Edwin Peraza

Fall 2023, CPSC 449, Section 1

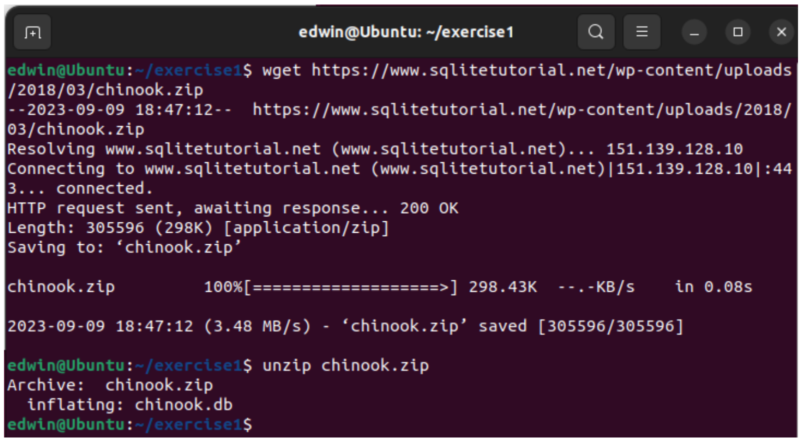
Exercise 1

Step 1:

As instructed, I ran the two given commands to download and extract the chinook SQLite sample database from sqlitetutorial.net.

Commands:

- wget https://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip
- unzip chinook.zip

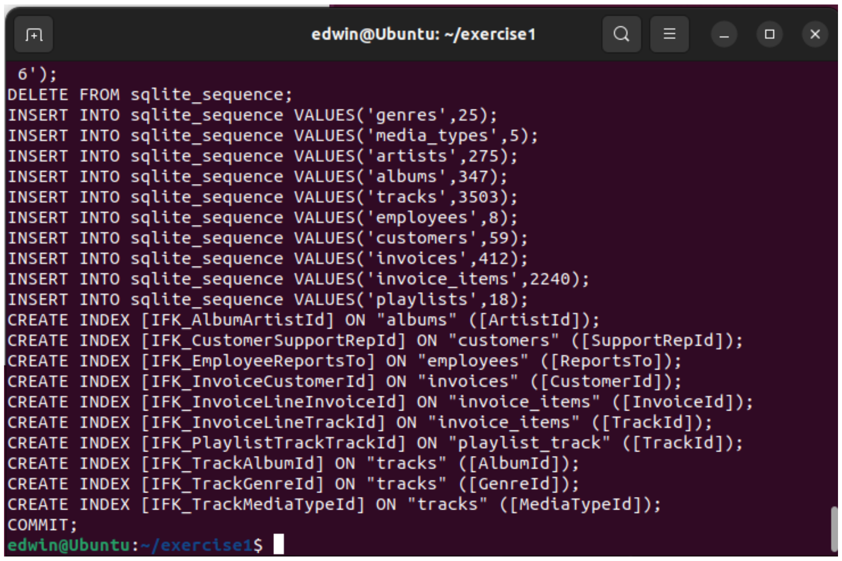


```
edwin@Ubuntu: ~/exercise1
edwin@Ubuntu:~/exercise1$ wget https://www.sqlitetutorial.net/wp-content/uploads
/2018/03/chinook.zip
--2023-09-09 18:47:12--  https://www.sqlitetutorial.net/wp-content/uploads/2018/
03/chinook.zip
Resolving www.sqlitetutorial.net (www.sqlitetutorial.net)... 151.139.128.10
Connecting to www.sqlitetutorial.net (www.sqlitetutorial.net)|151.139.128.10|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 305596 (298K) [application/zip]
Saving to: ‘chinook.zip’

chinook.zip         100%[===================>] 298.43K  --.-KB/s    in 0.08s

2023-09-09 18:47:12 (3.48 MB/s) - ‘chinook.zip’ saved [305596/305596]

edwin@Ubuntu:~/exercise1$ unzip chinook.zip
Archive:  chinook.zip
  inflating: chinook.db
edwin@Ubuntu:~/exercise1$
```

After that I ran the commands to install the Command Line Shell for SQLite and verified that the database has been extracted properly.

Commands:

- **`sudo apt update`**
- **`sudo apt install --yes sqlite3`**
- **`sqlite3 chinook.db .dump`**



```
edwin@Ubuntu: ~/exercise1
 6');
DELETE FROM sqlite_sequence;
INSERT INTO sqlite_sequence VALUES('genres',25);
INSERT INTO sqlite_sequence VALUES('media_types',5);
INSERT INTO sqlite_sequence VALUES('artists',275);
INSERT INTO sqlite_sequence VALUES('albums',347);
INSERT INTO sqlite_sequence VALUES('tracks',3503);
INSERT INTO sqlite_sequence VALUES('employees',8);
INSERT INTO sqlite_sequence VALUES('customers',59);
INSERT INTO sqlite_sequence VALUES('invoices',412);
INSERT INTO sqlite_sequence VALUES('invoice_items',2240);
INSERT INTO sqlite_sequence VALUES('playlists',18);
CREATE INDEX [IFK_AlbumArtistId] ON "albums" ([ArtistId]);
CREATE INDEX [IFK_CustomerSupportRepId] ON "customers" ([SupportRepId]);
CREATE INDEX [IFK_EmployeeReportsTo] ON "employees" ([ReportsTo]);
CREATE INDEX [IFK_InvoiceCustomerId] ON "invoices" ([CustomerId]);
CREATE INDEX [IFK_InvoiceLineInvoiceId] ON "invoice_items" ([InvoiceId]);
CREATE INDEX [IFK_InvoiceLineTrackId] ON "invoice_items" ([TrackId]);
CREATE INDEX [IFK_PlaylistTrackTrackId] ON "playlist_track" ([TrackId]);
CREATE INDEX [IFK_TrackAlbumId] ON "tracks" ([AlbumId]);
CREATE INDEX [IFK_TrackGenreId] ON "tracks" ([GenreId]);
CREATE INDEX [IFK_TrackMediaTypeId] ON "tracks" ([MediaTypeId]);
COMMIT;
edwin@Ubuntu:~/exercise1$
```

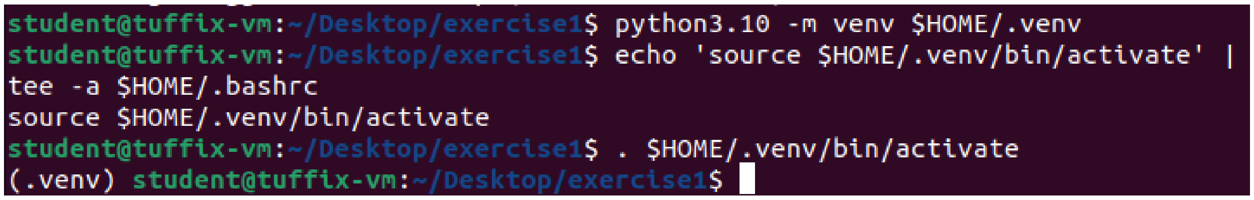
Step 2:

In this step, we create the virtual environment for python. Fist I installed pip and venv packages with the command:

- `sudo apt install --yes python3-pip python3-venv`

After that I setup the virtual environment with the commands:

- `python3.10 -m venv $HOME/.venv`
- `echo 'source $HOME/.venv/bin/activate' | tee -a $HOME/.bashrc`
- `. $HOME/.venv/bin/activate`



