Database Management Systems:

Assignment 4

Albert Adamson

Department of Computer Science. Saint Joseph's University

CSC 351: Database Management Systems

Dr. Forouraghi

November 14, 2025

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

Query + Results:

```
227   -- 1. What is the average length of films in each category? List the results in alphabetic order of categories.
228
229 • select cat.name, avg(film.length)              -- We only need these 2 values
230   from category as cat                           -- Alias category
231   inner join film_category using(category_id)    -- inner join: Category -> Film_category -> film
232   inner join film using(film_id)
233   group by cat.name                              -- Apply avg(film.length) to members of name group
234   order by cat.name;                             -- Alphabetical Ordering
235
236   -- 2. Which categories have the longest and shortest average film lengths?
237 • select cat.name, avg(film.length)                          -- Need Name + Avg. Film Length
```

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

#	name	avg(film.length
1	Action	111.6094
2	Animation	111.0152
3	Children	109.8000
4	Classics	111.6667
5	Comedy	115.8276
6	Documentary	108.7500
7	Drama	120.8387
8	Family	114.7826
9	Foreign	121.6986
10	Games	127.8361
11	Horror	112.4821
12	Music	113.6471
13	New	111.1270
14	Sci-Fi	108.1967
15	Sports	128.2027
16	Travel	113.3158

This query starts off by only pulling two columns: category.name , and avg(film.length). Then the tables, category, film_category, and film are merged together, respectively, since category contains the names of the categories, film_categories assigns categories to a film, and film contains the length of the films. Next it is grouped by the category name since we want to apply the avg() aggregate to the contents of each category group. Finally, the order by cat.name line is added to place the results in alphabetical order based on the name of the category.

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

Query + Results:

```
236   -- 2. Which categories have the longest and shortest average film lengths?
237 • select cat.name, avg(film.length)                          -- Need Name + Avg. Film Length
238   from category as cat                                       -- Category -> film_category -> film
239   inner join film_category using(category_id)
240   inner join film using(film_id)
241   group by cat.name                                          -- Group by category name, Apply these conditions AFTER
242   having avg(film.length) >= all (select avg(film2.length)                    -- Find Largest Average
243                                   from category as cat2                       -- Same grouping as main query
244                                   inner join film_category using(category_id)
245                                   inner join film as film2 using(film_id)
246                                   group by cat2.name )
247   or avg(film.length) <= all (select avg(film3.length)                        -- Find Shortest Average
248                               from category as cat3                           -- Same grouping as main query
249                               inner join film_category using(category_id)
250                               inner join film as film3 using(film_id)
251                               group by cat3.name );
```

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

#	name	avg(film.length
1	Sci-Fi	108.1967
2	Sports	128.2027

The query begins by only pulling the category name and the avg(film.length), similar to how it was done in problem 1, and the tables are joined for the same reason. The main difference is that an additional “having” line is added. The first part of this line checks for the category which has the greatest average film length by using a subquery which contains all of the average film lengths for each category. It then tries to find the film category that is greater than or equal to all of the contents of the first subquery. The second part of the line does the opposite and tries to find the category which has the shortest average. Just like the previous one, the subquery contains all of the average film lengths and finds the category which has an average length less than or equal to all averages. Finally, the ‘or’ keyword is used to combine the two results.

Problem 3: Which customers have rented action but not comedy or classic movies?

Query + Results:

```
select customer.first_name, customer.last_name                    -- Just need customer First name + last name
from customer
where customer.customer_id in ( select distinct rental.customer_id  -- Customers who rented action movies (Distinct is redundant)
                                 from rental                        -- Rental -> inventory -> film -> film_category -> category
                                 inner join inventory using (inventory_id)
                                 inner join film using (film_id)
                                 inner join film_category using (film_id)
                                 inner join category using (category_id)
                                 where category.name = "Action")    -- Pull only action movies
and customer.customer_id not in (   select distinct rental.customer_id  -- Customers who have rented comedy or classics (Apply not afterwards)
                                    from rental                         -- Rental -> inventory -> film -> film_category -> category
                                    inner join inventory using (inventory_id)
                                    inner join film using (film_id)
                                    inner join film_category using (film_id)
                                    inner join category using (category_id)
                                    where category.name = "Comedy"      -- Pull Comedy and Classics movies
                                    or category.name = "Classics");
```

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

#	first_name	last_name
1	LAWRENCE	LAWTON
2	MATTHEW	MAHAN
3	TOM	MILNER
4	JO	FOWLER
5	SCOTT	SHELLEY
6	EDWIN	BURK
7	JOANN	GARDNER
8	DONNA	THOMPSON
9	DON	BONE
10	JUAN	FRALEY
11	DOLORES	WAGNER
12	MICHEAL	FORMAN
13	AMBER	DIXON
14	MELINDA	FERNAND...
15	CONSTA...	REID
16	RUBY	WASHING...
17	GINA	WILLIAMS...

The query begins by only pulling the customer's first and last names since we only care about who the customers are. We then use the 'where' statement to filter results based on two conditions. Condition one is based upon customers who rented action movies. A subquery is used to generate a set of ids. Firstly, only customer_ids are outputted by the query. Then the tables rental, inventory, film, film_category, and category are joined together because we need to connect the rental table's customer_id to the category name "Action". As a result, the subquery generates all customer_ids who have rented an action movie. The second condition is connected with an "and" statement, and finds the name of all customer_ids of everyone who has rented a Comedy movie or a Classics movie via the same method as the prior one. The main query then finds people who have rented action movies but not Comedy nor Classics by checking if each customer's id is in the first subquery, but not in the second query.

