

Carlos Ariel Rubio Espinosa

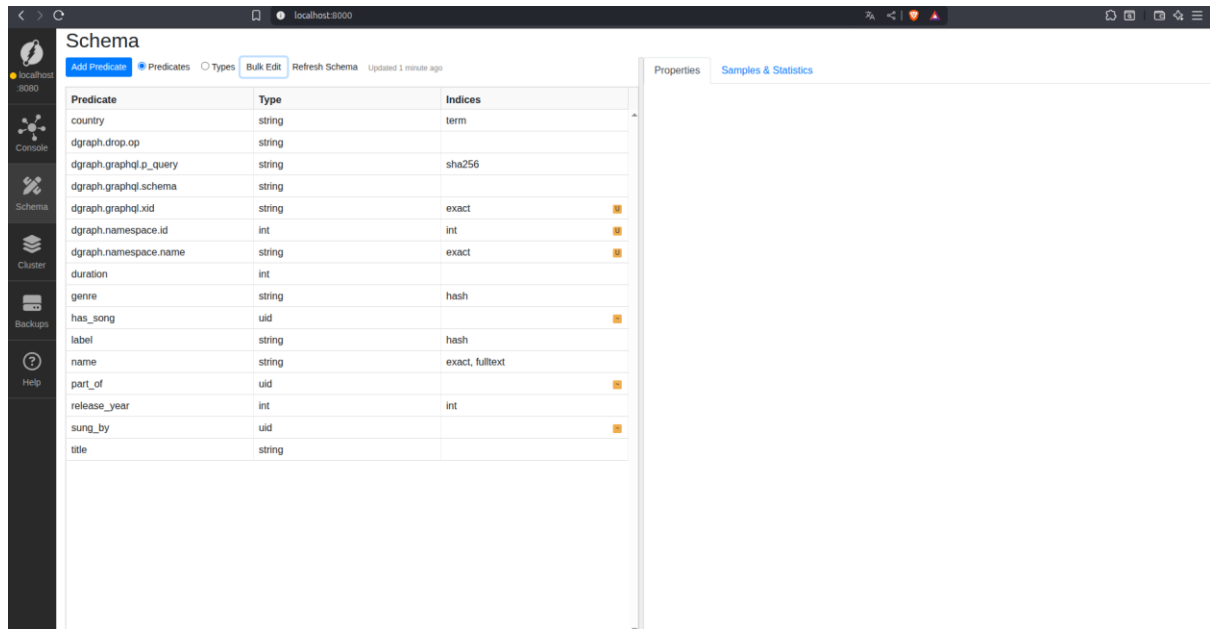
Creación de Schema:

```
<country>: string @index(term) .  
<dgraph.drop.op>: string .  
<dgraph.graphql.p_query>: string @index(sha256) .  
<dgraph.graphql.schema>: string .  
<dgraph.graphql.xid>: string @index(exact) @upsert .  
<dgraph.namespace.id>: int @index(int) @upsert .  
<dgraph.namespace.name>: string @index(exact) @upsert .  
<duration>: int .  
<genre>: string @index(hash) .  
<has_song>: uid @reverse .  
<label>: string @index(hash) .  
<name>: string @index(exact, fulltext) .  
<part_of>: uid @reverse .  
<release_year>: int @index(int) .  
<sung_by>: uid @reverse .  
<title>: string .  
type <dgraph.graphql> {  
  dgraph.graphql.schema  
  dgraph.graphql.xid  
}  
type <dgraph.graphql.persisted_query> {  
  dgraph.graphql.p_query  
}  
type <dgraph.namespace> {  
  dgraph.namespace.name
```

dgraph.namespace.id

}

Prueba del Schema hecho:

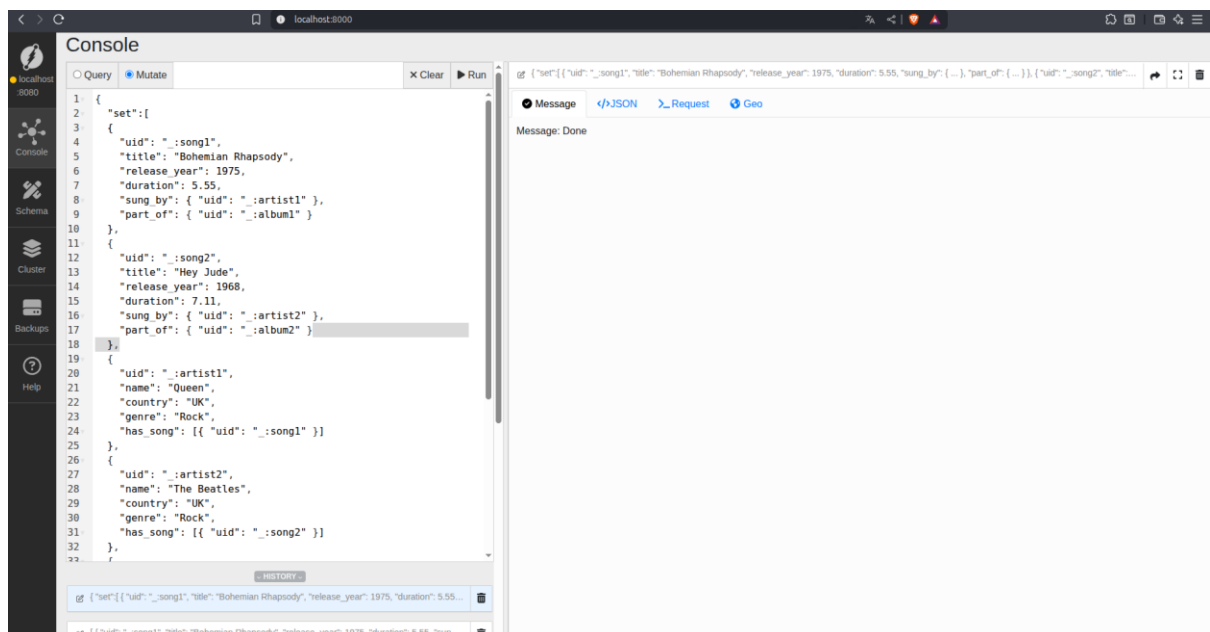


The screenshot shows the Dgraph Studio interface with the Schema tab selected. The left sidebar contains navigation options: localhost:8080, Console, Schema, Cluster, Backups, and Help. The main area displays a table of predicates with their types and indices.

Predicate	Type	Indices
country	string	term
dgraph.drop.op	string	
dgraph.graphql.p_query	string	sha256
dgraph.graphql.schema	string	
dgraph.graphql.xid	string	exact
dgraph.namespace.id	int	int
dgraph.namespace.name	string	exact
duration	int	
genre	string	hash
has_song	uid	
label	string	hash
name	string	exact, fulltext
part_of	uid	
release_year	int	int
sung_by	uid	
title	string	

On the right, there are tabs for Properties and Samples & Statistics.

Insertión de datos:



The screenshot shows the Dgraph Studio interface with the Console tab selected. The left sidebar is the same as in the previous image. The main area displays a JSON query for inserting data into the graph.

```
1 {
2   "set": [
3     {
4       "uid": "._:song1",
5       "title": "Bohemian Rhapsody",
6       "release_year": 1975,
7       "duration": 5.55,
8       "sung_by": { "uid": "._:artist1" },
9       "part_of": { "uid": "._:album1" }
10    },
11    {
12      "uid": "._:song2",
13      "title": "Hey Jude",
14      "release_year": 1968,
15      "duration": 7.11,
16      "sung_by": { "uid": "._:artist2" },
17      "part_of": { "uid": "._:album2" }
18    }
19  ],
20  {
21    "uid": "._:artist1",
22    "name": "Queen",
23    "country": "UK",
24    "genre": "Rock",
25    "has_song": [{ "uid": "._:song1" }]
26  },
27  {
28    "uid": "._:artist2",
29    "name": "The Beatles",
30    "country": "UK",
31    "genre": "Rock",
32    "has_song": [{ "uid": "._:song2" }]
33  }
34 }
```

On the right, there are tabs for Message, JSON, Request, and Geo. The Message tab is selected, showing "Message: Done".

Primera Consulta:

localhost:3000

Console

```

1 {
2   artist(func: eq(name, "Queen")) {
3     uid
4     name
5     country
6     has_song {
7       uid
8       title
9       release_year
10      duration
11    }
12  }
13 }

```

Query

Q { artist(func: eq(name, "Queen")) { uid name country has\_song { ... } } }

Graph

uid: 0x5

pred.	value
duration	5
release_year	1975
title	"Bohemian Rhapsody"
uid	"0x5"

artist has\_song

## Segunda Consulta:

localhost:3000

Console

```

1 {
2   song(func: eq(title, "Hey Jude")) {
3     uid
4     title
5     # Sigue la relación "part_of" para ver el álbum
6     part_of {
7       uid
8       title
9       release_year
10      label
11    }
12  }
13 }

```

Query

Q # ESTA CONSULTA SOLO FUNCIONA SI AÑADES @index(exact) A <title> { song(func: eq(title, "Hey Jude")) { uid title # Sigue la relación "part\_of" para ... }

Graph

Showing 2 nodes and 1 edges

song part\_of

## Tercera Consulta:

The screenshot shows the GraphQL IDE interface. On the left, the 'Console' tab displays a query: `1 { 2 songs_by_year(func: eq(release_year, 1975)) 3 @filter(has(duration)) { 4 uid 5 title 6 release_year 7 duration 8 sung_by { 9 name 10 } 11 }`. The 'Graph' tab on the right shows a visualization with two nodes: a green circle labeled 'Queen' and a blue circle. An edge labeled 'sung\_by' connects them. A sidebar on the right shows the details for the 'Queen' node (uid: 0x5), including its duration (5), release year (1975), and title ('Bohemian Rhapsody').

Mostrar todos los datos:

The screenshot shows the GraphQL IDE interface. On the left, the 'Console' tab displays a query: `1 { 2 artista(func: has(name)) { 3 uid 4 name 5 has_song { 6 uid 7 title 8 part_of { 9 uid 10 title 11 release_year 12 } 13 } 14 } 15 }`. The 'Graph' tab on the right shows a visualization with three nodes: a green circle, a blue circle labeled 'Queen', and a pink circle. Edges connect them: 'part\_of' (pink) from the green node to the pink node, 'has\_song' (green) from the green node to the blue node, and 'part\_of' (pink) from the pink node to the blue node. A sidebar on the right shows the details for the 'Queen' node (uid: 0x5), including its duration (5), release year (1975), and title ('Bohemian Rhapsody').

Conclusión:

Al ser esto una creación y consulta más guiada, siento que es más fácil de entender cómo funciona Dgraph y más rápida de hacer esto que el primer laboratorio.