Discover GraphQL with Python, Graphene and Odoo

FOSDEM 2019-02-03

Stéphane Bidoul <stephane.bidoul@acsone.eu>

Version 1.0.4

A short story

- Why this talk…

/me in a nutshell

- @sbidoul
- CTO of acsone (https://acsone.eu)
- Elected Board Member of oca (https://odoo-community.org)
- Python since 1996 (1.4)
- FLOSS, because…
- Have used a lot of RPC mechanisms

acsone oca

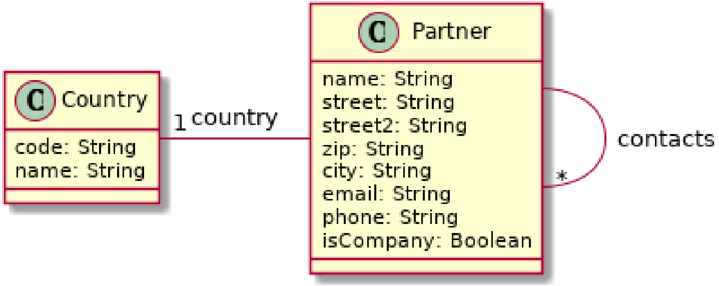
Content

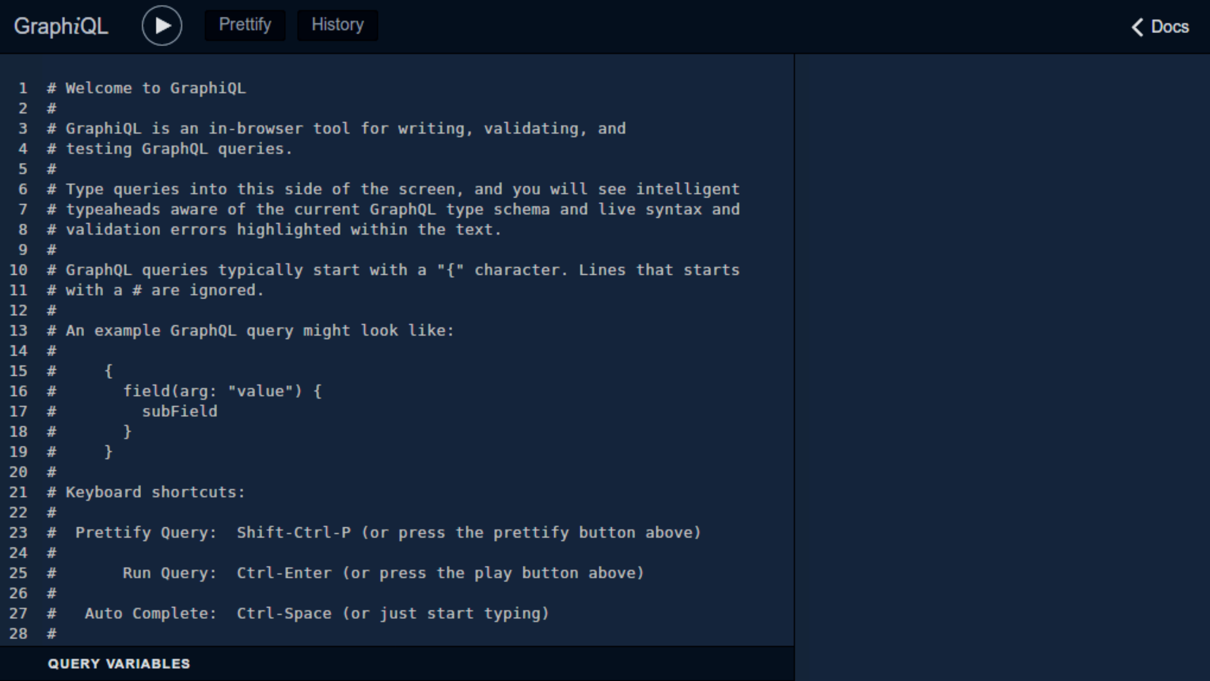
What is GraphQL?

- Yet another Remote Procedure Call protocol?
- Open Sourced by Facebook in 2015
- Basic characteristics
    - Requests: GraphQL data query language
    - Responses: json
    - Schema: GraphQL schema language
    - Transport: usually HTTPS (GET, POST)
    - Variety of server side libs, no need for client side lib

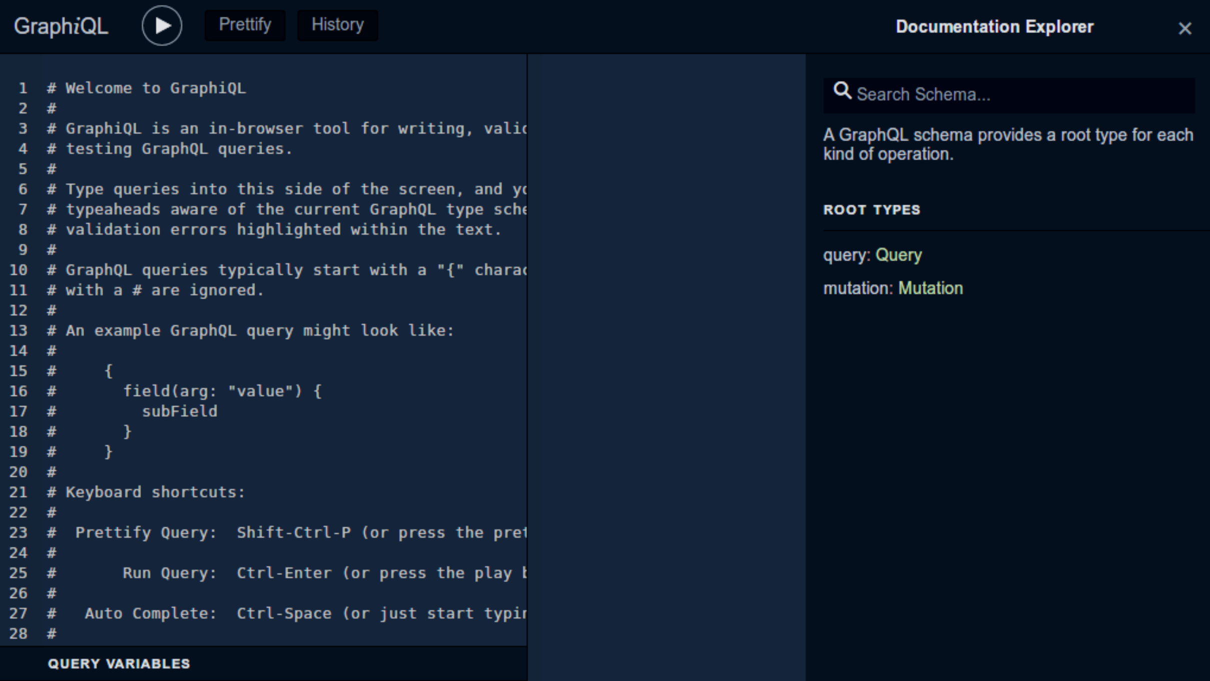
Demo

GraphQL Schema for Odoo Partners and Contacts.





```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#
```