Problem 4: Which actor has appeared in the most English-language movies?

Query + Results:

```
-- 4. Which actor has appeared in the most English-language movies?
select actor.first_name, actor.last_name                    -- Just need actor name
from actor
where actor.actor_id in(select film_actor.actor_id          -- Subquery for actor id in most english language movies
                        from film_actor                     -- Film_actor -> film -> language
                        inner join film using (film_id)
                        inner join language using (language_id)
                        where language.name = "English"     -- Pull only English movies
                        group by film_actor.actor_id        -- Group by film_actor.id since it is in film_actor
                        having count(*) >= all (select count(*)                     -- Subquery for findind person with most English Movies
                                                from film_actor                     -- Film_actor -> film -> language
                                                inner join film using (film_id)
                                                inner join language using (language_id)
                                                where language.name = "English"     -- Filter English Movies only
                                                group by film_actor.actor_id));     -- Group by film_actor.id since it is in film_actor
```

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

#	first_name	last_name
1	GINA	DEGENERES

The query begins by only pulling the names of the actors since we only need to know the name of the actor. It then filters for actors based on a subquery which finds the actor id of the actor which was in the most english language movies. It works by outputting only actor_ids from the joined tables: film_actor, film, language. These tables were chosen since we need to relate the actor_id in film_actor with the language table. However, before any grouping is done, the results are filtered to only include “English” language movies. The results are then grouped, but an additional having clause is added to ensure we find the actor with the most English movie appearances. An additional subquery is used to generate the count() of the appearances, to which the “having” clause is used to find the actor id with the highest amount.

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

Query + Results:

```
289   -- 5. How many distinct movies were rented for exactly 10 days from the store where Mike works?
290
291 • select count(distinct film.film_id)                                    -- Count only distinct films
292   from rental                                                           -- Rental -> inventory -> film
293   inner join inventory using (inventory_id)
294   inner join film using (film_id)
295   where inventory.store_id = (select store_id from staff where first_name = "Mike")      -- Make sure it is only from Mike's store
296   and DATE_SUB(DATE(rental.return_date), INTERVAL 10 DAY) = DATE(rental.rental_date);    -- Make sure it is exactly 10 days (Ignore Time)
297
```

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

#	count(distinct film.film_i
1	61

This query works by only outputting the count of distinct film ids. We then join the tables rental, inventory, and film since we want to relate the rental table's return_dates/rental_dates with a particular store_id. We filter the results before the counting is done with the "where" clause and find the inventory.store_id which is equal to the store_id of the store that Mike works at. Mike's store_id is found through a subquery that selects a store_id from staff and filters to only show employees with the first name "Mike". Since there is only one Mike who works at only one store, only one store_id is output so we can compare it in the main query's store_id. The second condition in the main clause is to make sure to filter for results which were returned in exactly 10 days. The DATE_SUB function is used for this as it is used to subtract 10 days from the rental return date (We first ignore the time it was rented) and compare it to when it was originally rented. If the two values are equal then it was returned in exactly 10 days (If we ignore the time which I did since nothing was returned otherwise). After all filtering is done, the results are counted to output the number 61.

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

Query + Results:

```
298   -- 6. Alphabetically list actors who appeared in the movie with the largest cast of actors.
299
300 • select actor.first_name, actor.last_name                          -- Only need actor names
301   from actor                                                        -- actor -> film_actor
302   inner join film_actor using (actor_id)
303   where film_actor.film_id = (select film_id                        -- Subquery for finding movie with most actors
304                               from film_actor
305                               group by film_id                      -- Group by film_id
306                               having count(*) >= all (select count(*)   -- Find max number of actors
307                                                       from film_actor
308                                                       group by film_id)
309                               limit 1)                              -- Make sure only one film_id is outputted
310   order by actor.last_name, actor.first_name;                       -- Sort Alphabetically
311
```

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

#	first_name	last_name
1	JULIA	BARRYMORE
2	VAL	BOLGER
3	SCARLETT	DAMON
4	LUCILLE	DEE
5	WOODY	HOFFMAN
6	MENA	HOPPER
7	REESE	KILMER
8	CHRISTIAN	NEESON
9	JAYNE	NOLTE
10	BURT	POSEY
11	MENA	TEMPLE
12	WALTER	TORN
13	FAY	WINSLET
14	CAMERON	ZELLWEGER
15	JULIA	ZELLWEGER

This query works by first only outputting the actors name since that is all we need. Within the main query the table actor is joined with the film_actor table since we need to find the name of actors to the film id of the films they appeared in. A subquery is used to filter for the film that has the most actors in it and the main query uses it to find the name of actors who appeared in that particular film. The subquery works by only outputting a single film id, grouping by the film_id and using the “having” clause to find the film_id that has the greatest number of actors. It finds that by comparing the count of actors to an additional subquery that outputs the number of actors in each movie. An additional limit of 1 is added, but it isn’t needed since there are no ties. The subquery then outputs the film_id with the most actors, and the main query uses it to output all of the actors inside of that movie. The results are then ordered alphabetically by their last names first and then their first names second in case there is a tie.