

Erin Zahner

Dr. F

CSC 351

31 October 2024

## Assignment 4

### Set-Up

```
-- make composite unique for rental
alter table rental
add constraint unique_combination
unique (rental_date(20), inventory_id, customer_id);

-- add all constraints

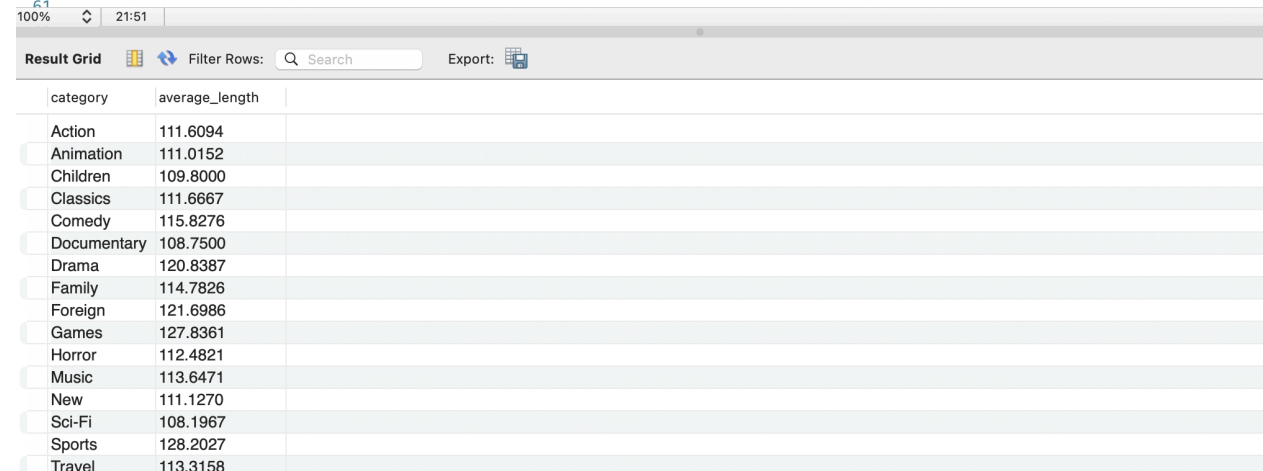
-- check category names
alter table category
add constraint chk_category_name
check (name in ('animation', 'comedy', 'family', 'foreign', 'sci-fi', 'travel', 'children', 'drama', 'horror', 'action', 'classics',

-- check special features, duration, rate, length, rating, and replacement cost
alter table film
add constraint chk_special_features
check (special_features in ('behind the scenes', 'commentaries', 'deleted scenes', 'trailers')),
add constraint chk_rental_duration
check (rental_duration between 2 and 8),
30 add constraint chk_rental_rate
31 check (rental_rate between 0.99 and 6.99),
32 add constraint chk_length
33 check (length between 30 and 200),
34 add constraint chk_rating
35 check (rating in ('pg', 'g', 'nc-17', 'pg-13', 'r')),
36 add constraint chk_replacement_cost
37 check (replacement_cost between 5.00 and 100.00);
38
39 -- active constraints for customer and staff
40 • alter table customer
41 add constraint chk_active
42 check (active in (0, 1));
43
44 • alter table staff
45 add constraint chk_staff_active
46 check (active in (0, 1));
47
48 -- make sure payment amount is greater than 0
49 • alter table payment
50 add constraint chk amount
```

To begin this assignment, I imported all the files from GitHub into MySQL. As I imported the files, I manually entered the required primary keys and foreign keys to ensure that all the data was imported correctly. Once the data was set up, I applied the required constraints across the database to meet the project specifications. These included defining the accepted category names for films, which come from a designated set of genres like 'Animation,' 'Comedy,' and 'Drama.' I enforced limits on the film table attributes: special features must come from a specific set, rental duration must be between 2 and 8 days, rental rates between 0.99 and 6.99, film lengths between 30 and 200 minutes, ratings from a set list, and replacement costs between 5.00 and 100.00. Additionally, I enforced a binary active status in the customer and staff tables, and ensured that all payment amounts are zero or higher. These constraints were implemented as specified to maintain data integrity across the tables.

