": ""
```


```
type ClassToDatasets
    @prefix(name: "void", iri: "http://rdfs.org/ns/void#")
    @prefix(name: "dcat", iri: "http://www.w3.org/ns/dcat#")
    @prefix(name: "owl", iri: "http://www.w3.org/2002/07/owl#")
    @pattern(of: """
      SELECT ?class (COUNT(DISTINCT ?dcatDataset) AS ?datasetCount) {
        ?dcatDataset a dcat:Dataset ;
        owl:sameAs ?voidDataset .
        ?voidDataset void:classPartition ?cp .
        ?cp void:class ?class .
      }
      GROUP BY ?class ORDER BY DESC(?datasetCount)
    """, from: "class", to: "class")
{
  class: Scalar @bind(of: "?class")
  dcatDatasetCount: Scalar @bind(of: "?datasetCount")
  dcatDatasets: [DcatDataset]
    @pattern(of: """
      SELECT ?class ?dcatDataset {
        ?dcatDataset a dcat:Dataset ;
        owl:sameAs ?voidDataset .
        ?voidDataset void:classPartition ?cp .
        ?cp void:class ?class .
      }
    """, from: "class", to: "dcatDataset")
}
```

Proxy Scenario

Decoupled SPARQL and GraphQL endpoints.



Future Work

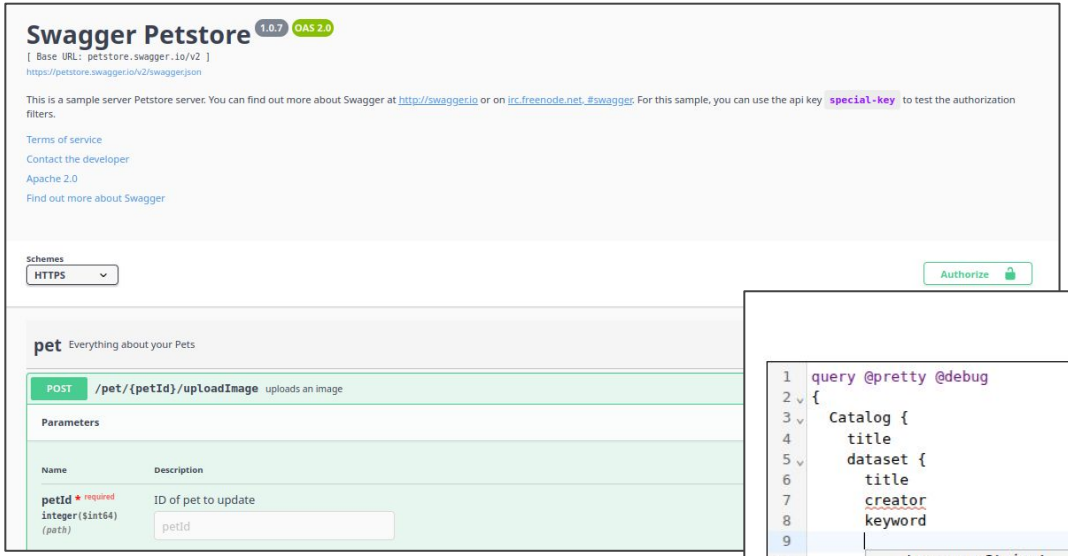
- Support for GraphQL Variables
- Top level array responses
  - Non standard because GraphQL requires a JSON object on the top level.
  - But: Useful for streaming arrays with many items
    - Seamless streaming of RDF data via JSON to OpenSearch / ElasticSearch
  - Arrays could be streamed as line-based JSON (JSONL/NDJSON)
- Improve documentation / specification
- Full release with Jena 5.5.0 (code is developed against Jena SNAPSHOT)
- Jena module if there is community interest
- Hasura-like filtering

