Netzwerke und Internet II - C

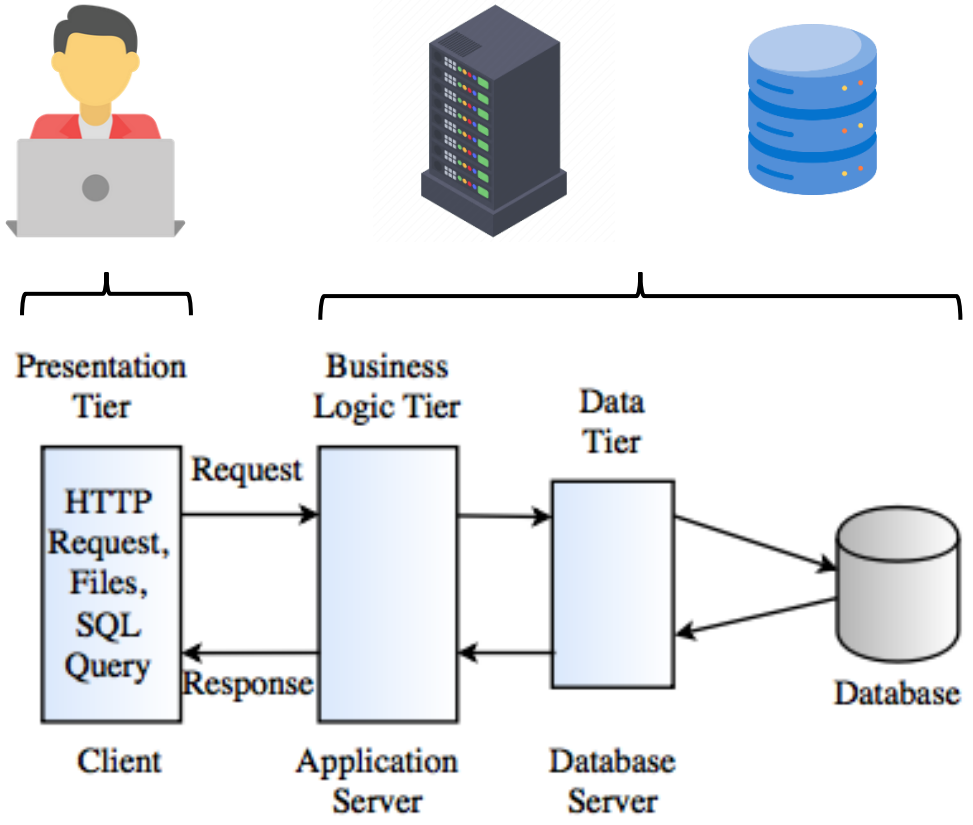
Dr. Philipp Hurni, Kantonsschule Sursee

Agenda

- Datenbanken
  - Architekturen und Typen
  - Relationale Systeme
  - File-basiert vs. Prozess-basiert
- Datenbank in SQ Lite Studio
  - Repetition SQL Befehle
- Einfache Applikation mit Datenbank
  - Flask und SQL kombinieren / Verbinden
  - Übungen

Die klassische 3-Tier-Architektur

- **Tier:** Schicht
- **Presentation Tier:** Browser des Clients (auch Client Tier genannt)
- **Business Logic Tier/Application Tier:** Webserver-Applikation auf Webserver-Computer. Bei uns: HTML-Files auf Apache Webserver
- **Data Tier:** Datenbankserver mit Zugriff auf Datenbank. Das haben wir bisher noch nicht gemacht



