1. Create the Django Project and App

```
django-admin startproject music
cd music
python manage.py startapp bands
```

2. Update `settings.py`

Add the 'bands' app to the `INSTALLED_APPS` list in your project's `settings.py` file:

```
INSTALLED_APPS = [
    # ... other apps
    'bands',
]
```

3. Define Models (follow indentation rules) in `bands/models.py`

```
from django.db import models

class Genre(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Member(models.Model):
    name = models.CharField(max_length=100)
    bands = models.ManyToManyField('Band',
related_name='members')

    def __str__(self):
        return self.name

class Band(models.Model):
    name = models.CharField(max_length=100)
    genre = models.ForeignKey(Genre,
on_delete=models.CASCADE)
    # members field automatically created because of
    # ManyToManyField in Member

    def __str__(self):
        return self.name
```

4. Create and Apply Migrations

```
python manage.py makemigrations bands
python manage.py migrate
```

5. Populate the Database Using the Django Shell

```
python manage.py shell
```

Then, inside the shell:

```
from bands.models import Genre, Member, Band

# Create genres
rock = Genre.objects.create(name='Rock')
pop = Genre.objects.create(name='Pop')

# Create members
john_lennon = Member.objects.create(name='John Lennon')
paul_mccartney = Member.objects.create(name='Paul
McCartney')
freddie_mercury = Member.objects.create(name='Freddie
Mercury')

# Create bands and associate members
beatles = Band.objects.create(name='The Beatles',
genre=rock)
beatles.members.add(john_lennon, paul_mccartney)

queen = Band.objects.create(name='Queen', genre=rock)
queen.members.add(freddie_mercury)

# You can add more bands and members as you like
```

To exit the Django shell, simply type `exit()` and press Enter, or the keyboard shortcut `Ctrl+D` (on Linux/macOS) or `Ctrl+Z` followed by Enter (on Windows).

To use the Django shell to query the database and print results to the terminal:

```
python manage.py shell

# Inside the shell

from bands.models import Band

all_bands = Band.objects.all()

for band in all_bands:

    print(band.name, "-", band.genre)
```

Review:

- **Models:**
  - `Genre`: Represents a music genre.
  - `Member`: Represents a band member. It has a many-to-many relationship with `Band`.
  - `Band`: Represents a band. It has a foreign key relationship with `Genre` and a many-to-many relationship with `Member`.
- **Shell:**
  - We use the Django shell to interactively create and save objects to the database.
  - `objects.create()` is used to create and save new objects.
  - `band.members.add()` is used to associate members with a band.

Optional: Create Templates

Create templates (HTML files) to display the band information you've stored in the database. Use Django's template language to loop through the data and present it in a user-friendly way.

1. **Create Views:**
   - In your `bands` app, create a `views.py` file.
   - Define a view function that queries the database for the band information and renders a template to display it.

2. **Create Templates:**
   - In your `bands` app, create a `templates` folder.
   - Inside the `templates` folder, create an HTML file (e.g., `band_list.html`).

    - Use Django's template language to loop through the band data and display it in the HTML.

3. **Configure URLs:**

    - In your project's main `urls.py` file, include the URLs for your `bands` app.
    - In your `bands` app, create a `urls.py` file and define the URL pattern for your band list view.

Example:

- `bands/views.py`

```
from django.shortcuts import render
from .models import Band

def band_list(request):
    bands = Band.objects.all()
    return render(request,
    'bands/band_list.html', {'bands': bands})
```

- `bands/templates/bands/band_list.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Bands</title>
</head>
<body>
    <h1>Bands</h1>
    <ul>
        {% for band in bands %}
            <li>{{ band.name }} - {{ band.genre }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

- `music/urls.py` (project's main urls.py)

```
from django.contrib import admin
from django.urls import path, include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('bands/',
    include('bands.urls')),  # Include the bands app's URLs
]
```

- `bands/urls.py`

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.band_list, name='band_list'),
]
```

Run the Development Server

```
python manage.py runserver
```

Now, open your web browser and go to `http://127.0.0.1:8000/bands/` to see your band list.

Optional:
To display band members along with the band information, you need to modify your view and template to include the members associated with each band.

1. Update the View (`bands/views.py`)

Prefetching the `members` related objects when you query the `Band` objects. This will improve performance by reducing the number of database queries.

```
from django.shortcuts import render
from .models import Band

def band_list(request):
    bands = Band.objects.all().prefetch_related('members')
# Prefetch members
    return render(request, 'bands/band_list.html',
{'bands': bands})
```

2. Update the Template (`bands/templates/bands/band_list.html`)

Loop through the `members` of each `band` and display their names.

```
<!DOCTYPE html>
<html>
<head>
    <title>Bands</title>
</head>
<body>
    <h1>Bands</h1>
    <ul>
        {% for band in bands %}
            <li>
                {{ band.name }} - {{ band.genre }}
                <ul>
                    {% for member in band.members.all %}
                        <li>{{ member.name }}</li>
                    {% endfor %}
                </ul>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Review:

- In the view, we use `prefetch_related('members')` to fetch the band members along with the bands in a single query, optimizing database access. Without `prefetch_related`, Django might execute a separate database query for each related object when you access it. This can lead to a large number of queries,
- In the template, we use an inner loop (`{% for member in band.members.all %}`) to iterate over the members of each band and display their names.

Now, when you run the development server and visit `http://127.0.0.1:8000/bands/`, you should see the band names, genres, and their respective members listed.