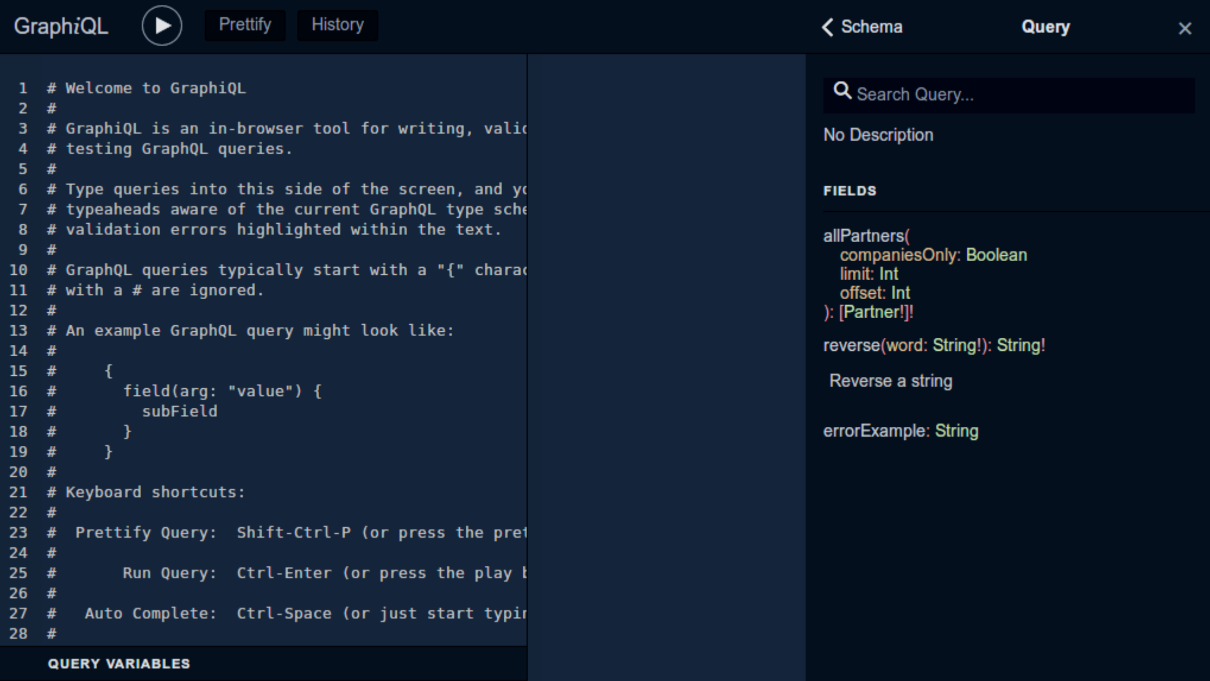
```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, valic
# testing GraphQL queries.
#
# Type queries into this side of the screen, and yc
# typeaheads aware of the current GraphQL type sche
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" charac
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the pret
#
#       Run Query:  Ctrl-Enter (or press the play b
#
#   Auto Complete:  Ctrl-Space (or just start typir
#
```

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation



```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, valic
# testing GraphQL queries.
#
# Type queries into this side of the screen, and yc
# typeaheads aware of the current GraphQL type sche
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" charac
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the pret
#
#       Run Query:  Ctrl-Enter (or press the play b
#
#   Auto Complete:  Ctrl-Space (or just start typir
#
```

QUERY VARIABLES

No Description

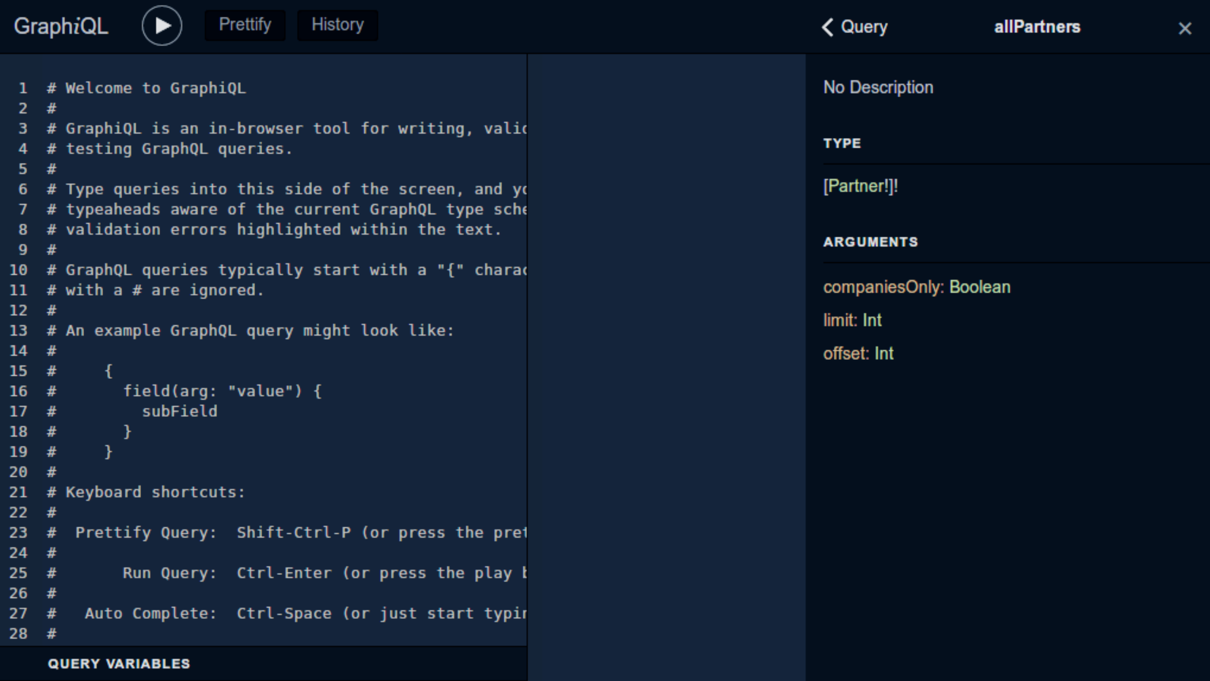
FIELDS

```
allPartners(
  companiesOnly: Boolean
  limit: Int
  offset: Int
): [Partner!]!

reverse(word: String!): String!
```

Reverse a string

```
errorExample: String
```



```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, valic
# testing GraphQL queries.
#
# Type queries into this side of the screen, and yc
# typeaheads aware of the current GraphQL type sche
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" charac
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the pret
#
#       Run Query:  Ctrl-Enter (or press the play b
#
#   Auto Complete:  Ctrl-Space (or just start typir
#
```

QUERY VARIABLES

No Description

TYPE

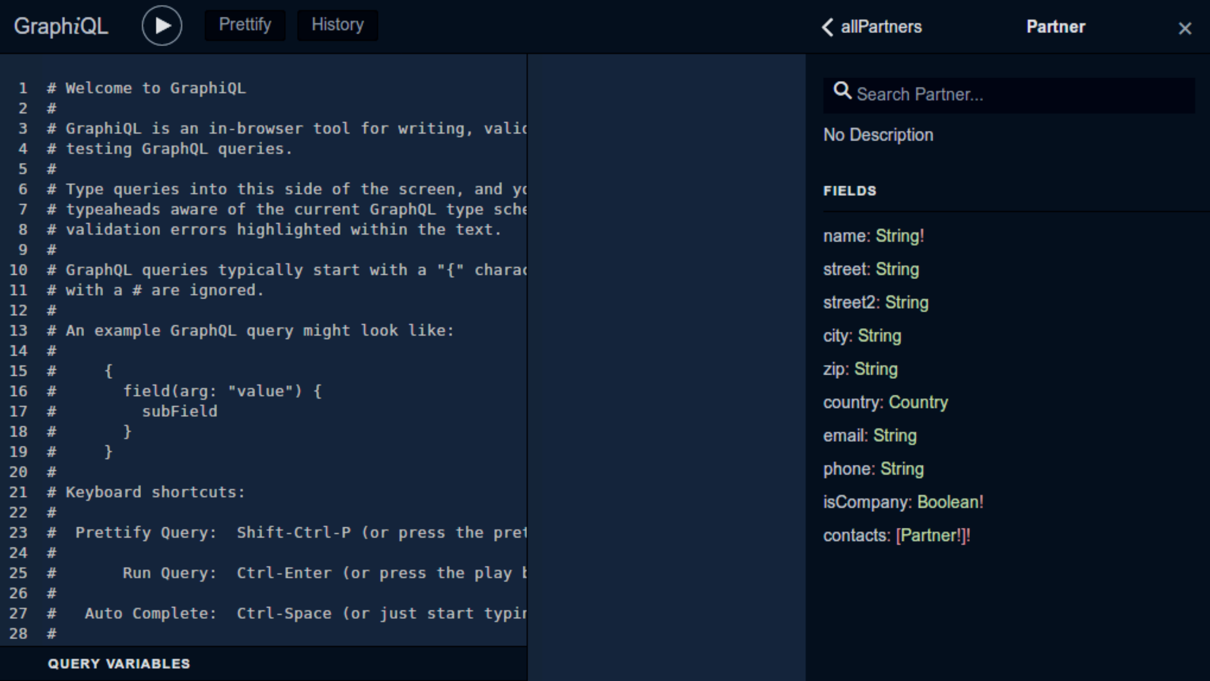
[Partner!]!

