Title: DB Assignment 4

Name: Aidan Elm

Date: 2024-11-04

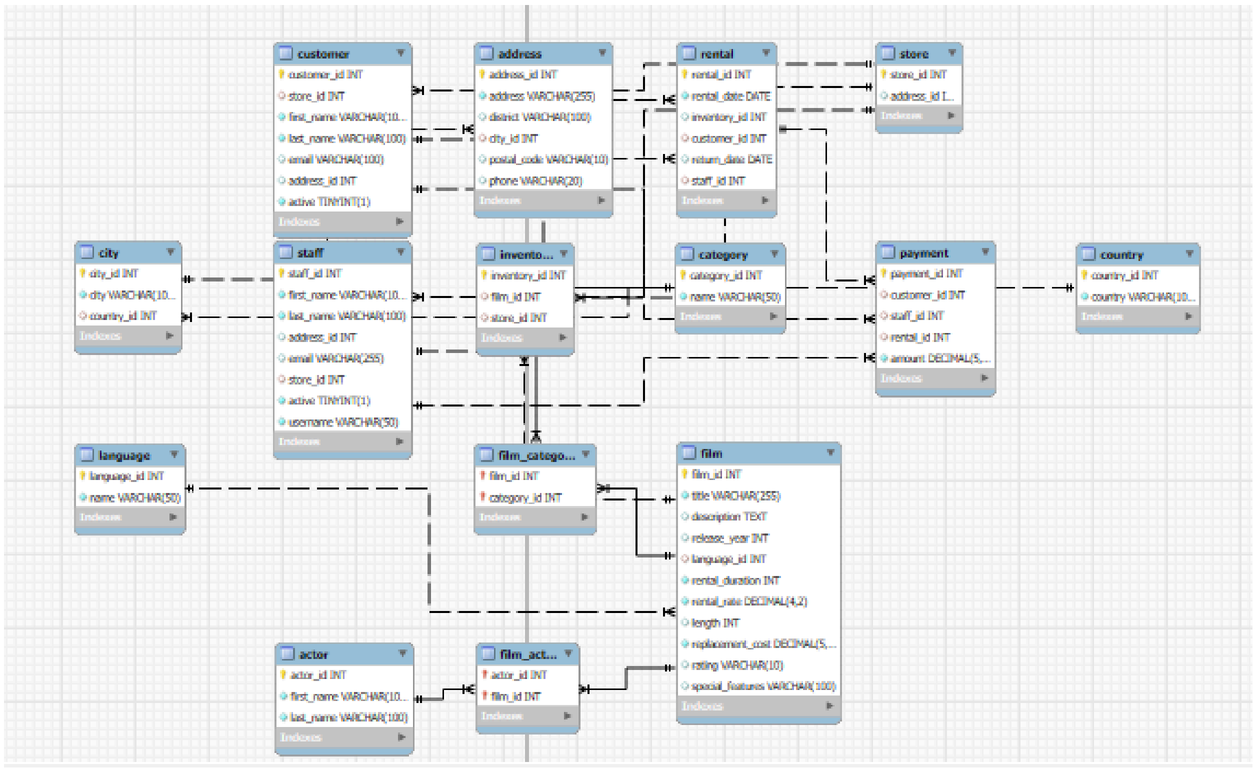
Creation of Tables and ER Diagram

```
-- create actor table
create table actor (
   actor_id int primary key,
   first_name varchar(100) not null,
   last_name varchar(100) not null
);

-- create category table
create table category (
   category_id int primary key,
   name varchar(50) not null,
   constraint check_category_name check (name in ('Animation', 'Comedy', 'Family',
'Foreign',
      'Sci-Fi', 'Travel', 'Children', 'Drama', 'Horror', 'Action', 'Classics', 'Games',
      'New', 'Documentary', 'Sports', 'Music'))
);

-- create country table
create table country (
   country_id int primary key,
   country varchar(100) not null
);

-- create city table
create table city (
   city_id int primary key,
   city varchar(100) not null,
   country_id int,
   foreign key (country_id) references country(country_id)
```

```
);

-- create address table
create table address (
   address_id int primary key,
   address varchar(255) not null,
   district varchar(100),
   city_id int,
   postal_code varchar(10),
   phone varchar(20),
   foreign key (city_id) references city(city_id)
);

-- create language table
create table language (
   language_id int primary key,
   name varchar(50) not null
);

-- create film table
create table film (
   film_id int primary key,
   title varchar(255) not null,
   description text,
   release_year int,
   language_id int,
   rental_duration int not null,
   rental_rate decimal(4,2) not null,
   length int,
   replacement_cost decimal(5,2) not null,
   rating varchar(10),
   special_features varchar(100),
   foreign key (language_id) references language(language_id),
   constraint check_rental_duration check (rental_duration between 2 and 8),
   constraint check_rental_rate check (rental_rate between 0.99 and 6.99),
   constraint check_length check (length between 30 and 200),
   constraint check_rating check (rating in ('PG', 'G', 'NC-17', 'PG-13', 'R')),
   constraint check_replacement_cost check (replacement_cost between 5.00 and 100.00),
   constraint check_special_features check (special_features in
     ('Behind the Scenes', 'Commentaries', 'Deleted Scenes', 'Trailers'))
);

-- create film_actor table
create table film_actor (
   actor_id int,
   film_id int,
   primary key (actor_id, film_id),
   foreign key (actor_id) references actor(actor_id),
   foreign key (film_id) references film(film_id)
);

-- create film_category table
create table film_category (
   film_id int,
   category_id int,
   primary key (film_id, category_id),
   foreign key (film_id) references film(film_id),
   foreign key (category_id) references category(category_id)
);
```

```
-- create inventory table
create table inventory (
   inventory_id int primary key,
   film_id int,
   store_id int,
   foreign key (film_id) references film(film_id),
   foreign key (store_id) references store(store_id)
);

-- create store table
create table store (
   store_id int primary key,
   address_id int
);

-- create staff table
create table staff (
   staff_id int primary key,
   first_name varchar(100) not null,
   last_name varchar(100) not null,
   address_id int,
   email varchar(255),
   store_id int,
   active boolean not null,
   username varchar(50) not null,
   foreign key (store_id) references store(store_id),
   constraint check_active check (active in (0,1))
);

-- create customer table
```