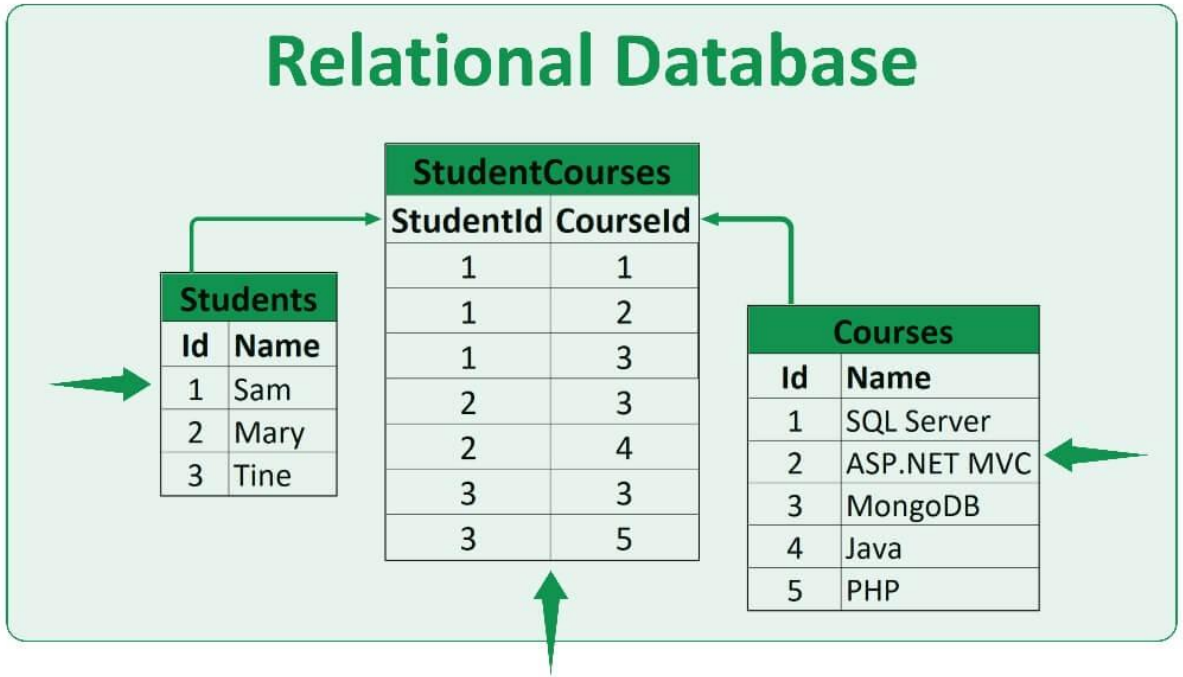
3 - Tier Architecture

Datenbank – Architekturen und Typen

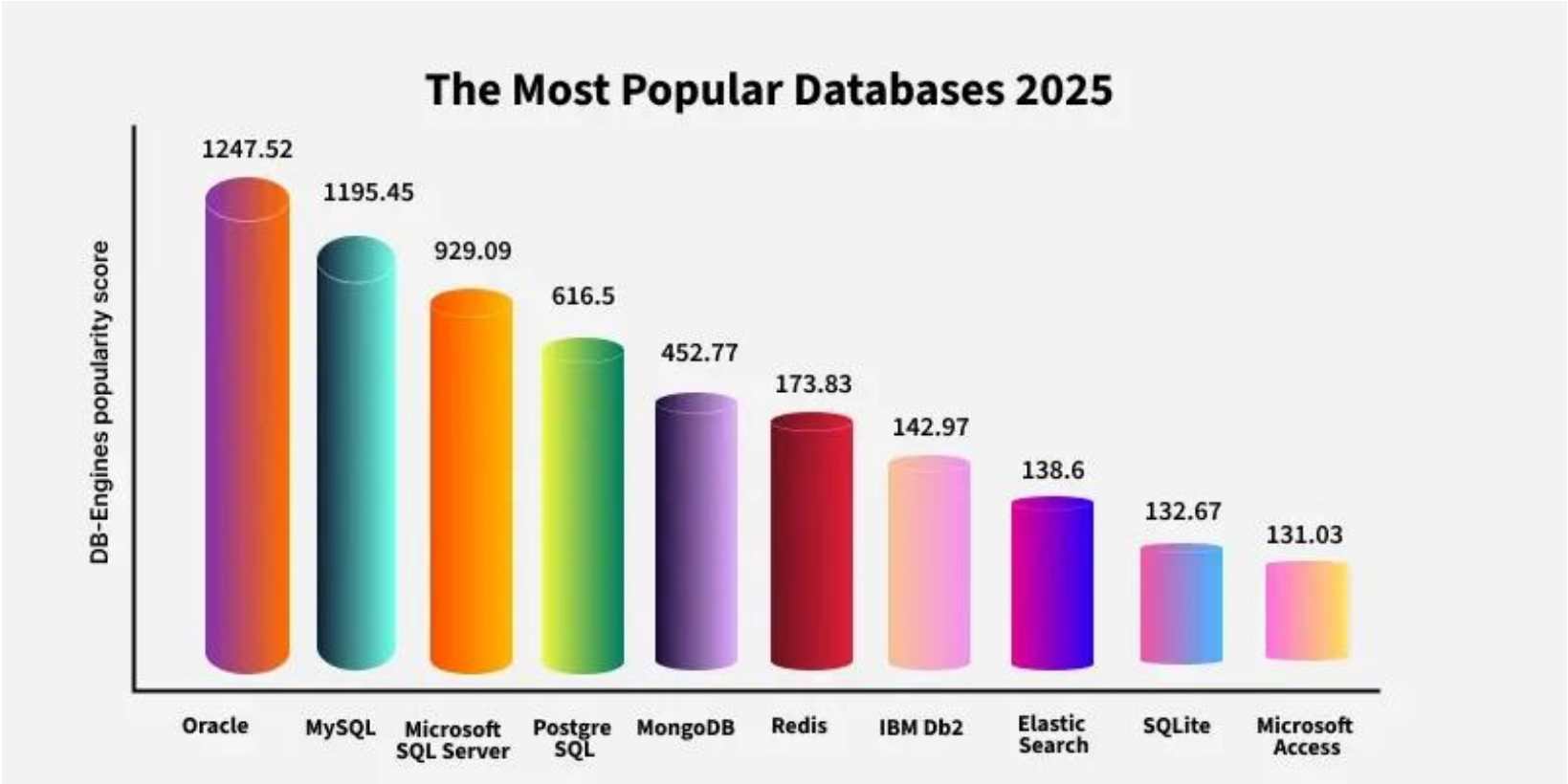
- Relationale Datenbanken
  - Tabellen
  - Attribute
  - Relationen
  - Kardinalitäten