ARGUMENTS

companiesOnly: Boolean

limit: Int

offset: Int



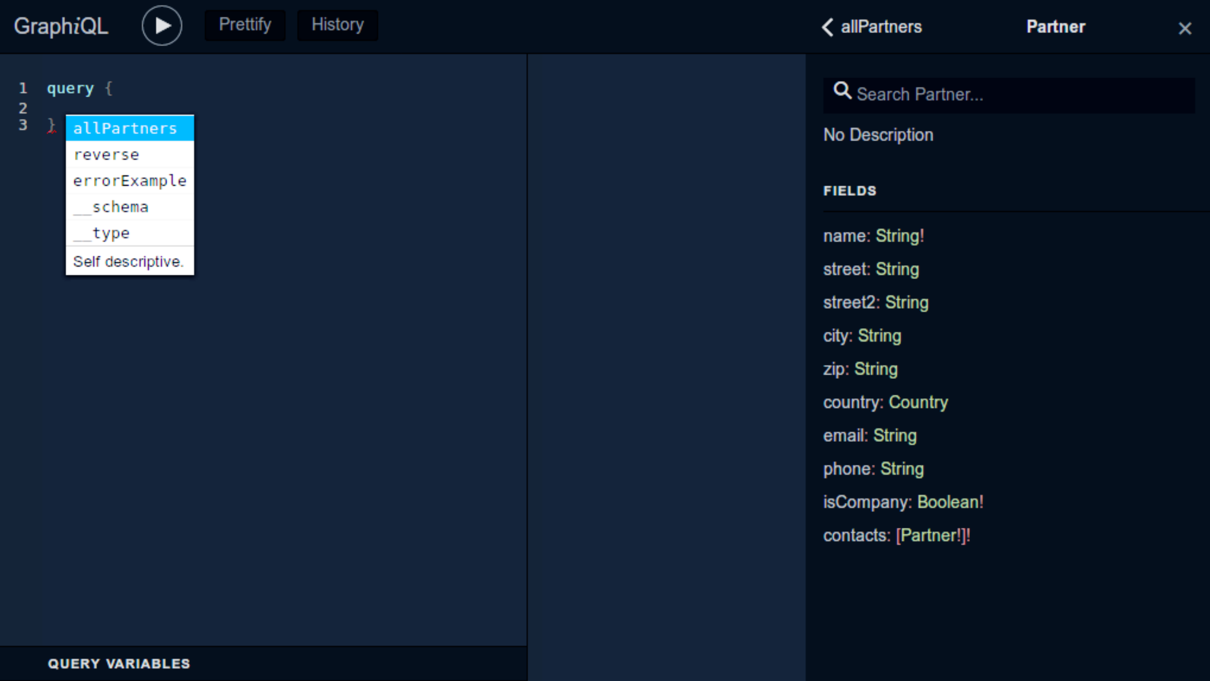
```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, valic
# testing GraphQL queries.
#
# Type queries into this side of the screen, and yc
# typeaheads aware of the current GraphQL type sche
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" charac
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the pret
#
#       Run Query:  Ctrl-Enter (or press the play b
#
#   Auto Complete:  Ctrl-Space (or just start typir
#
```

Search Partner...

No Description

FIELDS

name: String!
street: String
street2: String
city: String
zip: String
country: Country
email: String
phone: String
isCompany: Boolean!
contacts: [Partner!]!



```
query {

}
```

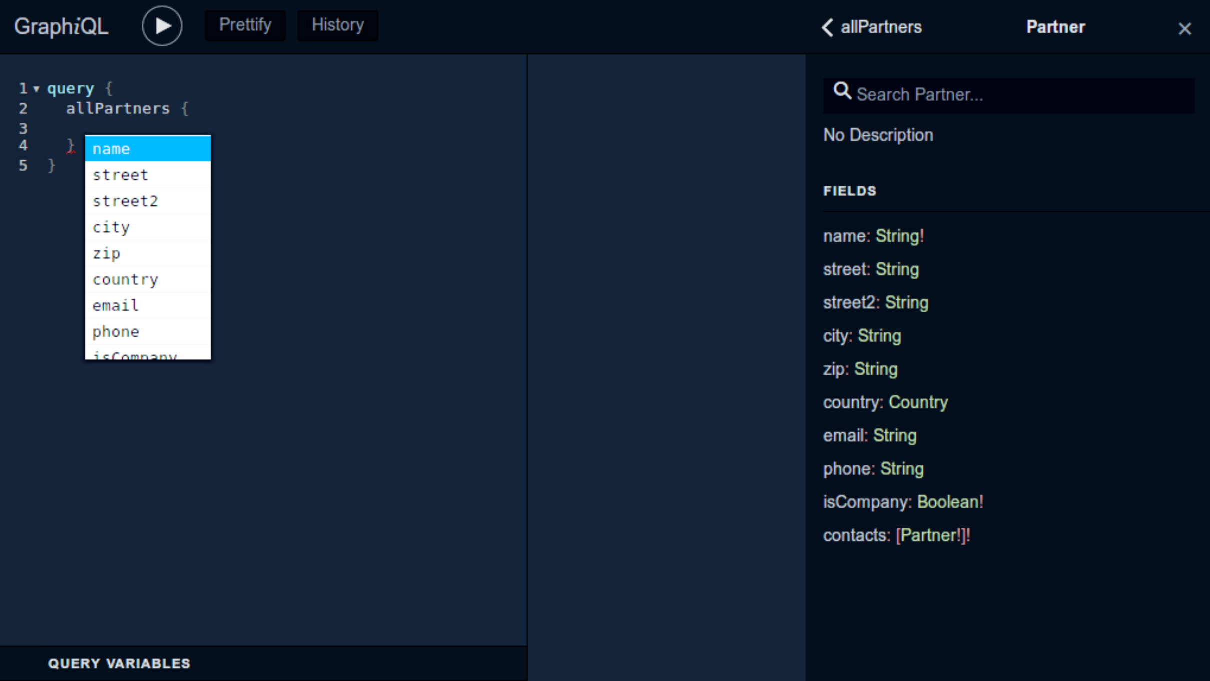
allPartners
reverse
errorExample
__schema
__type
Self descriptive.

No Description

FIELDS

name: String!
street: String
street2: String
city: String
zip: String
country: Country
email: String
phone: String
isCompany: Boolean!
contacts: [Partner!]!

