Report of HW4
Name: Deare
Date:2024/10/21

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

SQL Component	Explanation
SELECT category.name, AVG(f.length) AS lengthOFfilms	Selects the category name and calculates the average length of films within each category, labeling it as lengthOFfilms.
FROM film AS f	Specifies the source table for film data, using f as an alias for the film.
JOIN film_category USING (film_id)	Joins the film table to the film_category table by matching entries based on the film_id column.
JOIN category USING (category_id)	Joins the film_category table to the category table by matching entries based on the category_id column.
GROUP BY category.name	Group the results by the category name to calculate the average length for films in each category.
ORDER BY category.name ASC	Orders the grouped results alphabetically by category name for organized output.

```sql
-- ******************************************************************************
-- 1. What is the average length of films in each category? List the results in alphabetic order of categories.
-- ******************************************************************************
Select category.name ,avg(f.length) as lengthOFfilms
from film f
join film_category using (film_id)
join category using (category_id)
group by category.name
order by category.name asc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	lengthOFfilms
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

Toggle wrapping of cell contents

2. Which categories have the longest and shortest average film lengths?

-- UNION ALL is the best choice in situations where the inclusion of all results from multiple queries
-- Query for the category with the shortest average film length

SQL Part	Description
(Select category.name, avg(f.length) as lengthOFfilms	This part of the query selects the name of the film category and calculates the average length of films within that category.
from film f	Specifies that the data is sourced from the film table, aliased as f.
join film_category using (film_id)	Joins the film table with the film_category table on the film_id, linking films to their categories.
join category using (category_id)	Joins the film_category table with the category table on the category_id, providing access to category names.
group by category.name	Groups the results by the film category name to enable aggregation (average calculation) per category.
order by avg(f.length) desc	Orders the results by the average film length in descending order, so the category with the longest films appears first.
limit 1)	Limits the results to the single category with the longest average film length.
union all	Combines the results of the two queries. UNION ALL includes all results from both queries, including duplicates.
(Select category.name, avg(f.length) as lengthOFfilms	Starts the second part of the query, structurally similar to the first, to select the category with the shortest films.
order by avg(f.length) asc	Changes the order to ascending to find the category with the shortest films.
limit 1);	Limits the results to the single category with the shortest average film length.

```sql
-- ******************************************************************************
-- 1. Which categories have the longest and shortest average film lengths?
-- ******************************************************************************
-- Query for the category with the longest average film length
(Select category.name ,avg(f.length) as lengthOFfilms
from film f
join film_category using (film_id)
join category using (category_id)
group by category.name
order by avg(f.length) desc
limit 1)
union all -- UNION ALL is the best choice in situations where the inclusion of all results from multiple queries
-- Query for the category with the shortest average film length
(Select category.name ,avg(f.length) as lengthOFfilms
from film f
join film_category using (film_id)
join category using (category_id)
group by category.name
order by avg(f.length) asc
limit 1);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	lengthOFfilms
Sports	128.2027
Sci-Fi	108.1967

Longest average film length

shortest

3. Which customers have rented action but not comedy or classic movies?
-- Assuming combination of first_name and last_name is customer's name
-- Assuming the customer's name must be distinct
-- Assuming the catagory name 'classic' comes from the constraint set {Classics}
-- Assuming represented result by customer's name and customer_id

SQL Part	Description
select distinct	Selects unique combinations of customer first and last names, along with their customer IDs, formatted as "Customer_name" and "cid".
from customer	Indicates that the data retrieval starts from the customer table, establishing the base table for further joins.
join rental using(customer_id)	Joins the customer table to the rental table using the customer_id field, including data about each customer's rentals.
join inventory using(inventory_id)	Joins the rental table to the inventory table using the inventory_id, providing access to details of rented items.
join film using(film_id)	Joins the inventory table to the film table using the film_id, allowing access to film details related to the inventory.
join film_category using(film_id)	Joins the film table to the film_category table, linking films to their categories using the film_id.
join category using(category_id)	Joins the film_category table to the category table using the category_id, facilitating