- Definitionssprache
  CREATE TABLE…

- Abfragesprache
  SELECT * FROM Students…



Datenbank – Systeme nach Popularität



Datenbanken – Upper End

ORACLE®

EXADATA

Datenbank – Middle Tier

Datenbank – Lower end

Flatbase

Flatbase is a flat file database written in PHP which aims to be:

- Lightweight
- Very easy to install, with minimal/no configuration
- Simple intuitive API
- Suitable for small data sets, low-load applications, and testing/prototyping

PicoDb

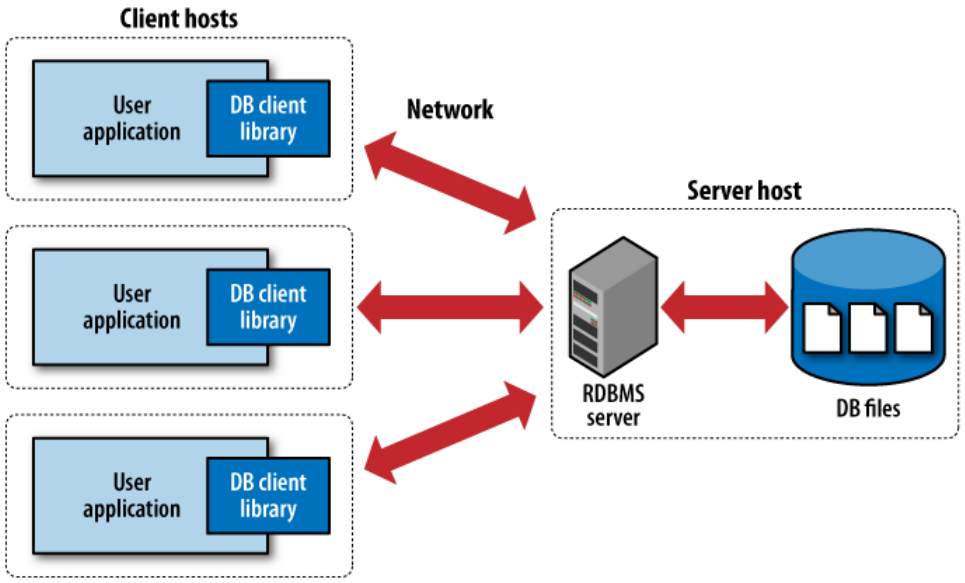
PicoDb is a minimalist database query builder for PHP.