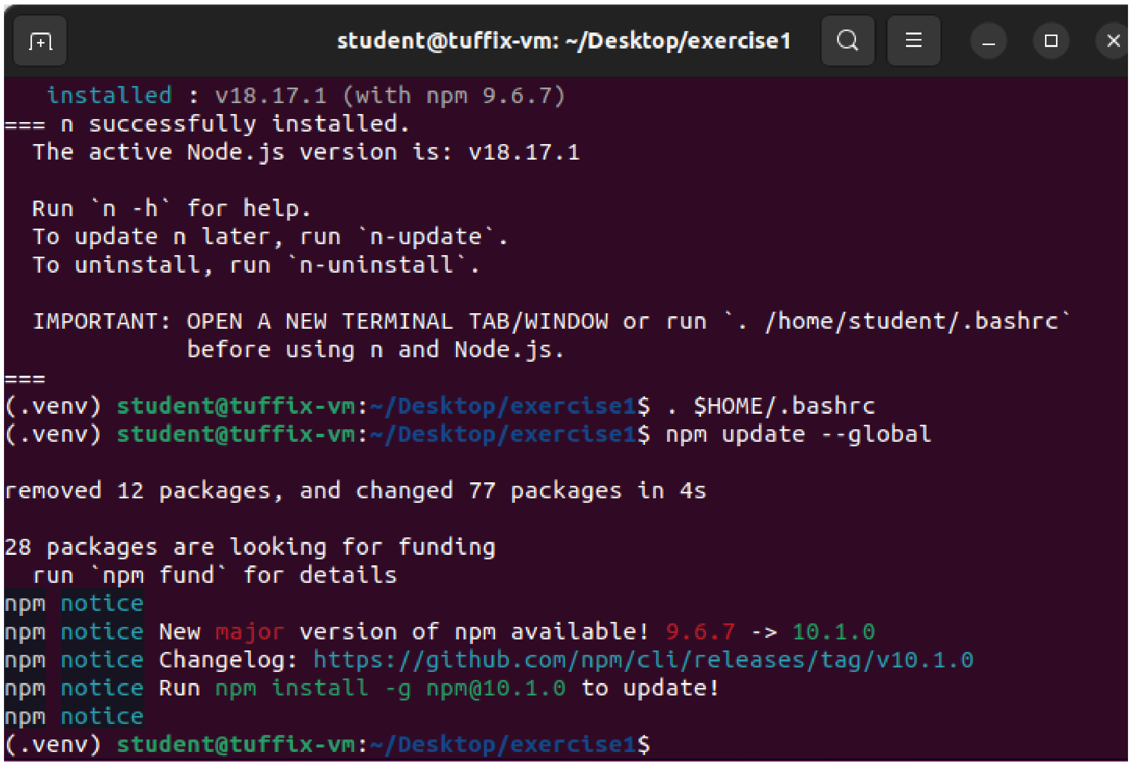
```
student@tuffix-vm:~/Desktop/exercise1$ python3.10 -m venv $HOME/.venv
student@tuffix-vm:~/Desktop/exercise1$ echo 'source $HOME/.venv/bin/activate' |
tee -a $HOME/.bashrc
source $HOME/.venv/bin/activate
student@tuffix-vm:~/Desktop/exercise1$ . $HOME/.venv/bin/activate
(.venv) student@tuffix-vm:~/Desktop/exercise1$
```

Step 3:

In this step I installed n version manager for node.js

Commands:

- curl -s -L http://git.io/n-install | bash -s -- -y
- . $HOME/.bashrc
- npm update --global



```
   installed : v18.17.1 (with npm 9.6.7)
=== n successfully installed.
  The active Node.js version is: v18.17.1

  Run `n -h` for help.
  To update n later, run `n-update`.
  To uninstall, run `n-uninstall`.

  IMPORTANT: OPEN A NEW TERMINAL TAB/WINDOW or run `. /home/student/.bashrc`
             before using n and Node.js.
===
(.venv) student@tuffix-vm:~/Desktop/exercise1$ . $HOME/.bashrc
(.venv) student@tuffix-vm:~/Desktop/exercise1$ npm update --global

removed 12 packages, and changed 77 packages in 4s

28 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New major version of npm available! 9.6.7 -> 10.1.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.1.0
npm notice Run npm install -g npm@10.1.0 to update!
npm notice
(.venv) student@tuffix-vm:~/Desktop/exercise1$
```

Now I used the command to install the soul server for the REST API.

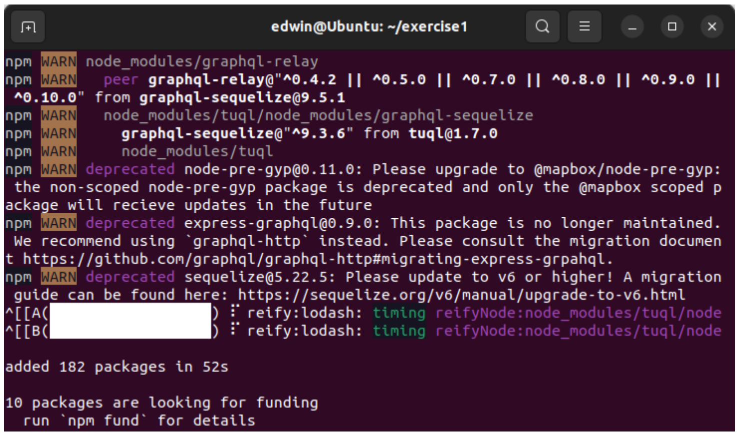
Command:

- `npm install --global soul-cli`

Subsequently, I ran the command to install tuql server for the GraphQL API.

Command:

- `npm install --global tuql`

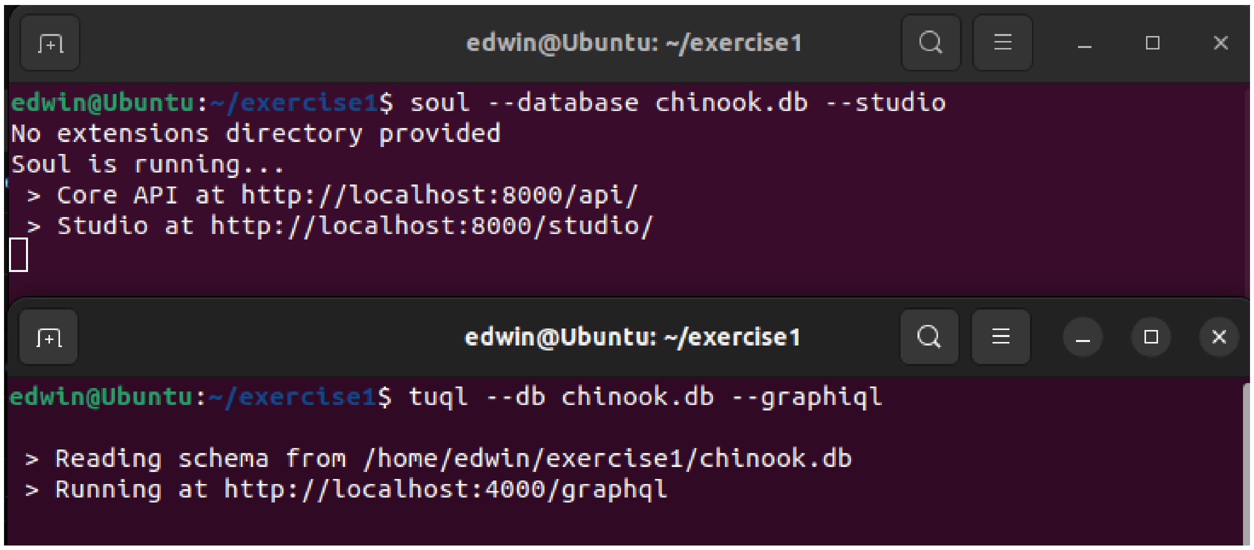


```
npm WARN node_modules/graphql-relay
npm WARN   peer graphql-relay@"^0.4.2 || ^0.5.0 || ^0.7.0 || ^0.8.0 || ^0.9.0 ||
 ^0.10.0" from graphql-sequelize@9.5.1
npm WARN   node_modules/tuql/node_modules/graphql-sequelize
npm WARN     graphql-sequelize@"^9.3.6" from tuql@1.7.0
npm WARN     node_modules/tuql
npm WARN deprecated node-pre-gyp@0.11.0: Please upgrade to @mapbox/node-pre-gyp:
 the non-scoped node-pre-gyp package is deprecated and only the @mapbox scoped p
ackage will recieve updates in the future
npm WARN deprecated express-graphql@0.9.0: This package is no longer maintained.
 We recommend using `graphql-http` instead. Please consult the migration documen
t https://github.com/graphql/graphql-http#migrating-express-grpahql.
npm WARN deprecated sequelize@5.22.5: Please update to v6 or higher! A migration
 guide can be found here: https://sequelize.org/v6/manual/upgrade-to-v6.html
^[[A(                  ) ⠏ reify:lodash: timing reifyNode:node_modules/tuql/node
^[[B(                  ) ⠏ reify:lodash: timing reifyNode:node_modules/tuql/node

added 182 packages in 52s

10 packages are looking for funding
  run `npm fund` for details
```

Step 4:

Opening two different terminals, I started API servers for the database with the following commands:

