1. Practice querying with Countries GraphQL API

Endpoint Url: https://countries.trevorblades.com/

API Used: https://github.com/trevorblades/countries

Schema: https://github.com/trevorblades/countries/blob/main/src/schema.ts

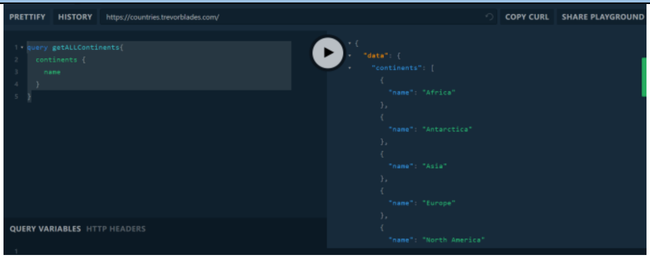
Overview:

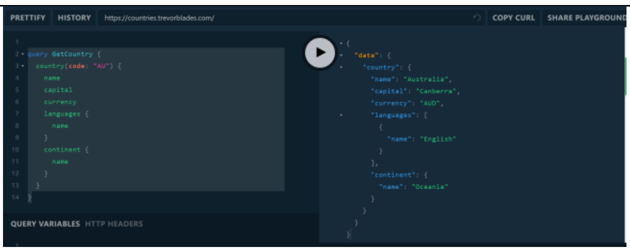
A public GraphQL API for information about countries, continents, and languages. This project uses Countries List and provinces as data sources, so the schema follows the shape of that data, with a few exceptions:

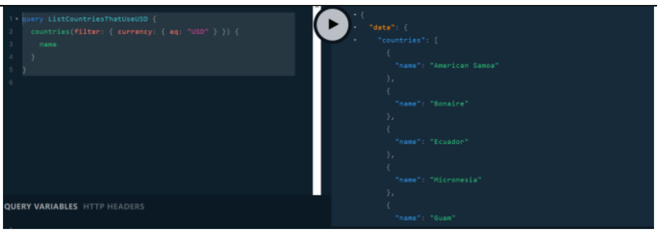
- The codes used to key the objects in the original data are available as a code property on each item returned from the API.
- The Country.continent and Country.languages are now objects and arrays of objects, respectively.
- The Country.currency and Country.phone fields sometimes return a comma-separated list of values. For this reason, this API also exposes currencies and phones fields that are arrays of all currencies and phone codes for a country.
- Each Country has an array of states populated by their states/provinces, if any.
- Each Country also has an awsRegion field that shows its nearest AWS region, powered by country-to-aws-region.

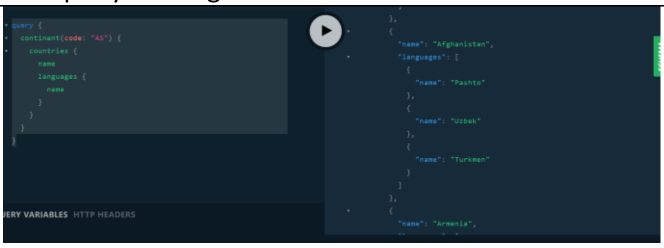
Summary of queries I tried & Reflection:

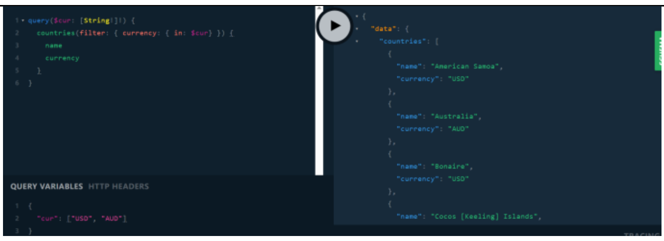
Aim of the query	**Query**	**Level**	**Result & Reflection**

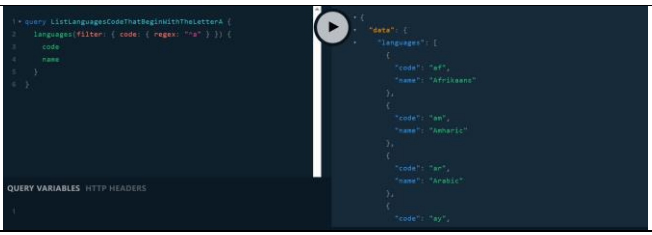
Get all continents' name.	query getAllContinents{
	  continents {
	    name
	  }
	}
		Easy	
			PRETTIFY HISTORY https://countries.trevorblades.com/ COPY CURL SHARE PLAYGROUND
			`query getALLContinents{`
			`  continents {`
			`    name`
			`  }`
			`}`
			`{`
			`  "data": {`
			`    "continents": [`
			`      {`
			`        "name": "Africa"`
			`      },`
			`      {`
			`        "name": "Antarctica"`
			`      },`
			`      {`
			`        "name": "Asia"`
			`      },`
			`      {`
			`        "name": "Europe"`
			`      },`
			`      {`
			`        "name": "North America"`
			QUERY VARIABLES HTTP HEADERS
			
