#Assignment27.1_Session27

#Problem

#1. Return the categories (names) of the longest film. NOTE that there may be several "longest"

#Films (i.e. with the same length), so you might need to return more than one category. Return

#the duration as well.

#2. Find the movies whose total number of actors is above the average. Return the movie names

#and its number of actors ordered by the title. IMPORTANT NOTE: this query should return many

#movies. Please write in your submission only the first TOP-10 results.

#Answers

Code :

```
SELECT title,
      length,
      categoryName
FROM ( SELECT CASE
                WHEN film_category.category_id = @currentCategory THEN
                  @currentRecord := @currentRecord + 1
                ELSE
                  @currentRecord := 1 AND
                  @currentCategory := film_category.category_id
            END AS recordNumber,
            film.title AS title,
            film.length AS length,
            categoryName AS categoryName
      FROM film
      JOIN film_category ON film.film_id = film_category.film_id
      JOIN ( SELECT film_category.category_id AS category_id,
                  category.name AS categoryName,
                  AVG( film.length ) AS categoryAvgLength
            FROM film
            JOIN film_category ON film.film_id = film_category.film_id
            JOIN category ON category.category_id = film_category.category_id
            GROUP BY film_category.category_id,
                    category.name
          ) AS categoryAvgLengthFinder ON film_category.category_id = categoryAvgLengthFinder.category_id,
          (
            SELECT @currentCategory := 0 AS currentCategory
          ) AS currentValuesInitialiser
      WHERE film.length > categoryAvgLength
      ORDER BY film_category.category_id,
              film.length
    ) AS allLarger
WHERE recordNumber <= 5;
```

Assumptions :

This statement starts by using the following subquery to form a "table" consisting of each `category`'s unique identifier `category_id`, the `name` of the `category` and the corresponding average `Film` `length`

This subquery's results are then joined to `film` and `film_catgory`. As the subquery retrieves all the details from `category` that we will need for the rest of the statement, no `JOIN` with `category` is needed.

The resulting dataset is then cross-joined with `SELECT @currentCategory := 0 AS currentCategory`to initialise the variable `@currentCategory` within the same statement. This does come at the cost of appending a field called `currentCategory` to the dataset generated above, so you may prefer to use the following code instead.
Once the `JOIN`'s are performed (and `@currentCategory` is initialised), the resulting dataset is refined to just those records whose value of `film.length' is greater than the corresponding average for that category. The refined dataset is then sorted (not grouped) by one of the`category_id`fields (of which there will be two sharing the same value owing to the joining) and subsorted by`film.length`.

When the fields are chosen, each record's value of `category_id` is compared to the value of `@currentCategory`. If they do not match then `@currentRecord` is initialised to `1` and `@currentCategory` is updated to the new value of `category_id`. If they do match, then `@currentRecord` is incremented. In either case, the value assigned to `@currentRecord` is returned to the `SELECT` statement into the field given the alias `recordNumber`. It is thus that we are able to prepend a record number to our refined dataset.
Then all that remains is to `SELECT` all the records from the refined dataset (sans record number) where the record number is less than or equal to `5`

```
SELECT film_id as MovieNumber, COUNT(actor_id) AS ActorCount
FROM film_actor
GROUP BY film_id
HAVING ActorCount > AVG(ActorCount);
```