- soul --database chinook.db --studio
- tuql --db chinook.db –graphiql

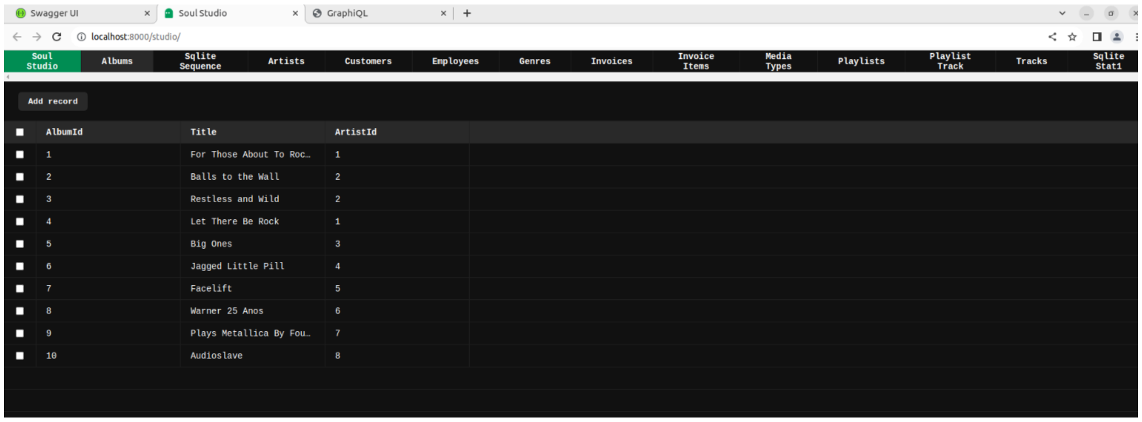


```
edwin@Ubuntu:~/exercise1$ soul --database chinook.db --studio
No extensions directory provided
Soul is running...
 > Core API at http://localhost:8000/api/
 > Studio at http://localhost:8000/studio/
```

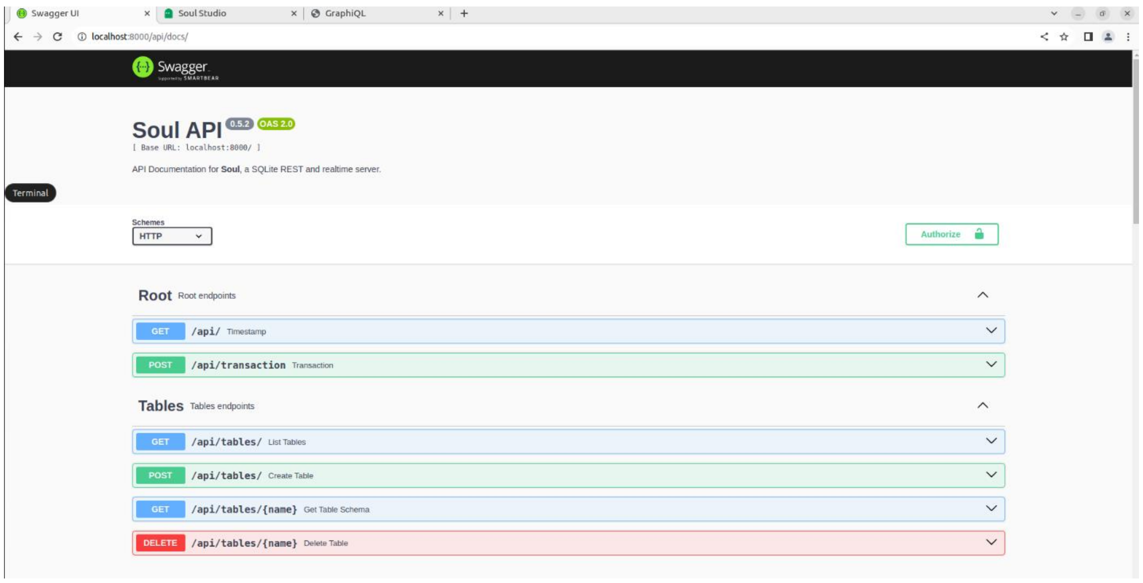
```
edwin@Ubuntu:~/exercise1$ tuql --db chinook.db --graphiql

 > Reading schema from /home/edwin/exercise1/chinook.db
 > Running at http://localhost:4000/graphql
```

Step 5:

I was able to access Soul Studio GUI, the Soul API documentation, and the GraphQL IDE for tuql.



AlbumId	Title	ArtistId
1	For Those About To Roc…	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3
6	Jagged Little Pill	4
7	Facelift	5
8	Warner 25 Anos	6
9	Plays Metallica By Fou…	7
10	Audioslave	8



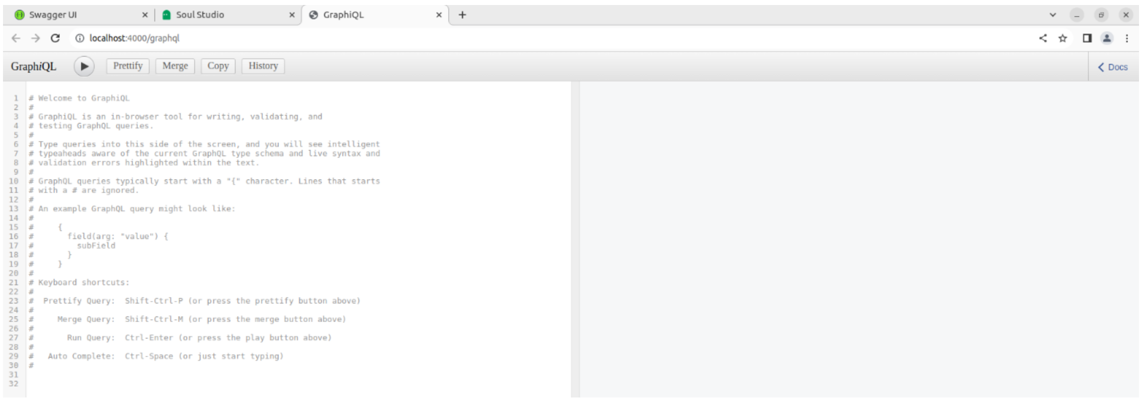
Soul API 0.5.2 OAS 2.0
[Base URL: localhost:8000/]
API Documentation for **Soul**, a SQLite REST and realtime server.

Root Root endpoints

- GET /api/ Timestamp
- POST /api/transaction Transaction

Tables Tables endpoints

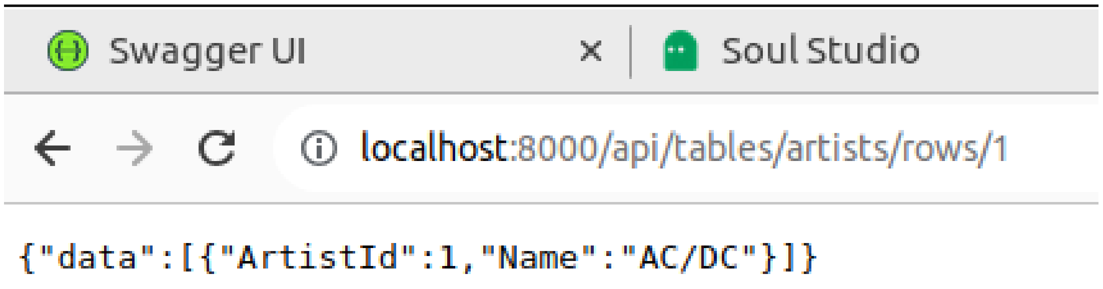
- GET /api/tables/ List Tables
- POST /api/tables/ Create Table
- GET /api/tables/{name} Get Table Schema
- DELETE /api/tables/{name} Delete Table



```
# 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)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#
```

Step 6:

I used the url http://localhost:8000/api/tables/artists/rows/1 to retrieve the information of the first artist in the database with the REST API.

