

Going Live with Enterprise Solutions



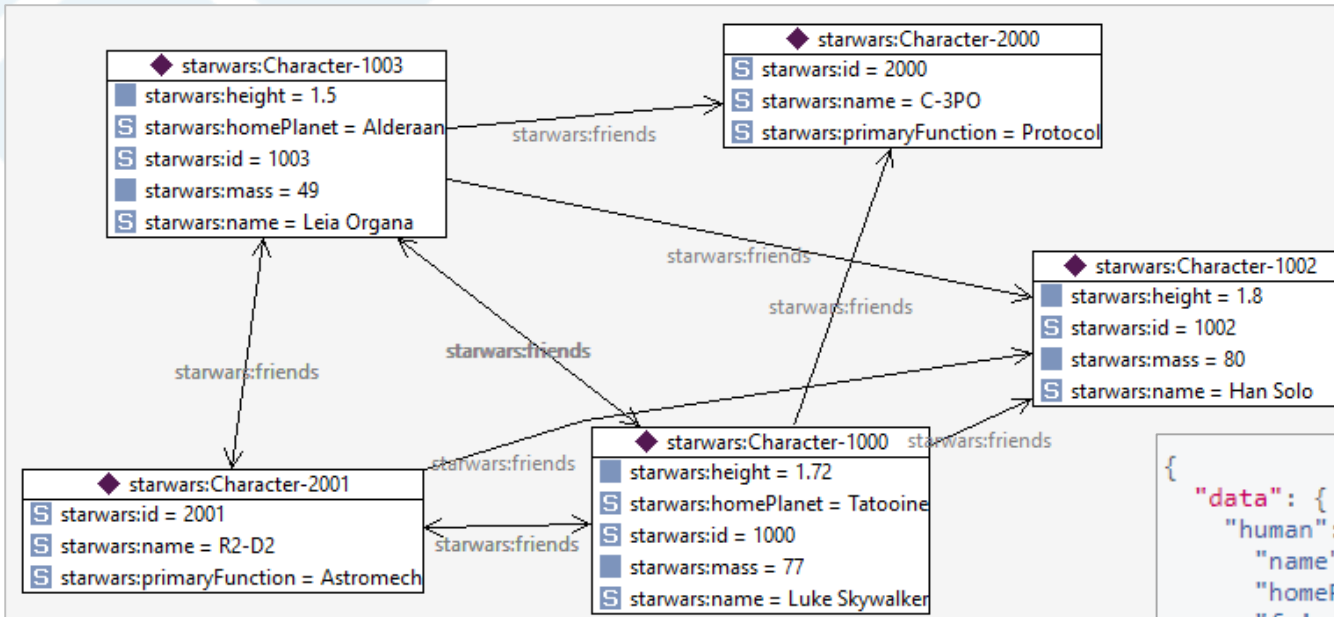
TopBraid GraphQL

Bringing the Worlds of JSON and RDF Together

Webinar, June 2018

Holger Knublauch

RDF Graphs to JSON Trees



```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "homePlanet": "Tatooine",
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
        {
          "name": "R2-D2"
        }
      ]
    }
  }
}
```

- What is GraphQL?
- How to bring together RDF technologies and GraphQL?
- GraphQL queries over RDF graphs
- GraphQL mutations (updates)
- JSON to RDF

- Questions

GraphQL (General Demo)

GraphiQL ▶ Prettify History

```
1 {
2   human(id: "1000") {
3     name
4     homePlanet
5     friends {
6       name
7     }
8   }
9 }
```

QUERY VARIABLES

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "homePlanet": "Tatooine",
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
        {
          "name": "R2-D2"
        }
      ]
    }
  }
}
```

< Schema **Query** ×

🔍 Search Query...

The query type, represents all of the entry points into our object graph

FIELDS

- hero(episode: Episode): Character
- reviews(episode: Episode!): [Review]
- search(text: String): [SearchResult]
- character(id: ID!): Character
- droid(id: ID!): Droid
- human(id: ID!): Human
- starship(id: ID!): Starship

TopBraid GraphiQL
▶
Prettify

```

1 {
2   humans(id: "1000") {
3     name
4     homePlanet
5     friends {
6       name
7     }
8   }
9 }

```

```

{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker",
        "homePlanet": "Tatooine",
        "friends": [
          {
            "name": "Leia Organa"
          },
          {
            "name": "C-3PO"
          },
          {
            "name": "R2-D2"
          },
          {
            "name": "Han Solo"
          }
        ]
      }
    ]
  }
}

```

< Schema
RootRDFQuery
✕

```

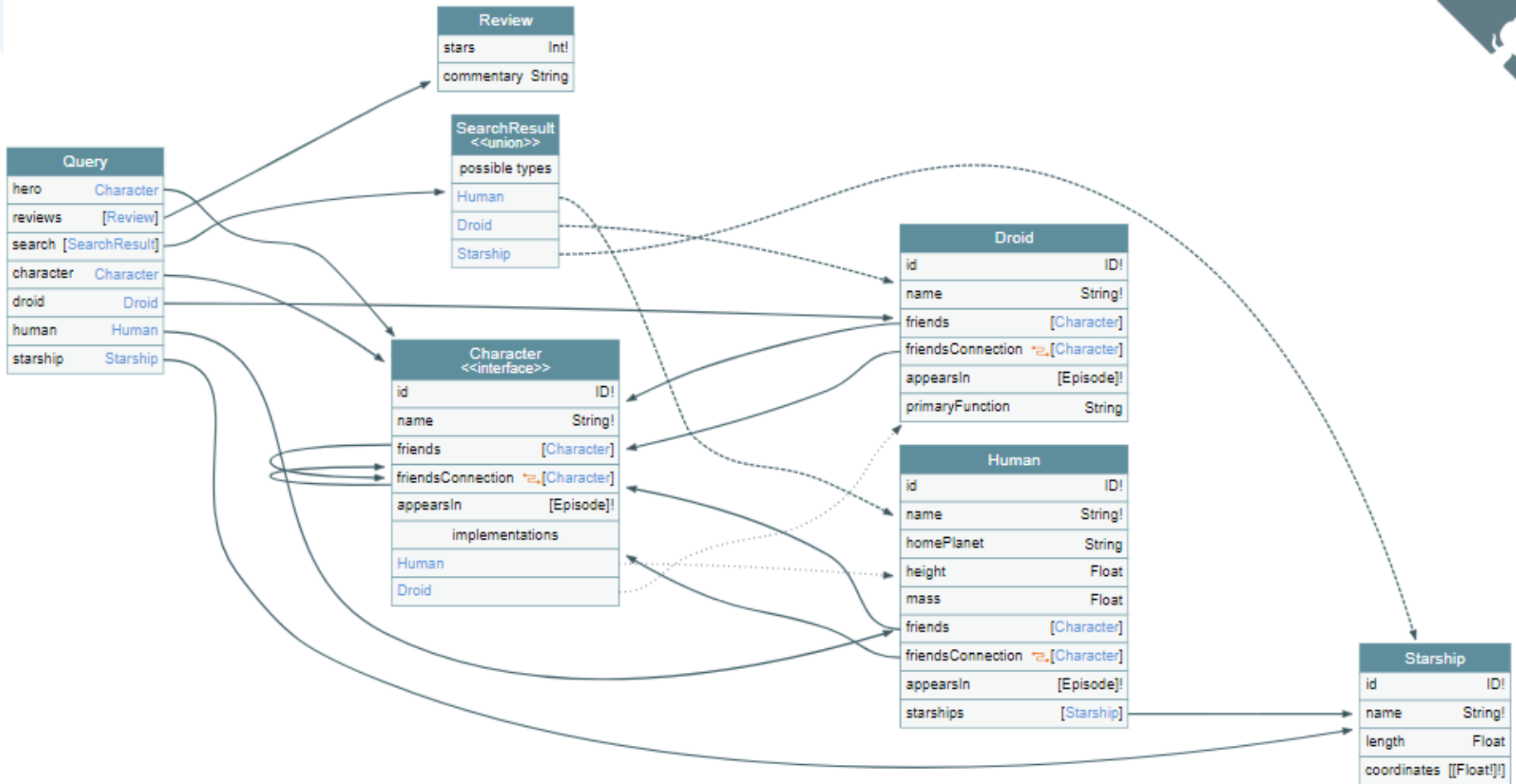
humans(
  queryText: String
  filter: String
  uri: ID
  orderBy: Human_FieldsEnum
  orderByExpr: String
  orderByDesc: Boolean
  appearsIn: String
  friends: ID
  height: Float
  homePlanet: String
  id: String
  mass: Float
  name: String
  starships: ID
  where: Human_where
  first: Int
  skip: Int
): [Human]

```

A humanoid creature from the Star Wars

QUERY VARIABLES

GraphQL Schemas



Star Wars Example Schema

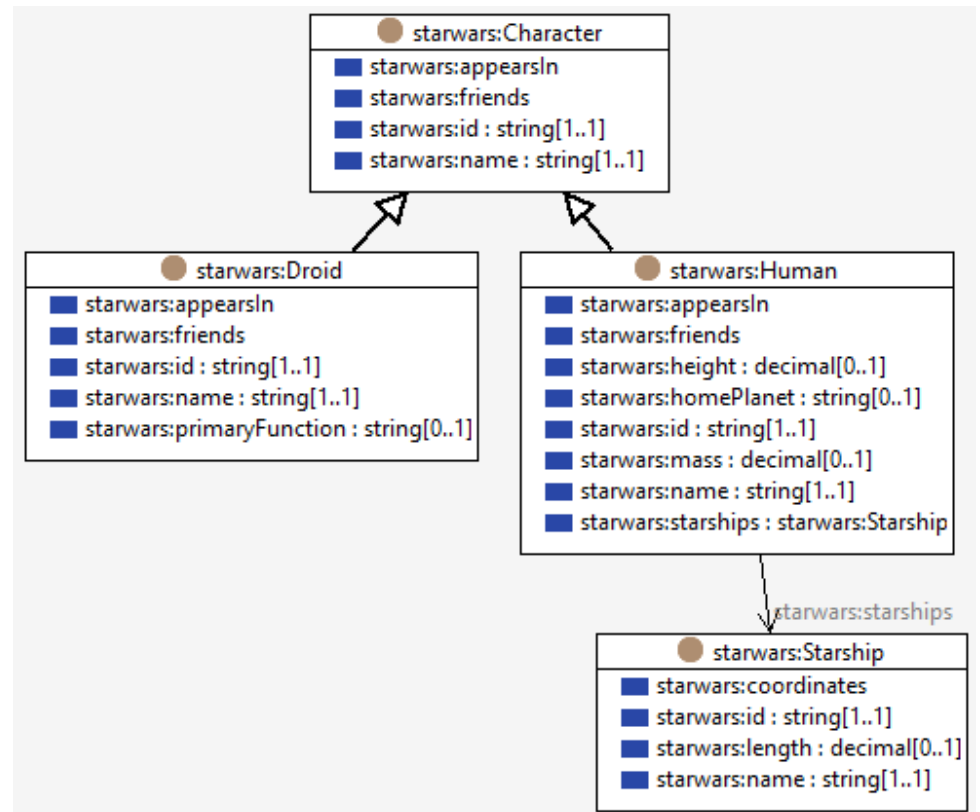
```
interface Character {  
  appearsIn: [Episode]!  
  friends: [Character]  
  id: ID!  
  name: String!  
}
```