build unknown

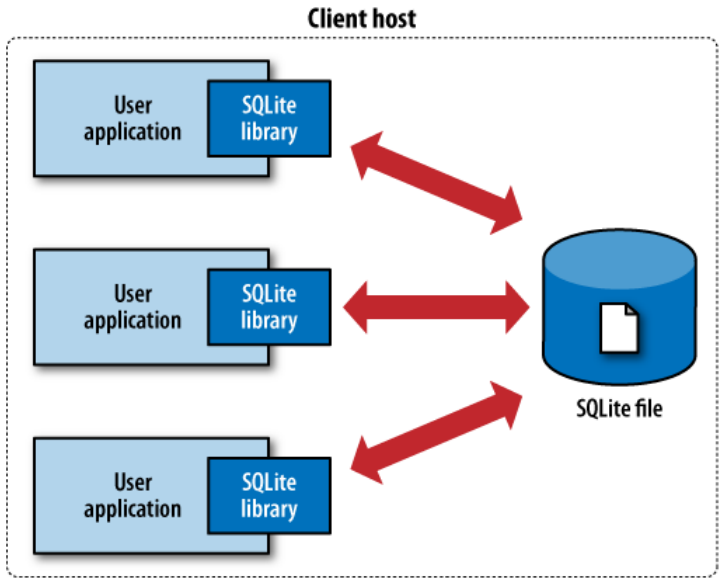
Datenbank – SQLite

- (Relationale) Datenbank ist gespeichert in einer einzigen Datei xyz.db
- «Schnell etwas aufbauen» - ohne grossen Overhead – dafür ist SQLite genau richtig
- SQLiteStudio – ein kostenfreies Tool um SQLite Datenbanken zu verwalten

Datenbank – SQLite



(a) Traditional client-server architecture



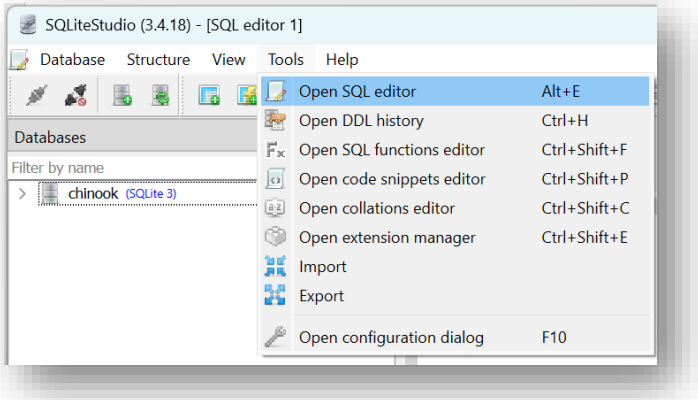
(b) SQLite serverless architecture

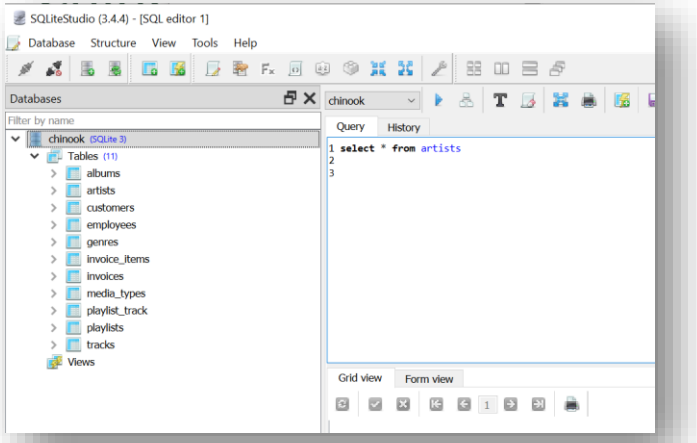
Chinook Database

- Die chinook.db ist auf Teams gespeichert
- Öffne die Datei mit SQLite Studio und gib ein paar Queries ein, z.B.

