Gatsby's `graphql` tag enables page components to query data via a GraphQL query.

In this guide, you will learn how to use the `graphql` tag in your pages, as well as go a little deeper into how the `graphql` tag works.

If you're curious, you can also read more about why Gatsby uses GraphQL.

How to use the `graphql` tag in pages

Page components can have their own query. The query can use variables passed in when creating the page to select data for that page.

Let's do a quick tutorial on writing a simple page query.

Add `description` to `siteMetadata`

Add a description to your site's metadata.

```
module.exports = {
  siteMetadata: {
    title: "My Homepage",
    description: "This is where I write my thoughts.",
  },
}
```

Make a basic index page

Create an home page (`src/pages/index.js`) like so:

```
import React from "react"

const HomePage = () => {
  return <div>Hello!</div>
}

export default HomePage
```

Add the `graphql` query

The first thing to do is import `graphql` from Gatsby. At the top of `index.js` add:

```
import * as React from 'react'
+ import { graphql } from 'gatsby'

const HomePage = () => {
  return (
    <div>
      Hello!
    </div>
  )
}
```

Below the `HomePage` component declaration, export a new constant called `query`. The name of the constant isn't important, as Gatsby looks for an exported `graphql` string from the file rather than a specific variable. Note that you can only have one page query per file.

Then, set the const variable's value to be a `graphql` tagged template with the query between two backticks:

```
const HomePage = () => {
  return (
    <div>
      Hello!
    </div>
  )
}

+ export const query = graphql`
+   # query will go here
+ `
```

The first part of writing the GraphQL query is including the operation (in this case "`query`") along with a name.

From exploring in the GraphQL IDE, GraphiQL, you've learned that one of the types that you can query is `site`, which in turn has its own `siteMetadata` field with subfields that correspond to the data provided in `gatsby-config.js`.

Putting this together, the completed query looks like:

```
export const query = graphql`
- # query will go here
+  query HomePageQuery {
+    site {
+      siteMetadata {
+        description
+      }
+    }
+  }
`
```

Use the query result in the `<HomePage />` component

The `data` prop contains the results of the page GraphQL query, and matches the shape of the query. With this in mind, the updated `HomePage` markup looks like:

```
import * as React from 'react'
import { graphql } from 'gatsby'

- const HomePage = () => {
+ const HomePage = ({data}) => {
  return (
    <div>
-     Hello!
```

```
+       {data.site.siteMetadata.description}
      </div>
    )
}

export const query = graphql`
  query HomePageQuery {
    site {
      siteMetadata {
        description
      }
    }
  }
`

export default HomePage
```

After restarting `gatsby develop`, your home page will now display "This is where I write my thoughts." from the description set in `gatsby-config.js`!

How does the `graphql` tag work?

`graphql` is a tag function. Behind the scenes Gatsby handles these tags in a particular way:

The short answer

During the Gatsby build process, GraphQL queries are pulled out of the original source for parsing.

The longer answer

The longer answer is a little more involved: Gatsby borrows a technique from Relay that converts your source code into an abstract syntax tree (AST) during the build step. `file-parser.js` and `query-compiler.js` pick out your `graphql`-tagged templates and effectively remove them from the original source code.

More information about how queries work is included in the Gatsby Internals section of the docs.

This means that the `graphql` tag isn't executed the way that JavaScript code is typically handled. For example, you cannot use expression interpolation with Gatsby's `graphql` tag. However, it's possible to pass variables into page queries with the `context` object when creating pages.

How to add query variables to a page query

Variables can be added to *page queries* (but not static queries) through the context object that is an argument of the `createPage` API.

Consider the following query:

```
export const query = graphql`
  query MdxBlogPost {
    mdx(title: { eq: "Using a Theme" }) {
      id
      title
```

```
    }
  }
`
```

The `MdxBlogPost` query will return an MDX node in a site where `gatsby-plugin-mdx` is installed and `.mdx` files have been sourced with `gatsby-source-filesystem`, so long as it matches the argument passed in for a `title` equaling (`eq`) the string `"Using a Theme"`.

In addition to hardcoding an argument directly into the page query, you can pass in a variable. The query can be changed to include a variable like this:

```
export const query = graphql`
  query MdxBlogPost($title: String) { // highlight-line
    mdx(title: {eq: $title}) { // highlight-line
      id
      title
    }
  }
`
```

When a page is created dynamically from this blog post template in `gatsby-node.js`, you can provide an object as part of the page's context. Keys in the context object that match up with arguments in the page query (in this case: `"title"`), will be used as variables. Variables are prefaced with `$`, so passing a `title` property will become `$title` in the query.

```
posts.forEach(({ node }, index) => {
  createPage({
    path: node.fields.slug,
    component: path.resolve(`./src/templates/blog-post.js`),
    // values in the context object are passed in as variables to page queries
    context: {
      title: node.title, // "Using a Theme"
    },
  })
})
```

For more information, check out the docs on creating pages programmatically.