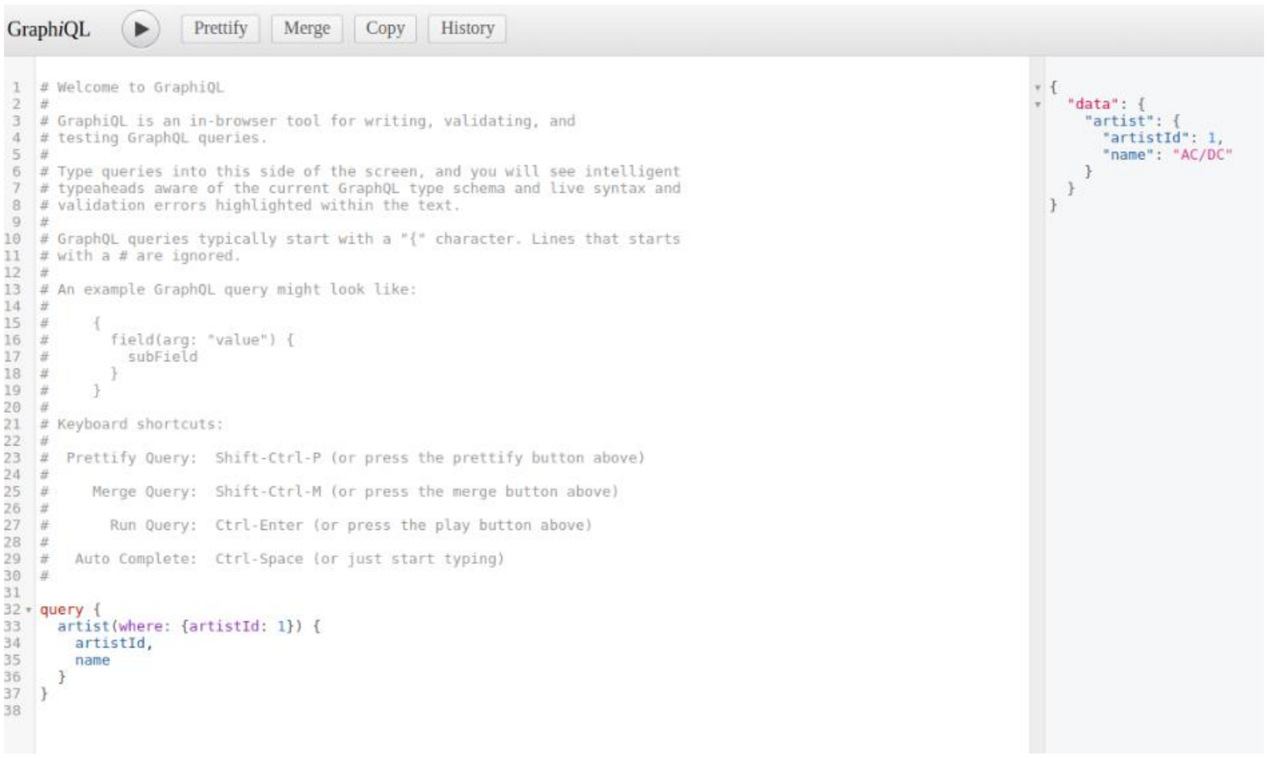


```
{"data":[{"ArtistId":1,"Name":"AC/DC"}]}
```

After that I ran the query in GraphiQL to retrieve the same record with the GraphQL API.

Query:

```
query {
  artist(where: {artistId: 1}) {
    artistId,
    name
  }
}
```



```
# 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)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#

query {
  artist(where: {artistId: 1}) {
    artistId,
    name
  }
}
```

```
{
  "data": {
    "artist": {
      "artistId": 1,
      "name": "AC/DC"
    }
  }
}
```

Step 7:

I wrote more queries.

1. Albums by the artist “Red Hot Chili Peppers”

REST API

First, I made a request for all the artists in the database to find the id for Red Hot Chili Peppers.

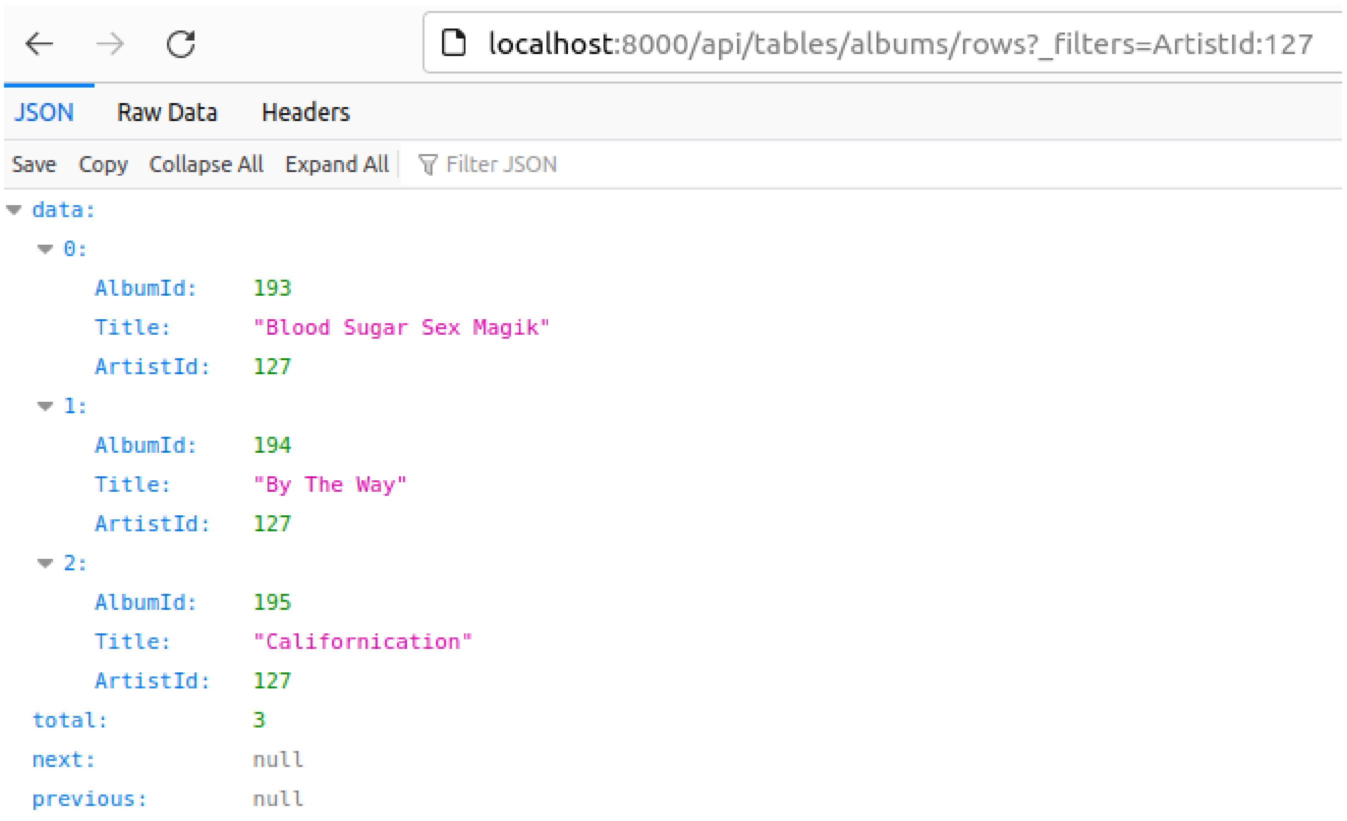
Query:

http://localhost:8000/api/tables/artists/rows?_filters=Name:Red%20Hot%20Chili%20Peppers

The id for Red Hot Chili Peppers is 127, with this we can find all the albums for this artist.

Query:

- http://localhost:8000/api/tables/albums/rows?_filters=ArtistId:127



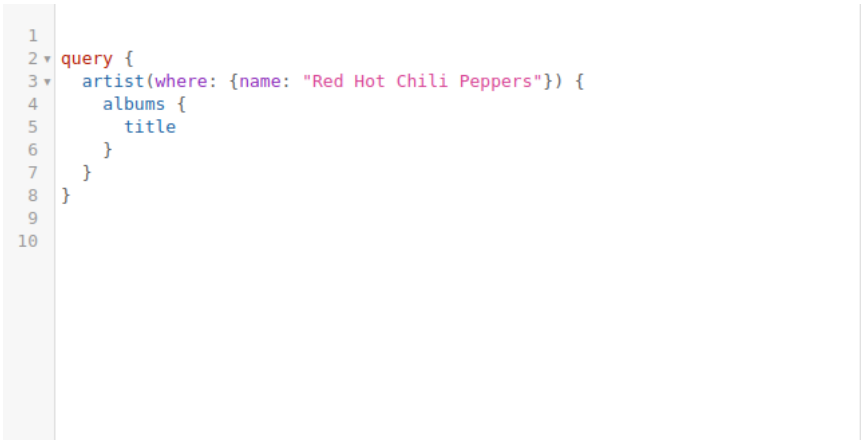
```
data:
  0:
    AlbumId:    193
    Title:      "Blood Sugar Sex Magik"
    ArtistId:   127
  1:
    AlbumId:    194
    Title:      "By The Way"
    ArtistId:   127
  2:
    AlbumId:    195
    Title:      "Californication"
    ArtistId:   127
total:          3
next:           null
previous:       null
```