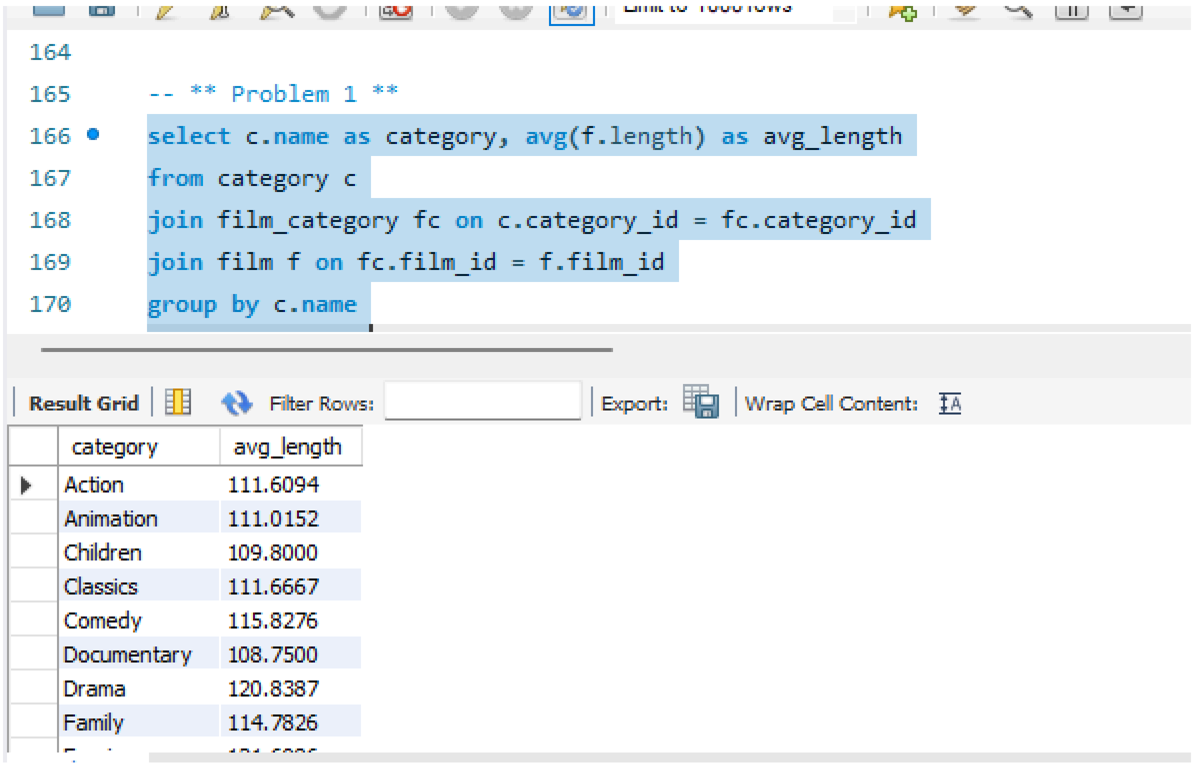
```
create table customer (
   customer_id int primary key,
   store_id int,
   first_name varchar(100) not null,
   last_name varchar(100) not null,
   email varchar(100),
   address_id int,
   active boolean not null,
   foreign key (store_id) references store(store_id)
);

-- create rental table
create table rental (
   rental_id int primary key,
   rental_date date not null,
   inventory_id int,
   customer_id int,
   return_date date,
   staff_id int,
   foreign key (customer_id) references customer(customer_id),
   foreign key (staff_id) references staff(staff_id),
   constraint check_dates check (return_date >= rental_date)
);

-- create payment table
create table payment (
   payment_id int primary key,
   customer_id int,
   staff_id int,
   rental_id int,
   amount decimal(5,2) not null,
   foreign key (customer_id) references customer(customer_id),
   foreign key (staff_id) references staff(staff_id),
   foreign key (rental_id) references rental(rental_id),
   constraint check_amount check (amount >= 0)
);
```



