Defining Property Graph Schemas using the GraphQL Schema Definition Language

Olaf Hartig
@olafhartig
What is GraphQL?

- State-of-the art approach to create Web APIs to retrieve data for Web and mobile applications
- Alternative to the notion of REST-based Web APIs
- Developed and used by Facebook since 2012
- Made available to the public (open source) in 2015
  - Spec and reference implementation
- Based on a simple, JSON-like query language
GraphQL Schema Definition Language (SDL)

- Language to define *GraphQL schema*
- Specifies the types of objects that can be queried when accessing a specific GraphQL Web API
GraphQL SDL Example

declaration of an object type with its fields and their types

type Starship {
    id: ID!
    name: String!
    length(unit: String): Float
}

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

type Droid implements Character {
    id: ID!
    name: String!
    friends: [Character]
    appearsIn: [Episode]!
    primaryFunction: String
}

declaration of an interface type and an implementation

type Human implements Character {
    id: ID!
    name: String!
    friends: [Character]
    appearsIn: [Episode]!
    starships: [Starship]
    totalCredits: Int
}

union SearchResult = Human | Droid | Starship

data Episode {
    NEWHOPE, EMPIRE, JEDI
}

data Query {
    hero(episode: Episode!): Character
    droid(id: ID!): Droid
    node(id: ID!): SearchResult
}

declaration of a union type

(possible root fields of queries)
Can we use this language to define schemas for Property Graphs?
Property Graph Schemas with GraphQL SDL

Example Property Graph schema defined using the GraphQL SDL

type person {
  name: String!
  age: Int
  knows(weight:Float!): [person]  @distinct @noloops
  created(weight:Float!): [software] @distinct @requiredForTarget
}

type software {
  name: String!
  lang: Language
}

enum Language {
  java
  javascript
  python
}
Node Types and Node Properties

type UserSession {
    id: ID!
    user: User!
    startTime: Time
    endTime: Time
}

scalar Time

type User {
    id: ID!
    loginName: String!
    name: String
}
(Outgoing) Edges

type A {
    name: String!
    favoriteB: B
    relatedA: [A]
}

type B {
    favoriteA: A!
    otherA: [A!]
}

type C {
    otherC: [C!] @distinct @noloops
    b: [B] @distinct
}

type D {
    a: [A] @uniqueForTarget
    b: [B] @requiredForTarget
    c: [C] @uniqueForTarget @requiredForTarget
}
Cardinality Restrictions

- 1:1 relationship

  rel: B @uniqueForTarget

- 1:N relationship

  rel: B

- N:1 relationship

  rel: [B] @uniqueForTarget

- N:M relationship

  rel: [B]
Multiple Types of Target Nodes 1/2

type Person {
  id: ID!
  name: String!
  favoriteVehicle: Vehicle
}

union Vehicle = Car | Motorcycle

type Car {
  brand: String!
  color: String
}

type Motorcycle {
  brand: String!
}
Multiple Types of Target Nodes 2/2

type Person {
  id: ID!
  name: String!
  favoriteVehicle: Vehicle
}

interface Vehicle {
  brand: String!
}

type Car implements Vehicle {
  brand: String!
  color: String
}

type Motorcycle implements Vehicle {
  brand: String!
}
Multiple Types of Source Nodes

type Car {
    brand: String!
    color: String
    owner: Person
}

type Motorcycle {
    brand: String!
    owner: Person
}

type Person {
    name: String!
}
Edge Properties

type UserSession {
    id: ID!
    user(certainty: Float! comment: String): User!
    startTime: Time
    endTime: Time
}