GraphQL

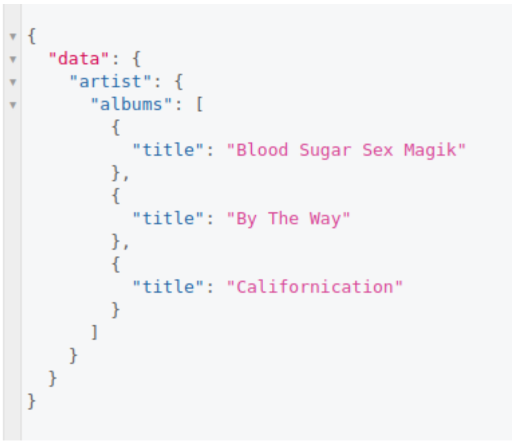
Here we use the following query to retrieve the albums by Red Hot Chili Peppers

Query:

```
query {

  artist (where: {name: "Red Hot Chili Peppers"}) {

    albums {

      title

    }

  }

}
```



```

query {
  artist(where: {name: "Red Hot Chili Peppers"}) {
    albums {
      title
    }
  }
}

```



```
{
  "data": {
    "artist": {
      "albums": [
        {
          "title": "Blood Sugar Sex Magik"
        },
        {
          "title": "By The Way"
        },
        {
          "title": "Californication"
        }
      ]
    }
  }
}
```

2. Genres associated with the artist “U2.”

REST API

Here we first find the ArtistId for U2 with the following query:

- http://localhost:8000/api/tables/artists/rows?_filters=name:U2

The ArtistId is 150.

Now we make a call in the albums table to find all the albums from U2

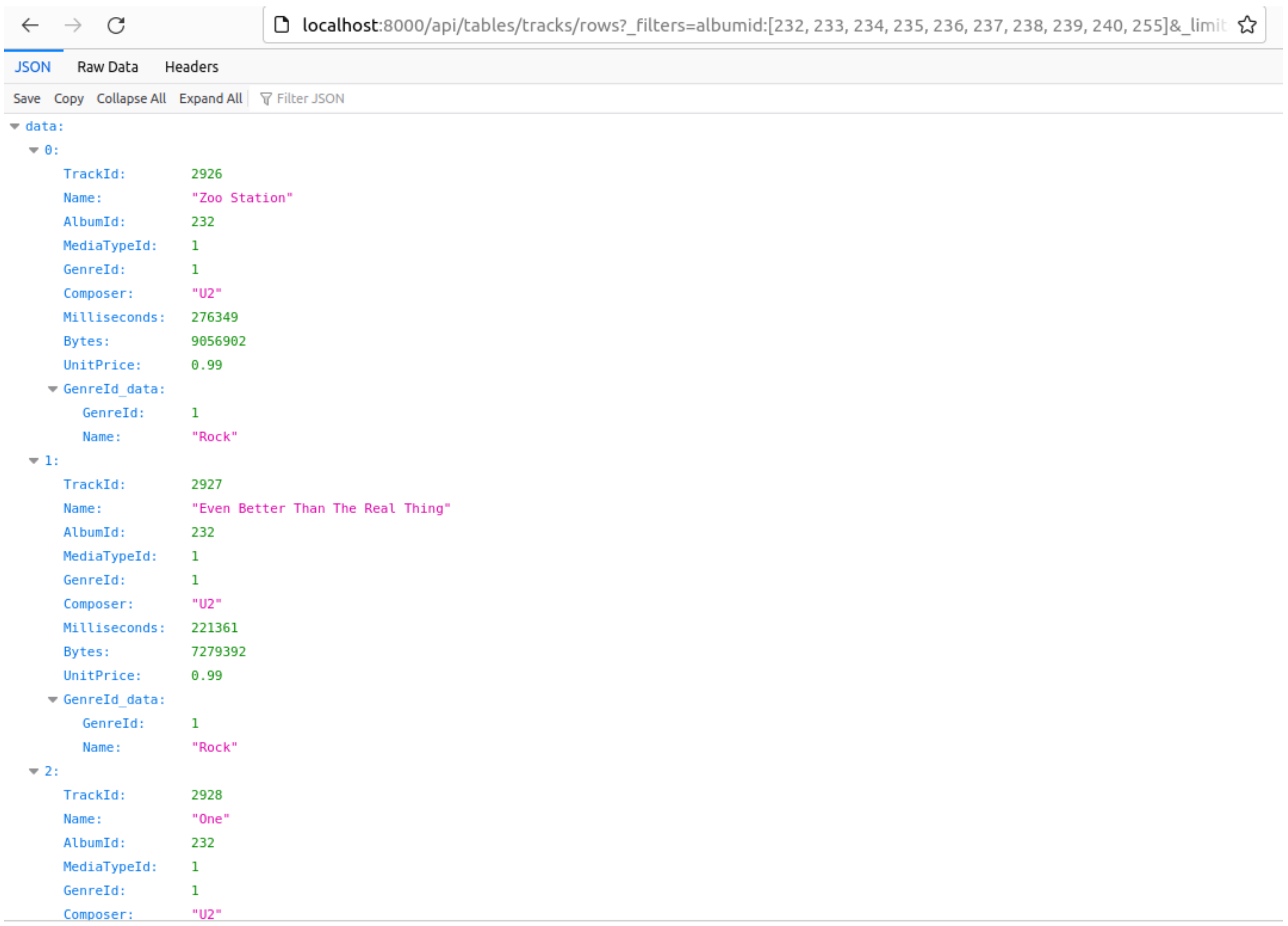
- http://localhost:8000/api/tables/albums/rows?_filters=artistid:150

The albums from U2 have the IDs: 232, 233, 234, 235, 236, 237, 238, 239, 240, and 255.

Now we can find all the tracks in these albums with the following query.

- [http://localhost:8000/api/tables/tracks/rows?_filters=albumid:[232,%20233,%20234,%20235,%20236,%20237,%20238,%20239,%20240,%20255]&_limit=135&_extend=GenreId](http://localhost:8000/api/tables/tracks/rows?_filters=albumid:[232,%20233,%20234,%20235,%20236,%20237,%20238,%20239,%20240,%20255]&_limit=135&_extend=GenreId)

By using extend, I can see the name of the genres which are “Rock” and “Pop”


localhost:8000/api/tables/tracks/rows?_filters=albumid:[232, 233, 234, 235, 236, 237, 238, 239, 240, 255]&_limit
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

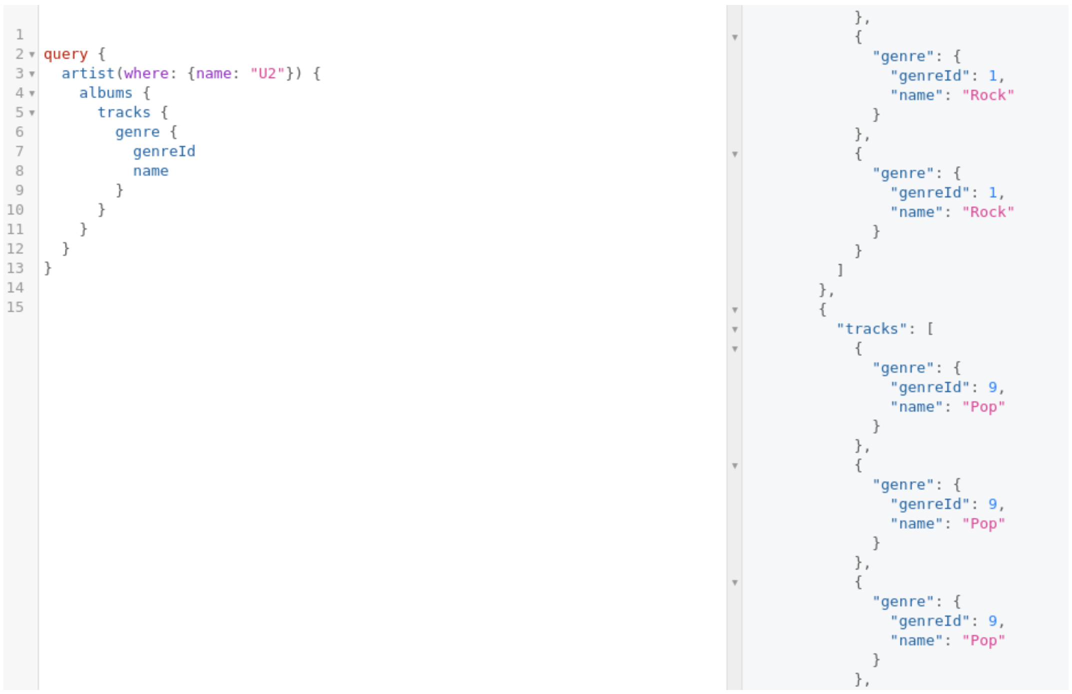
```
data:
  0:
    TrackId:          2926
    Name:             "Zoo Station"
    AlbumId:          232
    MediaTypeId:      1
    GenreId:          1
    Composer:         "U2"
    Milliseconds:     276349
    Bytes:            9056902
    UnitPrice:        0.99
    GenreId_data:
      GenreId:        1
      Name:           "Rock"
  1:
    TrackId:          2927
    Name:             "Even Better Than The Real Thing"
    AlbumId:          232
    MediaTypeId:      1
    GenreId:          1
    Composer:         "U2"
    Milliseconds:     221361
    Bytes:            7279392
    UnitPrice:        0.99
    GenreId_data:
      GenreId:        1
      Name:           "Rock"
  2:
    TrackId:          2928
    Name:             "One"
    AlbumId:          232
    MediaTypeId:      1
    GenreId:          1
    Composer:         "U2"
```

GraphQL

Here we use the following query to retrieve the genres associated with U2.