	filtering of films by category.
where category.name='Action'	Filters the results to include only entries where the film's category is 'Action'.
and customer.customer_id not in	Adds a condition to exclude certain customers. This part is incomplete and would typically include a subquery to specify customers to exclude (e.g., those who have rented 'Comedy' or 'Classics').

```
*/
select distinct(concat(customer.first_name,' ',last_name)) as Customer_name, customer.customer_id as cid
from customer
join rental using(customer_id)
join inventory using(inventory_id)
join film using(film_id)
join film_category using(film_id)
join category using(category_id)
where category.name='Action'
and customer.customer_id not in(
select customer.customer_id
from customer
join rental using(customer_id)
join inventory using(inventory_id)
join film using(film_id)
join film_category using(film_id)
join category using(category_id)
WHERE category.name IN ('Comedy', 'Classics')
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_name	cid
DONNA THOMPSON	17
RUBY WASHINGTON	90
AMBER DIXON	139
JOANN GARDNER	164
DOLORES WAGNER	171
GINA WILLIAMSON	213
MELINDA FERNANDEZ	223

Result 62

4. Which actor has appeared in the most English-language movies?
-- Assuming combination of first_name and last_name is actor's full name
-- Assuming representing result by actor_id and actor full name and number of films they participate in
-- Due to some actors having the same first name and last name like SUSAN DAVIS, it is better to represent the result by actor_id(PK), actor full name, number of films they participated in

SQL	Description
SELECT CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name, COUNT(film.film_id) AS num_films, actor.actor_id as actor_id	Selects the full name of the actor by concatenating the first and last names, counts the number of films each actor has appeared in, and selects the actor's ID.
FROM actor	Specifies the actor table as the source of data.
JOIN film_actor USING(actor_id)	Joins the actor table with the film_actor table using the

	actor_id to match records.
JOIN film USING(film_id)	Joins the film_actor table with the film table using the film_id to match records.
JOIN language USING(language_id)	Joins the film table with the language table using the language_id to match records.
WHERE language_id = '1'	Filters the results to include only films in language_id '1'.
GROUP BY actor_name, actor_id	Groups the results by actor name and actor ID, which is necessary for counting the films per actor.
ORDER BY num_films DESC	Orders the results by the number of films in descending order, so the actor with the most films is listed first.
LIMIT 1	Limits the results to only the top entry (actor with the most films).

```
105   */
106 • SELECT CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name,  COUNT(film.film_id) AS num_films, actor.actor_id as actor_id
107   from actor
108   JOIN film_actor USING(actor_id)
109   JOIN film USING(film_id)
110   JOIN language USING(language_id)
111   WHERE language_id = '1'
112   GROUP BY actor_name,actor_id
113   ORDER BY num_films DESC
114   LIMIT 1;
115
116   -- ***********************************************************************
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actor_name	num_films	actor_id
GINA DEGENERES	42	107

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

-- Assuming active row to check if the staff member is currently active in work, 1 means staff is working.

-- Assuming rentals for exactly 10 days means where the difference between the return_date and the rental_date.

SQL Part	Description
SELECT DISTINCT(film.film_id) AS distinct_films_id, staff.staff_id	Selects unique film IDs and the staff ID, ensuring no duplicates in the list of film IDs.
FROM rental	Specifies that the data is being selected from the rental table.
JOIN inventory ON rental.inventory_id =	Joins the rental table with the inventory table, linking them via the