			This is a straightforward query that fetches all countries. The API provides a dedicated "continents" field, making it simple to retrieve the required information.

Get the name, capital, currency, language name and continent name info of a country with its 2-alphabet code, e.g., “AU”.	query GetCountry {
	  country(code: "AU") {
	    name
	    capital
	    currency
	    languages {
	      name
	    }
	    continent {
	      name
	    }
	  }
	}
		Easy	
			 PRETTIFY
			HISTORY
			https://countries.trevorblades.com/
			COPY CURL
			SHARE PLAYGROUND
			QUERY VARIABLES
			HTTP HEADERS
			query GetCountry {
			  country(code: "AU") {
			    name
			    capital
			    currency
			    languages {
			      name
			    }
			    continent {
			      name
			    }
			  }
			}
			{
			  "data": {
			    "country": {
			      "name": "Australia",
			      "capital": "Canberra",
			      "currency": "AUD",
			      "languages": [
			        {
			          "name": "English"
			        }
			      ],
			      "continent": {
			        "name": "Oceania"
			      }
			    }
			  }
			}
			
			The API provides a "country" field that accepts the "code" argument. This simple query is efficient and returns a response specific to the query, thus leaving no room for over-fetching or under-fetching.

List countries using a certain currency, e.g. “USD”.	query ListCountriesThatUseUSD {
	  countries(filter: { currency: { eq: "USD" } }) {
	    name
	  }
	}
		Easy	
			
			query ListCountriesThatUseUSD {
			  countries(filter: { currency: { eq: "USD" } }) {
			    name
			  }
			}
			{
			  "data": {
			    "countries": [
			      {
			        "name": "American Samoa"
			      },
			      {
			        "name": "Bonaire"
			      },
			      {
			        "name": "Ecuador"
			      },
			      {
			        "name": "Micronesia"
			      },
			      {
			        "name": "Guam"
			
			Filtering countries based on currency requires passing a filter object with the desired condition ("eq" for equal). The query is straightforward.

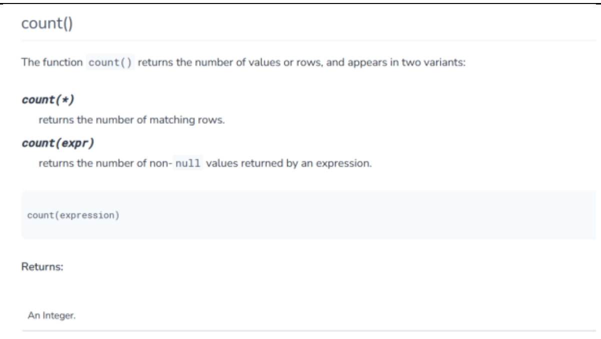
List countries within a continent, e.g. “Asia”, with all the languages each country use.	query {
	  continent(code: "AS") {
	    countries {
	      name
	      languages {
	        name
	      }
	    }
	  }
	}
		Intermediate	
			
			query {
			  continent(code: "AS") {
			    countries {
			      name
			      languages {
			        name
			      }
			    }
			  }
			}
			},
			{
			  "name": "Afghanistan",
			  "languages": [
			    {
			      "name": "Pashto"
			    },
			    {
			      "name": "Uzbek"
			    },
			    {
			      "name": "Turkmen"
			    }
			  ]
			},
			{
			  "name": "Armenia",
			
			Retrieving languages of countries with a specific continent involves querying the “language” field nested within the "country" field. This requires understanding the schema structure and how to access nested fields.