Query:

```
query {
  artist (where: {name: "U2"}) {
    albums {
      tracks {
        genre {
          genreId
          name
        }
      }
    }
  }
}
```



```

query {
  artist(where: {name: "U2"}) {
    albums {
      tracks {
        genre {
          genreId
          name
        }
      }
    }
  }
}


```

```
            },
            {
              "genre": {
                "genreId": 1,
                "name": "Rock"
              }
            },
            {
              "genre": {
                "genreId": 1,
                "name": "Rock"
              }
            }
          ]
        },
        {
          "tracks": [
            {
              "genre": {
                "genreId": 9,
                "name": "Pop"
              }
            },
            {
              "genre": {
                "genreId": 9,
                "name": "Pop"
              }
            },
            {
              "genre": {
                "genreId": 9,
                "name": "Pop"
              }
            },
```

3. Names of tracks on the playlist “Grunge” and their associated artists and albums.

REST API

First, we retrieve the PlaylistId for Grunge, which in this case is 16.

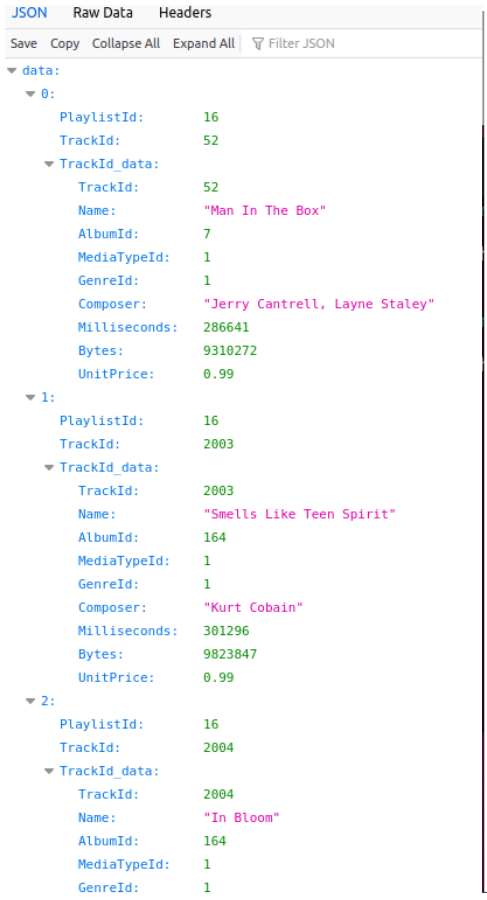
- http://localhost:8000/api/tables/playlists/rows?_filters=name:Grunge

Now we retrieve the tracks on the playlist.

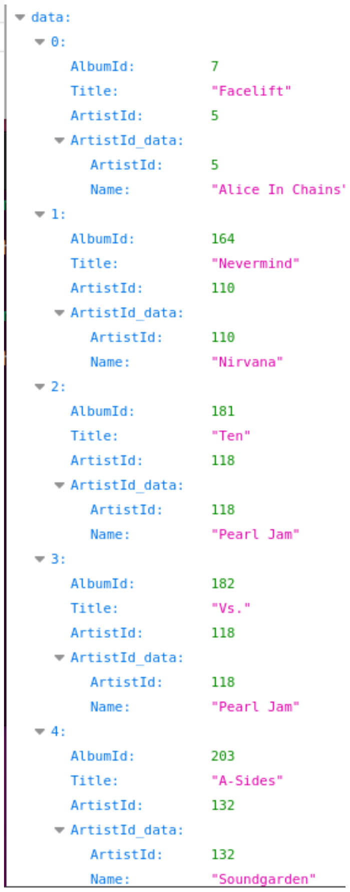
- http://localhost:8000/api/tables/playlist_track/rows?_filters=PlaylistId:16&_extend=TrackId&_limit=15

Lastly, we use the AlbumId to find the associated albums to the tracks in the playlist and their associated artist.

- http://localhost:8000/api/tables/albums/rows?_filters=AlbumId:[7,%20164,%20181,%20182,%20203,%20206,%20269]



```
data:
  0:
    PlaylistId:      16
    TrackId:         52
    TrackId_data:
      TrackId:       52
      Name:          "Man In The Box"
      AlbumId:       7
      MediaTypeId:   1
      GenreId:       1
      Composer:      "Jerry Cantrell, Layne Staley"
      Milliseconds:  286641
      Bytes:         9310272
      UnitPrice:     0.99
  1:
    PlaylistId:      16
    TrackId:         2003
    TrackId_data:
      TrackId:       2003
      Name:          "Smells Like Teen Spirit"
      AlbumId:       164
      MediaTypeId:   1
      GenreId:       1
      Composer:      "Kurt Cobain"
      Milliseconds:  301296
      Bytes:         9823847
      UnitPrice:     0.99
  2:
    PlaylistId:      16
    TrackId:         2004
    TrackId_data:
      TrackId:       2004
      Name:          "In Bloom"
      AlbumId:       164
      MediaTypeId:   1
      GenreId:       1
```



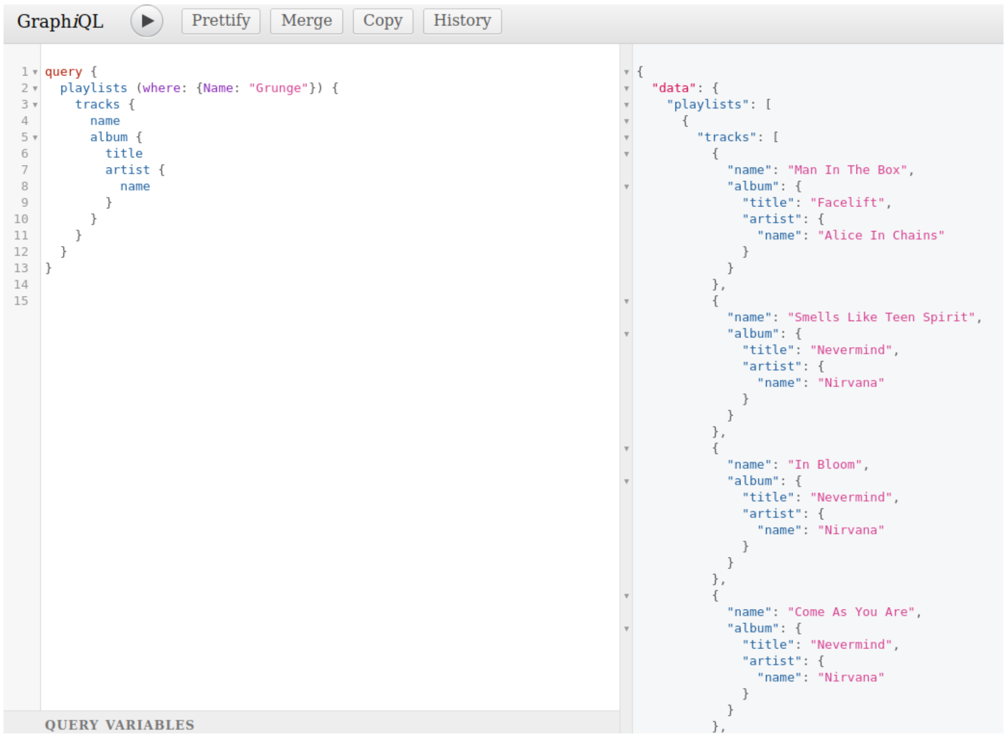
```
data:
  0:
    AlbumId:       7
    Title:         "Facelift"
    ArtistId:      5
    ArtistId_data:
      ArtistId:    5
      Name:        "Alice In Chains'
  1:
    AlbumId:       164
    Title:         "Nevermind"
    ArtistId:      110
    ArtistId_data:
      ArtistId:    110
      Name:        "Nirvana"
  2:
    AlbumId:       181
    Title:         "Ten"
    ArtistId:      118
    ArtistId_data:
      ArtistId:    118
      Name:        "Pearl Jam"
  3:
    AlbumId:       182
    Title:         "Vs."
    ArtistId:      118
    ArtistId_data:
      ArtistId:    118
      Name:        "Pearl Jam"
  4:
    AlbumId:       203
    Title:         "A-Sides"
    ArtistId:      132
    ArtistId_data:
      ArtistId:    132
      Name:        "Soundgarden"
```

GraphQL

The following query retrieves all the information that we need from the Grunge playlist.