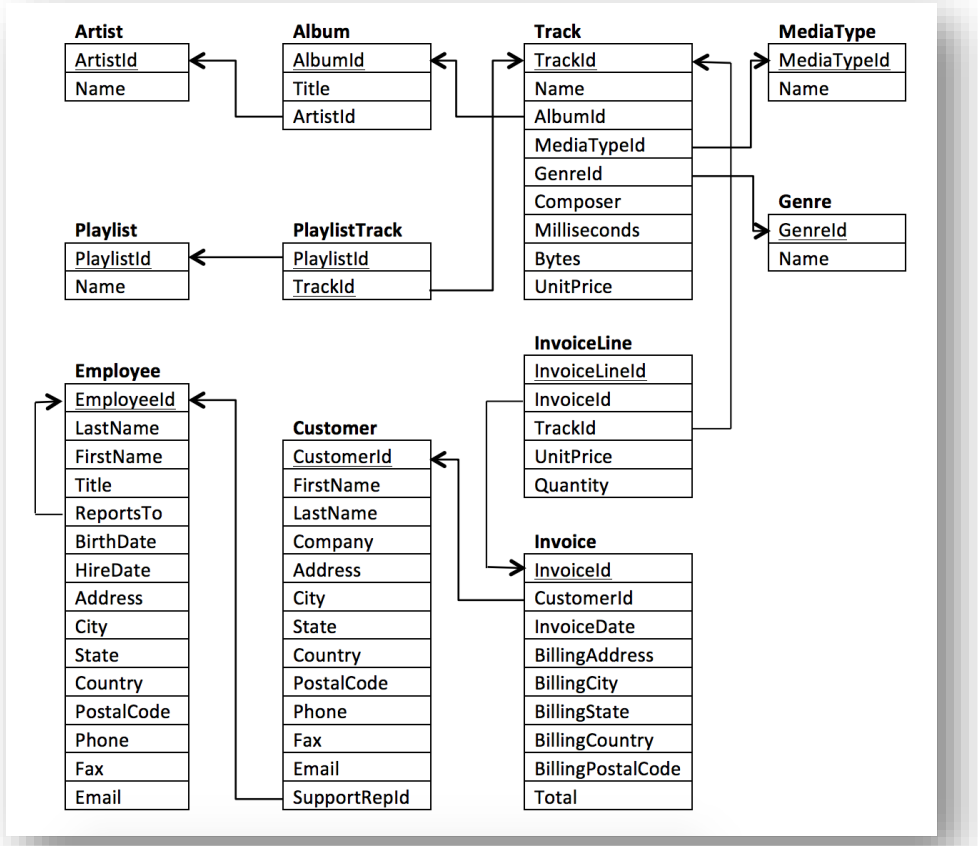
  SELECT * FROM artists

- Öffne den SQL Editor mit
  Tools -> Open SQL Editor (ALT-E)





Chinook Schema



Aufgaben SQL (Repetition) A - Einfach

- Gib alle "Artists" aus (=Künstler)
- Gib alle Künstler aus mit "The" im Namen (Hinweis: benutze LIKE)
- Finde alle Artisten, welche mindestens ein Lied mit dem Genre "Alternative & Punk" haben:

Aufgaben SQL (Repetition) A - Einfach

- Gib alle "Artists" aus (=Künstler)

```
SELECT * FROM artists
```

- Gib alle Künstler aus mit "The" im Namen (Hinweis: LIKE)

```
SELECT * FROM artists Where Name LIKE "%The%"
```

Aufgaben SQL (Repetition) B - Fortgeschritten

- Finde alle Liedernamen mit Genre "Alternative & Punk" (GenreId = 4):
- Du willst alle Lieder aller Künstler auflisten. Wie machst du das?
- Finde alle Artisten, welche mindestens ein Lied mit dem Genre "Alternative & Punk" haben:

Lösungen zu Aufgaben SQL B - Fortgeschritten

- Finde alle Liedernamen mit dem Genre «Alternative & Punk»:

  SELECT Name, AlbumId FROM Tracks WHERE GenreId = 4

- Du willst alle Lieder aller Künstler darstellen. Wie machst du das?