```
type Human implements Character {  
  appearsIn: [Episode]!  
  friends: [Character]  
  height(unit: LengthUnit = METER): Float  
  homePlanet: String  
  id: ID!  
  mass: Float  
  name: String!  
  starships: [Starship]  
}
```

Motivations and Design Goals for Providing GraphQL Access to RDF Data

- Benefits:
 - JSON output
 - Simple to use (especially for UI developers)
 - Widely accepted
- Challenges:
 - While the “why” is clear and compelling, there are also some shortcomings (expressivity, etc.) and differences
- Design Goals
 - Integration should offer the best of both worlds

- W3C Standard since 2017
- Schema Language for RDF
- Rich Semantic Constraints
- Shapes > Classes
- Views
- Paths



Types vs RDF Datatypes

GraphQL Scalar Type	RDF Data Type
Boolean	xsd:boolean
Float	xsd:decimal
ID	xsd:string
Int	xsd:integer
String	xsd:string

Object Types vs Shapes

Example GraphQL Schema

```
type Person {  
  name: String  
  friends: [Person]  
}
```

Example RDF/SHACL

```
ex:Person  
  a sh:NodeShape ;  
  sh:property [  
    sh:path ex:friends ;  
    sh:node ex:Person ;  
  ] ;  
  sh:property [  
    sh:path ex:name ;  
    sh:datatype xsd:string ;  
    sh:maxCount 1 ;  
  ] .
```

Interfaces & Simple Inheritance

Example GraphQL Schema

```
interface Character {  
  id: ID!  
}  
  
type Human implements Character {  
  id: ID!  
  friends: [Character]  
}
```

Example RDF/SHACL

```
ex:Character  
  a sh:NodeShape ;  
  graphql:isInterface true ;  
  sh:property [  
    sh:path ex:id ;  
    sh:datatype xsd:string ;  
    sh:maxCount 1 ;  
    sh:minCount 1 ;  
  ] .  
  
ex:Human  
  a sh:NodeShape ;  
  sh:node ex:Character ;  
  sh:property [  
    sh:path ex:friends ;  
    sh:node ex:Character ;  
  ] ;  
  sh:property [  
    sh:path ex:id ;  
    sh:datatype xsd:string ;  
    sh:maxCount 1 ;  
    sh:minCount 1 ;  
  ] .
```

Union Types

Example GraphQL Schema

```
type Human {  
  name: String  
}  
  
type Starship {  
  length: Int  
}  
  
union SearchResult = Human | Starship
```

Example RDF/SHACL

```
ex:Human  
  a sh:NodeShape ;  
  sh:property [  
    sh:path ex:name ;  
    sh:datatype xsd:string ;  
    sh:maxCount 1 ;  
  ] .  
  
ex:Starship  
  a sh:NodeShape ;  
  sh:property [  
    sh:path ex:length ;  
    sh:datatype xsd:integer ;  
    sh:maxCount 1 ;  
  ] .  
  
ex:SearchResult  
  a sh:NodeShape ;  
  sh:or (  
    ex:Human  
    ex:Starship  
  ) .
```

Enumerations

Example GraphQL Schema

```
type Unicorn {  
  colors: Color  
}
```

```
enum Color {  
  # Yellow is our least favorite color  
  YELLOW  
  RED  
  PINK  
}
```

Example RDF/SHACL

```
ex:Unicorn  
  a sh:NodeShape ;  
  sh:property [  
    sh:path ex:colors ;  
    sh:node Color ;  
    sh:maxCount 1 ;  
  ] .
```

```
ex:Color  
  a sh:NodeShape ;  
  sh:in (  
    "YELLOW"  
    "RED"  
    "PINK"  
  ) .
```

Semantic Constraints

Example GraphQL Schema

```
type Adult {  
  age: Int @shape(minInclusive: 18)  
}
```

GraphQL Directives
(Extension Mechanism)

Example RDF/SHACL

```
ex:Adult  
  a sh:NodeShape ;  
  sh:property [  
    sh:path ex:age ;  
    sh:datatype xsd:integer ;  
    sh:maxCount 1 ;  
    sh:minInclusive 18 ;  
  ] .
```

Names and URIs

Example GraphQL Schema

```
schema
  @prefixes(
    human: "http://example.org/human/",
    starwars: "http://starwars.com/data/ (default)"
  )
{
  query: Query
}

type Human @uri(template: "human:{$id}") {
  id: ID!
  ...
}
```


Linked Schemas

Example GraphQL Schema

```
schema
  @graph(
    uri: "http://starwars.com/data/",
    imports: [ "http://movies.org/data/" ]
  )
  @prefixes(
    starwars: "http://starwars.com/data/ (default)",
    movies: "http://movies.org/data/"
  )
  ...
  type Actor {
    appearedIn: movies_Movie
    ...
  }
```

Display Metadata (SHACL)

Example RDF/SHACL

```
ex:NamesGroup
  a sh:PropertyGroup ;
  rdfs:label "Names" ;
  sh:order "0"^^xsd:decimal .

ex:AddressGroup
  a sh:PropertyGroup ;
  rdfs:label "Address" ;
  rdfs:label "Adresse"@de ;
  sh:order "1"^^xsd:decimal .

ex:Customer
  a sh:NodeShape ;
  sh:property [
    sh:path ex:firstName ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    sh:order "0"^^xsd:decimal ;
    sh:group ex:NamesGroup ;
    sh:name "given name" .
  ]
  ...
```

Names:

```
given name:    Rolf-Michel
family name:   Massin
```

Address:

```
street:        9100 Oak Street
zip code:      91823
country:       USA
```

Display Metadata (GraphQL)

Example GraphQL Schema

```
schema
  @groups(
    NamesGroup: {
      label: "Names"
    },
    AddressGroup: {
      label: "Address"
      label_de: "Adresse"
    }
  )
  ...