**What is the average length of films in each category? List the results in alphabetic order of categories.**

```
54  -- What is the average length of films in each category? List the results in alphabetic order of categories.
55  •  select c.name as category, avg(f.length) as average_length
56      from film f
57      join film_category fc on f.film_id = fc.film_id
58      join category c on fc.category_id = c.category_id
59      group by c.name
60      order by c.name;
```



The screenshot shows a MySQL query window with the following SQL code and its results:

category	average_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
Foreign	121.6986
Games	127.8361
Horror	112.4821
Music	113.6471
New	111.1270
Sci-Fi	108.1967
Sports	128.2027
Travel	113.3158

In the query above, I started by selecting the category names and calculating the average film lengths. I joined the film table with the film\_category table on film\_id to link each film with its

categories. Next, I joined the film\_category table with the category table on category\_id to retrieve the category names. I grouped the results by category name to calculate the average film length for each category using the AVG function. Finally, I ordered the results alphabetically by category name to display the average length of films in each category in alphabetical order.

### Which categories have the longest and shortest average film lengths?

```
63  -- Which categories have the longest and shortest average film lengths?
64  • (select c.name as category, avg(f.length) as average_length
65    from film f
66   join film_category fc on f.film_id = fc.film_id
67   join category c on fc.category_id = c.category_id
68   group by c.name
69   order by average_length asc
70   limit 1)
71  union all
72  • (select c.name as category, avg(f.length) as average_length
73    from film f
74   join film_category fc on f.film_id = fc.film_id
75   join category c on fc.category_id = c.category_id
76   group by c.name
77   order by average_length desc
78   limit 1);
79
80
```

100% 10:78

**Result Grid** Filter Rows: Search Export:

category	average_length
Sci-Fi	108.1967
Sports	128.2027

In the query above, I started by calculating the average film length for each category. I joined the film table with the film\_category table on film\_id to link each film with its categories, and then joined the category table on category\_id to retrieve the category names. I grouped the results by category name and used the AVG function to calculate the average length of films in each category. To find the categories with the longest and shortest average film lengths, I ordered the results by average\_length in ascending order to get the shortest average (with LIMIT 1) and in descending order to get the longest average.

Finally, I combined the two results using UNION ALL to display both the shortest and longest average lengths in one result set.

### Which customers have rented action but not comedy or classic movies?

```
00
81  -- Which customers have rented action but not comedy or classic movies?
82  • select distinct c.customer_id, c.first_name, c.last_name
83  from customer c
84  join rental r on c.customer_id = r.customer_id
85  join inventory i on r.inventory_id = i.inventory_id
86  join film_category fc on i.film_id = fc.film_id
87  join category cat on fc.category_id = cat.category_id
88  where cat.name = 'Action'
89  and c.customer_id not in (
90      select r.customer_id
91      from rental r
92      join inventory i on r.inventory_id = i.inventory_id
93      join film_category fc on i.film_id = fc.film_id
94      join category cat on fc.category_id = cat.category_id
95      where cat.name in ('Comedy', 'Classics')
96  );
97
```

customer_id	first_name	last_name
361	LAWRENCE	LAWTON
323	MATTHEW	MAHAN
452	TOM	MILNER
250	JO	FOWLER
330	SCOTT	SHELLEY
432	EDWIN	BURK
164	JOANN	GARDNER
17	DONNA	THOMPSON
433	DON	BONE
350	JUAN	FRALEY
171	DOLORES	WAGNER
445	MICHEAL	FORMAN
139	AMBER	DIXON
223	MELINDA	FERNANDEZ
232	CONSTAN...	REID
90	RUBY	WASHING...
213	GINA	WILLIAMS...

For this problem, I started by selecting unique customer IDs, along with their first and last names, for customers who have rented Action movies but have not rented Comedy or Classics. I joined the customer table with the rental table on customer\_id to connect each customer to their rentals. Then, I joined the rental table with the inventory table on inventory\_id to link each rental to the specific films

rented. Next, I joined the inventory table with the film\_category table on film\_id and finally joined the category table on category\_id to retrieve the category names.

To filter for only customers who rented Action movies, I used a WHERE clause specifying cat.name = 'Action'. Then, I used a NOT IN clause with a subquery to exclude customers who have rented films in the Comedy or Classics categories. The subquery selects customer\_id from the rental, inventory, film\_category, and category tables for customers who rented Comedy or Classics. This ensures that only customers who rented Action movies, but not Comedy or Classics, are returned in the results.