Query:

```
query {
  playlists (where: {Name: "Grunge"}) {
       tracks {
      name
       album {
        title
        artist {
          name
        }
      }
    }
  }
}
```



```
query {
  playlists (where: {Name: "Grunge"}) {
    tracks {
      name
      album {
        title
        artist {
          name
        }
      }
    }
  }
}
```

```
{
  "data": {
    "playlists": [
      {
        "tracks": [
          {
            "name": "Man In The Box",
            "album": {
              "title": "Facelift",
              "artist": {
                "name": "Alice In Chains"
              }
            }
          },
          {
            "name": "Smells Like Teen Spirit",
            "album": {
              "title": "Nevermind",
              "artist": {
                "name": "Nirvana"
              }
            }
          },
          {
            "name": "In Bloom",
            "album": {
              "title": "Nevermind",
              "artist": {
                "name": "Nirvana"
              }
            }
          },
          {
            "name": "Come As You Are",
            "album": {
              "title": "Nevermind",
              "artist": {
                "name": "Nirvana"
              }
            }
          },
```

Step 8:

Now we make API request with python.

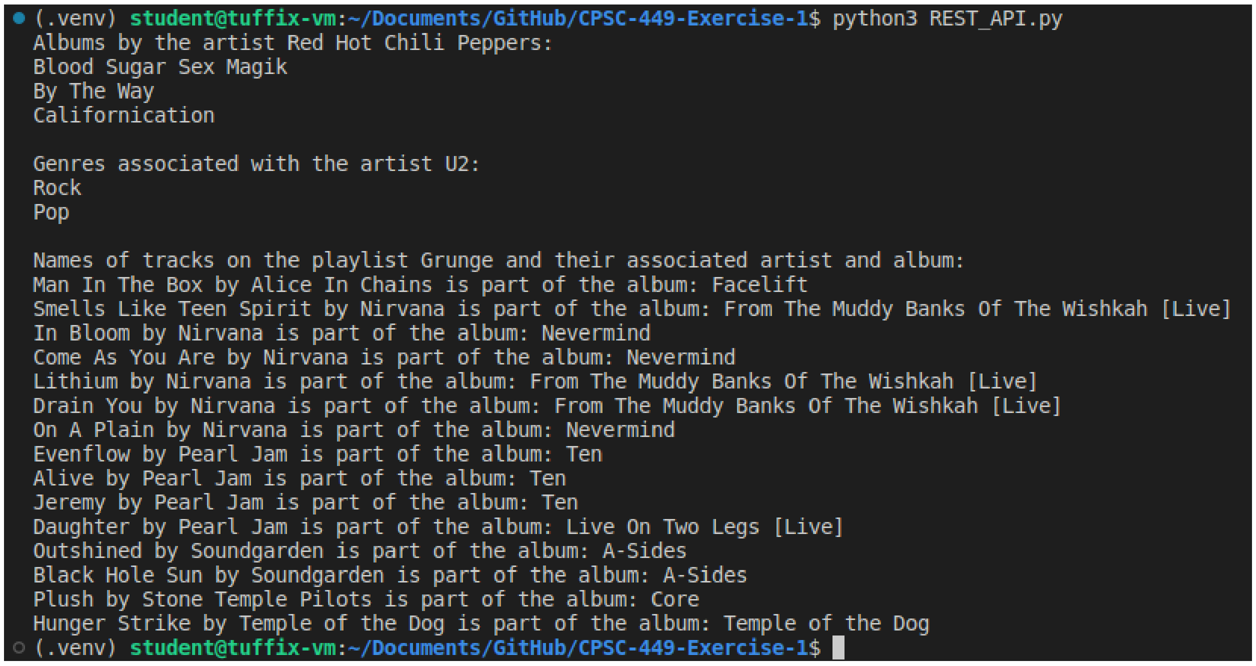
First, the code for my REST API is the following:

```
import requests

# url for REST API
REST_URL = "http://localhost:8000/api/"

artistTable = "tables/artists/rows"
albumsTable = "tables/albums/rows"
tracksTable = "tables/tracks/rows"
genresTable = "tables/genres/rows"
playlistTable = "tables/playlists/rows"
playlist_trackTable = "tables/playlist_track/rows"
retrieveID = {"_filters": "Name:Red Hot Chili Peppers"}

# use get to retrieve information
response = requests.get(REST_URL + artistTable, params={"_filters":
"Name:Red Hot Chili Peppers"})

# convert response to json
rhcp = response.json()

# get artist ID
artist_info = rhcp.get('data')
artistID = artist_info[0].get('ArtistId')

# now find albums by RHCP
response = requests.get(REST_URL + albumsTable, params={"_filters":
"ArtistId:" + str(artistID)})

# convert response to json
rhcp = response.json()

# print albums
print("Albums by the artist Red Hot Chili Peppers:")
albums_info = rhcp.get('data')
for album in albums_info:
    print(album.get('Title'))

# beginning of second REST API call
response = requests.get(REST_URL + artistTable, params={"_filters":
"Name:U2"})
```

```python
# convert response to json
u2 = response.json()

# get artist ID
artist_info = u2.get('data')
artistID = artist_info[0].get('ArtistId')

# now we find all the albums by U2
response = requests.get(REST_URL + albumsTable, params={"_filters":
"ArtistId:" + str(artistID)})

# convert response to json
u2 = response.json()

# retrieve albums ids
albums_info = u2.get('data')
albumIDs = []
for album in albums_info:
    albumIDs.append(album.get('AlbumId'))

# convert albumIDs to string
albumIDs = ','.join(map(str, albumIDs))

# now we find all the genres associated with the albums
response = requests.get(REST_URL + tracksTable, params={"_filters":
"AlbumId:[" + albumIDs + "]", "_limit":"135", "_extend": "GenreId"})
u2 = response.json()

# now we store the genres in a list
genres = set()
tracks_info = u2.get('data')
for track in tracks_info:
    genres.add(track.get('GenreId'))

# lastly, find the name of the genres
response = requests.get(REST_URL + genresTable, params={"_filters":
"GenreId:[" + ','.join(map(str, genres)) + "]"})
u2 = response.json()

# print the results to the screen
print("\nGenres associated with the artist U2:")
genres_info = u2.get('data')
for genre in genres_info:
    print(genre.get('Name'))
```

```python
# beginning of final REST API call
response = requests.get(REST_URL + playlistTable, params={"_filters":
"Name:Grunge"})
Grunge = response.json()

# get playlist ID
playlist_info = Grunge.get('data')
playlistID = playlist_info[0].get('PlaylistId')

# now we retrieve all the tracks on the playlist
response = requests.get(REST_URL + playlist_trackTable,
params={"_filters": "PlaylistId:" + str(playlistID), "_extend": "TrackId",
"_limit": "15"})
Grunge = response.json()

# store track ids
tracks_info = Grunge.get('data')
trackIDs = []
for track in tracks_info:
    trackIDs.append(track.get('TrackId'))

# retrieve track names and albums id
response = requests.get(REST_URL + tracksTable, params={"_filters":
"TrackId:[" + ','.join(map(str, trackIDs)) + "]", "_extend": "AlbumId",
"_limit": "15"})
Grunge = response.json()

# store track names and albums id
tracks_info = Grunge.get('data')
trackNames = []
albumIDs = []
for track in tracks_info:
    trackNames.append(track.get('Name'))
    albumIDs.append(track.get('AlbumId'))

# print the results to the screen
print("\nNames of tracks on the playlist Grunge and their associated
artist and album:")
for track in trackNames:
    response = requests.get(REST_URL + tracksTable, params={"_filters":
"Name:" + track, "_extend": "AlbumId", "_limit": "15"})
    Grunge = response.json()
    tracks_info = Grunge.get('data')
    album_ID = tracks_info[0]['AlbumId']
    album_name = tracks_info[0]['AlbumId_data']['Title']
    response = requests.get(REST_URL + albumsTable, params={"_filters":
"AlbumId:" + str(album_ID), "_extend": "ArtistId", "_limit": "15"})
    Grunge = response.json()
    albums_info = Grunge.get('data')
    artist_ID = albums_info[0]['ArtistId_data']['Name']
    print(track + " by " + artist_ID + " is part of the album: " +
album_name)

# end of REST API calls
```

This is a screenshot of the results of the program.