inventory.inventory_id	inventory_id.
JOIN store ON inventory.store_id = store.store_id	Joins the inventory table with the store table, using store_id to link inventory items to their stores.
JOIN staff ON store.store_id = staff.store_id	Joins the store table with the staff table, linking them by store_id to associate staff members with stores.
WHERE DATEDIFF(rental.return_date, rental.rental_date) = 10	Filters for rentals where the difference between the return_date and rental_date is exactly 10 days.
AND staff.first_name = 'Mike'	Narrows down records to those handled by staff members named Mike.
AND staff.staff_id = '1'	Further filters to ensure the specific staff member, Mike, has the staff_id of '1'. - First name Mike staff_id is 1, filter by Mike's id number, this is an extra step but it can ensure the exact staff Mike, because lot of staff's first name could be Mike
AND staff.active = 1	Includes only those entries where the staff member is currently active (active = 1).

```
-- ******************************************************************************
-- 5. How many distinct movies were rented for exactly 10 days from the store where Mike works?
-- ******************************************************************************
select distinct(film_id) as distinct_films_id, staff.staff_id
FROM rental
JOIN inventory using(inventory_id)
JOIN store using (store_id)
JOIN staff using(store_id)
where DATEDIFF(rental.return_date, rental.rental_date) = 10 -- Assuming rentals to exactly 10 days means where the difference between the re
and staff.first_name='Mike'-- staff first name is Mike
and staff.staff_id='1'-- First name Mike staff_id is 1, filter by Mike's id number, this is an extra step but it can ensure the exact staff
AND staff.active = 1; -- Assuming active row to check if the staff member is currently active in work,1 means staff is working.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

distinct_films_id	staff_id
16	1
56	1
78	1
98	1
118	1
139	1
143	1
153	1
200	1

Result 54 × Read

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

SQL Part	Description
WITH MovieCastCounts AS (...)	Common Table Expression (CTE): This section calculates the cast size for each film by counting the number of

	distinct actors associated with each film.
SELECT film.film_id, COUNT(actor.actor_id) AS cast_size	This command selects the film ID and counts the number of unique actor IDs associated with each film, effectively measuring the size of the cast per film.
FROM film JOIN film_actor USING(film_id) JOIN actor USING(actor_id)	Joins the film, film_actor, and actor tables to link films with their actors. These joins ensure that all necessary data about films and their actors can be collated.
GROUP BY film.film_id	Groups the results by film ID, which is necessary for the COUNT function to calculate the number of actors per film.
MaxCastMovie AS (...)	CTE: Identifies the film(s) with the largest cast. It uses the results of the MovieCastCounts CTE to find the maximum cast size.
SELECT film_id FROM MovieCastCounts WHERE cast_size = (SELECT MAX(cast_size) FROM MovieCastCounts)	Selects the film ID from the previous CTE where the cast size equals the maximum cast size found across all movies. This identifies the movie with the largest number of actors.
SELECT DISTINCT CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name	Selects distinct actor names by concatenating first and last names, ensuring that each actor is listed only once.
FROM actor JOIN film_actor USING(actor_id) JOIN MaxCastMovie USING(film_id)	After identifying the movie with the largest cast, this joins the actor and film_actor tables, and then with the MaxCastMovie to only include actors from that specific movie.
ORDER BY actor_name ASC	Orders the list of actors alphabetically by their full name.

```sql
WITH MovieCastCounts AS (
SELECT film.film_id,COUNT(actor.actor_id) AS cast_size  -- Count of actors in each movie
FROM film
JOIN film_actor using(film_id)
join actor using(actor_id)
GROUP BY film.film_id
),
MaxCastMovie AS (
SELECT film_id
from MovieCastCounts
WHERE cast_size = (SELECT MAX(cast_size) FROM MovieCastCounts)  -- Identifies the movie with the largest cast
)

-- Final Query to list actors from the movie with the largest cast
SELECT DISTINCT CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name
FROM actor
JOIN film_actor using (actor_id)
JOIN MaxCastMovie using(film_id)
ORDER BY actor_name ASC;  -- Orders the actor names alphabetically
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actor_name
BURT POSEY
CAMERON ZELLWEGER
CHRISTIAN NEESON
FAY WINSLET
JAYNE NOLTE
JULIA BARRYMORE
JULIA ZELLWEGER

Result 61