type Customer {
  firstName: String      @display(group: NamesGroup,   label: "given name")
  lastName: String       @display(group: NamesGroup,   label: "family name")
  street: String          @display(group: AddressGroup)
  postalCode: String      @display(group: AddressGroup, label: "zip code", label_de: "Postleitzahl")
  country: String         @display(group: AddressGroup, defaultValue: "USA")
}
```

Names:

given name: Rolf-Michel

family name: Massin

Address:

street: 9100 Oak Street

zip code: 91823

country: USA

GraphQL Schema + SHACL = Better Together

- Lots of similarities (types, fields, etc)
- GraphQL has a huge user base & tool support
- GraphQL has user-friendly syntaxes
- RDF is a flexible model for knowledge graphs
- RDF offers URIs and subclasses
- SHACL offers rich constraints & UI metadata

- All RDF knowledge graphs can be turned into GraphQL query endpoints with SHACL

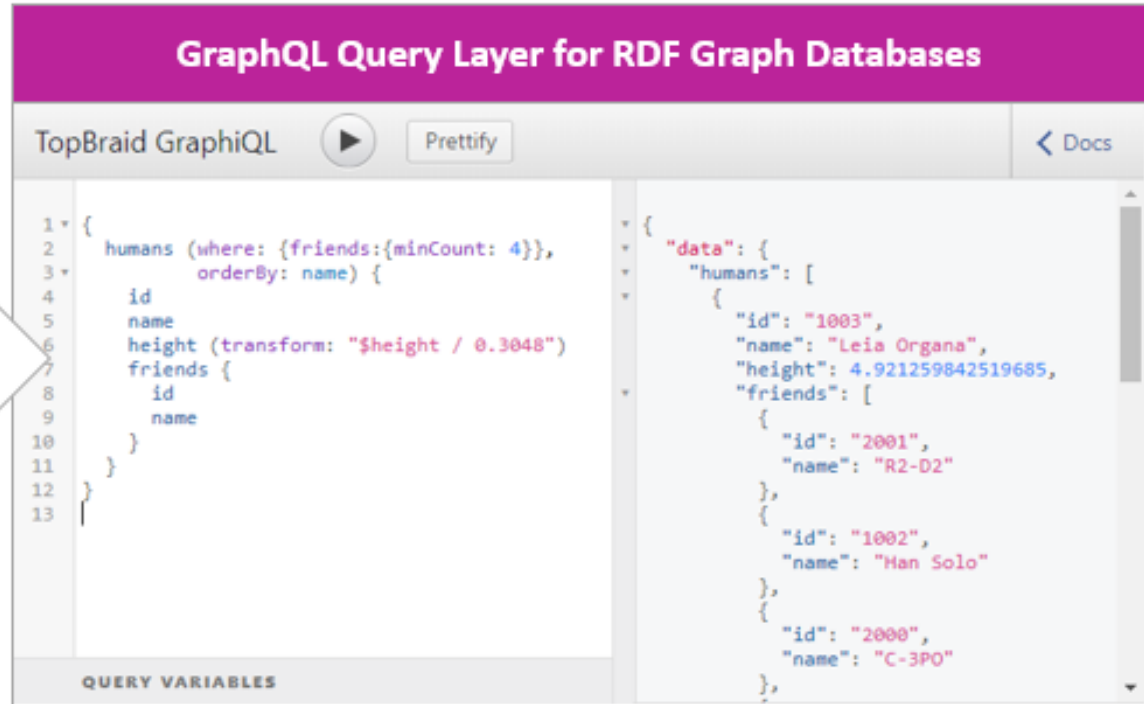
GraphQL in TopBraid

RDF/SHACL Schemas
(Turtle files etc.)

GraphQL Schemas
converted to SHACL

RDFS/OWL Models
converted to SHACL

GraphQL
Schema
Generator

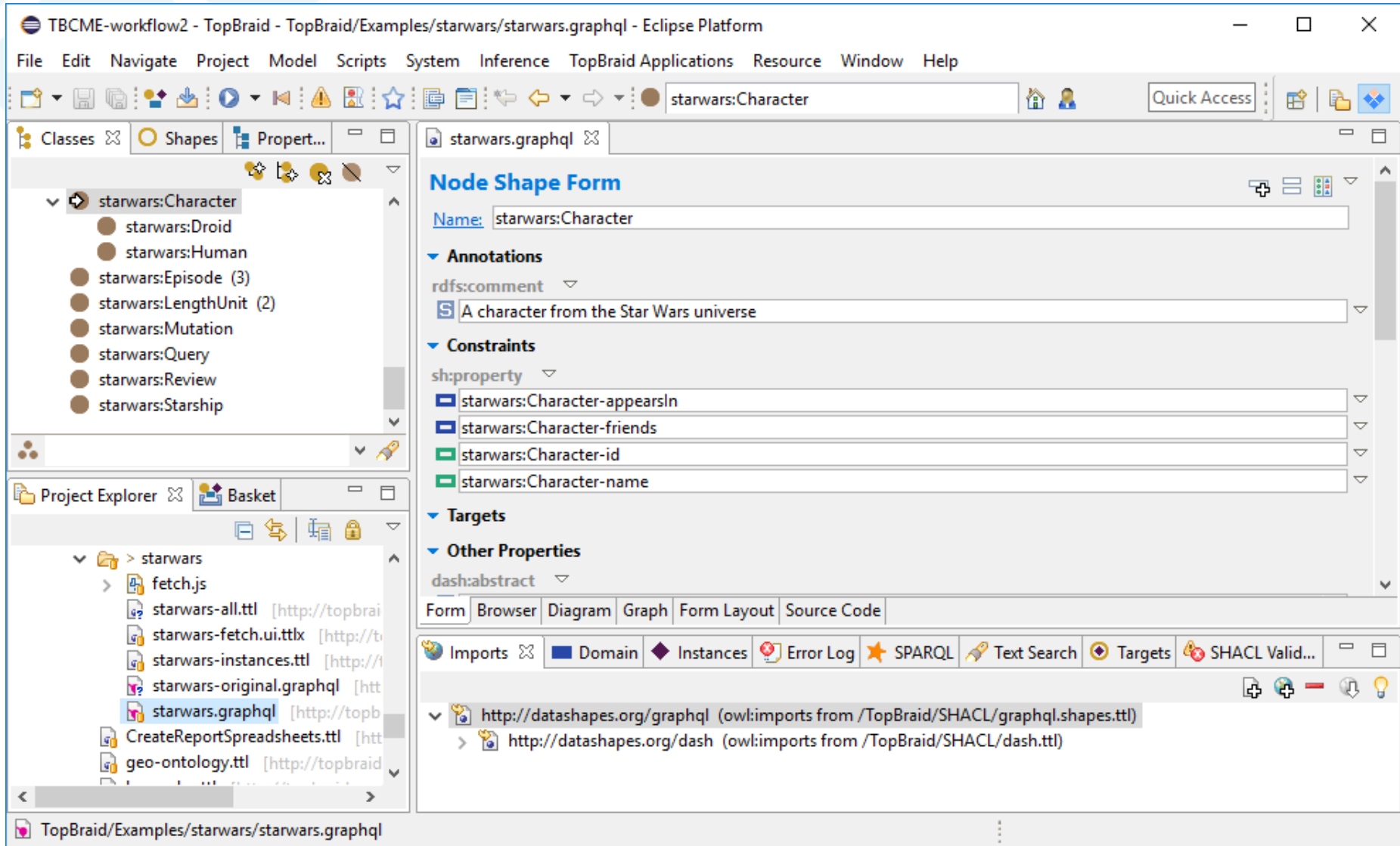


The screenshot shows the TopBraid GraphQL Query Layer for RDF Graph Databases interface. The title bar is purple and contains the text "GraphQL Query Layer for RDF Graph Databases". Below the title bar, there is a search bar with "TopBraid GraphiQL" and a play button, a "Prettify" button, and a "Docs" link. The main area is split into two panes. The left pane shows a GraphQL query with line numbers 1 through 13. The right pane shows the JSON response. At the bottom, there is a "QUERY VARIABLES" section.

```
1 {
2   humans (where: {friends:{minCount: 4}},
3           orderBy: name) {
4     id
5     name
6     height (transform: "$height / 0.3048")
7     friends {
8       id
9       name
10    }
11  }
12 }
13 }
```