```
query {
  authors(where: { name: { _eq: "Sidney" } }) {
    id
    name
  }
}
```

Thank You! Questions?

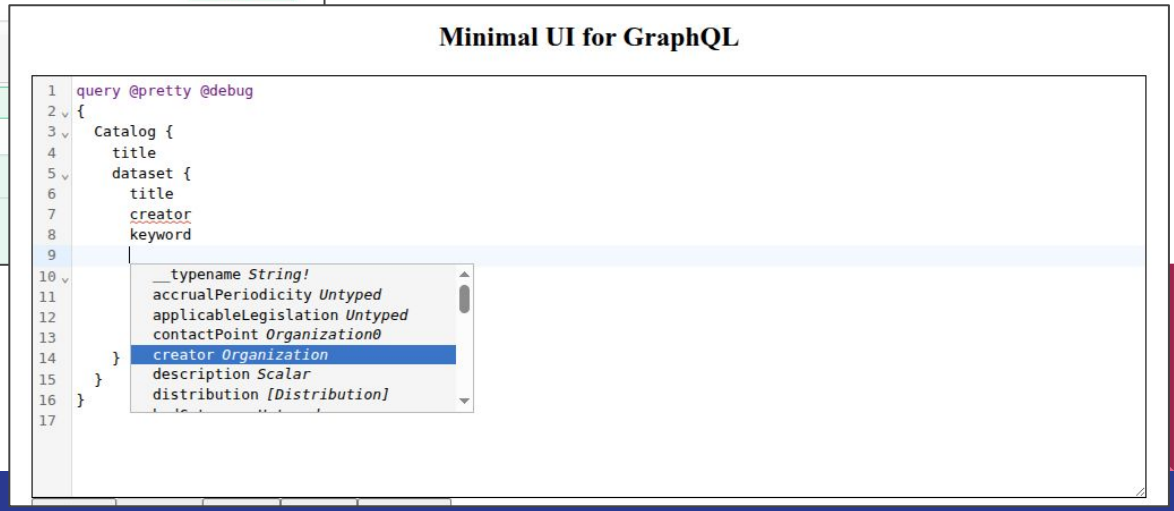
https://github.com/AKSW/graphql-over-sparql

GraphQL vs OpenAPI



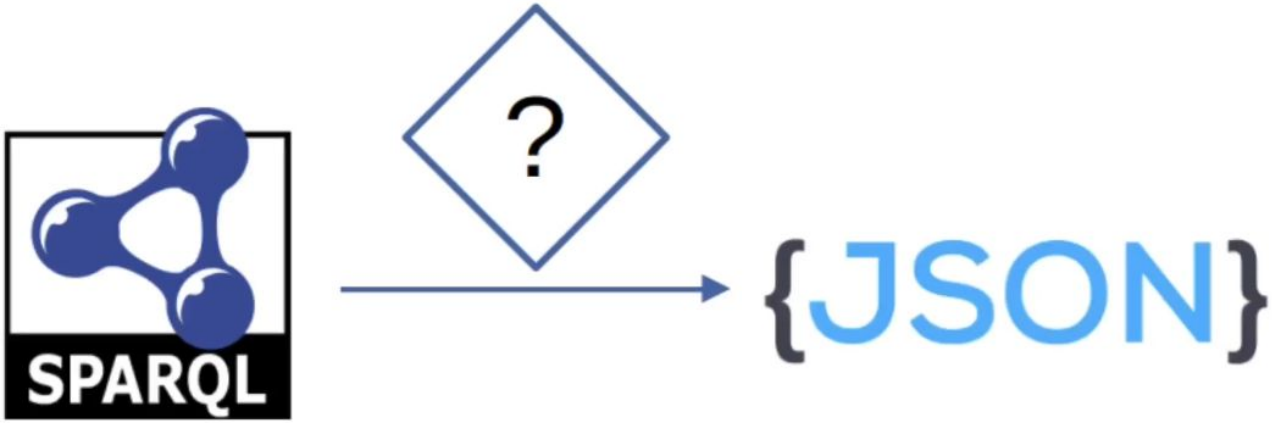
OpenAPI: Listing of methods and arguments.
Custom responses.

GraphQL Schema: Type-ahead for all schema elements (types, fields, arguments, directives, …)
Must return JSON.



GraphQL-to-SPARQL

https://drive.google.com/file/d/1i7EtlvYYhahcu-0OZdzaeS3a0BadtwSM/view?usp=sharing



http://localhost:3030/#/dataset/qlever/info

Apache Jena Fuseki datasets manage help server status

/qlever

query | add data | edit | info

Available Services

GraphQL query service	/qlever/graphql
GraphQL query service	/qlever/graphql
Graph Store Protocol	/qlever/data
SPARQL Query	/qlever/
Spatial index computation service	/qlever/update-spatial
SPARQL Update	/qlever/update

Statistics

Endpoint	Requests	Good	Bad
GraphQL query service (graphql)	0	0	0
Graph Store Protocol (data)	0	0	0
SPARQL Query	0	0	0
Spatial index computation service (update-spatial)	0	0	0
SPARQL Update (update)	0	0	0
Overall	**0**	**0**	**0**

Dataset size

count triples in all graphs

graph name	triples
No data	

GraphQL Editor

http://localhost:3030/qlever/graphql

Minimal UI for GraphQL

