</talentlabs>

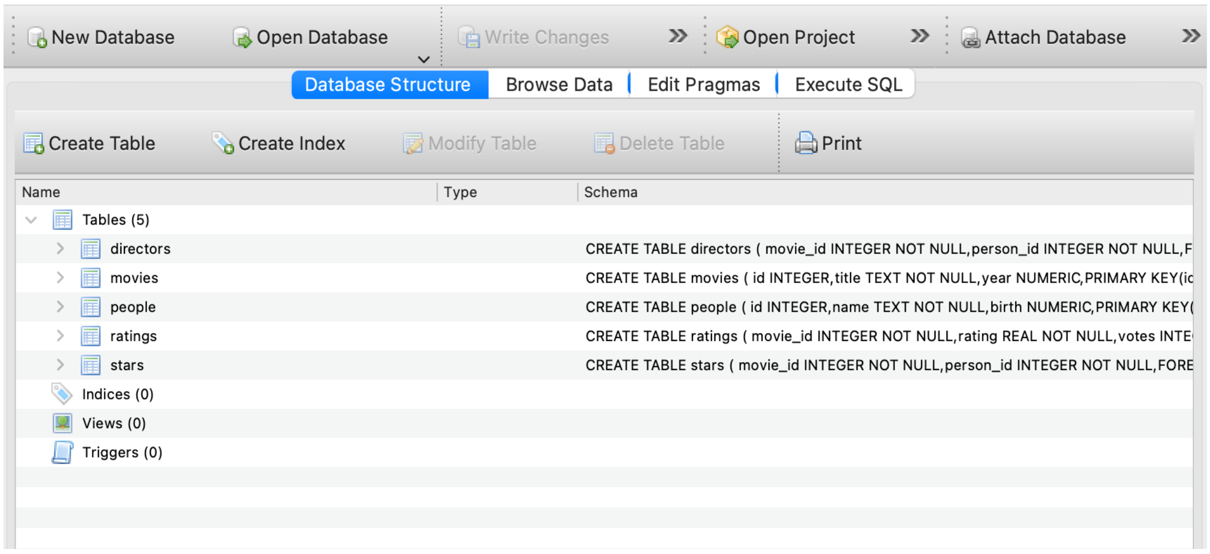
Lab Assignment 6 - Complex SELECT statements

Instructions

1. Answer the below question in the boxes.
2. Please submit the assignment through TalentLabs Learning System.

Open the Movies database

Follow the step illustrated in Chapter 3 to open the Movies database using DB Browser for SQLite. You should see 5 tables in the database.



Name	Type	Schema
Tables (5)		
directors		CREATE TABLE directors (movie_id INTEGER NOT NULL,person_id INTEGER NOT NULL,F
movies		CREATE TABLE movies (id INTEGER,title TEXT NOT NULL,year NUMERIC,PRIMARY KEY(ic
people		CREATE TABLE people (id INTEGER,name TEXT NOT NULL,birth NUMERIC,PRIMARY KEY(
ratings		CREATE TABLE ratings (movie_id INTEGER NOT NULL,rating REAL NOT NULL,votes INTE
stars		CREATE TABLE stars (movie_id INTEGER NOT NULL,person_id INTEGER NOT NULL,FORE
Indices (0)		
Views (0)		
Triggers (0)		

Understanding the database

1. Study the table schema and the data in the “people” and “directors” table and describe the relation between the tables “people” and “directors”

The relationship between the “people” and “directors” tables can be understood as one-to-many. The detailed breakdown is as follows:

Table Descriptions:

- **People Table**:
  - **Columns**:
    i. **id:** A unique identifier for each person.

ii. **name:** The name of the person.
iii. **birth:** The birthdate of the person.

➢ **Directors Table**:
- **Columns**:
  i. **movie_id:** The unique identifier for a movie.
  ii. **person_id:** The identifier linking to a person in the "people" table.

<u>Relationship:</u>

- **Foreign Key**: The `person_id` in the "directors" table acts as a foreign key that references the `id` column in the "people" table. This indicates that each entry in the "directors" table is linked to a specific individual in the "people" table.