```
{
  "data": {
    "humans": [
      {
        "id": "1003",
        "name": "Leia Organa",
        "height": 4.921259842519685,
        "friends": [
          {
            "id": "2001",
            "name": "R2-D2"
          },
          {
            "id": "1002",
            "name": "Han Solo"
          },
          {
            "id": "2000",
            "name": "C-3PO"
          }
        ]
      }
    ]
  }
}
```

Loading GraphQL Files in TopBraid Composer



The screenshot displays the Eclipse Platform interface for TopBraid Composer. The main window is titled "TBCME-workflow2 - TopBraid - TopBraid/Examples/starwars/starwars.graphql - Eclipse Platform". The menu bar includes File, Edit, Navigate, Project, Model, Scripts, System, Inference, TopBraid Applications, Resource, Window, and Help. The toolbar contains various icons for file operations and navigation.

The left sidebar shows the "Classes" view with a tree structure of classes under "starwars:Character":

- starwars:Character
 - starwars:Droid
 - starwars:Human
 - starwars:Episode (3)
 - starwars:LengthUnit (2)
 - starwars:Mutation
 - starwars:Query
 - starwars:Review
 - starwars:Starship

The "Project Explorer" view shows the project structure:

- starwars
 - fetch.js
 - starwars-all.ttl [http://topbrai]
 - starwars-fetch.ui.ttlx [http://t]
 - starwars-instances.ttl [http://]
 - starwars-original.graphql [htt]
 - starwars.graphql [http://topb]
 - CreateReportSpreadsheets.ttl [htt]
 - geo-ontology.ttl [http://topbrai]

The main editor area displays the "Node Shape Form" for "starwars:Character". The form includes the following sections:

- Name:** starwars:Character
- Annotations:**
 - rdfs:comment: A character from the Star Wars universe
- Constraints:**
 - sh:property:
 - starwars:Character-appearsIn
 - starwars:Character-friends
 - starwars:Character-id
 - starwars:Character-name
- Targets:**
- Other Properties:**
 - dash:abstract

The bottom of the interface shows the "Imports" view with the following entries:

- http://datashapes.org/graphql (owl:imports from /TopBraid/SHACL/graphql.shapes.ttl)
 - http://datashapes.org/dash (owl:imports from /TopBraid/SHACL/dash.ttl)

SHACL/Ontology Editor in TopBraid EDG

TopBraid EDG Enterprise Data Governance Hello, Administrator

Ontology | Dashboard | Settings | Users | Import | Transform | Export | Reports | Workflows | Tasks | Comments | Manage

Class Hierarchy

Look up Class

- Character
 - Droid
 - Human**
 - appears in
 - friends
 - id
 - name
- Episode
- LengthUnit
- Review
- Starship
 - coordinates
 - id
 - length

Human (Class, Node shape)

Enter log message

<http://topbraid.org/examples/starwars/Human>

Labels and Description

label: Lang

comment:

A humanoid creature from the Star Wars universe

Class Characteristics

sub-class of:

abstract:

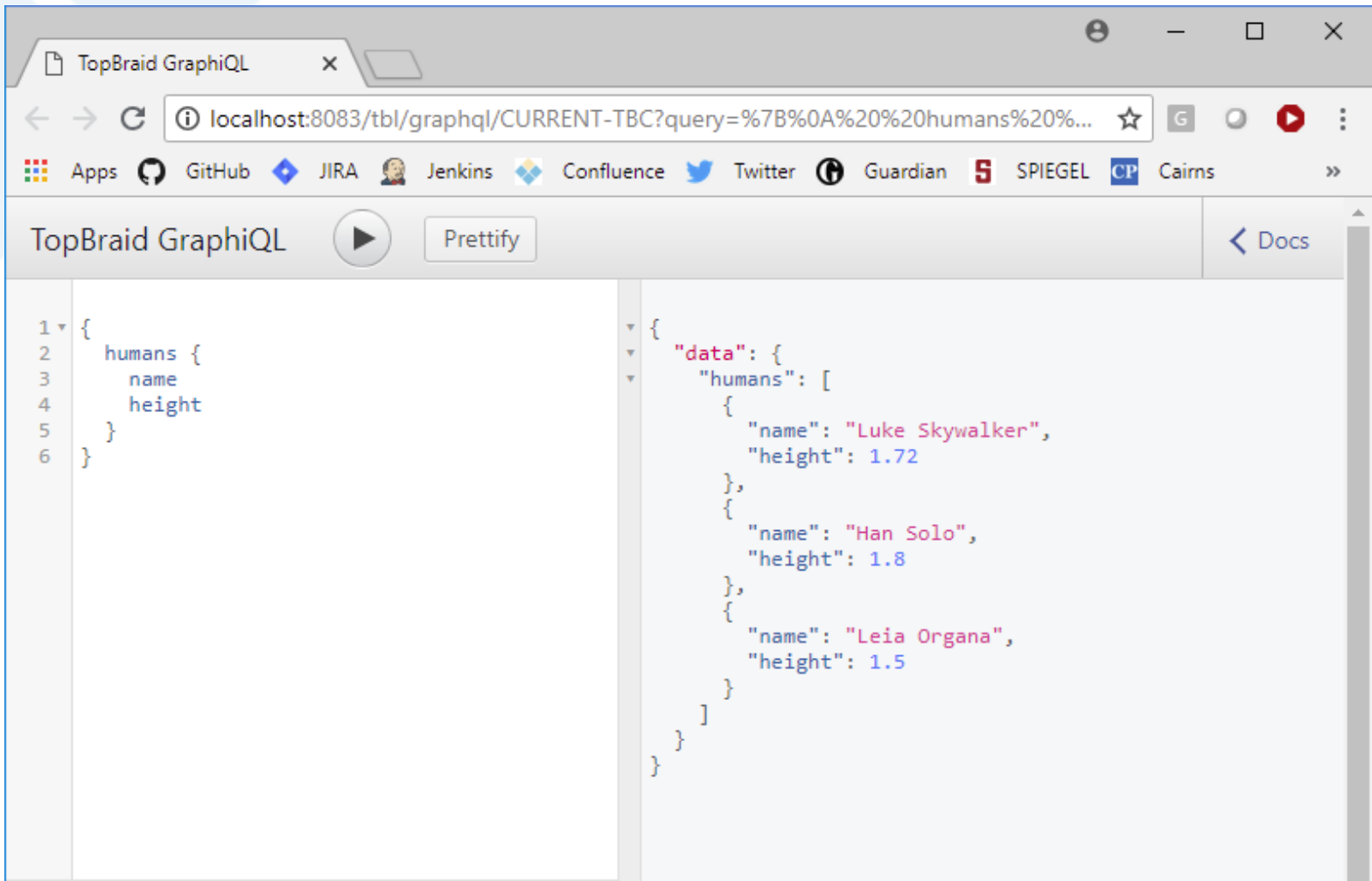
Constraints

property shapes:

Instances of Human

Show entries

Human
Character-1000
Character-1002



The screenshot shows a web browser window with the URL `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20humans%20%20...`. The browser's address bar and tabs are visible. Below the browser, the TopBraid GraphQL interface is shown. It includes a 'Prettify' button and a 'Docs' link. The interface is split into two panes. The left pane shows a GraphQL query:

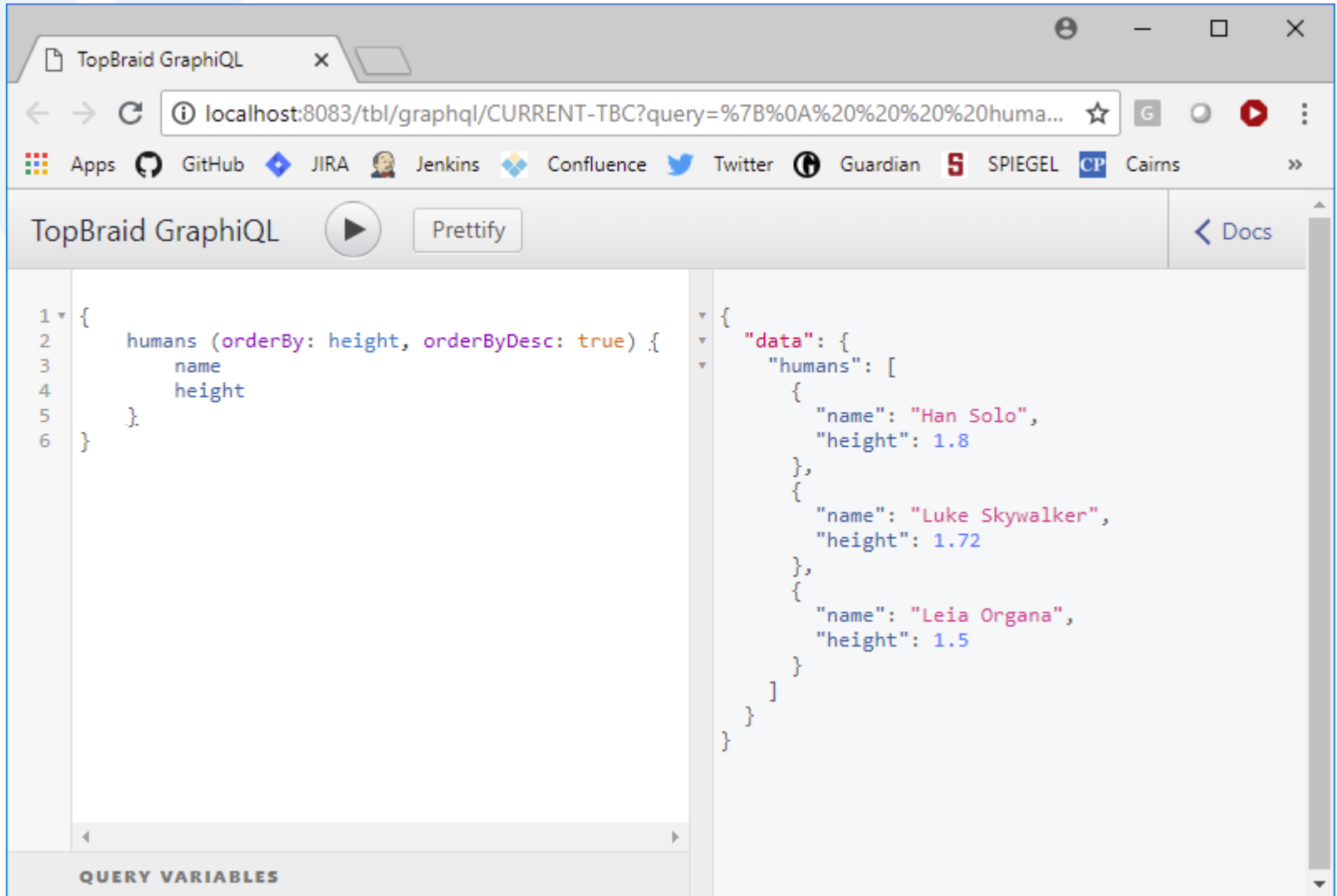
```
1 {
2   humans {
3     name
4     height
5   }
6 }
```

The right pane shows the JSON response:

```
{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker",
        "height": 1.72
      },
      {
        "name": "Han Solo",
        "height": 1.8
      },
      {
        "name": "Leia Organa",
        "height": 1.5
      }
    ]
  }
}
```

Available in all server products (e.g., TopBraid EDG) and in TBC-ME

Ordering



The screenshot shows the TopBraid GraphQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20%20%20%20huma...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following GraphQL query:

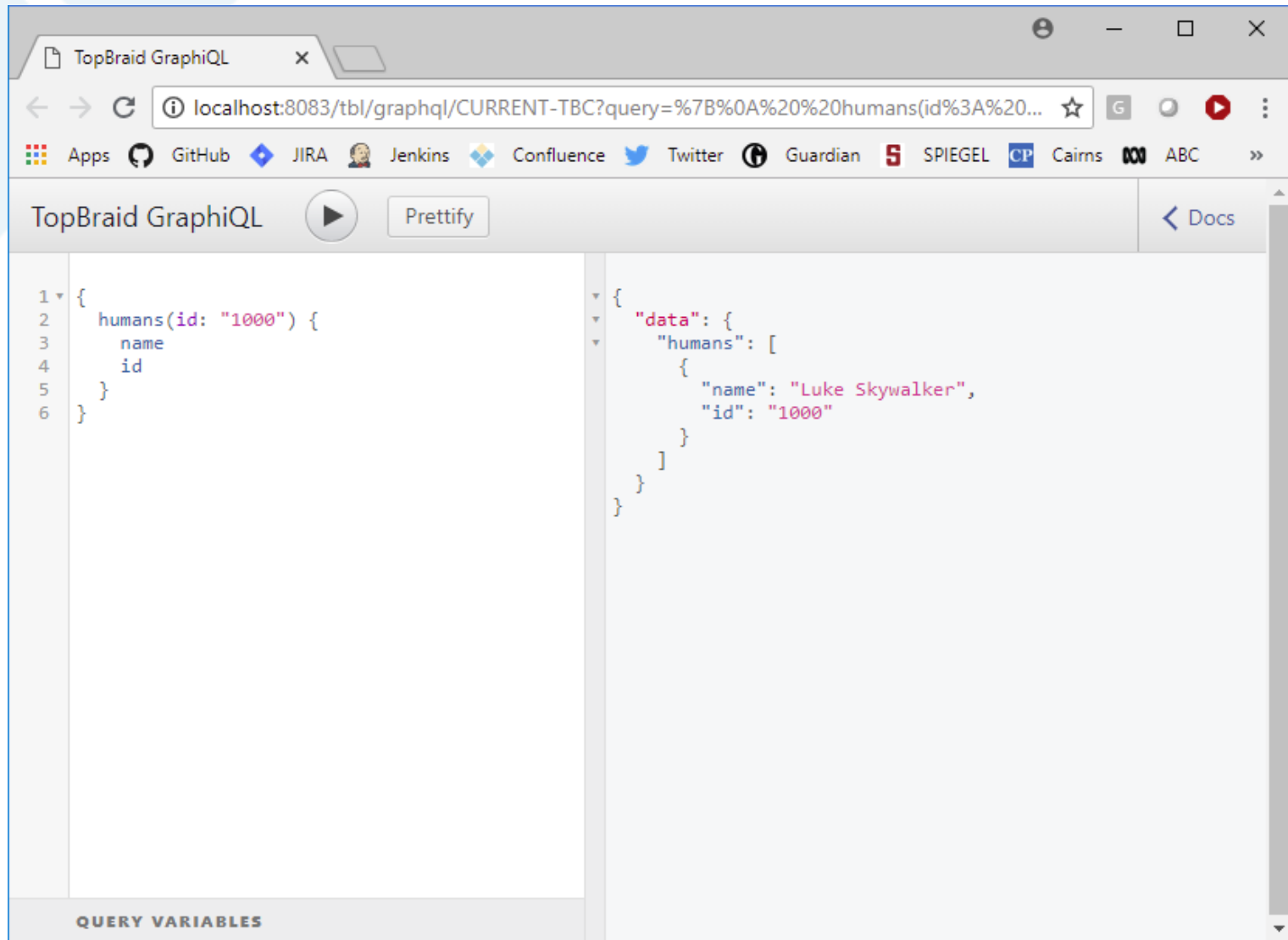
```
1 {
2   humans (orderBy: height, orderByDesc: true) {
3     name
4     height
5   }
6 }
```

The JSON response on the right is:

```
{
  "data": {
    "humans": [
      {
        "name": "Han Solo",
        "height": 1.8
      },
      {
        "name": "Luke Skywalker",
        "height": 1.72
      },
      {
        "name": "Leia Organa",
        "height": 1.5
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Matching



The screenshot shows the TopBraid GraphiQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20humans(id%3A%20...`. The interface includes a "Prettify" button and a "Docs" link. The left pane shows the following GraphQL query:

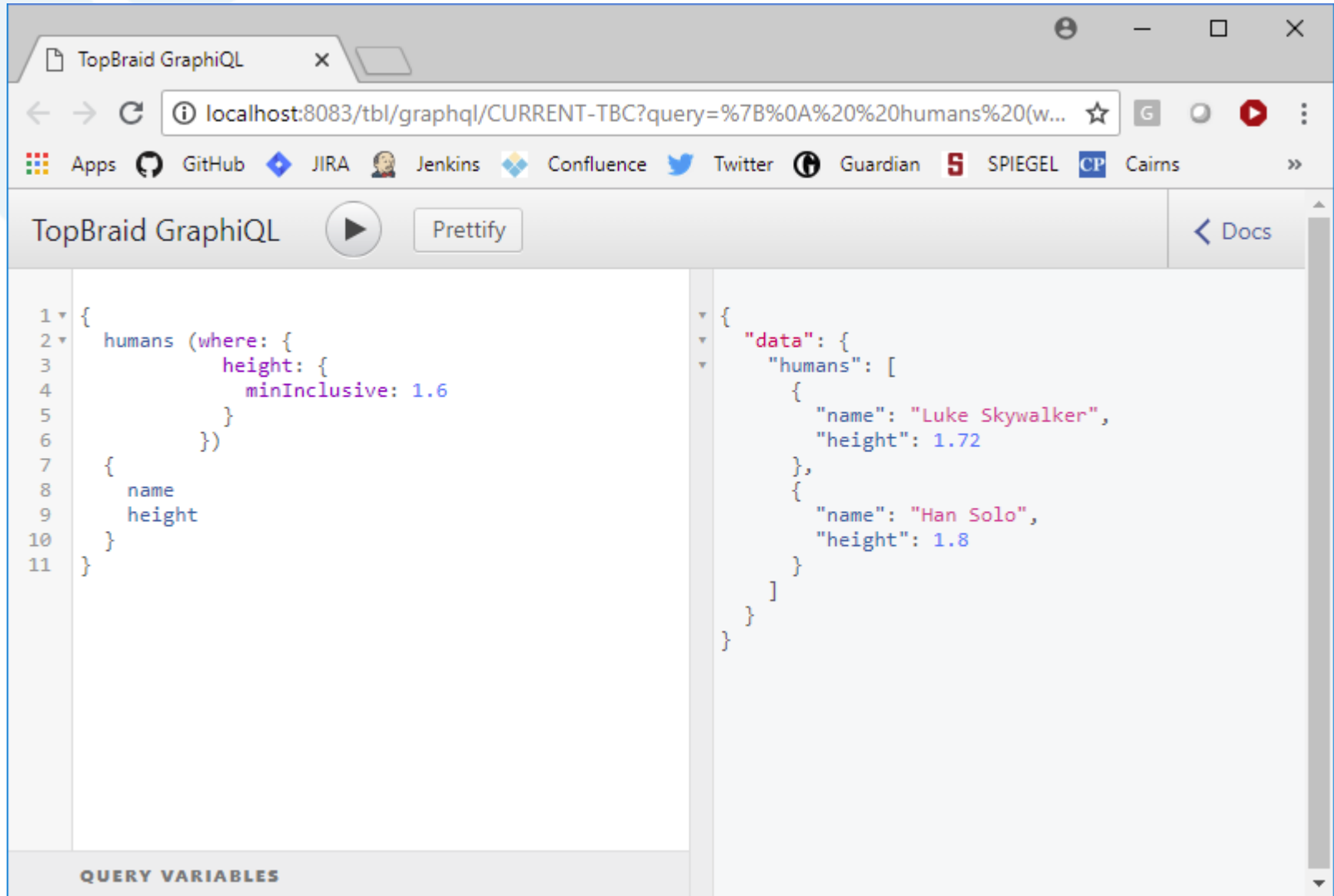
```
1 {
2   humans(id: "1000") {
3     name
4     id
5   }
6 }
```

The right pane shows the JSON response:

```
{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker",
        "id": "1000"
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Filtering



The screenshot shows the TopBraid GraphiQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20humans%20(w...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following GraphQL query:

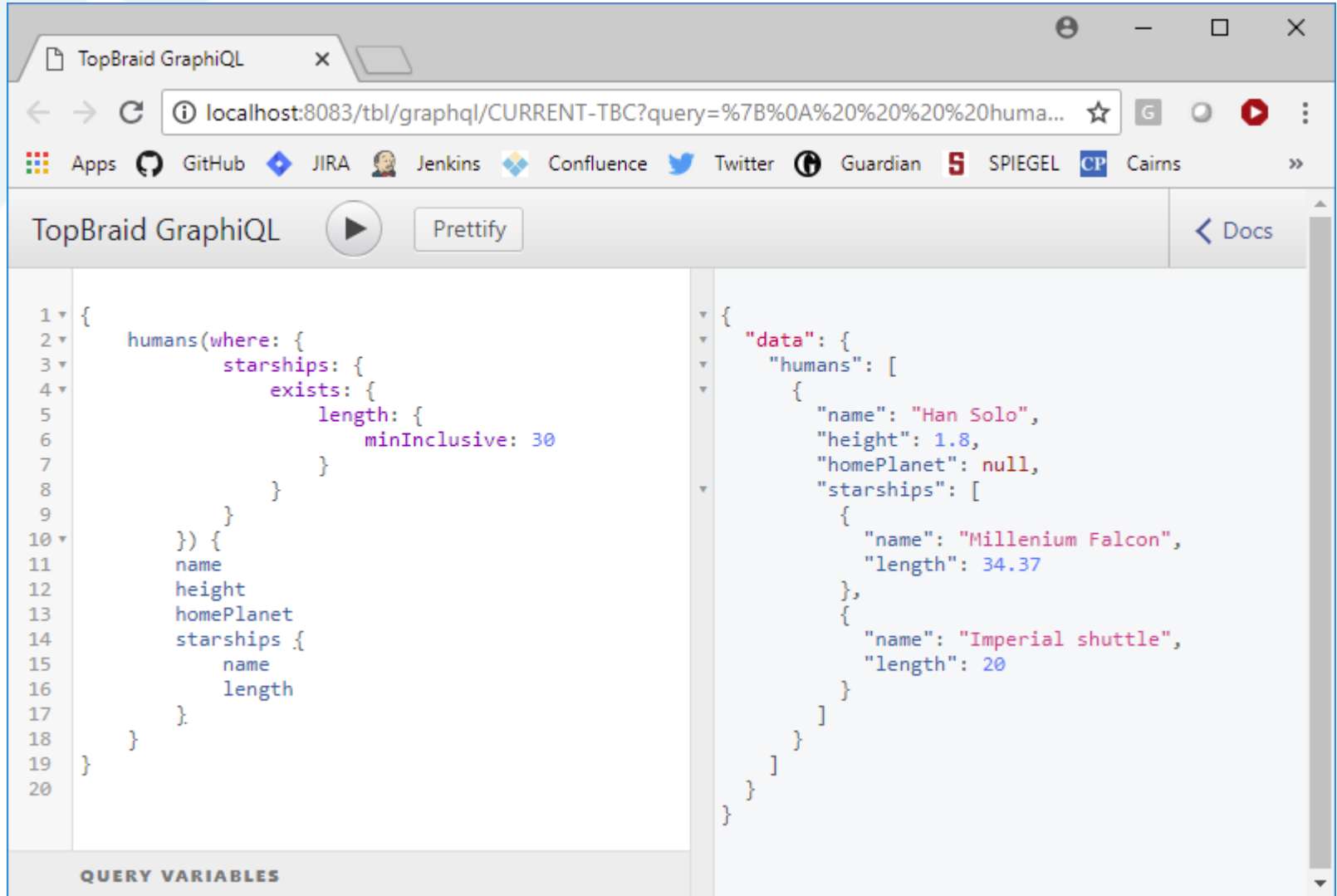
```
1 {
2   humans (where: {
3     height: {
4       minInclusive: 1.6
5     }
6   })
7   {
8     name
9     height
10  }
11 }
```

The response editor on the right shows the JSON output:

```
{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker",
        "height": 1.72
      },
      {
        "name": "Han Solo",
        "height": 1.8
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Filtering (nested)



The screenshot shows the TopBraid GraphiQL interface. The browser address bar contains the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20%20%20huma...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following GraphQL query:

```

1 {
2   humans(where: {
3     starships: {
4       exists: {
5         length: {
6           minInclusive: 30
7         }
8       }
9     }
10  }) {
11    name
12    height
13    homePlanet
14    starships {
15      name
16      length
17    }
18  }
19 }
20

```

The response editor on the right shows the JSON output:

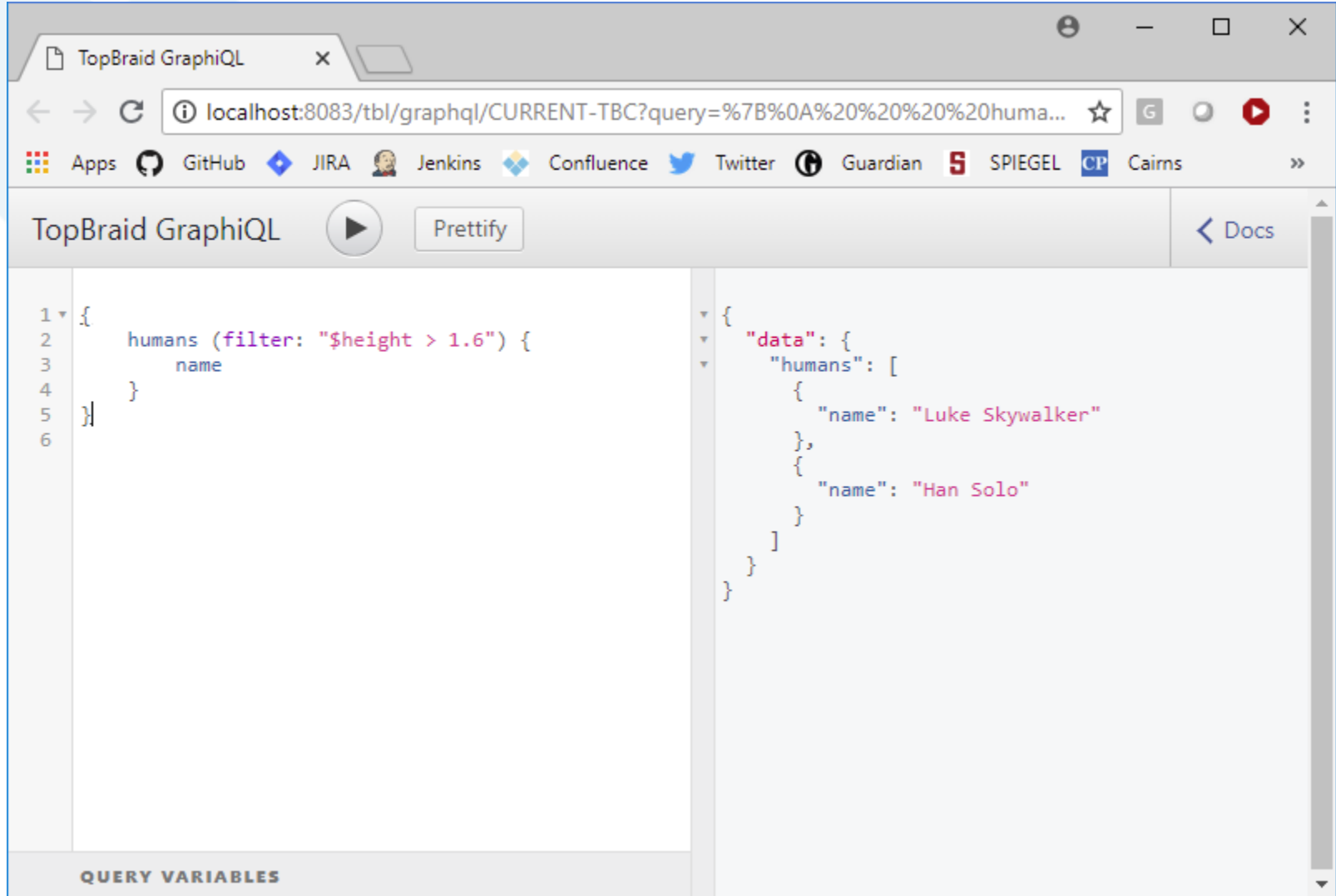
```

{
  "data": {
    "humans": [
      {
        "name": "Han Solo",
        "height": 1.8,
        "homePlanet": null,
        "starships": [
          {
            "name": "Millenium Falcon",
            "length": 34.37
          },
          {
            "name": "Imperial shuttle",
            "length": 20
          }
        ]
      }
    ]
  }
}

```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Filtering with SPARQL



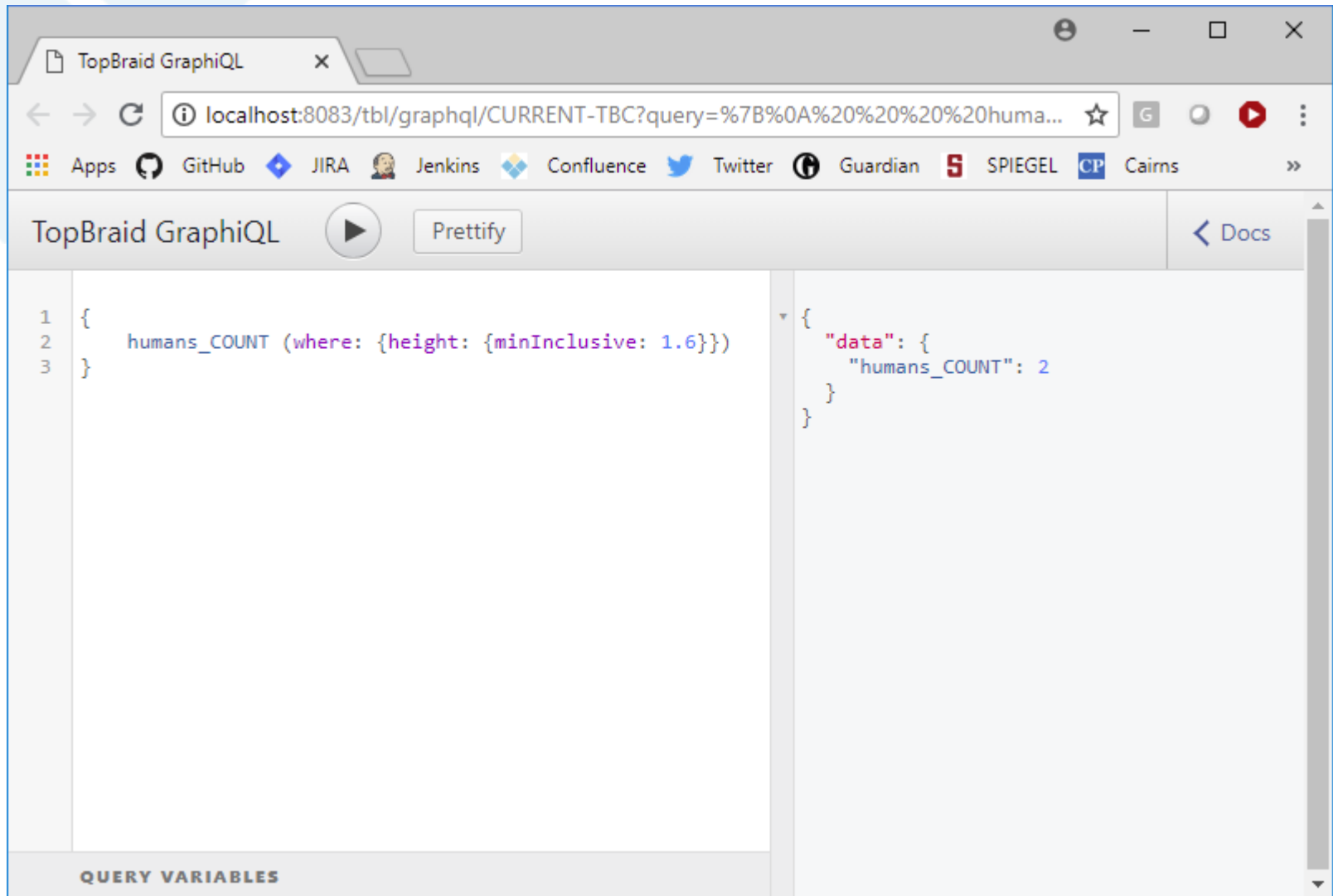
The screenshot shows the TopBraid GraphiQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20%20%20huma...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following SPARQL query:

```
1 {
2   humans (filter: "$height > 1.6") {
3     name
4   }
5 }
6
```