```
query @debug {
  test(limit: 10) @pattern(of: "?s ?p ?o", from: "s", to: "o")   @index(by: "?p")
}
```

Send | Abort | Curl | Link | Sparql

```
{
  "data": {
    "http://ld.company.org/prod-vocab/addressCountry": [
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India",
      "India"
    ]
  },
```

```
SELECT  ?state ?v_0 ?v_1 ?v_2
WHERE
  {   { BIND("state_0" AS ?state)}
    UNION
      { { SELECT  *
          WHERE
            { BIND("state_1" AS ?state)
              { SELECT  *
                WHERE
                  { ?field1_s  ?field1_p  ?field1_o }
                ORDER BY ASC(?field1_p)
              }
              BIND(?field1_s AS ?v_0)
              BIND(?field1_p AS ?v_1)
              BIND(?field1_o AS ?v_2)
            }
          LIMIT   10
        }
      }}
```

./mystart.sh

```
08:42:24 INFO  RDFEngineBuilderQlever :: 2025-05-20 06:42:24.399 - INFO: Requested media type of result is "application/sparql-results+json"
08:42:24 INFO  RDFEngineBuilderQlever :: 2025-05-20 06:42:24.399 - INFO: Processing the following SPARQL query:
08:42:24 INFO  RDFEngineBuilderQlever :: SELECT  ?state ?v_0 ?v_1 ?v_2
08:42:24 INFO  RDFEngineBuilderQlever :: WHERE
08:42:24 INFO  RDFEngineBuilderQlever ::   {   { BIND("state_0" AS ?state)}
08:42:24 INFO  RDFEngineBuilderQlever ::     UNION
08:42:24 INFO  RDFEngineBuilderQlever ::       { { SELECT  *
08:42:24 INFO  RDFEngineBuilderQlever ::           WHERE
08:42:24 INFO  RDFEngineBuilderQlever ::             { BIND("state_1" AS ?state)
08:42:24 INFO  RDFEngineBuilderQlever ::               { SELECT  *
08:42:24 INFO  RDFEngineBuilderQlever ::                 WHERE
08:42:24 INFO  RDFEngineBuilderQlever ::                   { ?field1_s  ?field1_p  ?field1_o }
08:42:24 INFO  RDFEngineBuilderQlever ::                 ORDER BY ASC(?field1_p)
08:42:24 INFO  RDFEngineBuilderQlever ::               }
08:42:24 INFO  RDFEngineBuilderQlever ::               BIND(?field1_s AS ?v_0)
08:42:24 INFO  RDFEngineBuilderQlever ::               BIND(?field1_p AS ?v_1)
08:42:24 INFO  RDFEngineBuilderQlever ::               BIND(?field1_o AS ?v_2)
08:42:24 INFO  RDFEngineBuilderQlever ::             }
08:42:24 INFO  RDFEngineBuilderQlever ::           LIMIT   10
08:42:24 INFO  RDFEngineBuilderQlever ::         }
08:42:24 INFO  RDFEngineBuilderQlever ::       }}
08:42:24 INFO  RDFEngineBuilderQlever ::
08:42:24 INFO  RDFEngineBuilderQlever :: 2025-05-20 06:42:24.399 - INFO: Query planning done in 0 ms
08:42:24 INFO  RDFEngineBuilderQlever :: 2025-05-20 06:42:24.399 - INFO: Result has size 11 x 7
08:42:24 INFO  RDFEngineBuilderQlever :: 2025-05-20 06:42:24.399 - INFO: Done processing query and sending result, total time was 1 ms
08:42:24 INFO  Fuseki          :: [9]   <= Content-Type:        application/json
08:42:24 INFO  Fuseki          :: [9]   <= Server:              Apache Jena Fuseki (5.5.0-SNAPSHOT)
08:42:24 INFO  Fuseki          :: [9] 200 OK (13 ms)
```

GraphQL-to-SPARQL: Approach

- **Self-contained** GraphQL queries
  - GraphQL document contains all information to
    - build a SPARQL query and
    - post-process the result set into JSON
