Assessment: Databases

Download code <../databases.zip>

Warning: Assessments

Remember, assessments are meant to be completed **by you**, not as a shared exercise with friends or other members of your cohort.

All code submitted should be **written by you**. If you incorporate code from elsewhere, it must be clearly specified.

Please do not put your assessment on GitHub.

Part 1 - Conceptual

Answer the following questions inside the ***conceptual.md*** file.

Part 2 - Practice SQL Queries

Answer the following questions inside the ***queries.md*** file.

Part 3 - Playlists!

It's time to build a Flask application! This application allows a user to create songs and playlists and add a song to a playlist. Your data model will allow for many songs to be part of many different playlists and allow for many different playlists to include many different songs.

This application must make use of server-side validation using WTForms and SQLAlchemy.

Warning: JavaScript

This app should not use any JavaScript.

Step One - Getting setup

Make sure you download the starter code and run the following steps in Terminal:

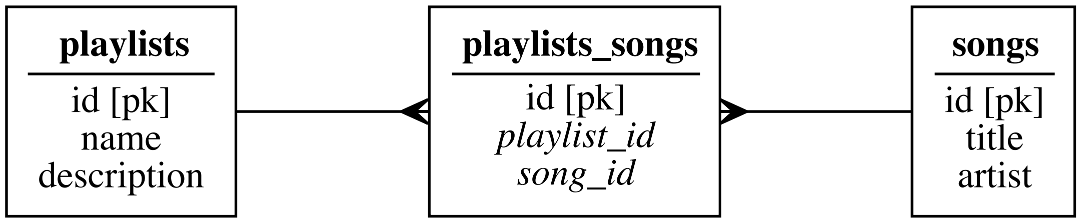
```
$ python3 -m venv venv
$ source venv/bin/activate
```

```
(venv) $ pip install -r requirements.txt
(venv) $ createdb playlist-app
```

If you try to run the app locally it will not work because the ***models.py*** needs to be implemented.

Step Two - Creating the models

In the starter code, you will notice that the ***models.py*** file is almost empty. Add the necessary models for the following schema:



You **must** use these column names, the data type you choose for each column is up to you.

Make sure you also create your tables using ***db.create_all()*** in ***ipython***

Step Three - Creating Songs and Playlists

Now that you have your models working, it's time to add the necessary code to create new songs and playlists.

In the ***app.py*** file you will find the view functions that need to be written to successfully create a new song and a new playlist.

You **must** display a form using WTForms which means you will have to modify the the ***PlaylistForm*** and ***SongForm*** classes in the ***forms.py*** file.

Make sure that your forms have appropriate validations and that you are using the ***.validate_on_submit()*** method to ensure that inputs are valid.

Step Four - Adding a song to a Playlist

For the view function to add a specific playlist you can find the code below, or try it out on your own!

Make sure you add the necessary code in the template to display the form.

Adding a song to a playlist

```
@app.route("/playlists/<int:playlist_id>/add-song", methods=["GET", "POST"])
def add_song_to_playlist(playlist_id):
```

```
"""Add a playlist and redirect to list."""

  playlist = Playlist.query.get_or_404(playlist_id)
  form = NewSongForPlaylistForm()

  # Restrict form to songs not already on this playlist

  curr_on_playlist = [s.id for s in playlist.songs]
  form.song.choices = (db.session.query(Song.id, Song.title)
                      .filter(Song.id.notin_(curr_on_playlist))
                      .all())

  if form.validate_on_submit():

      # This is one way you could do this ...
      playlist_song = PlaylistSong(song_id=form.song.data,
                                  playlist_id=playlist_id)
      db.session.add(playlist_song)

      # Here's another way you could that is slightly more ORM-ish:
      #
      # song = Song.query.get(form.song.data)
      # playlist.songs.append(song)

      # Either way, you have to commit:
      db.session.commit()

      return redirect(f"/playlists/{playlist_id}")

  return render_template("add_song_to_playlist.html",
                         playlist=playlist,
                         form=form)
```

Solution

You can view Our Solution <solution/index.html> (password required)