Question 1

```
select c.name as category, avg(f.length) as avg_length
from category c
join film_category fc on c.category_id = fc.category_id
join film f on fc.film_id = f.film_id
group by c.name
order by c.name;
```

To find the average length of films in each category, we join the category table to film through film_category. We group by category name and order alphabetically to get the average length for each category.



```
-- ** Problem 1 **
select c.name as category, avg(f.length) as avg_length
from category c
join film_category fc on c.category_id = fc.category_id
join film f on fc.film_id = f.film_id
group by c.name
```

category	avg_length
Action	111.6094
Animation	111.0152
Children	109.8000
Classics	111.6667
Comedy	115.8276
Documentary	108.7500
Drama	120.8387
Family	114.7826

Question 2

(select c.name as category, avg(f.length) as avg_length

from category c

join film_category fc on c.category_id = fc.category_id

join film f on fc.film_id = f.film_id

group by c.name

order by avg_length desc

limit 1)

union all

(select c.name as category, avg(f.length) as avg_length

from category c

join film_category fc on c.category_id = fc.category_id

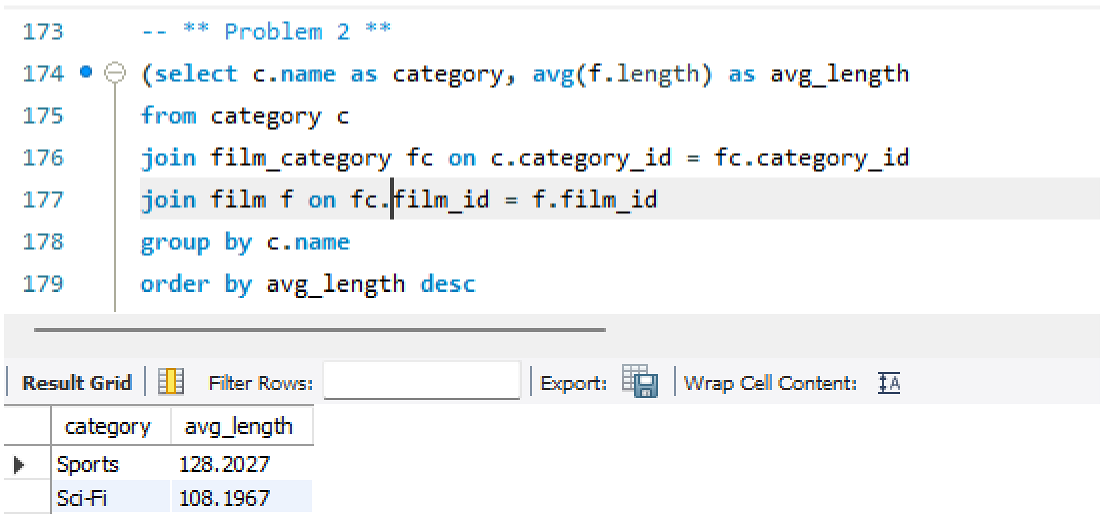
join film f on fc.film_id = f.film_id

group by c.name

order by avg_length asc

limit 1);

Similar to above, but we use two separate queries combined with “union all.” The first query orders by length descending to get the longest average, while the second orders ascending to get the shortest. We limit each to 1 result.



```
-- ** Problem 2 **
(select c.name as category, avg(f.length) as avg_length
from category c
join film_category fc on c.category_id = fc.category_id
join film f on fc.film_id = f.film_id
group by c.name
order by avg_length desc
```

category	avg_length
Sports	128.2027
Sci-Fi	108.1967

Question 3

select distinct c.first_name, c.last_name

from customer c

join rental r on c.customer_id = r.customer_id

join inventory i on r.inventory_id = i.inventory_id

join film f on i.film_id = f.film_id

join film_category fc on f.film_id = fc.film_id

join category cat on fc.category_id = cat.category_id