- New: Support for annotated **GraphQL Schema**
- Engine can run as a proxy on SPARQL endpoints
  - Requires stable inter/intra ordering with UNIONs (TODO study which vendors support this)
  - Requires LATERAL, supported by Jena, Oxigraph
  - JenaX SPARQL polyfill for LATERAL
- Generate a single SPARQL query
  - Using **LATERAL** - see https://github.com/w3c/sparql-dev/issues/100
  - Let the SPARQL server handle everything in a **single request** (reduce HTTP overhead)
  - **Streaming** result set post-processing for **scalability**

Approaches

- 2 Levels for RDF Data Acess
  - Graph Level: Stream<Triple> stream = graph.find(s, p, o)
    - Matches all triples
    - *Triple pattern fragments* is the name of a spec to expose such *find* method via a REST API.
  - SPARQL Level: Stream<Binding> resultSet = engine.exec(graph, sparqlQuery)
    -
- 1 SPARQL query per GraphQL nesting
- 1 SPARQL query per GraphQL query
- Native execution of GraphQL
  -

SPARQL Rewrite (Outline)

- The GraphQL document encodes a state automaton
  - Each field corresponds to a state, and if the field's pattern matches something, then transition to the state of the child field.
  - Our GraphQL-over-SPARQL engine requires the SPARQL result set to encode

- For each GraphQL field:
  - emit its connective.
  - BIND state id and target variables
  -

Workaround by ordering Result Sets

- Engine **may** preserve order.
  - For example: Apache Jena does it (so far). Oxigraph and GraphDB do not.

- Order conditions can be used to sort the SPARQL result set. But:
  - **Order conditions become extremely complex and expensive.**
  - Imposes limitation: All field values must be unique.
    Example: ?o must never bind to the same value twice for a given ?s

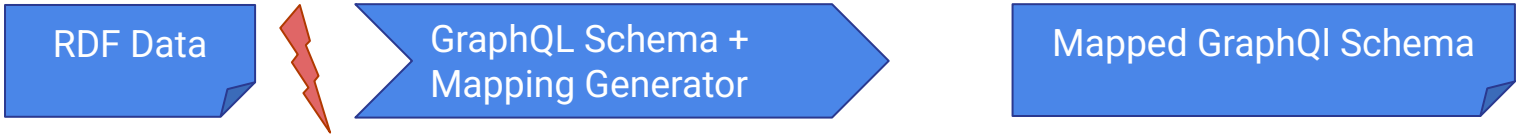
```
employees @many @pattern(of: “?s :department/:employee ?o”, from: “s”, to: “o”) {
    name
}
```

    Reason: If the e.g. the employee Bob appears multiple times, then we cannot unambiguously relate the attributes. E.g. is “Bob” the name of the first or second occurrence of Bob?

Workaround with multiple requests

- Graph patterns in UNION and LATERAL can be executed as seperate requests.
  - + Gives correct results, works with any engine.
  - - Great performance impact.

From RDF to mapped GraphQL Schema



Problem: Feeding large data to the schema generator is not feasible.

Solution: Summarization of relevant data.

Workaround by ordering Result Sets

- Engine **may** preserve order.
  - For example: Apache Jena does it (so far). Oxigraph and GraphDB do not.

- Order conditions can be used to sort the SPARQL result set. But:
  - **Order conditions become extremely complex and expensive.**
  - Imposes limitation: All field values must be unique.
    Example: ?o must never bind to the same value twice for a given ?s

```
employees @many @pattern(of: “?s :department/:employee ?o”, from: “s”, to: “o”) {
    name
}
```

    Reason: If the e.g. the employee Bob appears multiple times, then we cannot unambiguously relate the attributes. E.g. is “Bob” the name of the first or second occurrence of Bob?