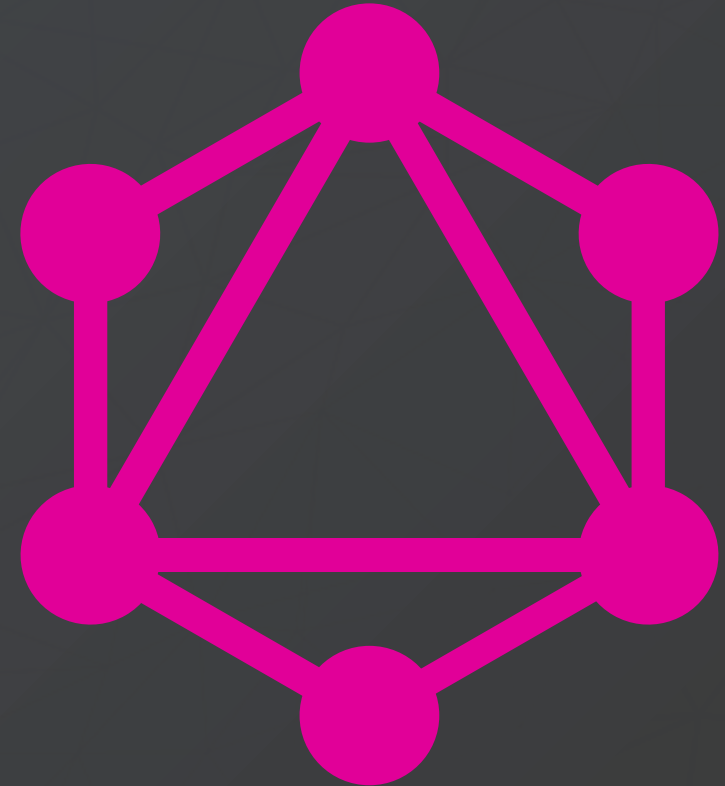


GraphQL, FHIR, and JavaScript

Robert Winterbottom
Software Engineer, Asymmetrik LTD

What is GraphQL?

- Specification designed and built by Facebook
- Implemented in many languages and database agnostic
- Query language for your API



Describe your data

- GraphQL has it's own type language
- Define your own custom scalar types
- Validation and type coercion

Get exactly what you ask for

Query

```
{
  Patient(_id: "1") {
    id
    text {
      status
      div
    }
    name {
      given
    }
  }
}
```

Response

```
{
  "data": {
    "Patient": {
      "id": "1",
      "text": {
        "status": "generated",
        "div": "<div>...</div>"
      },
      "name": [{
        "given": [ "Peter" ]
      }]
    }
  }
}
```

Sample REST response

```
{
  "resourceType": "Patient",
  "id": "example",
  "text": { ... },
  "identifier": [ ... ],
  "active": true,
  "name": [ ... ],
  "telecom": [ ... ],
  "gender": "male",
  "birthDate": "1974-12-25",
  "_birthDate": {
    "extension": [ ... ]
  },
  "deceasedBoolean": false,
  "address": [ ... ],
  "contact": [ ... ],
  "managingOrganization": {
    "reference": "Organization/1"
  }
}
```

