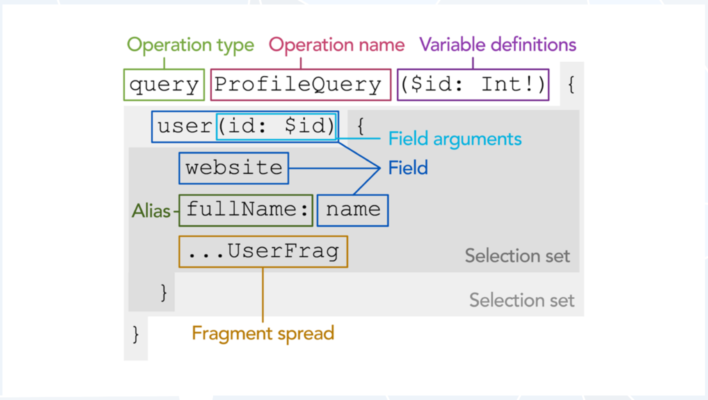
GraphQL Operation Cheatsheet

A request to a GraphQL schema is called an "operation," and each operation has an "operation type":

`query` — read and traverse data
`mutation` — modify data or trigger an action
`subscription` — run a query when an event occurs

Anatomy of a GraphQL operation

All operations use the same syntax. Here's an example of a GraphQL query with the various parts labelled:



```
query ProfileQuery ($id: Int!) {
  user(id: $id) {
    website
    fullName: name
    ...UserFrag
  }
}
```

The **operation type** ("`query`") is required for mutation and subscription operations, but for a query with no operation name and no variables it is optional.

The **operation name** is always optional, it is generally used for referring to queries conveniently, logging and debugging.

The **variable definitions** (including brackets) should only be included if there are variables; each variable starts with a dollar symbol (`$`) and must declare its type.

Each operation must have a **selection set**.

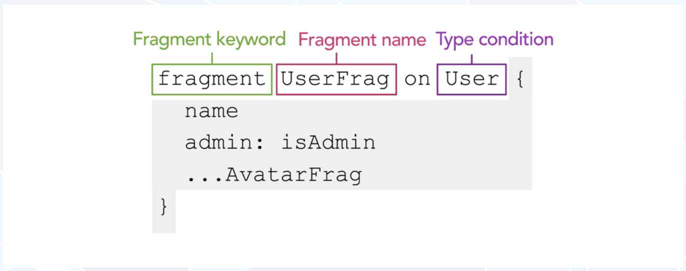
Each selection set must contain at least one **field** or **fragment spread**. Some fields can accept **arguments**. All fields (except scalar (leaf) fields) require a selection set. Operations must also include the definitions of any fragments they use (and only fragments that are used).

Fragments

Fragments in GraphQL are shared pieces of query logic, which allow you to reduce repetition in your queries, or to request fields on a subtype of a union or interface type. There are two types of fragments: named fragments and inline fragments.

Anatomy of a named fragment

The fragment spread in the above `ProfileQuery` refers to a fragment named "`UserFrag`". Here's how the "named fragment" UserFrag might be defined:



```
fragment UserFrag on User {
  name
  admin: isAdmin
  ...AvatarFrag
}
```

A named fragment is composed of the "`fragment`" keyword, a name, a "type condition" and a selection set. The type condition tells the system which types the fragment can be used on. The selection set, like elsewhere, can include fields, nested selection sets, arguments, aliases and additional fragment spreads; however a fragment spread may not create a cycle.

Inline fragment

When you are querying a field which returns a union or interface type, there are multiple concrete types the data may end up being. An inline fragment allows you to query fields on one concrete type from the interface or union without causing errors if the other types do not support that field, and without having to create a named fragment.

```
{
  search(phrase: "cream") {
    ... on Food {
      manufacturer
    }
  }
}
```

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.

graphile.org **@GraphileHQ** **graphile** **team@graphile.org**

GraphQL Operation Cheatsheet Continued...

Mutations

The syntax for mutations is the same as for queries, but the root fields are expected to perform a data mutation or action of some kind. For this reason, GraphQL mutation root fields run in series rather than parallel. Only the root fields in a mutation request perform mutations, nested fields are used to query the result of the mutation (if there is one).

```
mutation TagPost($id: ID!, $tag: String!) {
  addTagToPost (postId: $id, tag: $tag) {
    post {
      id
      tags
    }
  }
}
```

Subscriptions

Some GraphQL servers support real-time functionality thanks to a feature called "subscriptions." These subscriptions tell the server that whenever a particular event occurs (for example a new post is added, or someone likes your status), the server should execute the given query and send the result to the user.

Unlike queries and mutations, subscriptions only have one root-level field, and can send the client zero or more payloads conforming to the selection set. Subscriptions are generally long-running requests; they can be executed over a wide array of transports but they typically involve websockets.

```
subscription NotificationSubscription {
  notificationReceived {
    eventTimestamp
    notification {
      id
      text
    }
  }
}
```

GraphQL over HTTP

It's very common to expose a GraphQL schema over HTTP. When doing so, the main inputs are the **`query`** string (the GraphQL request document, could be a query or a mutation), and the **`variables`**. If the request document contains multiple operations, then the **`operationName`** of the operation to execute should also be specified.

The response from a successful GraphQL HTTP request will typically be a JSON object with the result "`data`" and any "`errors`" that occurred.

```
window.fetch("/graphql", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Accept": "application/json"
  },
  body: JSON.stringify({
    query,
    variables,
    operationName
  })
})
```

Normally you'd use a specialised GraphQL client to communicate with a GraphQL server, these clients can lead to a great developer experience. GraphiQL (pronounced like "graphical") is most people's first GraphQL client, and it's a great way to explore a GraphQL API. There are GraphQL clients available for most major programming languages; in JavaScript the big two are Apollo Client and Facebook's Relay.

Introspection

GraphQL has a powerful introspection system built in for finding out all about your GraphQL schema — this is how GraphiQL populates the documentation browser and auto-complete. Advanced tooling in many editors can use this to give instant feedback of the validity of your GraphQL queries. The introspection system in GraphQL uses and reserves names that begin with two underscores, such as '`__typename`'. Try exploring the '`__schema`' root query field in GraphiQL .

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.

graphile.org @GraphileHQ graphile team@graphile.org