The JSON response on the right is:

```
{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker"
      },
      {
        "name": "Han Solo"
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".



The screenshot shows the TopBraid GraphQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20%20%20huma...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following GraphQL query:

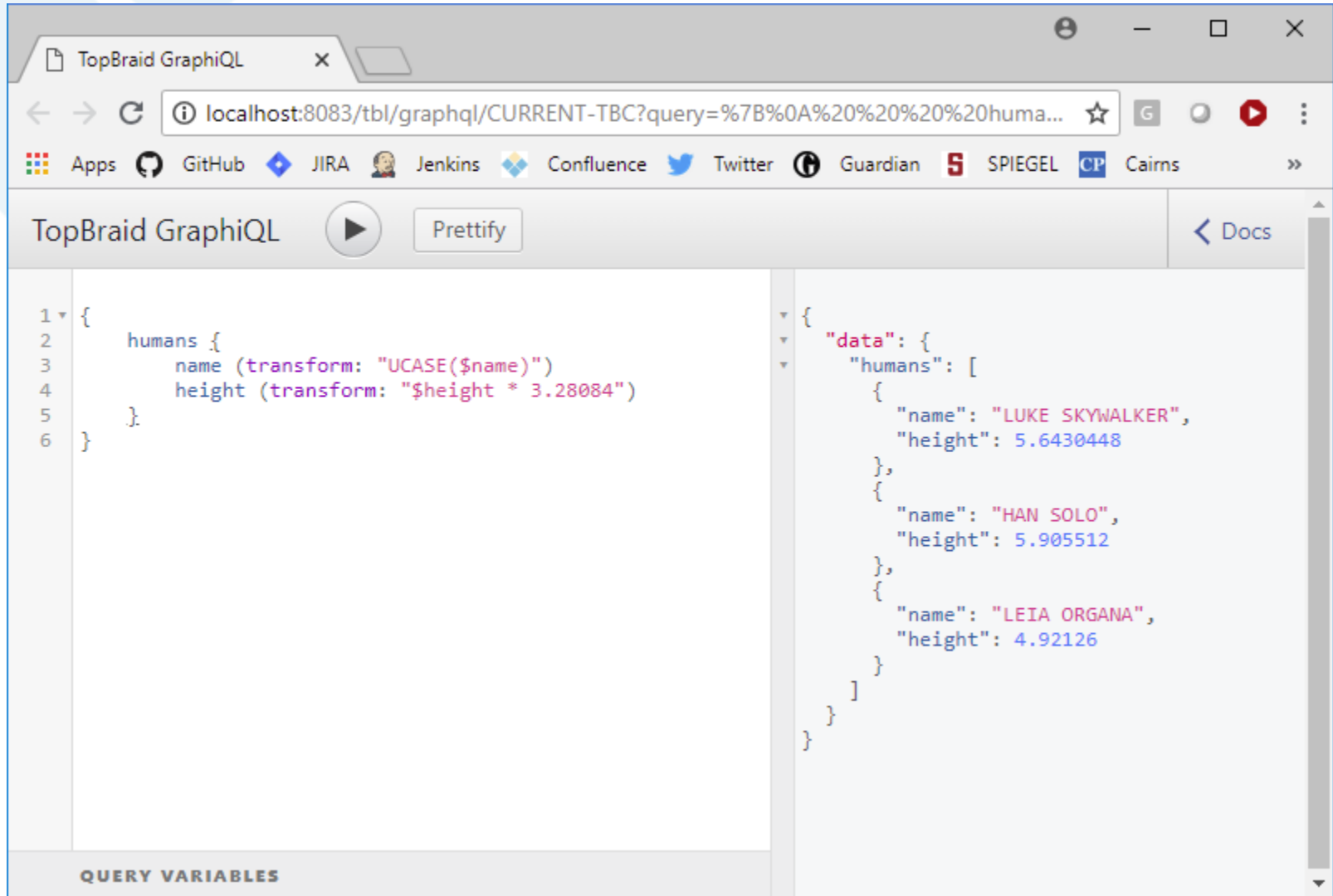
```
1 {  
2   humans_COUNT (where: {height: {minInclusive: 1.6}})  
3 }
```

The JSON response on the right is:

```
{  
  "data": {  
    "humans_COUNT": 2  
  }  
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Transforming



The screenshot shows the TopBraid GraphiQL interface. The browser address bar displays the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%7B%0A%20%20%20%20%20huma...`. The interface includes a "Prettify" button and a "Docs" link. The left pane contains the following GraphQL query:

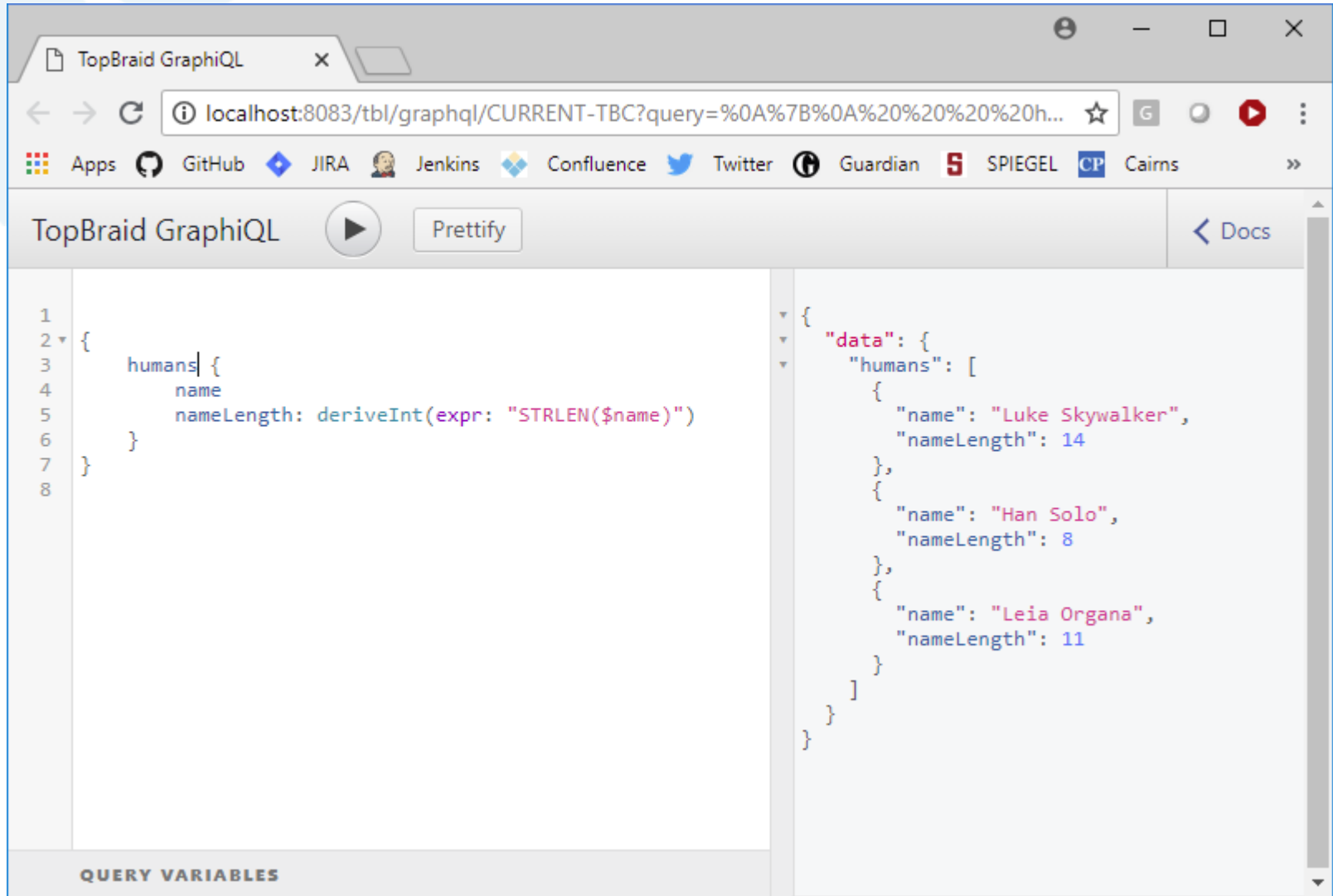
```
1 {
2   humans {
3     name (transform: "UCASE($name)")
4     height (transform: "$height * 3.28084")
5   }
6 }
```

The right pane shows the resulting JSON response:

```
{
  "data": {
    "humans": [
      {
        "name": "LUKE SKYWALKER",
        "height": 5.6430448
      },
      {
        "name": "HAN SOLO",
        "height": 5.905512
      },
      {
        "name": "LEIA ORGANA",
        "height": 4.92126
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

Deriving Values



The screenshot shows the TopBraid GraphiQL interface. The browser address bar shows the URL: `localhost:8083/tbl/graphql/CURRENT-TBC?query=%0A%7B%0A%20%20%20%20h...`. The interface includes a "Prettify" button and a "Docs" link. The query editor on the left contains the following GraphQL query:

```
1 {
2   {
3     humans {
4       name
5       nameLength: deriveInt(expr: "STRLEN($name)")
6     }
7   }
8 }
```

The JSON response on the right is:

```
{
  "data": {
    "humans": [
      {
        "name": "Luke Skywalker",
        "nameLength": 14
      },
      {
        "name": "Han Solo",
        "nameLength": 8
      },
      {
        "name": "Leia Organa",
        "nameLength": 11
      }
    ]
  }
}
```

At the bottom of the interface, there is a section labeled "QUERY VARIABLES".

TopBraid EDG™ Enterprise Data Governance Northwind Global Lookup Hello, Administrator

Assets Dashboard Settings Users Import Transform Export Reports Workflows Tasks Comments Manage

Type Database Column Free Text Advanced Filters Columns

is primary key any... any value

New Details Bulk Edit Delete Clone Export

Show 25 entries Filter:

<input type="checkbox"/>	Database Column	type	total number of values	is primary key
<input type="checkbox"/>	CATEGORYID (NORTHWIND.DBO.CATEGORIES)	Database Column	8	true
<input type="checkbox"/>	CUSTOMERID (NORTHWIND.DBO.CUSTOMERCUSTOMERDEMO)	Database Column	0	true
<input type="checkbox"/>	CUSTOMERID (NORTHWIND.DBO.CUSTOMERS)	Database Column	91	true
<input type="checkbox"/>	CUSTOMERTYPEID (NORTHWIND.DBO.CUSTOMERCUSTOMERDEMO)	Database Column	0	true
<input type="checkbox"/>	CUSTOMERTYPEID (NORTHWIND.DBO.CUSTOMERDEMO)	Database Column	0	true
<input type="checkbox"/>	EMPLOYEEID (NORTHWIND.DBO.EMPLOYEES)	Database Column	9	true
<input type="checkbox"/>	EMPLOYEEID (NORTHWIND.DBO.EMPLOYEEETERMINATIONS)	Database Column	0	true
<input type="checkbox"/>	ORDERID (NORTHWIND.DBO.ORDER DETAILS)	Database Column	99	true

```

{
  results: databaseColumns(where: {isPrimaryKey: {minCount: 1}}) {
    uri
    label
    type {
      uri
      label
    }
    nonNullValuesCount
    isPrimaryKey
  }
}

```

GraphQL Mutations (Updates)

Example GraphQL Query

```
mutation {
  createHuman (input: {
    uri: "http://example.org/Humans/123",
    id: "123",
    name: "Darth Vader"
  })
  updateHuman (input: {
    uri: "http://example.org/Humans/456",
    father: {
      uri: "http://example.org/Humans/123"
    }
  })
  results {
    addedCount
    deletedCount
  }
  commit (message: "Added Luke's dad")
}
```

Example JSON Result

```
{
  "data": {
    "createHuman": true,
    "updateHuman": true,
    "results": {
      "addedCount": 4,
      "deletedCount": 0
    },
    "commit": "Added Luke's dad"
  }
}
```

JSON to RDF Conversion

```
{
  "human": {
    "id": "HAN",
    "name": "Han Solo",
    "friends": [
      { "id": "LEIA" },
      { "id": "LUKE" }
    ]
  }
}
```

```
{
  "human": {
    "id": "LUKE",
    "name": "Luke Skywalker",
    "friends": [
      { "id": "HAN" }
    ]
  }
}
```

```
{
  "human": {
    "id": "LEIA",
    "name": "Leia Organa",
    "friends": null
  }
}
```

Example GraphQL Schema

```
schema {
  query: Query
}

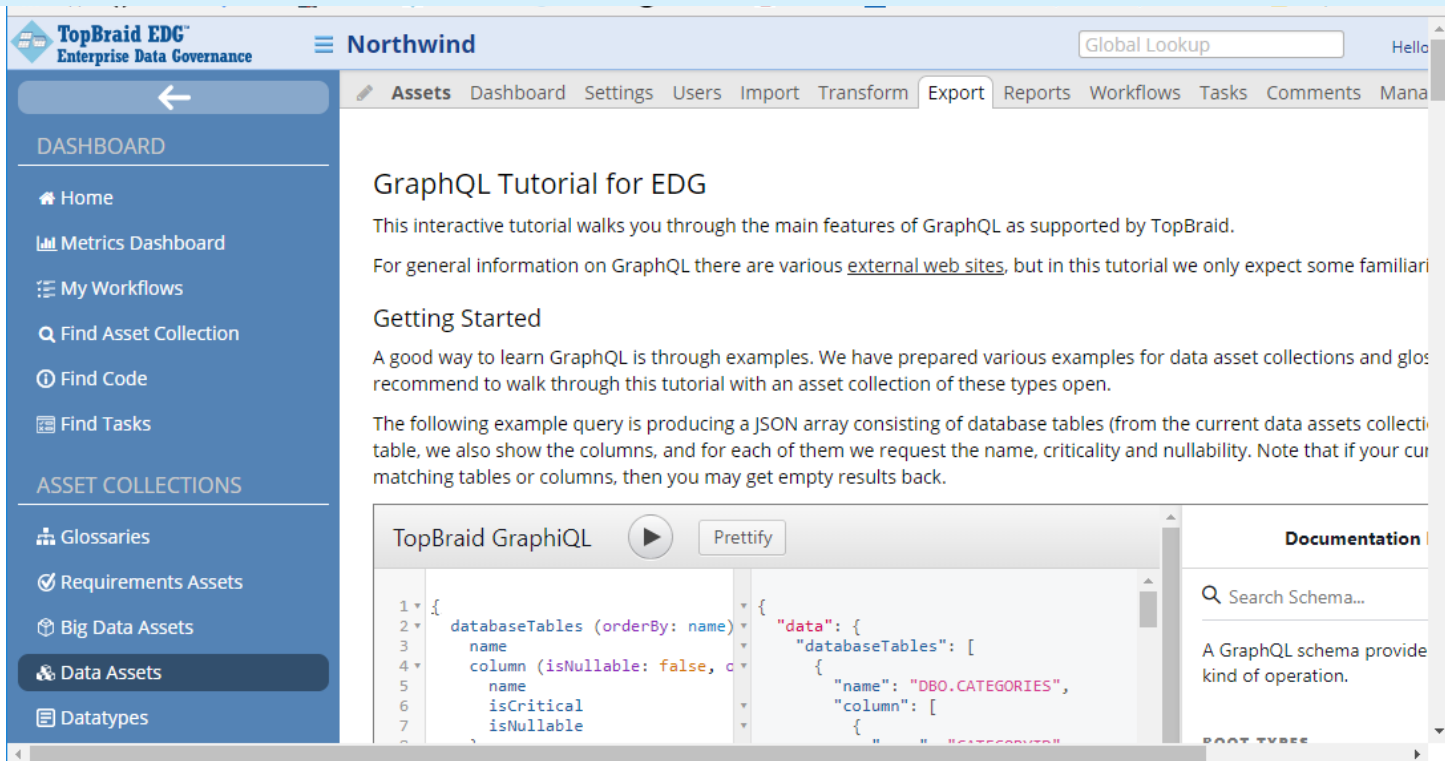
type Query {
  human (id: ID!): [Human]
}

type Human @uri(template: "http://example.org/human/{$id}") {
  id: ID!
  name: String!
  friends: [Human]
}
```

URI from JSON Object	Field	Value
http://example.org/human/HAN	id	"HAN"
http://example.org/human/HAN	name	"Han Solo"
http://example.org/human/HAN	friends	http://example.org/human/LEIA
http://example.org/human/HAN	friends	http://example.org/human/LUKE
http://example.org/human/LEIA	id	"LEIA"
http://example.org/human/LEIA	name	"Leia Organa"
http://example.org/human/LUKE	id	"LUKE"
http://example.org/human/LUKE	name	"Luke Skywalker"
http://example.org/human/LUKE	friends	http://example.org/human/HAN

Next Steps

To try this out, download TBC-ME 6.0 or request a server evaluation account for TopBraid EDG at <https://www.topquadrant.com/products/topbraid-enterprise-data-governance/request-edg-evaluation-account/> GraphQL tutorials and samples are built in. To access, go to an Export tab of any asset collection.



The screenshot shows the TopBraid EDG Enterprise Data Governance interface. The top navigation bar includes "Assets", "Dashboard", "Settings", "Users", "Import", "Transform", "Export", "Reports", "Workflows", "Tasks", "Comments", and "Manage". The left sidebar contains navigation options like "Home", "Metrics Dashboard", "My Workflows", "Find Asset Collection", "Find Code", "Find Tasks", "Glossaries", "Requirements Assets", "Big Data Assets", "Data Assets", and "Datatypes". The main content area displays a "GraphQL Tutorial for EDG" with introductory text and a "Getting Started" section. Below the text is a "TopBraid GraphiQL" editor showing a query and its JSON response. The query is:

```
1 {
2   databaseTables (orderBy: name) {
3     name
4     column (isNullable: false, c
5       name
6       isCritical
7       isNullable
```

 The response is:

```
{
  "data": {
    "databaseTables": [
      {
        "name": "DBO.CATEGORIES",
        "column": [
          {
            "name": "CATEGORYID",
            "isCritical": true,
            "isNullable": false
```

also read <https://www.topquadrant.com/technology/graphql/>