QUERY VARIABLES



```
query {
  allPartners {

  }
}
```

```
name
street
street2
city
zip
country
email
phone
isCompany
```

Search Partner...

No Description

FIELDS

name: String!

street: String

street2: String

city: String

zip: String

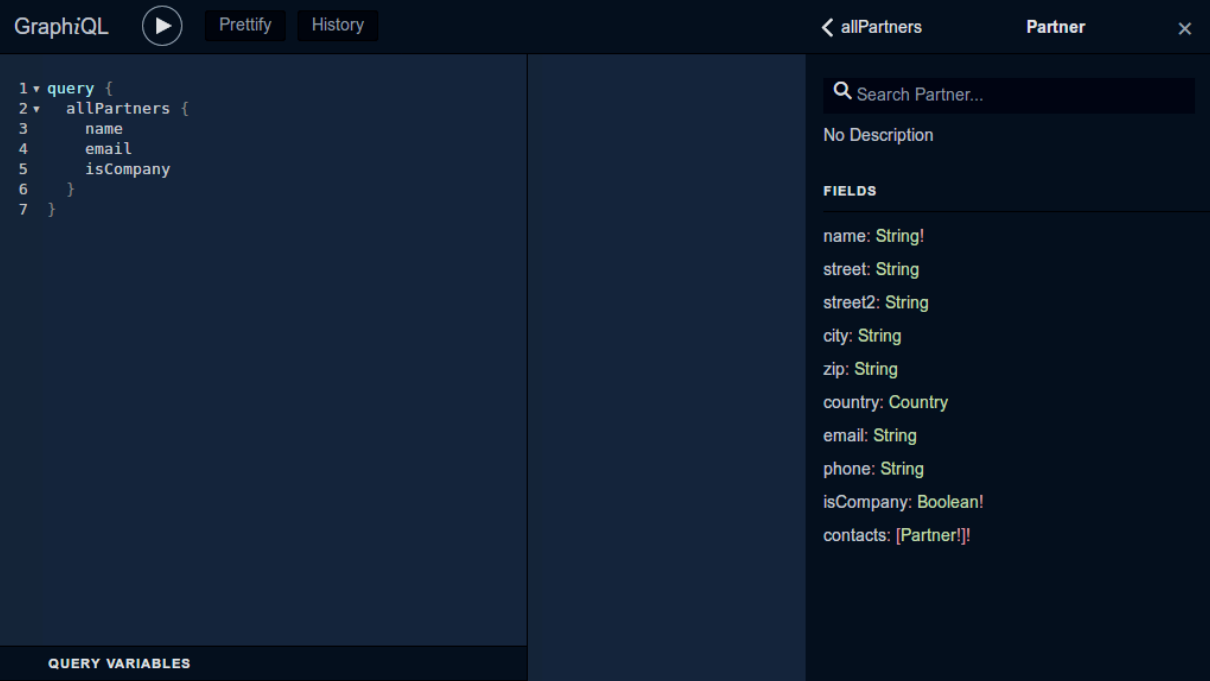
country: Country

email: String

phone: String

isCompany: Boolean!

contacts: [Partner!]!



```
query {
  allPartners {
    name
    email
    isCompany
  }
}
```

allPartners Partner

Search Partner...

No Description

FIELDS

name: String!

street: String

street2: String

city: String

zip: String

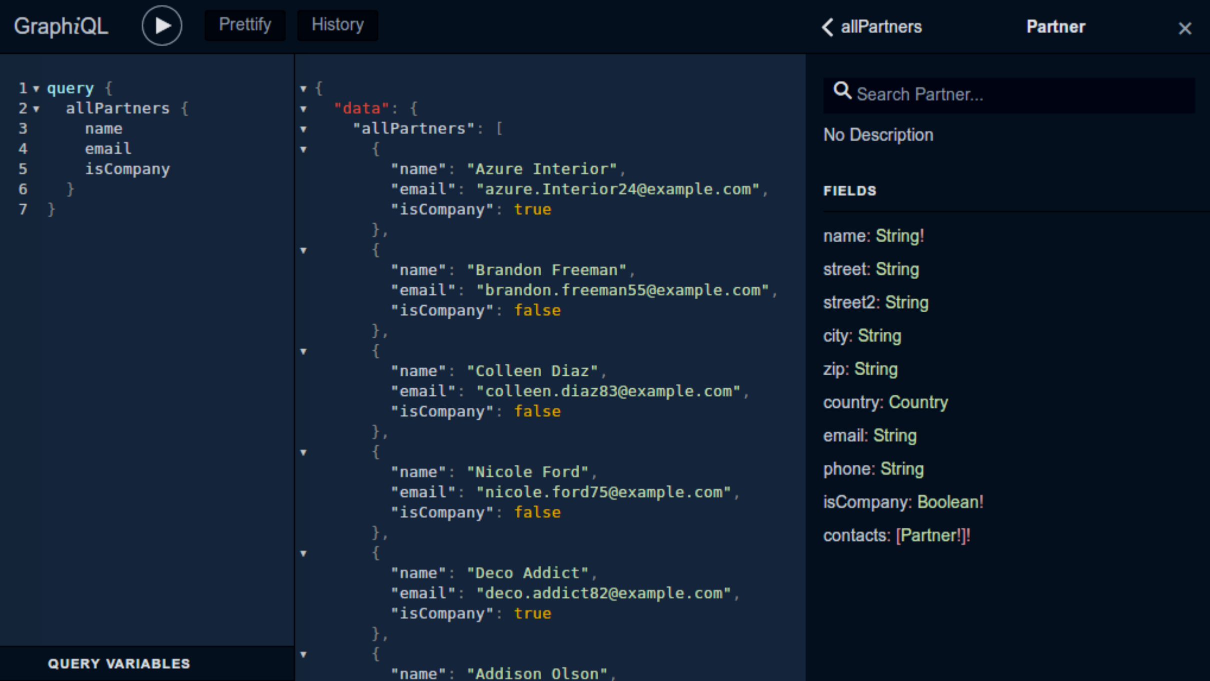
country: Country

email: String

phone: String

isCompany: Boolean!

contacts: [Partner!]!



```
query {
  allPartners {
    name
    email
    isCompany
  }
}
```