  SELECT Artists.name, Albums.Title, Tracks.Name FROM Artists, Albums, Tracks WHERE Artists.ArtistId = Albums.ArtistId and Albums.AlbumId = Tracks.AlbumId

- Finde alle Artisten, welche mindestens ein Lied mit dem Genre "Alternative & Punk" haben:

  SELECT DISTINCT Artists.Name FROM Artists, Albums, Tracks WHERE Artists.ArtistId = Albums.ArtistId AND Albums.AlbumId = Tracks.AlbumId AND Tracks.GenreId = 4

Mit Python aus einer Datenbank lesen

Datenbank in Python Programm (searcher.py)

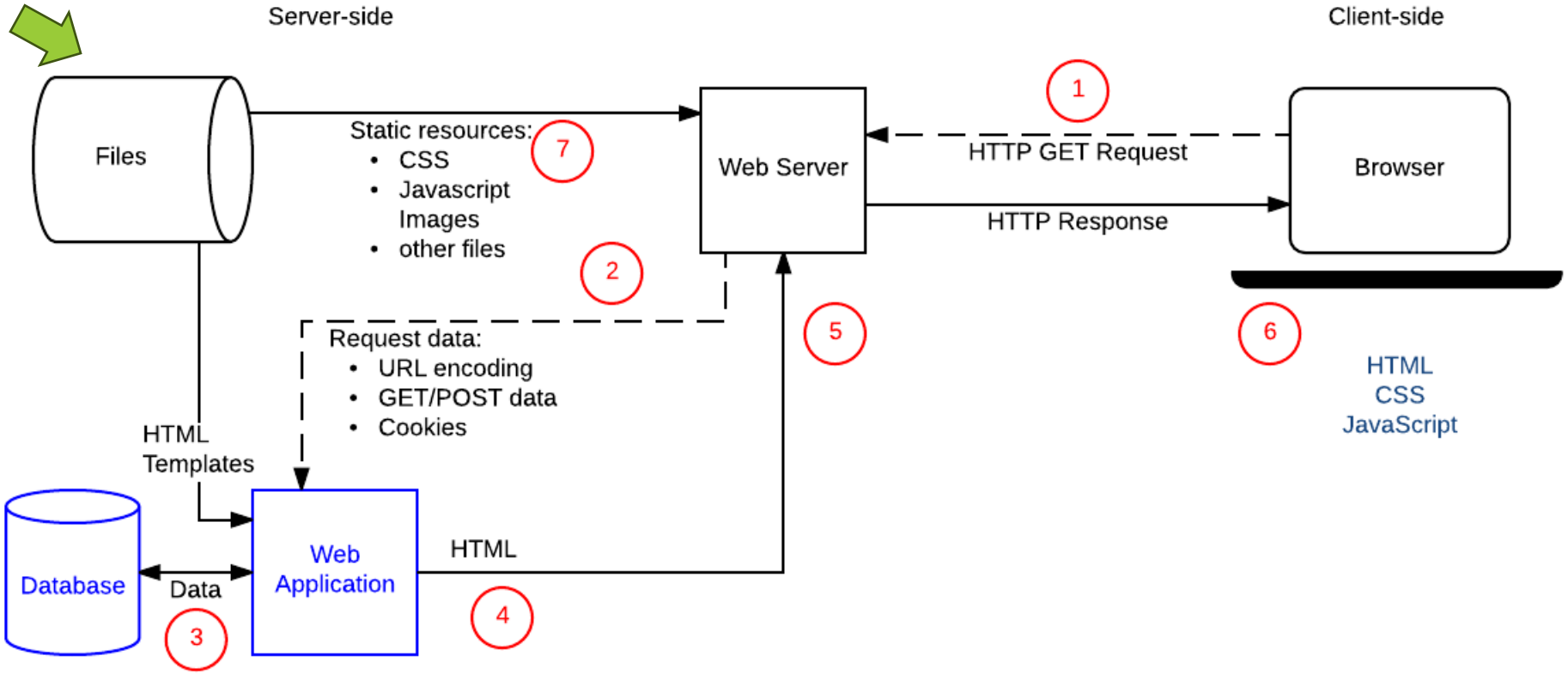
- Benutzereingabe: Fragt den Anwender per Konsole nach einem Musikgenre (z. B. "Rock").
- Datenbankverbindung: Öffnet die Datei chinook.db mittels sqlite3-Bibliothek
- Tabellenverknüpfung: Verbindet vier Tabellen (tracks, genres, albums, artists) über deren Ids
- Filterung: Sucht nach Genres, die den eingegebenen Text enthalten, wobei Gross-/Kleinschreibung durch LOWER ignoriert wird.

```python
import sqlite3

genre = input("Welches Genre suchst du? ")

conn = sqlite3.connect('chinook.db')
conn.row_factory = sqlite3.Row
cursor = conn.cursor()

query  = "SELECT artists.Name AS artist, tracks.Name AS title "
query += "FROM tracks, genres, albums, artists "
query += "WHERE tracks.GenreId = genres.GenreId "
query += "AND tracks.AlbumId = albums.AlbumId "
query += "AND albums.ArtistId = artists.ArtistId "
query += "AND LOWER(genres.Name) LIKE LOWER('%" + genre + "%') "

print(query)

# Ausführung
results = conn.execute(query).fetchall()
conn.close()

# Ergebnisse ausgeben
if results:
    print(f"Gefundene Songs für Genre '{genre}':\n")
    for row in results:
        print(f"{row['artist']} - {row['title']}")
else:
    print(f"Keine Einträge für '{genre}' gefunden.")

conn.close()
```