Multiple resources in one request

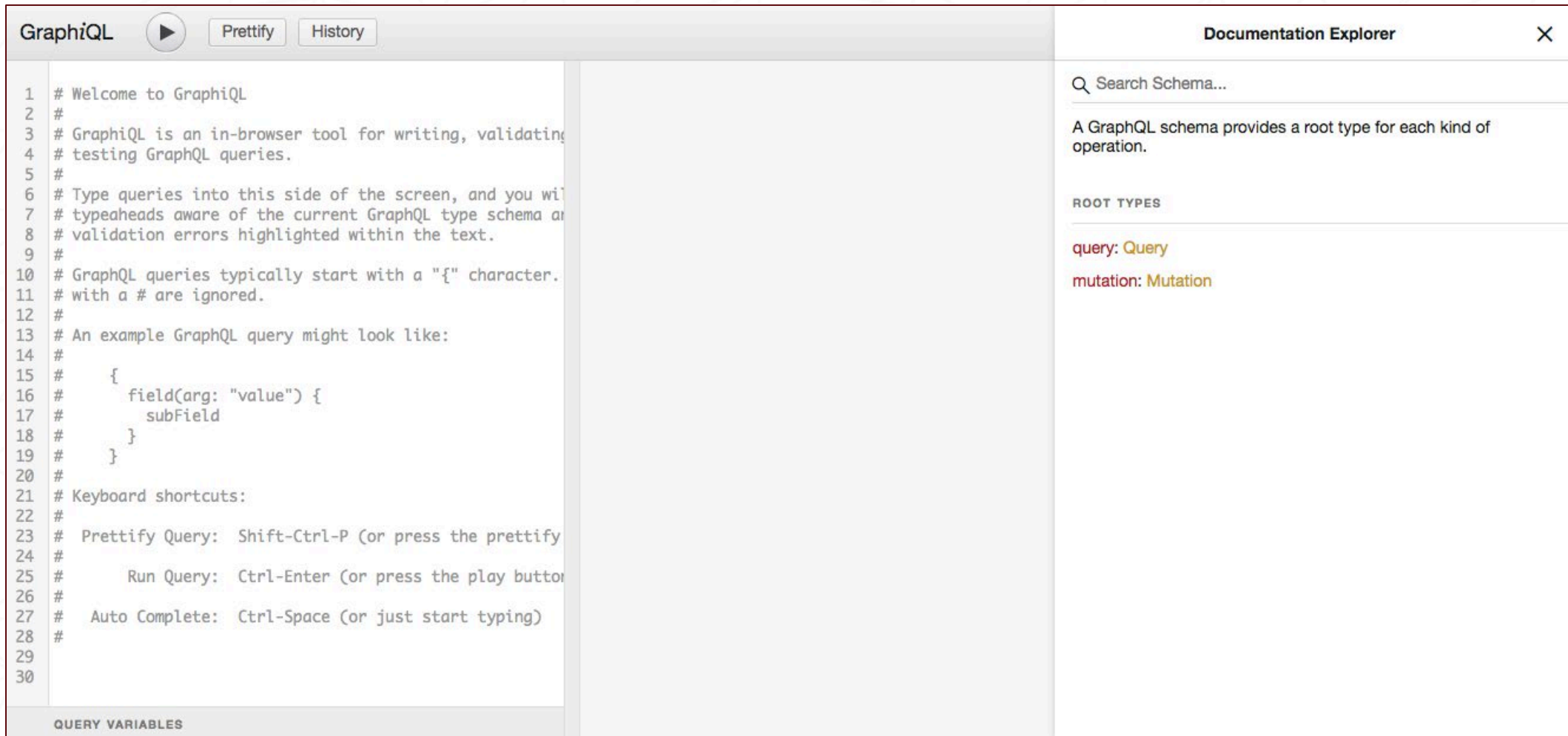
```
query {  
  Patient (_id:"example") { ... }  
  Observation { ... }  
  MedicationRequest { ... }  
}
```



Partial error handling

```
{
  "errors": [{
    "path": [ ... ],
    "message": ...,
    "locations": [{ ... }],
    "extensions": {
      "resource": {
        "resourceType": "OperationOutcome",
        "issue": [{
          "severity": "error",
          "code": "forbidden",
          "diagnostics": "You do not have sufficient scope for read access on this Observation resource."
        }]
      }
    }
  }],
  "data": {
    "Patient": {
      "id": "123456789"
    },
    "Observation": null
  }
}
```

Powerful developer tools



The screenshot displays the GraphQL IDE interface, which is split into two main panels. The left panel is a code editor titled "GraphQL" with a play button and "Prettify" and "History" buttons. It contains a series of numbered lines of text, including a welcome message, instructions on how to use the tool, and a sample GraphQL query. The right panel is titled "Documentation Explorer" and features a search bar labeled "Search Schema...". Below the search bar, there is a paragraph of text explaining that a GraphQL schema provides a root type for each kind of operation. Underneath, there is a section titled "ROOT TYPES" which lists two types: "query: Query" and "mutation: Mutation". At the bottom of the code editor, there is a section labeled "QUERY VARIABLES".

```
1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser tool for writing, validating
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will
7 # typeahead aware of the current GraphQL type schema and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character.
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify
24 #
25 #   Run Query: Ctrl-Enter (or press the play button)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29
30
```

Documentation Explorer

Q Search Schema...

A GraphQL schema provides a root type for each kind of operation.

