GraphQL Schema Language Cheatsheet

GraphQL Schema Definition Language (SDL) allows teams of programmers and non-programmers alike to easily describe the shape of a GraphQL API using non-ambiguous shared terminology.

Operation type

GraphQL currently supports 3 operation types:

`query` - read data
`mutation` - modify data / trigger action
`subscription` - run a query whenever an event occurs

Schema keyword

Defines the object type associated with each operation type.

```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}
```

Built-in scalars

`Boolean` - true or false
`Int` - 32-bit signed integer (± 2,147,483,647)
`Float` - double precision floating point number
`String` - a sequence of UTF8 characters
`ID` - opaque string to identify a node

User defined custom scalars:

```
scalar JSON
scalar DateTime
```

Object type

Composed of fields which may optionally accept arguments:

```
"""Documentation for MyType"""
type MyType {
  """textField description"""
  textField: String

  moreComplexField(
    """argument1 description"""
    argument1: Int
    argument2: Float
  ): Int!
}
```

Object field resolvers

Each field in an object type is backed by an (optionally) asynchronous resolver function which can be called in parallel and fetches the required data.

Nullable by default

Types in GraphQL are nullable by default to allow for partial successes (e.g. where part of the query cannot be served, but another part can). When a error is raised by a non-null field, it will cascade up the tree until it reaches a nullable field.

Non-null variants

```
nonNullString: String!
nonNullMedia: Media!
```

List (array) variants:

```
listOfPeople: [Person]
listOfNonNullInts: [Int!]
nonNullListOfNonNullMedia: [Media!]!
```

Enumeration type

Specifies a list of allowed values

```
enum AcceptableUseOfBakedBeans {
  ON_TOAST
  DIPPING_CHIPS
  BATHING
}
```

PostGraphile instantly builds a best-practices GraphQL API from your PostgreSQL database.
By converting each GraphQL query tree into a single SQL statement, PostGraphile solves server-side under- and over-fetching and eliminates the N+1 problem, leading to an incredibly high-performance GraphQL API.

PostGraphile is open source on GitHub, try it out today.

GraphQL Schema Language Cheatsheet Continued...

Interface type

Abstract type representing fields shared by multiple object types.

```
interface Media {
  price: Int
  title: String!
}
type Book implements Media {
  price: Int
  title: String!
  numberOfPages: Int
  authors: [Person!]
}
type Film implements Media {
  price: Int
  title: String!
  duration: Float
  directors: [Person!]
}
```

Union type

Represents one of a list of types that don't need common fields.

```
union Entity = Person | Film | Book | Publisher
```

Input object type

Allows user to input structured data via field arguments (particularly useful for mutations, also useful for search/filters/etc):

```
input HumanSearchFilter {
  id: ID
  name: String
  profession: String
}

type Query {
  humans(filter: HumanFilter): [Human!]

}
```

Versionless design

GraphQL enables and encourages versionless API design; enabling APIs to evolve over time without breaking the assumptions of existing code, or inflating payloads with new fields that were not requested. This helps to reduce or even eliminate the burden of maintaining multiple versioned API endpoints.

"Non-breaking" (safe) schema changes include:

- Adding a field
- Adding a nullable argument
- Changing an output field from nullable to non-nullable
- Adding a new type or scalar
- Adding an interface to a type
- Deprecating a field
- Changing descriptions of types, fields, etc
- Adding another operation type

"Breaking changes" include:

- Making a non-nullable output field nullable
- Making a nullable argument non-nullable
- Removing a field or type
- Renaming a field or type
- Changing the type of a field

Notes...

PostGraphile

PostGraphile instantly builds a best-practices GraphQL API from your PostgreSQL database.
By converting each GraphQL query tree into a single SQL statement, PostGraphile solves server-side under- and over-fetching and eliminates the N+1 problem, leading to an incredibly high-performance GraphQL API.

PostGraphile is open source on GitHub, try it out today.