Chordity Database Management Information

Table of Contents

This document will be broken down into the following categories:

- Industry and Application Overview

- Business Rules

- Entity Relationship Diagram

- Entity Model Information

Industry and Application Overview

Chordity is a web-based chat platform focused around connecting musicians and collaborating together to inspire and kickstart new tracks! Because of the target audience of musicians and music producers, and the intended use-cases and business rules of the application (which will be covered more in the next section of the document), this application is designed for the social media industry, with an emphasis or niche towards the music industry as well. This application was designed to be similar to platforms like Reddit (https://www.reddit.com/) or Discord (https://discord.com/), which are also social media chat platforms where users can interact with each other through messages or posts.

In the social media industry, one crucial problem companies face regarding database management is understanding optimization. Social media platforms often store large amounts of data, which is constantly expanding at a more rapid rate than ever, and companies are always looking for new and innovative solutions (whether cloud based or on-prem) to optimize their database. This can include ways to normalize their database to ensure only useful data is stored and that duplicated data is minimized. This is important for these companies, since users of these applications anticipate instant access to their data once they open the app. Because of this need for instant feedback and response times, speed and

efficiency are very important needs to the database developers, so the normalization or management solutions they use need to allow quick access to the data inside of the database.

Business Rules

Anyone on the application can view chat rooms or messages on Chordity, but in order to post a comment or make a new chat room, a user needs to login with an existing user account or create a new one (similar to Reddit). The user is required to enter their name, username, email, and password. They are also required to confirm their password by entering it again and agreeing to the terms and conditions, but these fields are not stored specifically in Chordity's database. The userID (unique identifier and primary key for the user's entity), superuserStatus, avatar, and bio fields are all populated with default values for the user, and avatar and bio can be changed by the user after their profile has been created (by manually navigating to their profile page or by selecting the settings icon in the navbar).

As of now, there is not a separate administrator role or entity, as all users created through the login page have their superuser status set to false, but current superusers can change this status for any user using the Django admin console. There are no intentions to refactor or restructure this method at this time. At this time, a user also can not update their username, name, or email through the application, but a request can be made to a superuser on the platform, since they can also change these fields through the Django admin console.

Once logged in, a user now has the functionality to create a new room (chat room) or add a comment to an existing one.

To create a new room, a user needs to enter the topic (either new or existing) that the room is about (ie. - Music Theory, Song Ideas, General Discussion, etc.), the name of the room, and the room's description. Once the room is created, the roomID will be automatically filled and the user that created the room is assigned to be the host of that room. These fields (except roomID) can be updated or the room can be deleted by the host after the room has been created. Deleting a room will also delete all of the comments inside of that room as well.

A user can also create a message inside of any existing room, and only the user that created their comment can delete it as well (with the exception of

superusers). Once a user posts a message in a particular room, they are also listed as a participant in that room.

Both room and message entities also have datetime fields for when the entity was created (roomCreated and messageCreated) and last updated (roomUpdated and roomUpdated).

Entity Relationship Diagram
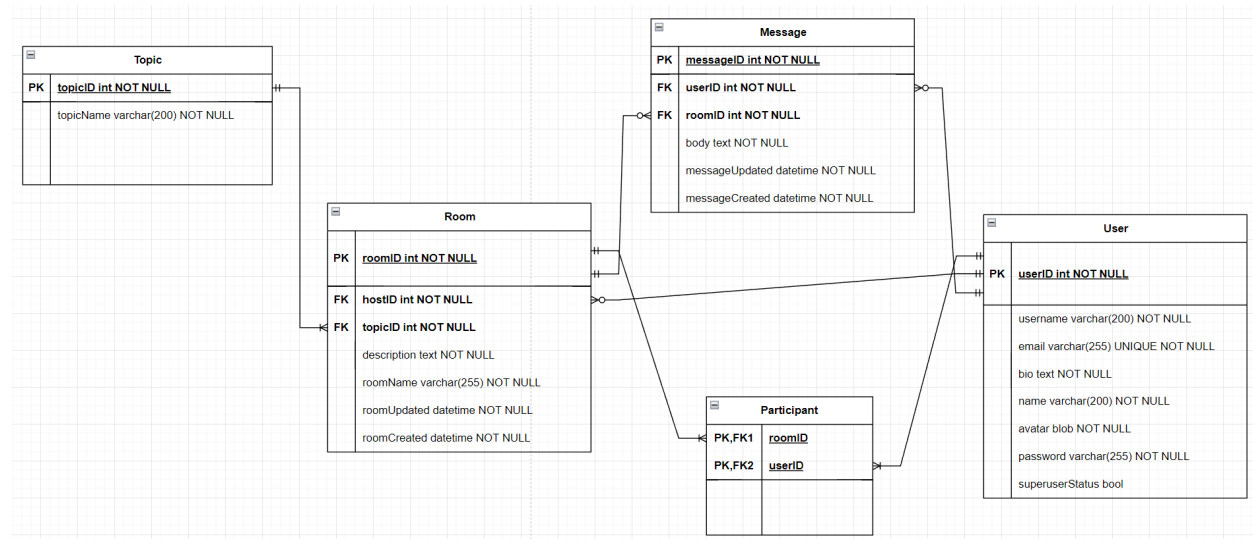
Here are the relationships described in the diagram and the business rules above:

- A user can create zero to many rooms, and a room can have one and only one host.

- A user can create zero to many messages, and a message can have one and only one user.

- Many users can participate in many rooms, and many rooms can have many participants.

- Each message belongs to one and only one room, and a room can have zero to many messages.

- A room can have one and only one topic, while a topic can be tied with one to many rooms.