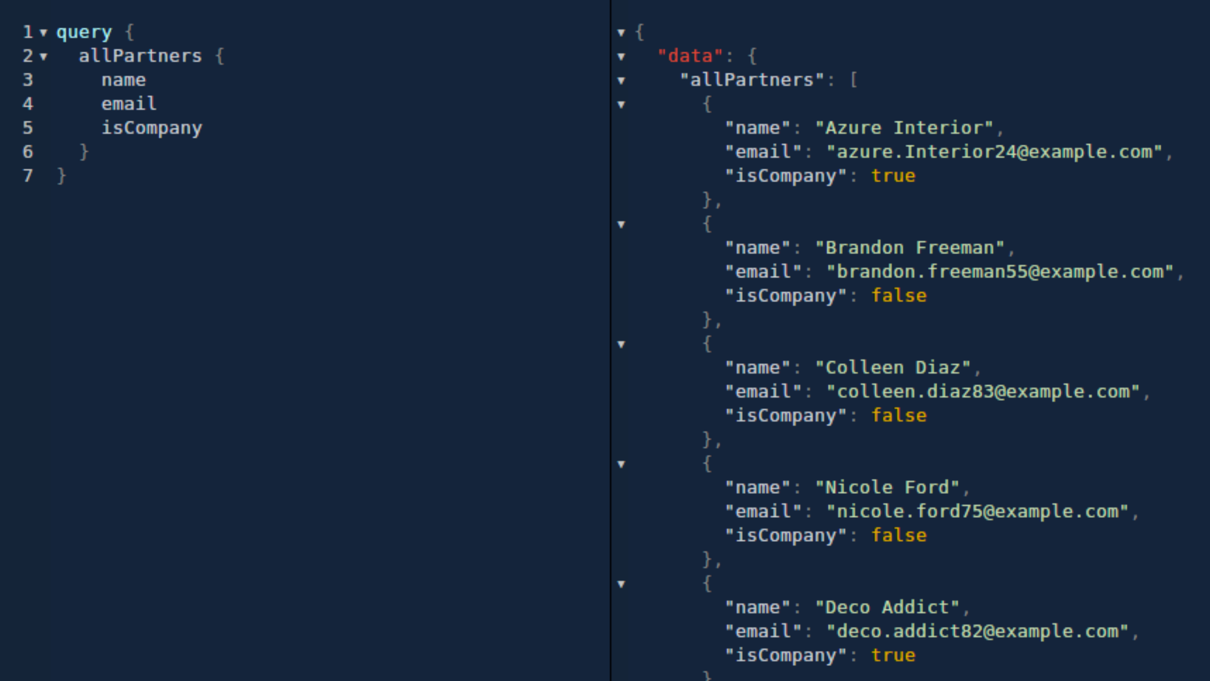
```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior",
        "email": "azure.Interior24@example.com",
        "isCompany": true
      },
      {
        "name": "Brandon Freeman",
        "email": "brandon.freeman55@example.com",
        "isCompany": false
      },
      {
        "name": "Colleen Diaz",
        "email": "colleen.diaz83@example.com",
        "isCompany": false
      },
      {
        "name": "Nicole Ford",
        "email": "nicole.ford75@example.com",
        "isCompany": false
      },
      {
        "name": "Deco Addict",
        "email": "deco.addict82@example.com",
        "isCompany": true
      },
      {
        "name": "Addison Olson",
```

Search Partner...

No Description

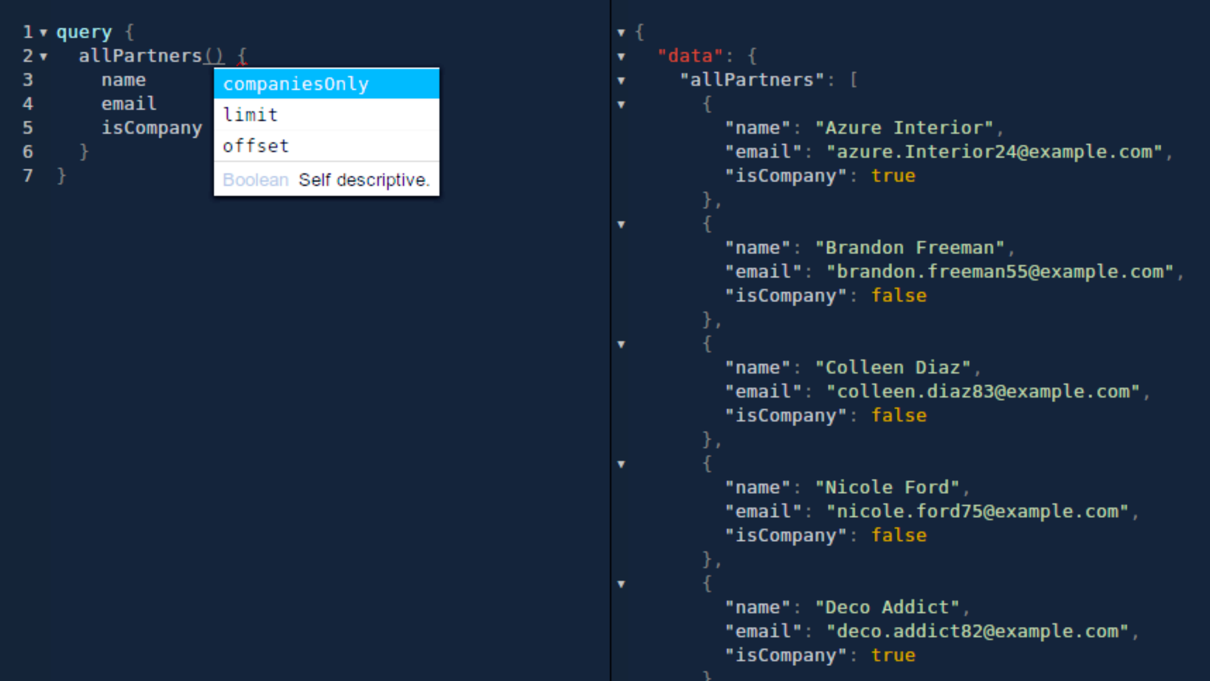
FIELDS

name: String!
street: String
street2: String
city: String
zip: String
country: Country
email: String
phone: String
isCompany: Boolean!
contacts: [Partner!]!



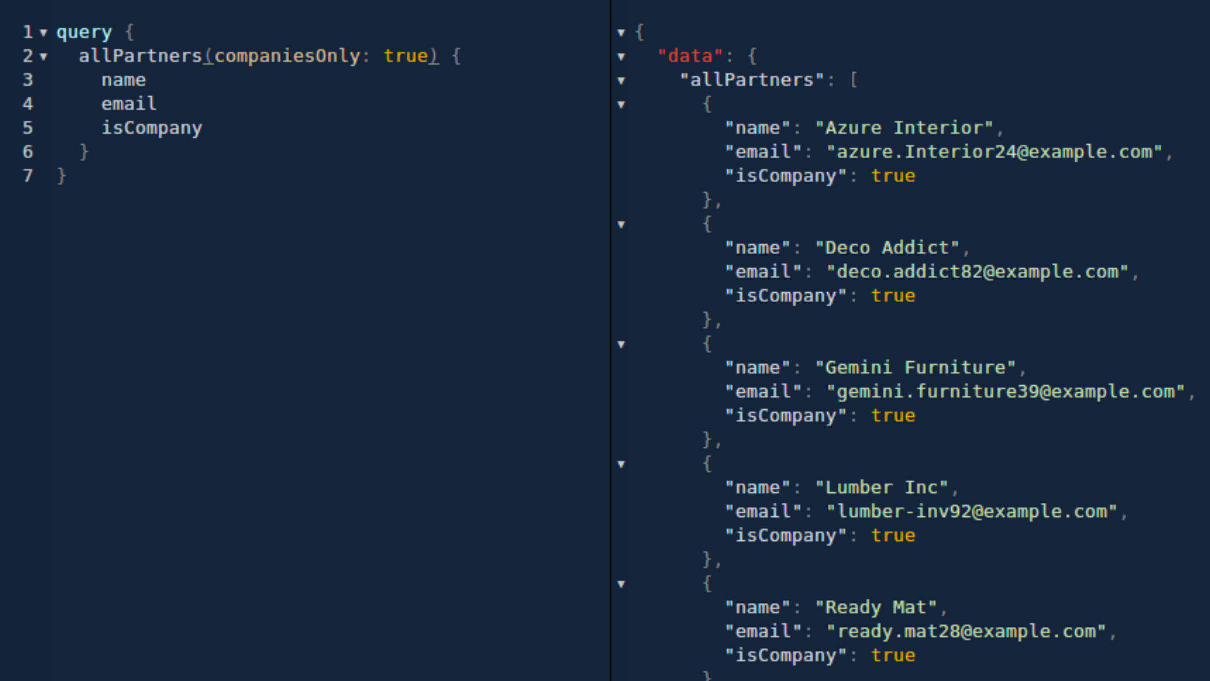
```
query {
  allPartners {
    name
    email
    isCompany
  }
}
```

```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior",
        "email": "azure.Interior24@example.com",
        "isCompany": true
      },
      {
        "name": "Brandon Freeman",
        "email": "brandon.freeman55@example.com",
        "isCompany": false
      },
      {
        "name": "Colleen Diaz",
        "email": "colleen.diaz83@example.com",
        "isCompany": false
      },
      {
        "name": "Nicole Ford",
        "email": "nicole.ford75@example.com",
        "isCompany": false
      },
      {
        "name": "Deco Addict",
        "email": "deco.addict82@example.com",
        "isCompany": true
```