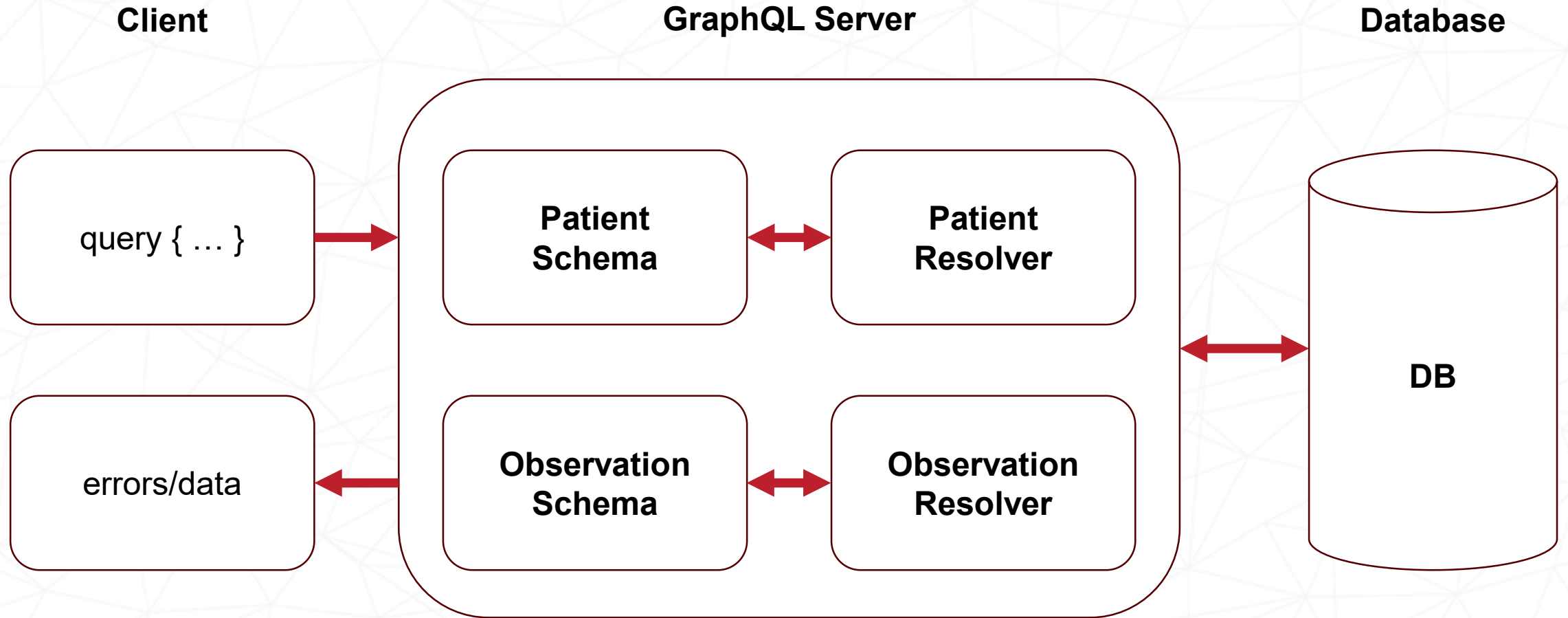
ROOT TYPES

query: Query

mutation: Mutation

QUERY VARIABLES

Example query



GraphQL + FHIR

- Not yet a formal standard
- Very much in the early phases of development
- Can be standalone or an add on to your current stack



Data

The data kind of looks like FHIR, but it's not exactly FHIR

```
{  
  "errors": [ ... ],  
  "data": {  
    "Patient": {  
      ...  
    }  
  }  
}
```

Errors – GraphQL vs FHIR

- GraphQL returns errors in an errors property in the response.
- Every error must contain an entry with the key “message”.
- Errors return an Operation Outcome.
- Operation outcomes are often returned with http status codes.

```
{  
  "errors": [{  
    "message": "Error"  
  }],  
  "data": { ... }  
}
```

```
{  
  "resourceType": "OperationOutcome",  
  "issue": [{  
    "severity": "error",  
    "code": "forbidden",  
    "diagnostics": " ... "  
  }]  
}
```

Errors – FHIR in GraphQL

```
{
  "errors": [{
    "path": [ ... ],
    "message": ...,
    "locations": [{ ... }],
    "extensions": {
      "resource": {
        "resourceType": "OperationOutcome",
        "issue": [{
          "severity": "error",
          "code": "forbidden",
          "diagnostics": "You do not have sufficient scope for read access on this Observation resource."
        }]
      }
    }
  }],
  "data": { ... }
}
```

Bundles

- Bundles aren't as easy in GraphQL as they are in REST.
- Reading a bundle in GraphQL requires a union and a different query syntax
- Writing a Bundle in GraphQL may require some more thought

Querying a Bundle

```
query {  
  PatientList(identifier:"example") {  
    entry {  
      resource {  
        ... on Patient { id }  
        ... on Observation { id }  
      }  
    }  
  }  
}
```

Bundles in Mutations

```
mutation {  
  BundleCreate(  
    resource: {  
      resourceType: Bundle  
      type: "transaction"  
      entry: {  
        patientResource: { ... },  
        observationResource: { ... }  
      }  
    }  
  ){  
    id  
  }  
}
```



Bundles in Mutations

```
mutation {  
  BundleCreate(  
    resource: {  
      resourceType: Bundle  
      type: "transaction"  
      entry: {  
        resource: "..."  
      }  
    }  
  ) {  
    id  
  }  
}
```



Demo



Thank You!

<https://asymmetrik.com/fhir>

Robert Winterbottom

rwinterbottom@asymmetrik.com

<https://github.com/Robert-W>

Twitter: **@RWinter85**