### Which actor has appeared in the most English-language movies?

```
99      -- Which actor has appeared in the most English-language movies?
100 •   select a.actor_id, a.first_name, a.last_name, count(f.film_id) as film_count
101      from actor a
102      join film_actor fa on a.actor_id = fa.actor_id
103      join film f on fa.film_id = f.film_id
104      join language l on f.language_id = l.language_id
105      where l.name = 'English'
106      group by a.actor_id, a.first_name, a.last_name
107      order by film_count desc
108      limit 1;
```

109

100% 9:108

Result Grid				Filter Rows:	Search	Export:	Fetch rows:
actor_id	first_name	last_name	film_count				
107	GINA	DEGENERES	42				

For this question, I started by selecting the actor's ID, first name, and last name, along with a count of the films they've appeared in. I joined the actor table with the film\_actor table on actor\_id to connect each actor with their films. Then, I joined the film table on film\_id to access the details of each film, and I joined the language table on language\_id to get the language of each film. To filter for only English-language movies, I added a WHERE clause specifying l.name = 'English'. I grouped the results by actor ID, first name, and last name, and used the COUNT function to count the number of

English-language films for each actor. Finally, I ordered the results in descending order by the film count and used LIMIT 1 to return only the actor who has appeared in the most English-language movies.

### How many distinct movies were rented for exactly 10 days from the store where Mike works?

```
113  -- How many distinct movies were rented for exactly 10 days from the store where Mike works?
114  • select count(distinct f.film_id) as distinct_movies_rented
115  from rental r
116  join inventory i on r.inventory_id = i.inventory_id
117  join film f on i.film_id = f.film_id
118  join staff s on i.store_id = s.store_id
119  where s.first_name = 'Mike'
120  and datediff(r.return_date, r.rental_date) = 10;
121
122
```

100% 49:120

Result Grid Filter Rows: Search Export:

distinct_movies_rented
61

In the query above, I started by counting the distinct movies rented for exactly 10 days from the store where Mike works. I joined the rental table with the inventory table on `inventory_id` to link each rental to its corresponding film. Then, I joined the film table on `film_id` to access film details, and I joined the staff table on `store_id` to identify the store associated with each rental. To filter for rentals that lasted exactly 10 days, I used a WHERE clause that checks the difference between the return date and rental date. Additionally, I specified that the rentals should be from Mike's store by including a condition to match the staff member's first name to 'Mike'. The query counts the distinct film IDs that satisfy these conditions and returns the result as `distinct_movies_rented`.



**Alphabetically list actors who appeared in the movie with the largest cast of actors.**

```
121  -- Alphabetically list actors who appeared in the movie with the largest cast of actors.
122  • ⊖ with film_cast_count as (
123      select fa.film_id, count(fa.actor_id) as actor_count
124      from film_actor fa
125      group by fa.film_id
126  ),
127  ⊖ max_cast_film as (
128      select film_id
129      from film_cast_count
130      where actor_count = (select max(actor_count) from film_cast_count)
131  )
132  select a.first_name, a.last_name
133  from actor a
134  join film_actor fa on a.actor_id = fa.actor_id
135  join max_cast_film mcf on fa.film_id = mcf.film_id
136  order by a.last_name, a.first_name;
---
```

first_name	last_name
JULIA	BARRYMORE
VAL	BOLGER
SCARLETT	DAMON
LUCILLE	DEE
WOODY	HOFFMAN
MENA	HOPPER
REESE	KILMER
CHRISTIAN	NEESON
JAYNE	NOLTE
BURT	POSEY
MENA	TEMPLE
WALTER	TORN
FAY	WINSLET
CAMERON	ZELLWEGER
JULIA	ZELLWEGER

For this last question, I started by finding actors who appeared in the movie with the largest cast. First, I created a Common Table Expression (CTE) named `film_cast_count` to count the number of actors in each movie. This CTE groups the `film_actor` table by `film_id` and counts the number of `actor_ids` to get the cast size for each film. Next, I created another CTE called `max_cast_film` to identify the film(s) with the largest cast. This CTE selects the `film_id` where the actor count matches the maximum cast size, using a subquery to find the maximum count from `film_cast_count`. Finally, in the main query, I selected the first and last names of actors by joining the `actor` table with the `film_actor` table on `actor_id` to link each actor

with their movies. I joined this result with `max_cast_film` on `film_id` to filter for actors in the movie with the largest cast. I ordered the final list alphabetically by the actors' last names and then by first names.