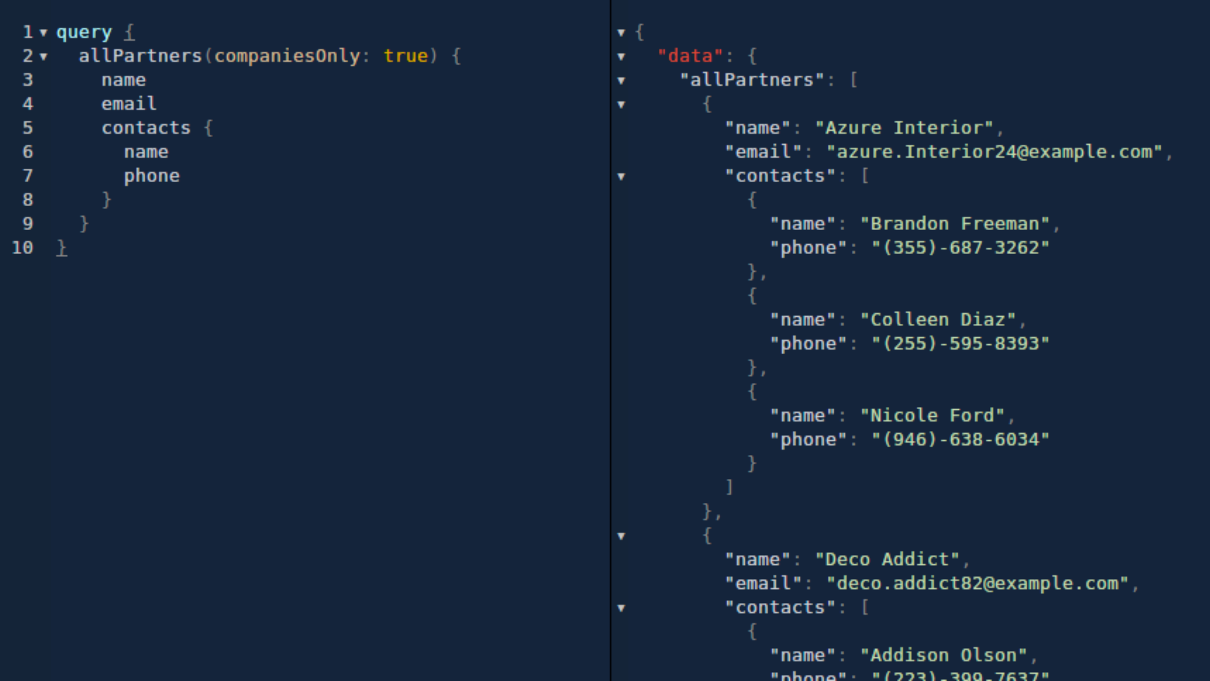
```
query {
  allPartners() {
    name
    email
    isCompany
  }
}
```

```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior",
        "email": "azure.Interior24@example.com",
        "isCompany": true
      },
      {
        "name": "Brandon Freeman",
        "email": "brandon.freeman55@example.com",
        "isCompany": false
      },
      {
        "name": "Colleen Diaz",
        "email": "colleen.diaz83@example.com",
        "isCompany": false
      },
      {
        "name": "Nicole Ford",
        "email": "nicole.ford75@example.com",
        "isCompany": false
      },
      {
        "name": "Deco Addict",
        "email": "deco.addict82@example.com",
        "isCompany": true
      }
```



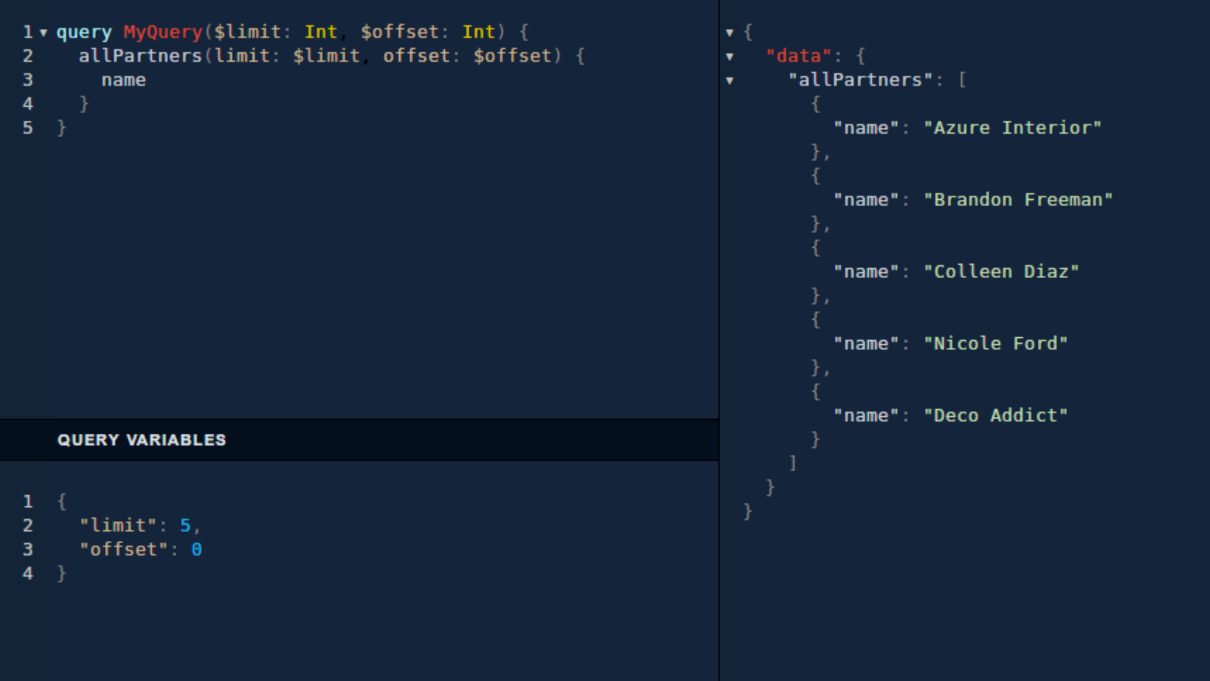
```
query {
  allPartners(companiesOnly: true) {
    name
    email
    isCompany
  }
}
```

```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior",
        "email": "azure.Interior24@example.com",
        "isCompany": true
      },
      {
        "name": "Deco Addict",
        "email": "deco.addict82@example.com",
        "isCompany": true
      },
      {
        "name": "Gemini Furniture",
        "email": "gemini.furniture39@example.com",
        "isCompany": true
      },
      {
        "name": "Lumber Inc",
        "email": "lumber-inv92@example.com",
        "isCompany": true
      },
      {
        "name": "Ready Mat",
        "email": "ready.mat28@example.com",
        "isCompany": true
      }
```



```
query {
  allPartners(companiesOnly: true) {
    name
    email
    contacts {
      name
      phone
    }
  }
}
```

```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior",
        "email": "azure.Interior24@example.com",
        "contacts": [
          {
            "name": "Brandon Freeman",
            "phone": "(355)-687-3262"
          },
          {
            "name": "Colleen Diaz",
            "phone": "(255)-595-8393"
          },
          {
            "name": "Nicole Ford",
            "phone": "(946)-638-6034"
          }
        ]
      },
      {
        "name": "Deco Addict",
        "email": "deco.addict82@example.com",
        "contacts": [
          {
            "name": "Addison Olson",
            "phone": "(223)-399-7637"
```