Datenbank in Webapplikation einbauen

Datenbank in Webapplikation

templates/index.html
static/styles.css



chinook.db

app.py

Erweiterbarkeit

app.py File (Flask) – die Logik ist einfach

```python
from flask import Flask, render_template, request
import sqlite3

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect('chinook.db')
    conn.row_factory = sqlite3.Row
    return conn

@app.route('/', methods=['GET', 'POST'])
def index():
    results = []
    if request.method == 'POST':
        # wir erhalten das Genre aus dem Formular
        genre = request.form.get('genre', '').strip()
        # prüfen, ob das Genre nicht leer ist
        if genre:
            # wir erhalten eine Verbindung zur Datenbank
            conn = get_db_connection()

            query  = "SELECT artists.Name AS artist, tracks.Name AS title "
            query += "FROM tracks, genres, albums, artists "
            query += "WHERE tracks.GenreId = genres.GenreId "
            query += "AND tracks.AlbumId = albums.AlbumId "
            query += "AND albums.ArtistId = artists.ArtistId "
            query += "AND LOWER(genres.Name) LIKE LOWER('%" + genre + "%')"

            results = conn.execute(query).fetchall()
            conn.close()
    return render_template('index.html', results=results)

if __name__ == '__main__':
    app.run(debug=True, port=8086, use_reloader=False)

```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Music Genre Search</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>

    <div class="search-box">
        <h2>Search Music by Genre</h2>
        <form method="POST">
            <input type="text" name="genre" placeholder="Enter genre (e.g., Rock, Pop, Jazz)"
            <button type="submit">Search</button>
        </form>

        {% if results %}
        <div class="results">
            <h3>Results for "{{ request.form.genre }}":</h3>
            <ul>
                {% for result in results %}
                <li>{{ result['artist'] }} - {{ result['title'] }}</li>
                {% endfor %}
            </ul>
        </div>
        {% endif %}
    </div>

</body>
</html>
```

Aufgaben:

- A) Falls es keine Songs zu diesem Genre findet, soll im index.html eine Meldung kommen

Hinweis: nur Jinja-Skript anpassen

- B) Die Suche soll nicht nur in den Genres suchen, sondern auch in den Artists (Interpreten)

Hinweis: Query in app.py anpassen

A) Nur Änderung an index.html nötig

```
<!DOCTYPE html>
<html>
<head>
    <title>Music Genre Search</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>

    <div class="search-box">
        <h2>Search Music by Genre</h2>
        <form method="POST">
            <input type="text" name="genre" placeholder="Enter genre (e.g., Rock, Pop, Jazz)"
            <button type="submit">Search</button>
        </form>

        {% if results %}
        <div class="results">
            <h3>Results for "{{ request.form.genre }}":</h3>
            <ul>
                {% for result in results %}
                <li>{{ result['artist'] }} - {{ result['title'] }}</li>
                {% endfor %}
            </ul>
        </div>
        {% else %}
            <h3>No results found for "{{ request.form.genre }}":</h3>
        {% endif %}
    </div>