where cat.name = 'Action'

```
and c.customer_id not in (
   select distinct c2.customer_id
   from customer c2
   join rental r2 on c2.customer_id = r2.customer_id
   join inventory i2 on r2.inventory_id = i2.inventory_id
   join film f2 on i2.film_id = f2.film_id
   join film_category fc2 on f2.film_id = fc2.film_id
   join category cat2 on fc2.category_id = cat2.category_id
   where cat2.name in ('Comedy', 'Classics')
);
```

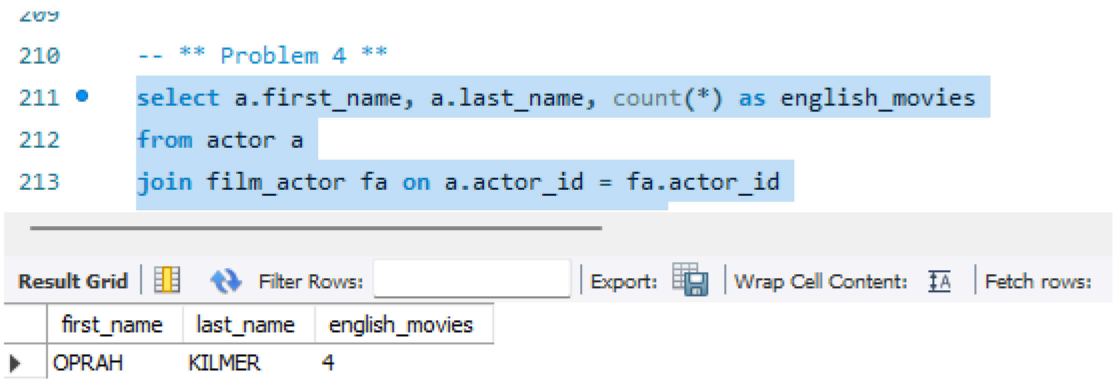
This query requires multiple joins to connect customers to their rented films' categories. We first find customers who rented action films, then use a “not in” subquery to exclude those who also rented comedy or classic films.

*** I couldn’t get an image for this one because I can’t get all the .csv imports to work but I’m pretty sure this one will run ***

Question 4

```
select a.first_name, a.last_name, count(*) as english_movies
from actor a
join film_actor fa on a.actor_id = fa.actor_id
join film f on fa.film_id = f.film_id
join language l on f.language_id = l.language_id
where l.name = 'English'
group by a.actor_id, a.first_name, a.last_name
order by english_movies desc
limit 1;
```

To find the actor with the most English-language movies, we join actor through film_actor to film and language tables. We count the occurrences for each actor where the language is English, group by actor, and order by the count descending.



```
-- ** Problem 4 **
select a.first_name, a.last_name, count(*) as english_movies
from actor a
join film_actor fa on a.actor_id = fa.actor_id
```

```
first_name  last_name  english_movies
OPRAH       KILMER     4
```

Question 5

select count(distinct f.film_id) as movies_count

from film f

join inventory i on f.film_id = i.film_id

join rental r on i.inventory_id = r.inventory_id

join staff s on r.staff_id = s.staff_id

where s.first_name = 'Mike'

and datediff(r.return_date, r.rental_date) = 10;

This query joins film through inventory and rental to staff to count distinct films that were rented for exactly 10 days and only from the store where Mike works. We use datediff to calculate the rental duration.