Get all countries with the currency as a variable.	query($cur: [String!]!) {
	  countries(filter: { currency: { in: $cur} }) {
	    name
	    currency
	  }
	}
	
	{
	  "cur": ["USD", "AUD"]
	}
		Intermediate	
			
			query($cur: [String!]!) {
			  countries(filter: { currency: { in: $cur} }) {
			    name
			    currency
			  }
			}
			{
			  "cur": ["USD", "AUD"]
			}
			{
			  "data": {
			    "countries": [
			      {
			        "name": "American Samoa",
			        "currency": "USD"
			      },
			      {
			        "name": "Australia",
			        "currency": "AUD"
			      },
			      {
			        "name": "Bonaire",
			        "currency": "USD"
			      },
			      {
			        "name": "Cocos [Keeling] Islands",
			
			Querying countries based on the currency as a variable requires using a GraphQL variable ("$cur" in this case). Handling variables adds complexity compared to static queries, but it provides flexibility by allowing dynamic inputs.

Get all languages with code begin with the letter “a”.	query ListLanguages {
	  languages(filter: { code: { regex: "^a" } }) {
	    code
	    name
	  }}
		Intermediate	
			
			query ListLanguagesCodeThatBeginWithTheLetterA {
			  languages(filter: { code: { regex: "^a" } }) {
			    code
			    name
			  }
			}
			{
			  "data": {
			    "languages": [
			      {
			        "code": "af",
			        "name": "Afrikaans"
			      },
			      {
			        "code": "am",
			        "name": "Amharic"
			      },
			      {
			        "code": "ar",
			        "name": "Arabic"
			      },
			      {
			        "code": "ay",

List the top 3 countries in Asia based on the number of languages spoken.	query {
	  continent(code: "AS") {
	    countries {
	      name
	      languages {
	        name
	      }
	    }
	  }
	}
		Hard	Since the existing API does not provide sorting and pagination options, we'll need to retrieve all the countries in Asia and perform the sorting and limiting on the client-side. If there is sorting and pagination option provided by the API, we could do query similar to below:
			query {
			  continent(code: "AS") {
			    countries(sort: { field: LANGUAGES, order: DESC }, first: 3) {
			      name
			    }
			  }
			}
			
			In comparison to Cypher, Cypher provides powerful aggregation functions like COUNT(), SUM(), MIN(), MAX(), which could be very useful in this case. We could use COUNT() to count the number of languages each country speaks, and use MAX() to find country with the most languages. We could use order by to sort the result.
			
			GraphQL with Prisma: Prisma Client enables more filtering, sorting, and pagination capabilities.
			
			Possible query with Prisma client and GraphQL:
			query {
			  countries(first: 3, orderBy: { languages: desc }) {
			    name
			      }
			}
			
			See reference:
			https://www.prisma.io/graphql
			[GraphQL Filtering, Pagination & Sorting Tutorial with JavaScript (howtographql.com)]

Get the total number of continents.	query totalNum {
	  continents {
	    name
	  }
	}
		Hard	In this case we may need to fetch all continents then count them on the client-side. After receiving the response, we can count the number of continents by accessing the length of the returned continent data.
			
			In comparison to Cypher, such query could be easily achieved in Cypher via aggregation function COUNT().
			Possible query:
			MATCH (c:Continent)
			RETURN count(c) AS totalNum

			 count() The function `count()` returns the number of values or rows, and appears in two variants: ***count(*)*** returns the number of matching rows. ***count(expr)*** returns the number of non- `null` values returned by an expression. `count(expression)` Returns: An Integer.

Retrieve a list of continents along with the total unique languages for each continent.	query {
	  continents {
	    name
	    countries {
	      languages {
	        name
	      }
	    }
	  }
	}
		Hard	The basic query provides a long nested list of languages used in each country in each continent. Achieving this in GraphQL may require additional client-side processing or making multiple requests to calculate the aggregates.
			
			In Cypher, we can aggregate and group data based on continents, then calculate the total languages for each continent easily.
			
			Possible query in Cypher:
			
			MATCH (c:Continent)-[:HAS_COUNTRY]->(country)-[:SPEAKS]->(language)
			RETURN c.name AS continent, COUNT(DISTINCT language) AS totalUniqueLanguages