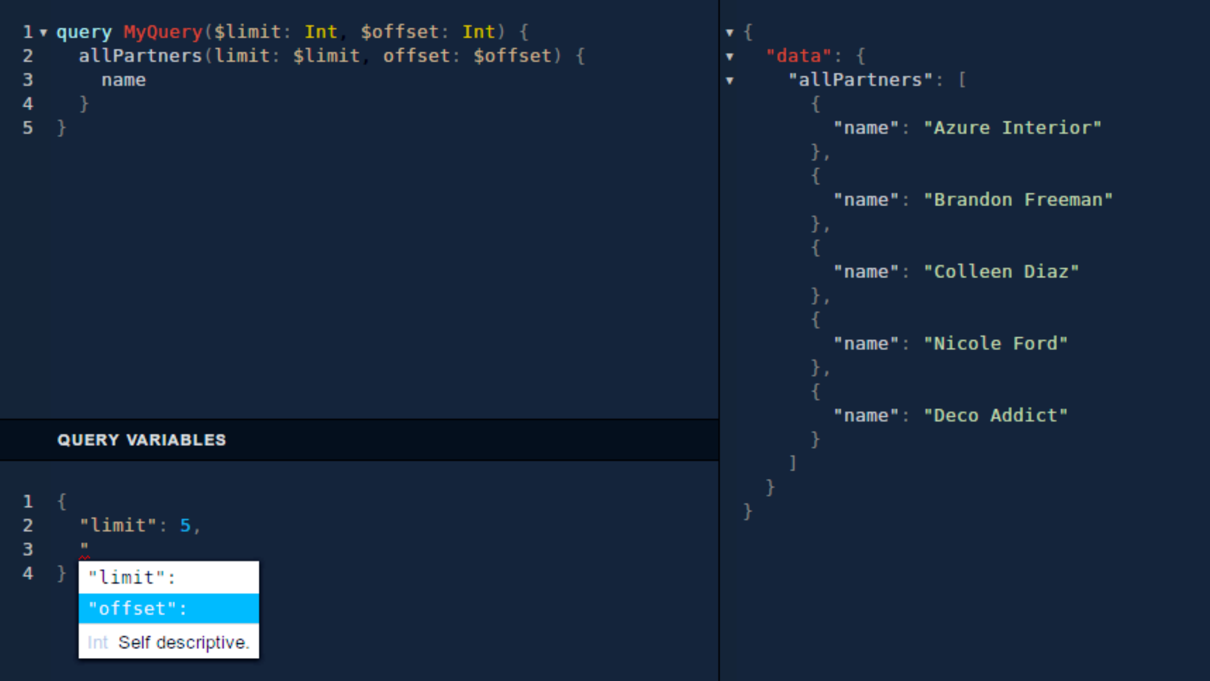


```graphql
query MyQuery($limit: Int, $offset: Int) {
  allPartners(limit: $limit, offset: $offset) {
    name
  }
}
```

QUERY VARIABLES

```json
{
  "limit": 5,
  "offset": 0
}
```

```json
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior"
      },
      {
        "name": "Brandon Freeman"
      },
      {
        "name": "Colleen Diaz"
      },
      {
        "name": "Nicole Ford"
      },
      {
        "name": "Deco Addict"
      }
    ]
  }
}
```

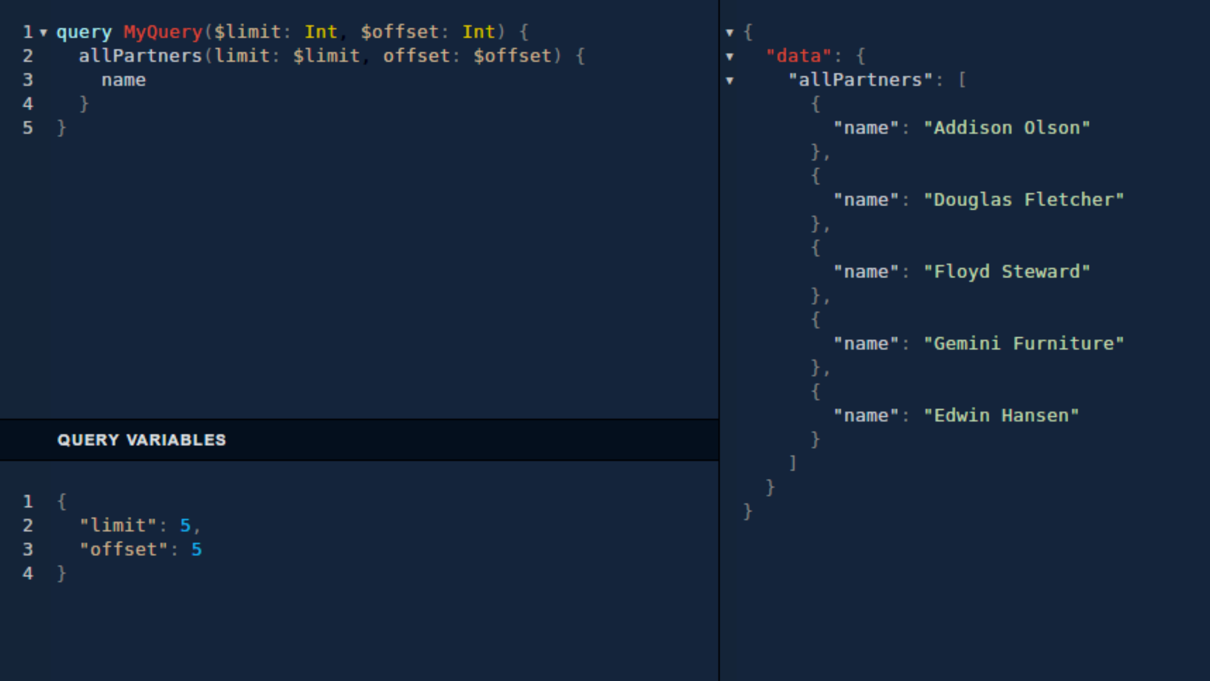


```
query MyQuery($limit: Int, $offset: Int) {
  allPartners(limit: $limit, offset: $offset) {
    name
  }
}
```