Entity Model Information

While Chordity's database is created using SQLite (which is Django's default relational database system), SQL queries or the MySQL software was not used to directly create, seed, or update this database. The models and commands used were instead handled by Django's default modeling tools, and a custom user model was also created as opposed to using Django's default user model.

The code that we used to specify and declare the models can be found here at this link (https://github.com/rexherndon/chordity/blob/main/chordity/base/models.py) or pasted down below (code is from 4/30/2024, and the newest versions can be found at that GitHub link):

```python
from django.db import models
from django.contrib.auth.models import AbstractUser

# Create your models here.

class User(AbstractUser):
    name = models.CharField(max_length=200, null=True)
    email = models.EmailField(unique=True, null=True)
    bio = models.TextField(null=True)

    # path for default image is at static\images
    avatar = models.ImageField(null=True, default="avatar.svg")

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

class Topic(models.Model):
    name = models.CharField(max_length=200)

    def __str__(self):
        return self.name

class Room(models.Model):
```

```python
    host = models.ForeignKey(User, on_delete=models.SET_NULL, nu
    topic = models.ForeignKey(Topic, on_delete=models.SET_NULL,
    name = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    participants = models.ManyToManyField(User, related_name='pa
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-updated', '-created']


    def __str__(self):
        return self.name

class Message(models.Model):

    user = models.ForeignKey(User, on_delete=models.CASCADE)
    # w/ CASCADE, if a room gets deleted, all the children (ie.
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    body = models.TextField()
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-updated', '-created']

    # only returns first 50 characters when initially viewing
    def __str__(self):
        return self.body[0:50]

    # unneeded boilerplate?
    # class Meta:
    #     verbose_name = _("")
    #     verbose_name_plural = _("s")


    # unneeded boilerplate?
```
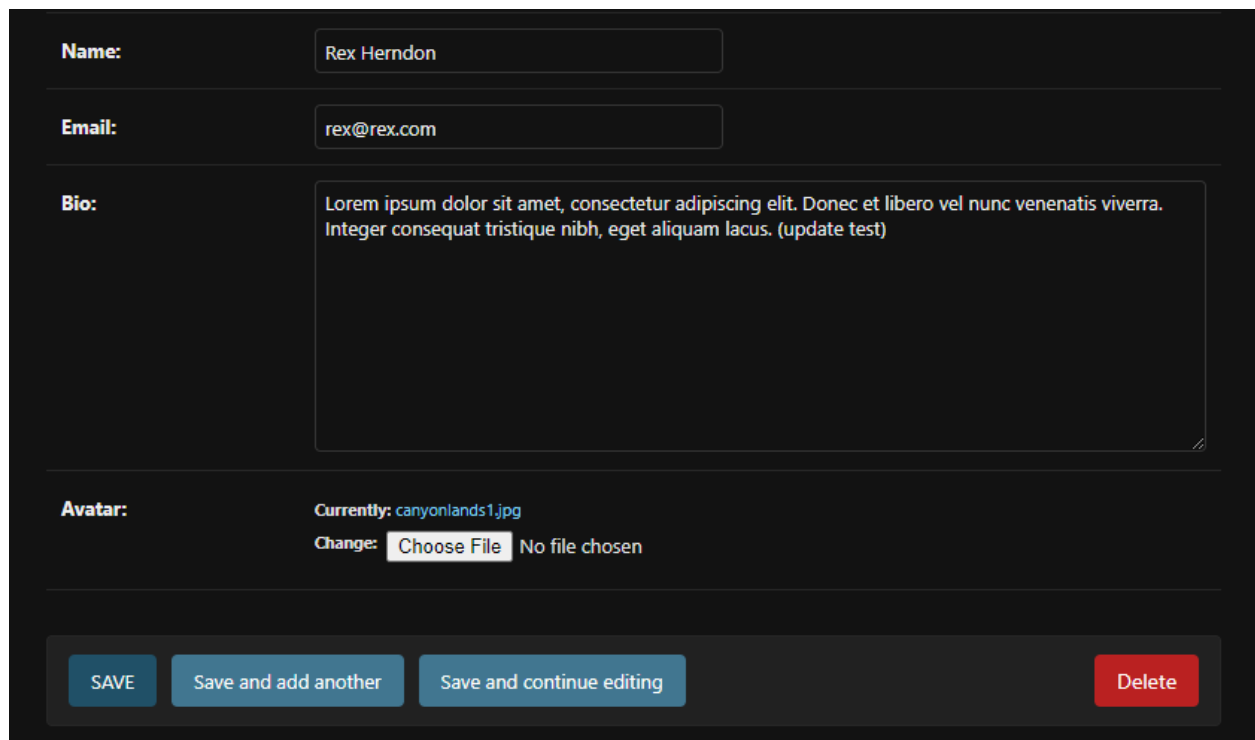
```
# def get_absolute_url(self):
#     return reverse("_detail", kwargs={"pk": self.pk})
```

The actual SQLite database can also be found here at this link (https://github.com/rexherndon/chordity/blob/main/chordity/db.sqlite3).

Snippets of example records from the Django admin console are also shown down below:

Snippet of example user record

Snippet of example topic record

Music Theory

Name: Music Theory

SAVE   Save and add another   Save and continue editing   Delete

Snippet of example room record

major or minor scale?

Host: rex@rex.com

Topic: Music Theory

Name: major or minor scale?

Description: what do you prefer when you produce?

Participants:
rex@rex.com
tim@tim.com
john@john.com

Hold down "Control", or "Command" on a Mac, to select more than one.

SAVE   Save and add another   Save and continue editing   Delete

Snippet of example message record