*** Same with this one - I think it will work, but MySQL keeps throwing errors when importing rentals.csv ***

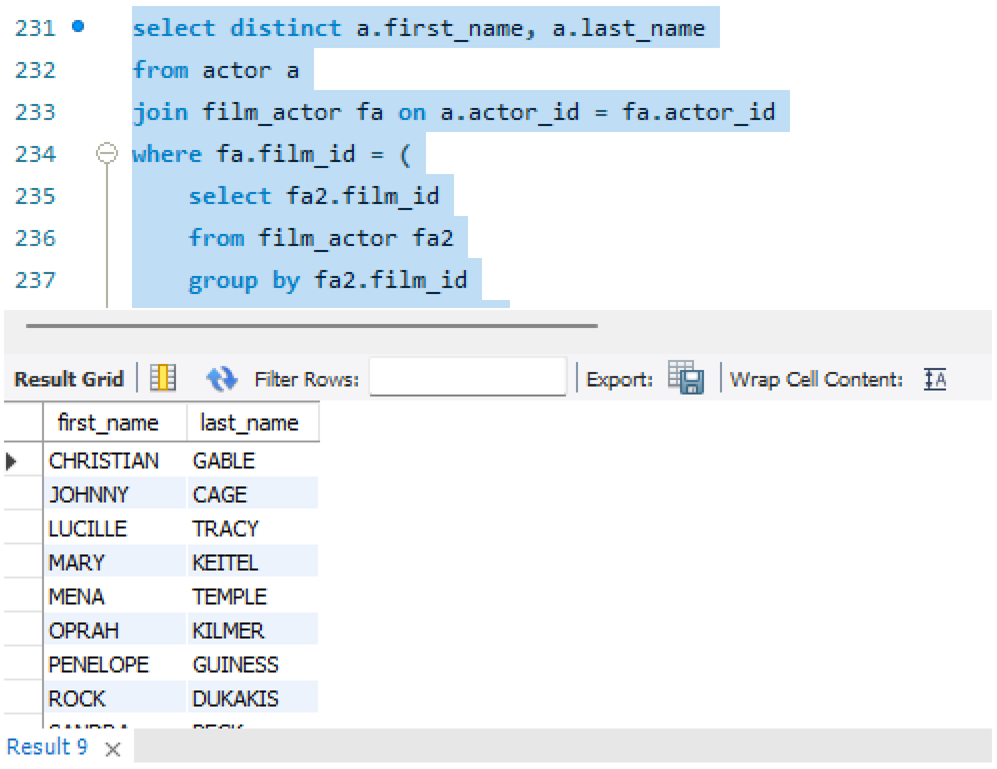
Question 6

select distinct a.first_name, a.last_name

from actor a

```
join film_actor fa on a.actor_id = fa.actor_id
where fa.film_id = (
  select fa2.film_id
  from film_actor fa2
  group by fa2.film_id
  order by count(*) desc
  limit 1
)
order by a.first_name, a.last_name;
```

We use a subquery to first find the film_id with the most actors (by counting entries in film_actor). Then we join this back to actor to get all actors who appeared in that film, ordering them alphabetically.



```
select distinct a.first_name, a.last_name
from actor a
join film_actor fa on a.actor_id = fa.actor_id
where fa.film_id = (
    select fa2.film_id
    from film_actor fa2
    group by fa2.film_id
```

first_name	last_name
CHRISTIAN	GABLE
JOHNNY	CAGE
LUCILLE	TRACY
MARY	KEITEL
MENA	TEMPLE
OPRAH	KILMER
PENELOPE	GUINESS
ROCK	DUKAKIS