QUERY VARIABLES

```
{
  "limit": 5,
  "
}
```

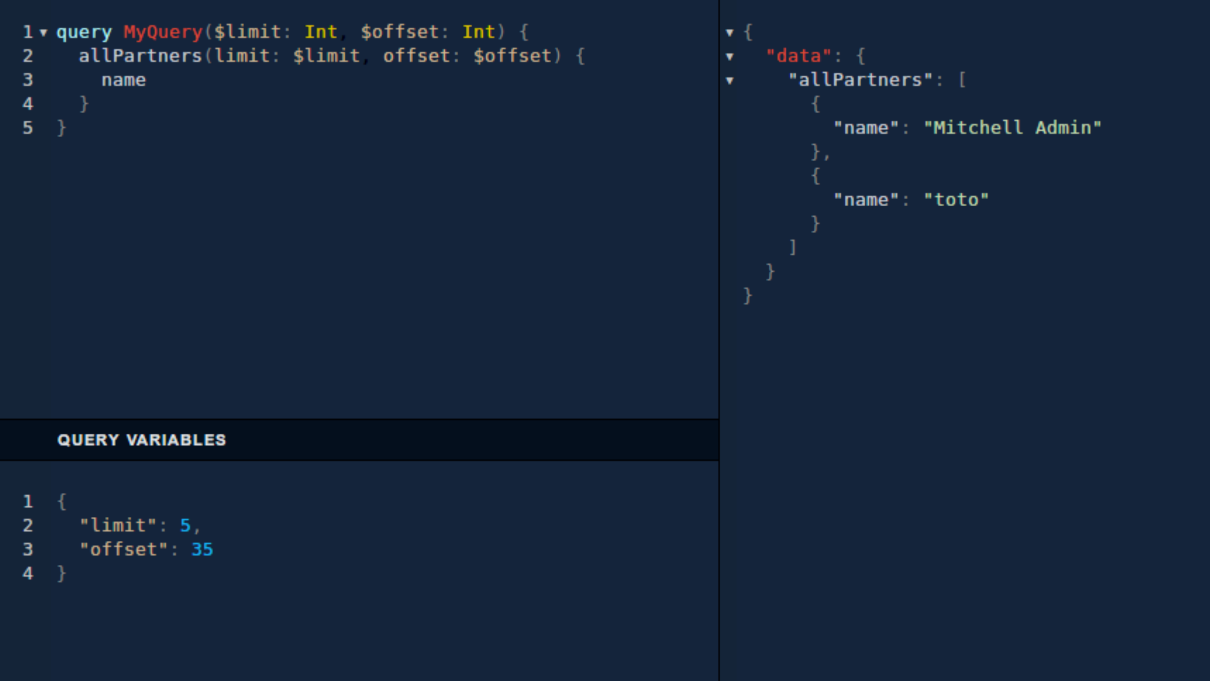
```
{
  "data": {
    "allPartners": [
      {
        "name": "Azure Interior"
      },
      {
        "name": "Brandon Freeman"
      },
      {
        "name": "Colleen Diaz"
      },
      {
        "name": "Nicole Ford"
      },
      {
        "name": "Deco Addict"
      }
    ]
  }
}
```



```graphql
query MyQuery($limit: Int, $offset: Int) {
  allPartners(limit: $limit, offset: $offset) {
    name
  }
}
```

```json
{
  "limit": 5,
  "offset": 5
}
```

```json
{
  "data": {
    "allPartners": [
      {
        "name": "Addison Olson"
      },
      {
        "name": "Douglas Fletcher"
      },
      {
        "name": "Floyd Steward"
      },
      {
        "name": "Gemini Furniture"
      },
      {
        "name": "Edwin Hansen"
      }
    ]
  }
}
```



```
query MyQuery($limit: Int, $offset: Int) {
  allPartners(limit: $limit, offset: $offset) {
    name
  }
}
```

QUERY VARIABLES

```
{
  "limit": 5,
  "offset": 35
}
```

```
{
  "data": {
    "allPartners": [
      {
        "name": "Mitchell Admin"
      },
      {
        "name": "toto"
      }
    ]
  }
}
```

How to… for Odoo with Graphene

```python
import graphene

class Country(graphene.ObjectType):
    code = graphene.String(required=True)
    name = graphene.String(required=True)
```

How to… for Odoo with Graphene

```python
from odoo.addons.graphql_base import OdooObjectType

class Partner(OdooObjectType):
    name = graphene.String(required=True)
    street = graphene.String()
    street2 = graphene.String()
    city = graphene.String()
    zip = graphene.String()
    email = graphene.String()
    phone = graphene.String()
    is_company = graphene.Boolean(required=True)
    # ...
```

How to... for Odoo with Graphene

```python
class Partner(OdooObjectType):
    # ...
    country = graphene.Field(Country)

    @staticmethod
    def resolve_country(root, info):
        return root.country_id or None
```

How to… for Odoo with Graphene

```python
class Partner(OdooObjectType):
    # ...
    contacts = graphene.List(
        graphene.NonNull(lambda: Partner),
        required=True,
    )

    def resolve_contacts(root, info):
        return root.child_ids
```

How to… for Odoo with Graphene

```
class Query(graphene.ObjectType):

    all_partners = graphene.List(
        graphene.NonNull(Partner),
        required=True,
        companies_only=graphene.Boolean(),
        limit=graphene.Int(),
        offset=graphene.Int(),
    )

    # ...
```

How to... for Odoo with Graphene

```
class Query(graphene.ObjectType):
    # ...
    def resolve_all_partners(
        root, info, companies_only=False, limit=limit, offset=offset
    ):
        # ... check for max limit
        domain = []
        if companies_only:
            domain.append(("is_company", "=", True))

        ResPartner = info.context["env"]["res.partner"]
        return ResPartner.search(domain, limit=limit, offset=offset)
```

How to… for Odoo with Graphene

```
schema = graphene.Schema(query=Query)
```

How to... for Odoo with Graphene

```python
from odoo import http
from odoo.addons.graphql_base import GraphQLControllerMixin
from ..schema import schema

class GraphQLController(http.Controller, GraphQLControllerMixin):

    @http.route("/graphiql/demo", auth="user")  # GraphiQL IDE
    def graphiql(self, **kwargs):
        return self._handle_graphiql_request(schema)

    @http.route("/graphql/demo", auth="user")
    def graphql(self, **kwargs):
        return self._handle_graphql_request(schema)
```

How is GraphQL different? A long ancestry

- ASN.1, DCOM, CORBA, SOAP, REST+OpenAPI and many more
- Some sort of schema language
- Schema is machine readable (eg for automatic message validation)
- “On the wire” representation of corresponding messages
- Rigid request/response data structures
- The service developer interprets and validates the request, does stuff, and prepares the response

How is GraphQL different? What about SQL?

- Machine readable schema
- “On the wire” message representation is proprietary (database “drivers” instead)
- Flexible queries, written by the client developer
- There is no service developer, the database does it (stored procedures fall in previous category)

How is GraphQL different?

- Client-side freedom of SQL.
- Server-side freedom of REST.

Caveats and thoughts: a better REST?

- What is REST?
- REST + OpenAPI
- Crafting a pure REST API is an art that few master
- GraphQL breaks HTTP semantics
  - Little leverage of HTTP infrastructure (caching, firewalls, etc)
  - With pure REST it’s “easy”, see above
- Attention to wild clients, complex queries
- As always, it’s a matter of tradeoffs

Caveats and thoughts: Performance

Beware of naive implementation of resolvers!

DON'T (one database query per returned record):

```python
def resolve_contacts(root, info):
    ResPartner = info.context["env"]["res.partners"]
    return ResPartner.search([('parent_id', '=', root.id)])
```

DO (use ORM prefetching strategies):

```python
def resolve_contacts(root, info):
    return root.child_ids
```

Caveats and thoughts: Façadism

- Temptation to expose all your domain model?
    - Easy with generic GraphQL adapters (Django, SQLAlchemy, …)
- It depends on the use case
- Often better to create a façade dedicated to the client use cases
    - Don’t expose your guts and break clients when your domain model changes

Caveats and thoughts: Access Control

- With traditional RPC (eg REST), access control is typically done at the façade/service level
- GraphQL typically binds at the domain model level
- Built-in security in your domain model or data sources?

Key takeaways

- GraphQL is easier than it sounds, try it!
- Powerful alternative to REST
- *Very* easy to integrate in any Python web application thanks to Graphene
- High productivity for backend devs
- High flexibility to frontend devs

Resources

- Start here
  - https://graphql.org/learn/
- With Python
  - https://graphene-python.org/
  - Incl. support for different frameworks (eg Django, SQLAlchemy)
- With Odoo
  - https://pypi.org/project/odoo12-addon-graphql-base/
  - https://pypi.org/project/odoo12-addon-graphql-demo/

Questions?

@sbidoul
stephane.bidoul@acsone.eu