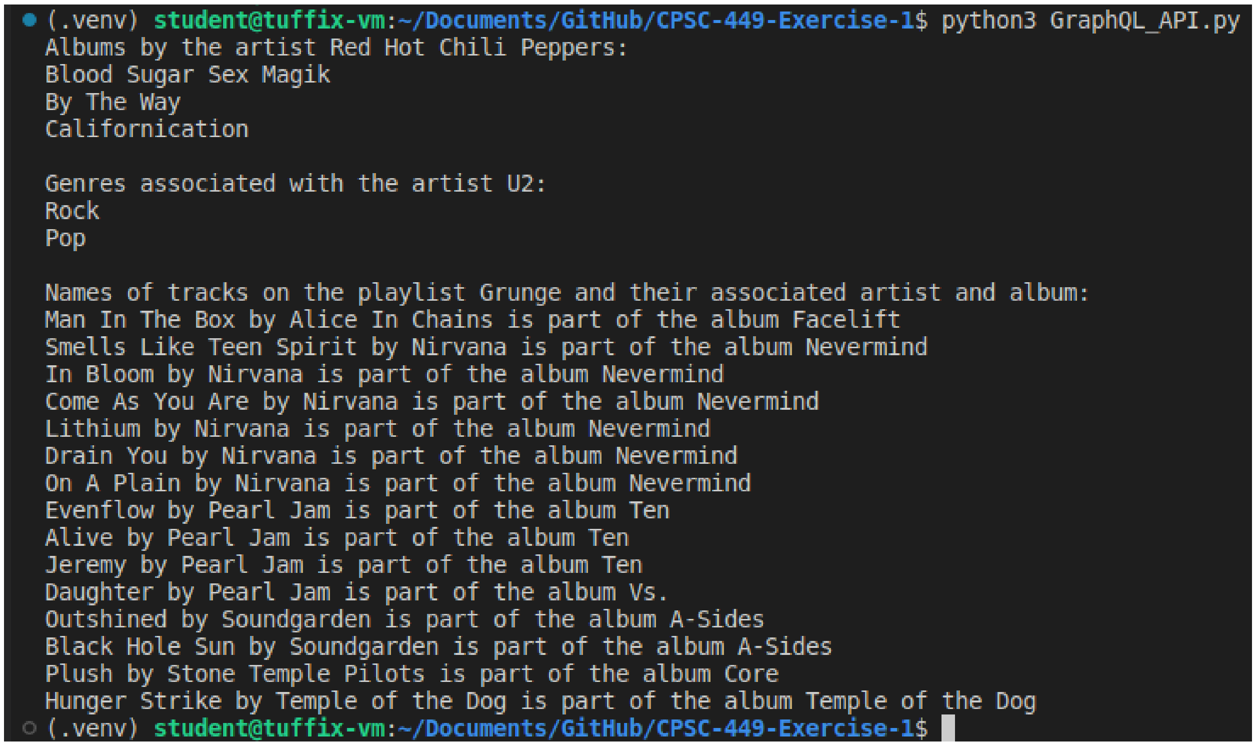
```
● (.venv) student@tuffix-vm:~/Documents/GitHub/CPSC-449-Exercise-1$ python3 REST_API.py
  Albums by the artist Red Hot Chili Peppers:
  Blood Sugar Sex Magik
  By The Way
  Californication

  Genres associated with the artist U2:
  Rock
  Pop

  Names of tracks on the playlist Grunge and their associated artist and album:
  Man In The Box by Alice In Chains is part of the album: Facelift
  Smells Like Teen Spirit by Nirvana is part of the album: From The Muddy Banks Of The Wishkah [Live]
  In Bloom by Nirvana is part of the album: Nevermind
  Come As You Are by Nirvana is part of the album: Nevermind
  Lithium by Nirvana is part of the album: From The Muddy Banks Of The Wishkah [Live]
  Drain You by Nirvana is part of the album: From The Muddy Banks Of The Wishkah [Live]
  On A Plain by Nirvana is part of the album: Nevermind
  Evenflow by Pearl Jam is part of the album: Ten
  Alive by Pearl Jam is part of the album: Ten
  Jeremy by Pearl Jam is part of the album: Ten
  Daughter by Pearl Jam is part of the album: Live On Two Legs [Live]
  Outshined by Soundgarden is part of the album: A-Sides
  Black Hole Sun by Soundgarden is part of the album: A-Sides
  Plush by Stone Temple Pilots is part of the album: Core
  Hunger Strike by Temple of the Dog is part of the album: Temple of the Dog
○ (.venv) student@tuffix-vm:~/Documents/GitHub/CPSC-449-Exercise-1$
```

Now for the GraphQL API:

```
import requests

# url for GraphQL API
GRAPHQL_URL = "http://localhost:4000/graphql/"


# Define your GraphQL query for artists
albumsQuery = """
{
  artists(where: { name: "Red Hot Chili Peppers" }) {
    albums {
        title
    }
  }
}
"""

# Send the GraphQL query
response = requests.get(GRAPHQL_URL, json={"query": albumsQuery})

# Parse the JSON response
RHCP = response.json()
albums = RHCP["data"]["artists"][0]["albums"]

# Print the albums by RHCP
print("Albums by the artist Red Hot Chili Peppers:")
for album in albums:
    print(album["title"])

# beginning of second GraphQL API call
genresQuery = """
{
  artist (where: {name: "U2"}) {
    albums {
      tracks {
        genre {
          genreId
          name
        }
      }
    }
  }
}
"""
```

```python
# Send the GraphQL query
response = requests.get(GRAPHQL_URL, json={"query": genresQuery})

# create set to store genres
genres = set()

# Parse the JSON response
U2 = response.json()
for album in U2["data"]["artist"]["albums"]:
    for track in album["tracks"]:
        genres.add(track["genre"]["name"])

# print genres associated with U2
print("\nGenres associated with the artist U2:")
for genre in genres:
    print(genre)

# beginning of last GraphQL API call
grungeQuery = """
{
  playlists (where: {Name: "Grunge"}) {
    tracks {
      name
        album {
        title
        artist {
          name
        }
      }
    }
  }
}
"""

# Send the GraphQL query
response = requests.get(GRAPHQL_URL, json={"query": grungeQuery})

# Parse the JSON response
Grunge = response.json()
# print track name, album title, artist name
print("\nNames of tracks on the playlist Grunge and their associated
artist and album:")
for track in Grunge["data"]["playlists"][0]["tracks"]:
    print(track["name"] + " by " + track["album"]["artist"]["name"] + " is
part of the album " + track["album"]["title"])

# end of GraphQL API calls
```

This is a screenshot of the results of the program.



```
(.venv) student@tuffix-vm:~/Documents/GitHub/CPSC-449-Exercise-1$ python3 GraphQL_API.py
Albums by the artist Red Hot Chili Peppers:
Blood Sugar Sex Magik
By The Way
Californication

Genres associated with the artist U2:
Rock
Pop

Names of tracks on the playlist Grunge and their associated artist and album:
Man In The Box by Alice In Chains is part of the album Facelift
Smells Like Teen Spirit by Nirvana is part of the album Nevermind
In Bloom by Nirvana is part of the album Nevermind
Come As You Are by Nirvana is part of the album Nevermind
Lithium by Nirvana is part of the album Nevermind
Drain You by Nirvana is part of the album Nevermind
On A Plain by Nirvana is part of the album Nevermind
Evenflow by Pearl Jam is part of the album Ten
Alive by Pearl Jam is part of the album Ten
Jeremy by Pearl Jam is part of the album Ten
Daughter by Pearl Jam is part of the album Vs.
Outshined by Soundgarden is part of the album A-Sides
Black Hole Sun by Soundgarden is part of the album A-Sides
Plush by Stone Temple Pilots is part of the album Core
Hunger Strike by Temple of the Dog is part of the album Temple of the Dog
(.venv) student@tuffix-vm:~/Documents/GitHub/CPSC-449-Exercise-1$
```