</body>
</html>
```

B) Nur Änderung an Query in app.py nötig

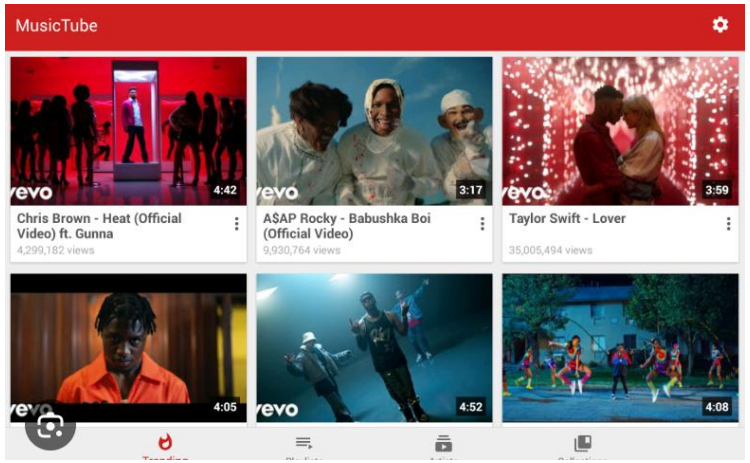
```python
@app.route('/', methods=['GET', 'POST'])
def index():
    results = []
    if request.method == 'POST':
        # wir erhalten das Genre aus dem Formular
        genre = request.form.get('genre', '').strip()
        # prüfen, ob das Genre nicht leer ist
        if genre:
            # wir erhalten eine Verbindung zur Datenbank
            conn = get_db_connection()

            query  = "SELECT artists.Name AS artist, tracks.Name AS title "
            query += "FROM tracks, genres, albums, artists "
            query += "WHERE tracks.GenreId = genres.GenreId "
            query += "AND tracks.AlbumId = albums.AlbumId "
            query += "AND albums.ArtistId = artists.ArtistId "
            query += "AND (LOWER(genres.Name) LIKE LOWER('%" + genre + "%') "
            query += "OR LOWER(artists.Name) LIKE LOWER('%" + genre + "%'))"

            results = conn.execute(query).fetchall()
            conn.close()
    return render_template('index.html', results=results)

if __name__ == '__main__':
    app.run(debug=True, port=8086, use_reloader=False)
```

Such-Link einbauen für jedes Lied

Links einbauen mit Parameter

Wir ergänzen jedes Lied um einen Link zur Google Suche nach dem Youtube Video des Liedes

```
<a href="https://www.google.com/search?q={{(result['artist']+' '+result['title'])|urlencode}}">

Find on Google

</a>
```



Zeige Lyrics an!

Auf Teams ist lyrics.py. Dort wird für einen Artist und ein Song die Lyrics gesucht.

Wir bauen das in app.py ein – falls jemand die Lyrics eines Songs lesen will, soll dieser gesucht werden und angezeigt werden

```python
import requests
from bs4 import BeautifulSoup
import urllib3

# This line suppresses the InsecureRequestWarning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

def scrape_azlyrics(artist, song):
    # Format names: remove spaces and non-alphanumeric characters
    artist = "".join(filter(str.isalnum, artist.lower()))
    song = "".join(filter(str.isalnum, song.lower()))

    url = f"https://www.azlyrics.com/lyrics/{artist}/{song}.html"

    # Using a User-Agent is highly recommended to avoid 403 Forbidden errors
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
    }

    try:
        # verify=False ignores SSL certificate errors
        response = requests.get(url, headers=headers, verify=False)

        if response.status_code != 200:
            return f"Error: Received status code {response.status_code}. Check artist/song spelling."

        soup = BeautifulSoup(response.text, 'html.parser')
```

Nochmals:
Was passiert eigentlich wann? Und wo?

Wie ein Browser eine Webseite lädt und aufbaut

- **Der "Bauplan":** Wenn man eine URL eintippt, lädt der Browser als allererstes die **HTML-Datei** herunter. Diese Datei ist rein technischer Text – quasi der Bauplan oder die "Einkaufsliste" der Seite.
- **Das Scannen (Parsing):** Der Browser liest diesen Text von oben nach unten durch. Sobald er auf ein Tag wie `<img src="bild.jpg">` stösst, merkt er, dass dort noch etwas fehlt
- **Zusätzliche Anfragen (Requests):** Für jedes Bild, jedes Video und jedes Design-Element (CSS) muss der Browser eine neue, separate HTTP-Anfrage an den Server schicken, welcher diesen Content bereitstellt. Das kann ein anderer Server sein als der ursprüngliche!
- **Asynchrones Laden:** Während die Bilder noch geladen werden, zeigt der Browser oft schon den Text an. Deshalb springt das Layout manchmal noch nachträglich hin und her, wenn ein grosses Bild plötzlich Platz einnimmt.

HTTP Requests bei externen Quellen