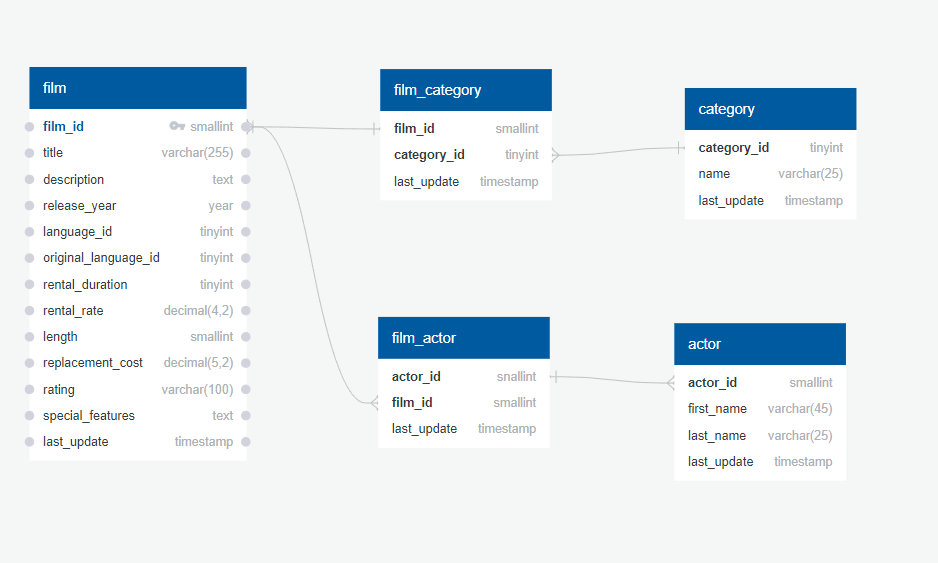
Sequel Premiere Films

Advanced SQL Project

Task 1: Database Schema

Part 1:

Record the missing table names and identify the relationships between tables. Make sure that the lines connect the correct two columns.



Part 2:

List any and all primary and foreign keys for the five tables included in the diagram.

|  |  |
| --- | --- |
| Primary Key | Foreign Key |
| * film\_id | -category\_id |
| * actor\_id | -actor\_id |
| * category\_id |  |

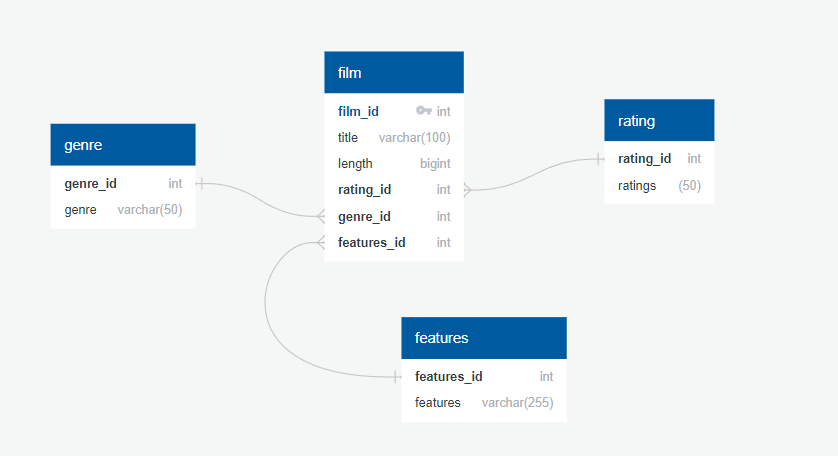
Task 2: Normalization

Below is a section of a table stored in the Sakila database showing information about a specific film. The table has not been normalized. Using the premade tables on the next page, normalize the table into 3NF form. You may also choose to create your own tables if you would like to normalize the table a different way.

Name at least 2 issues with how the table is currently organized.



