- Issues I encountered when setting up the environment
  - The graphql-engine service is using the wrong port mapping. It's mapping to port 80 inside the container, but Hasura typically uses port 8080.
  - The GraphQL engine's manifest/version was incorrectly set to use an invalid version (2.48)
  - The HASURA_GRAPHQL_METADATA_DATABASE_URL is pointing to a non-existent service called postgresdb.
  - I had to add the Metadata file as a volume in the compose file so that the GraphQL engine knows where to look at for metadata at the boot.
  - I was not able to connect my database initially. I ran into several errors, including not being able to see my database in the Hasura console. To resolve this, I manually added the database from the console. Then, I copied the Chinook database that I had downloaded on my host machine to the container running the Hasura engine. Finally, I ran the Docker/PostgreSQL execute command to load all the tables within the container so that the database had all the tables attached to it.

1. For running query as an admin, I had to change the id name to get the correct answer.

```
# Execute as an administrator
   query getTracks($genre: String, $limit: Int, $offset: Int) {
     track(limit: $limit, offset: $offset, where: {genre: {name: {_eq: $genre}}})
{
name
track_id
} }
```

```
   {
     "genre":"Metal",
     "limit": 5,
     "offset":50
}
```

Result

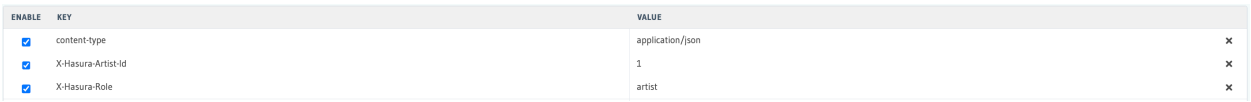
```
{
  "data": {
    "track": [
      {
        "name": "Trupets Of Jericho",
        "track_id": 190
      },
      {
        "name": "Machine Men",
        "track_id": 191
      },
      {
        "name": "The Alchemist",
        "track_id": 192
      },
      {
        "name": "Realword",
        "track_id": 193
      },
      {
        "name": "Free Speech For The Dumb",
        "track_id": 408
      }
    ]
  }
}
```


2.

Query as an artist

```
query getAlbumsAsArtist {
  album {
    title
  }
}
```

Headers

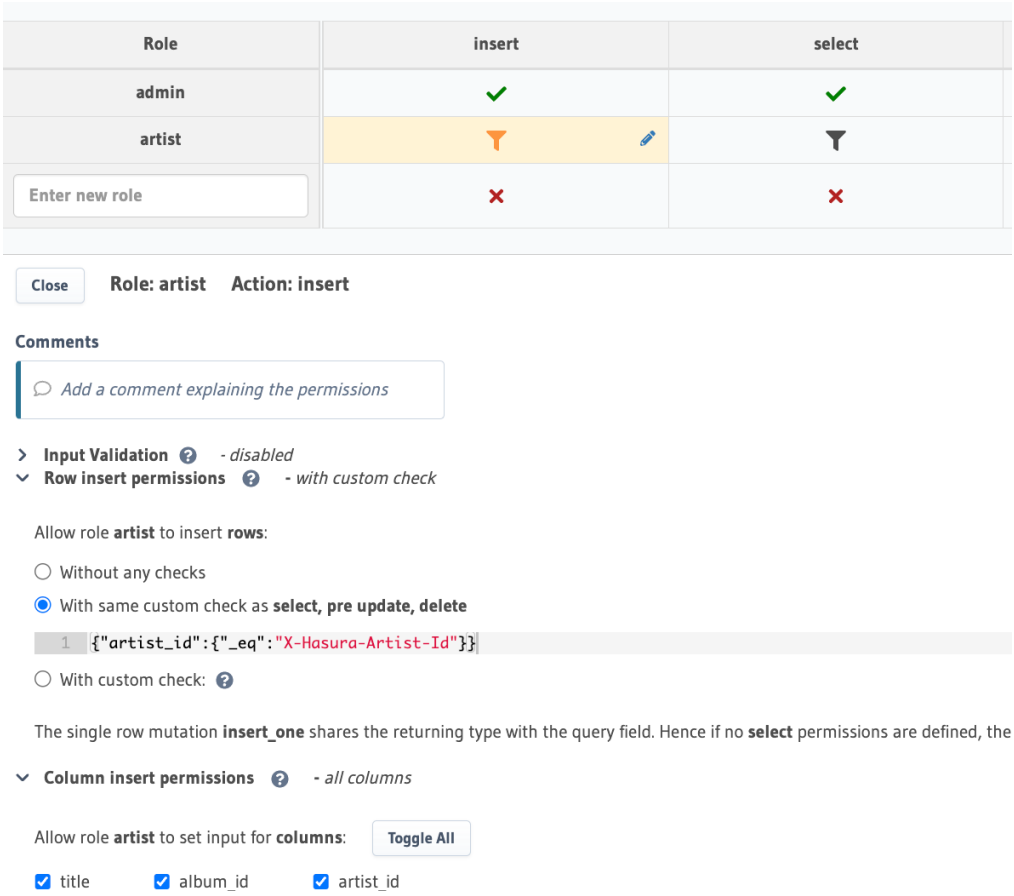


Result


```
{
  "data": {
    "album": [
      {
        "title": "For Those About To Rock We Salute You"
      },
      {
        "title": "Let There Be Rock"
      }
    ]
  }
}
```


Permissions

The custom header check. Additionally, I had to give access to the columns in order for the query to detect “title” of the album, and kept aggregation disabled.



3. After doing multiple tests on getting the aggregate results, I do not believe as an artist I can see the tracks_aggregate given the policy restriction. I would have to give a very broad level permission to an artist to track, album, artist table to get the results we want.
4. Sample complex query without caching:

```
query complexQuery1 {
  artist {
    name
    albums {
      title
      tracks_aggregate {
        aggregate {
          sum {
            unit_price
          }
        }
        nodes {
          name
          genre {
            name
          }
          milliseconds
          unit_price
        }
      }
    }
  }
}
```

a.

This query retrieves all albums by each artist, including the names, genres, durations, and prices of the tracks within those albums, and calculates the total revenue (sum of track prices) for each album.

Since I don't have Enterprise plan, I am not able to use the Cache directive. Without that, I would have to install a middleware and some type of backend server to implement caching. While I have not implemented this for exercise, I wanted to share the following steps that I would take if I were to implement this logic

- **Set Up a Proxy Server**: Deploy a middleware (e.g., a Node.js server) that will intercept GraphQL requests before they reach Hasura. This server will handle caching logic.
- **Connect to Redis**: Integrate Redis with the proxy server to store and retrieve cached responses. Use a Redis client library (e.g., redis for Node.js) to interact with the Redis cache.
- **Cache Query Results**: When a query is received, the proxy server checks Redis for a cached response. If the result is found in the cache, it is returned immediately. If not, the server forwards the query to Hasura, caches the response, and then returns it to the client.

- **Set Cache Expiry**: Implement a cache expiration strategy (e.g., TTL or time-based invalidation) to ensure that cached data is refreshed periodically and stays up-to-date.