2. Practice querying with Star Wars GraphQL API

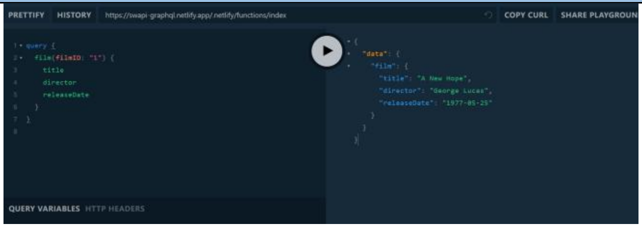
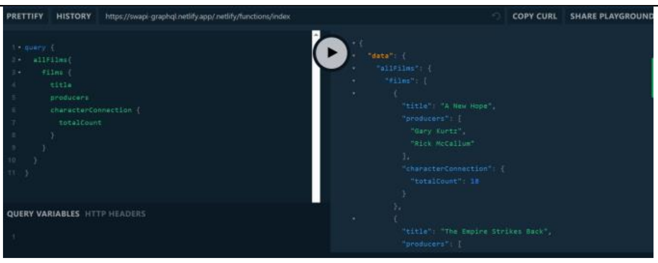
Endpoint url: https://swapi-graphql.netlify.app/.netlify/functions/index

API used: https://studio.apollographql.com/public/star-wars-swapi/variant/current/home

Schema: https://studio.apollographql.com/public/star-wars-swapi/variant/current/schema/reference

Overview: This GraphQL API retrieves all the Star Wars data: Planets, Spaceships, Vehicles, People, Films and Species from all seven Star Wars films.

Summary of queries I tried & Reflection:

Aim of the query	Query	Level	Result & Reflection
Get a specific film by its filmID.	`query { film(filmID: "1") { title director releaseDate } }`	Easy	 This is a straightforward query as the API provides a dedicated "film" field that accepts the "filmID" argument.
Get all films and their associated character connection count.	`query { allFilms{ films { title producers characterConnection { totalCount } } } }`	Easy	 The API offers a "films" field with nested "characterConnection" field, allowing for easy retrieval of the desired information.

PRETTIFY HISTORY https://swapi-graphql.netlify.app/.netlify/functions/index COPY CURL SHARE PLAYGROUN

```
query {
  film(filmID: "1") {
    title
    director
    releaseDate
  }
}
```

```
{
  "data": {
    "film": {
      "title": "A New Hope",
      "director": "George Lucas",
      "releaseDate": "1977-05-25"
    }
  }
}
```

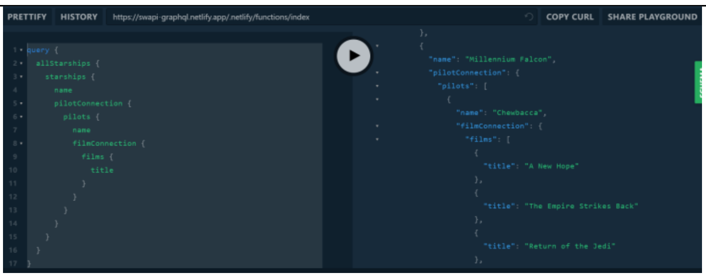
QUERY VARIABLES HTTP HEADERS

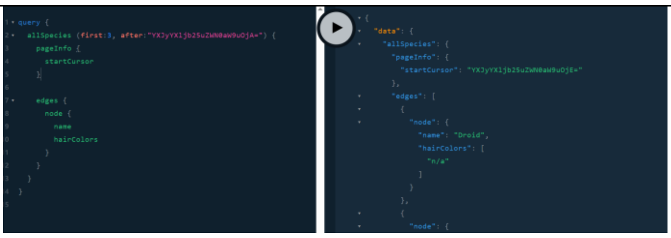
PRETTIFY HISTORY https://swapi-graphql.netlify.app/.netlify/functions/index COPY CURL SHARE PLAYGROUND

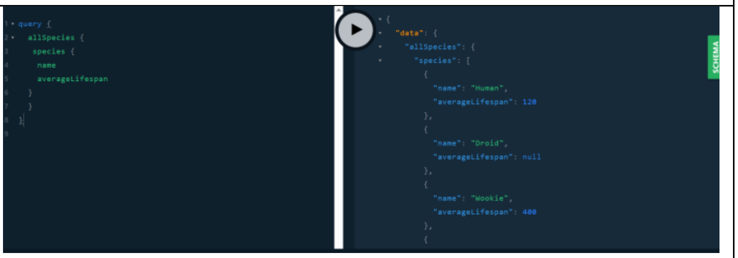
```
query {
  allFilms{
    films {
      title
      producers
      characterConnection {
        totalCount
      }
    }
  }
}
```

```
{
  "data": {
    "allFilms": {
      "films": [
        {
          "title": "A New Hope",
          "producers": [
            "Gary Kurtz",
            "Rick McCallum"
          ],
          "characterConnection": {
            "totalCount": 18
          }
        },
        {
          "title": "The Empire Strikes Back",
          "producers": [
```