- **One-to-Many**: Each individual (director) can be linked to multiple movies; therefore, a single id in the "people" table may correspond to several entries in the "directors" table, signifying that the same person has directed multiple movies. Conversely, each entry in the "directors" table pertains to only one individual from the "people" table.

- **Example Scenario**: If a director named "John Doe" has an id of 1 in the "people" table, then there could be several rows in the "directors" table with person_id set to 1, each linking to different movie_ids that John Doe has directed. Conversely, the "people" table will contain all the directors along with their birth details, providing a complete profile for each individual.

2. Study the table schema and the data in the "movies" and "directors" table and describe the relation between the tables "movies" and "directors"

The relationship between the "movies" and "directors" tables can be characterized as one-to-many. The detailed breakdown is as follows:

<u>Table Descriptions:</u>

➢ **Movies Table**:
- **Columns**:
  i. id: A unique identifier for each movie.
  ii. title: The title of the movie.
  iii. year: The release year of the movie.

➢ **Directors Table**:
- **Columns**:
  i. movie_id: A foreign key that links to a movie in the "movies" table.

ii. person_id: A foreign key that references a person in the “people” table.

Relationship:

- **Foreign Key**: The movie_id in the “directors” table serves as a foreign key that references the id column in the “movies” table. This means that each entry in the “directors” table is associated with a specific movie in the “movies” table.

- **One-to-Many**: Each movie can have multiple directors. Thus, a single id in the “movies” table may correspond to several entries in the “directors” table, indicating that a movie may have been directed by multiple individuals. Conversely, each entry in the “directors” table relates to only one specific movie from the “movies” table.

- **Example Scenario**: If a movie titled "Epic Adventure" has an id of 1 in the “movies” table, there could be multiple rows in the “directors” table with movie_id set to 1, each linked to different person_ids representing the directors of that film. This structure allows for flexible representation of collaborations among directors for a single movie while maintaining a clear connection to the movie's details.

Query Exercises

1. Write a SQL query to obtain the movie_id who is directed by “Joris Ivens” without using WITH keyword
   Expected Output: a table with a single column for the movie_id of the director’s movie.

```
SELECT movie_id
FROM directors
WHERE person_id IN (
    SELECT id
    FROM people
    WHERE name LIKE 'Joris Ivens'
);
```

2. Write a SQL query to obtain the movie title who is directed by “Joris Ivens”
   Expected Output: a table with a single column for the movie title of the director’s movie.

```
SELECT title
FROM movies
WHERE id IN (
    SELECT movie_id
    FROM directors
    WHERE person_id IN (
        SELECT id
        FROM people
        WHERE name LIKE 'Joris Ivens'
    )
)
```

3. Organize and rewrite the SQL query of Q1 using WITH keyword
   Expected Output: The SQL query in WITH keyword

```
WITH joris_id AS (
   SELECT id
   FROM people
   WHERE name LIKE 'Joris Ivens'
)

SELECT movie_id
FROM directors
WHERE person_id IN (
    SELECT id
    FROM joris_id
)
```

4. Write a SQL query to show each person’s name and whether the person is born before 1970, born in 1970, born after 1970
   Expected Output: The SQL query fulfilling the required data

```
SELECT name,
     CASE
      WHEN birth < 1970 THEN 'born before 1970'
      WHEN birth = 1970 THEN 'born in 1970'
      WHEN birth > 1970 THEN 'born after 1970'
      END AS birth_period
FROM people
```

5. Write a SQL query to count the number of people in the “people” table by each birth year.
   Expected Output: The SQL query fulfilling the required data. Note that having the NULL birth year on the query result is normal.

```
SELECT birth AS year,
COUNT(birth) AS number_of_people
FROM people
GROUP BY birth
```

6. Write a SQL query to count the number of directors by each birth year. Only the years with more than 500 directors born are interested.
   Expected Output: a table with two columns for the birth year and count of directors.

```
SELECT birth AS birth_year, COUNT(id) AS
count_of_directors
FROM people
WHERE id IN (
SELECT person_id FROM directors)
GROUP BY birth
HAVING COUNT(id) > 500
```

- End of Assignment -