* The issue with the table above is that many of ratings and genres are repeated often in the table. We can create separate tables with primary keys that show case unique genres and ratings and connect them back to the main table. Same can be said with the features attribute.



Task 3: Complex Views

Sequel Premier Films specializes is in sequels and remakes (surprised?). One of the ways we decide which film to remake is by looking at which actors stared in the original. If an actor is still a household name many years later, the film is more likely to be a success.

Create a view that contains the following information:

• Film ID

• Film Title

• Film Description

• Film Category

• Actor First Name

• Actor Last Name

Copy the text for your view into the box below.

CREATE VIEW film\_actors AS (

SELECT f.film\_id, title, description, name AS category,

first\_name, last\_name

FROM FILM f

INNER JOIN FILM\_CATEGORY fc

ON f.film\_id = fc.film\_id

INNER JOIN CATEGORY c

ON fc.category\_id = c.category\_id

INNER JOIN FILM\_ACTOR fa

ON f.film\_id = fa.film\_id

INNER JOIN ACTOR a

ON fa.actor\_id = a.actor\_id

ORDER BY f.film\_id);

SELECT \*

FROM film\_actors;

Task 4: Advanced Stored Procedures

Create a stored procedure where users can input a keyword (‘pirates’, ‘cowboys’, ‘space’, ‘romance’, etc.) and get results of all movies whose description contains that word. For a challenge, order the results from most to least relevant (however, that is not required). Copy the text for the stored procedure in the box below.

DELIMITER $

CREATE PROCEDURE keyword(IN DESCR VARCHAR(255))

BEGIN

SELECT \*

FROM FILM

WHERE description LIKE CONCAT('%',DESCR,'%')

ORDER BY release\_year DESC;

END $

CALL keyword('EPIC');

Task 5: Encryption

To increase security, we are going to create a new table to keep track of who has access to the database. Run the MySQL script below to create the table.

CREATE table employee\_records(

employee\_id smallint NOT NULL AUTO\_INCREMENT,

f\_name VARCHAR(20) NOT NULL,

l\_name VARCHAR(30) NOT NULL,

job\_title VARCHAR(20),

username VARCHAR(20),

password TEXT,

PRIMARY KEY(employee\_ID)

);

INSERT employee\_records VALUES(1,'Dalia','Database','Executive','dalia\_d', SHA1('movielover1'));

INSERT employee\_records VALUES(2,'Dante','Database','Executive','dante\_d', SHA1('Bossman99'));

SELECT \*

FROM employee\_records;

TASK 6: DOCUMENTATION

Before you leave to go film Doctor Kangaroo 2 in the Australian Outback, we want to make sure that there is a smooth transition between you and the new data analyst who is taking your place. The new data analyst has already received the database schema. Add comments to both the view and stored procedure that you wrote earlier in the project. If you commented your code as you wrote it, then this will be a piece of cake! Copy the code and your comments from the view and stored procedure in the boxes below.

# VIEW TABLE FOR FILM, CATEGORY, ACTOR FULL NAME

CREATE VIEW film\_actors AS (

SELECT f.film\_id, title, description, name AS category,

first\_name, last\_name

FROM FILM f

INNER JOIN FILM\_CATEGORY fc

ON f.film\_id = fc.film\_id

INNER JOIN CATEGORY c

ON fc.category\_id = c.category\_id

INNER JOIN FILM\_ACTOR fa

ON f.film\_id = fa.film\_id

INNER JOIN ACTOR a

ON fa.actor\_id = a.actor\_id

ORDER BY f.film\_id);

# PROCEDURE FOR KEYWRODS IN THE DESCRIPTION ATTRIBUTE

DELIMITER $

CREATE PROCEDURE keyword(IN DESCR VARCHAR(255))

BEGIN

SELECT \*

FROM FILM

WHERE description LIKE CONCAT('%',DESCR,'%')

ORDER BY release\_year DESC;

END $

CALL keyword('EPIC');