QUERY VARIABLES HTTP HEADERS

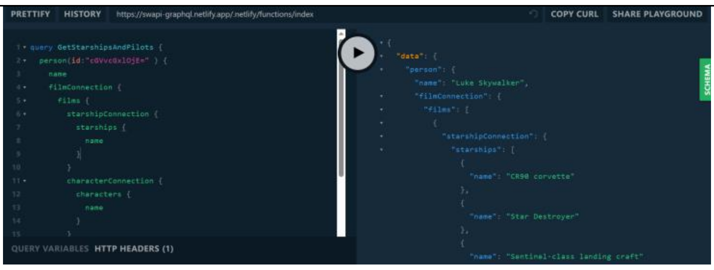
Get all starships with their pilots and the name of films these pilots appeared.	`query {`
	`  allStarships {`
	`    starships {`
	`      name`
	`      pilotConnection {`
	`        pilots {`
	`          name`
	`          filmConnection {`
	`            films {`
	`              title`
	`            }`
	`          }`
	`        }`
	`      }`
	`    }`
	`  }`
	`}`
		Intermediate	
			PRETTIFY HISTORY https://swapi-graphql.netlify.app/.netlify/functions/index COPY CURL SHARE PLAYGROUND
			`query {`
			`  allStarships {`
			`    starships {`
			`      name`
			`      pilotConnection {`
			`        pilots {`
			`          name`
			`          filmConnection {`
			`            films {`
			`              title`
			`            }`
			`          }`
			`        }`
			`      }`
			`    }`
			`  }`
			`}`
			`},`
			`{`
			`  "name": "Millennium Falcon",`
			`  "pilotConnection": {`
			`    "pilots": [`
			`      {`
			`        "name": "Chewbacca",`
			`        "filmConnection": {`
			`          "films": [`
			`            {`
			`              "title": "A New Hope"`
			`            },`
			`            {`
			`              "title": "The Empire Strikes Back"`
			`            },`
			`            {`
			`              "title": "Return of the Jedi"`
			`            },`
			Querying all starships and their associated pilots is relatively straightforward in GraphQL. However, retrieving deeply nested data like pilots' film connection for all starships require additional looping.

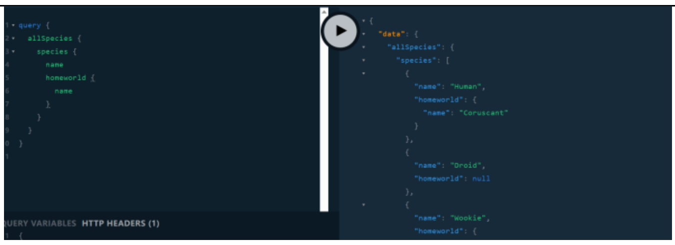
Get the hair colours of first 3 species after the start cursor.	`query {`
	`  allSpecies (first:3, after:"…") {`
	`    pageInfo {`
	`      startCursor`
	`    }`
	
	`    edges {`
	`      node {`
	`        name`
	`        hairColors`
	`      }`
	`    }`
	`  }`
	`}`
		Intermediate	
			`query {`
			`  allSpecies (first:3, after:"YXJyYXljb25uZWN0aW9uOjA=") {`
			`    pageInfo {`
			`      startCursor`
			`    }`
			
			`    edges {`
			`      node {`
			`        name`
			`        hairColors`
			`      }`
			`    }`
			`  }`
			`}`
			`{`
			`  "data": {`
			`    "allSpecies": {`
			`      "pageInfo": {`
			`        "startCursor": "YXJyYXljb25uZWN0aW9uOjE="`
			`      },`
			`      "edges": [`
			`        {`
			`          "node": {`
			`            "name": "Droid",`
			`            "hairColors": [`
			`              "n/a"`
			`            ]`
			`          }`
			`        },`
			`        {`
			`          "node": {`
			This is a basic query, but with pagination applied.
			
			The query requests the first 3 species after a specific start cursor. It fetches the name and hairColors fields for each species. The pageInfo field is included to retrieve the start cursor, which can be used for pagination purposes.

Fetch all species with their average lifespan and sort them in descending order of average lifespan.	`query {`
	`  allSpecies {`
	`   species {`
	`    name`
	`    averageLifespan`
	`  }`
	`  }`
	`}`
		Hard	
			`query {`
			`  allSpecies {`
			`   species {`
			`    name`
			`    averageLifespan`
			`  }`
			`  }`
			`}`
			`{`
			`  "data": {`
			`    "allSpecies": {`
			`      "species": [`
			`        {`
			`          "name": "Human",`
			`          "averageLifespan": 120`
			`        },`
			`        {`
			`          "name": "Droid",`
			`          "averageLifespan": null`
			`        },`
			`        {`
			`          "name": "Wookie",`
			`          "averageLifespan": 400`
			`        },`
			`        {`
			Observing the query result, we noticed that lifespan is an integer field, but is nullable. We could use Dgraph with GraphQL and perform sorting. We can leverage the order argument provided by Dgraph's GraphQL API. The order argument allows us to specify the field and direction for sorting. Here's an example query:
			`query {`
			`  querySpecies(order: {asc: averageLifespan}) {`
			`    species {`
			`      name`
			`      averageLifespan`
			`    }`

			} } Sort Query Results | Graphqlbasic | Dgraph Tour Note Dgraph's GraphQL API can handle null cases when sorting. By default, when you sort a field that contains null values, Dgraph returns null values at the end of the results, irrespective of their sort. Sorting - Query language (dgraph.io)

Get the characters who have appeared in all of the Star Wars films.	query {
	  allFilms {
	    films {
	      characterConnection {
	        edges {
	          node {
	            name
	          }
	        }
	      }
	    }
	  }
	}
		Hard	Finding the characters who have appeared in all the Star Wars films requires comparing the appearances of characters across multiple films. Achieving this in GraphQL is more challenging without built-in counting mechanism.
			
			Thus, after getting the response to the initial query, we need to iterate over the films and their associated characters and count the number of appearances for each character. By comparing this count with the total number of films, we can determine the characters who have appeared in all films.

Retrieve starships and characters that have appeared in the same film as a specific character.	query GetStarshipsAndPilots($pid: ID!) {
	  person(id: $pid) {
	    name
	    filmConnection {
	      films {
	        starshipConnection {
	          starships {
	            name
	          }
	        }
	        characterConnection {
	          characters {
	            name
	          }
	        }
	      }
	    }
	  }
	}
	
	{
	  "pid":"cGVvcGxlOjE="
	}
		Hard	
			PRETTIFY HISTORY https://swapi-graphql.netlify.app/.netlify/functions/index COPY CURL SHARE PLAYGROUND
			query GetStarshipsAndPilots {
			  person(id:"cGVvcGxlOjE=") {
			    name
			    filmConnection {
			      films {
			        starshipConnection {
			          starships {
			            name
			          }
			        }
			        characterConnection {
			          characters {
			            name
			          }
			        }
			QUERY VARIABLES HTTP HEADERS (1)
			{
			  "data": {
			    "person": {
			      "name": "Luke Skywalker",
			      "filmConnection": {
			        "films": [
			          {
			            "starshipConnection": {
			              "starships": [
			                {
			                  "name": "CR90 corvette"
			                },
			                {
			                  "name": "Star Destroyer"
			                },
			                {
			                  "name": "Sentinel-class landing craft"
			SCHEMA
			
			In this query, first traverse the filmConnection field to get the films associated with the character.
			
			Within each film, we further traverse the starshipConnection field to retrieve the starships, and the characterConnection field to retrieve the characters who appeared in the same film.
			
			After the complex traversal, we need to find 2 lists of names of starships and characters, exclude duplicates and find unique values.

Retrieve the species that share the same homeworld planet, sort by number of species in a planet in descending order.	`query {`
	`  allSpecies {`
	`    species {`
	`      name`
	`      homeworld {`
	`        name`
	`        }`
	`    }`
	`  }`
	`}`
		Hard	 `query {` `allSpecies {` `species {` `name` `homeworld {` `name` `}` `}` `}` `}` — QUERY VARIABLES HTTP HEADERS (1) — `{` `"data": {` `"allSpecies": {` `"species": [` `{` `"name": "Human",` `"homeworld": {` `"name": "Coruscant"` `}` `},` `{` `"name": "Droid",` `"homeworld": null` `},` `{` `"name": "Wookie",` `"homeworld": {`
			Querying films and species based on the shared homeworld planet involves complex relationship traversals and filtering based on shared properties.
			
			For this scenario, we use a basic query to retrieve all species along with their respective homeworld planet names. Then we need to perform additional processing to get the lists of species in each planet first, then count length of each list, and eventually sort the overall list.