Jeremy Obach

CareerFoundry DA Immersion

Task 3.6

1a. Checking film table for duplicate data:

SELECT DISTINCT title,

release\_year,

rental\_duration,

rental\_rate

FROM film;

1000 titles returned, so they are all unique values, as the complete film table contains 1000 entries. Eschewed the film\_id category but included title and release year (and sort of arbitrarily duration and rate as well, wondering if I should have included every category besides film\_id to check for uniqueness…).

Checking actor table for duplicate data:

SELECT DISTINCT first\_name,

last\_name

FROM actor

For this one, there were 199 returns for this query vs 200 for the entire table, so we know there’s one duplicate.

SELECT first\_name,

last\_name,

COUNT(\*)

FROM actor

GROUP BY first\_name,

last\_name

HAVING COUNT (\*) >1;

Susan Davis resulted in 2 counts. Probably could have started with this query… Hey we’re learning!

1b. Checking film table for non-uniform values:

SELECT title,

rating,

release\_year,

rental\_duration,

rental\_rate

FROM film

GROUP BY title,

rating,

release\_year,

rental\_duration,

rental\_rate

ORDER BY rating

Used GROUP BY then ORDER BY rating to check for inconsistencies in the rating column. If there were irregular values there, they would be easier to spot in between the correct values in the table. Could repeat this process for every category that is likely to have non-uniform values, or categories that probably should have been built with a specific data type but were not.

Checking actor table for non-uniform values:

SELECT \*

FROM actor

Given the four categories of this table (actor\_id, first\_name, last\_name, and last\_update, we’re just going to check the entire table and see if any values don’t fit the data type (integer, charvar45, charvar45, and timestamp without time zone, respectively). Looks good.

1c. Checking film table for null values:

SELECT COUNT(rental\_rate) AS count\_rent\_values,

COUNT (title) AS "number of titles"

FROM film

Rinse and repeat for all categories, see if number of values equals number of records.

Checking actor table for null values:

SELECT \*

FROM actor

A manageably sized table, I just looked it over and saw no null values.

2a. Film table summary statistics.

Numerical columns include:

Release year – 200 36 is MIN/MAX/AVG

Rental duration – Min: 3, Max: 7, avg 4.985

Rental rate – Min: .99, Max: 4.99, avg 2.98

Length – Min: 46, Max: 185, avg: 115.272

Replacement cost – Min: 9.99, Max: 29.99, avg: 19.98

SELECT MIN(release\_year) AS min\_release\_year,

MAX(release\_year) AS max\_release\_year,

AVG(release\_year) AS avg\_release\_year

FROM film

SELECT MIN(rental\_duration),

MAX(rental\_duration),

AVG(rental\_duration)

FROM film

Etc.

Non-numeric but the MODE could still bring insight:

Rating – PG-13

Language\_ID (FK) – 1 (English)

Title – Academy Dinosaur (Wasn’t sure if it was going to return anything with all titles only being exhibited once, but it looks like it output the first title in the table).

Special features – {Trailers, Commentaries, “Behind the Scenes”}

SELECT MODE() WITHIN GROUP (ORDER BY [non-numeric category])

AS modal\_value

FROM film

Etc.

2b. Actor Table Summary Statistics

Will check for the MODE of first names and last names.

First name – Julia

Last name – Kilmer

SELECT MODE() WITHIN GROUP (ORDER BY first\_name)

AS "first name mode",

MODE() WITHIN GROUP (ORDER BY last\_name)

AS "last name mode"

FROM actor

3. As of right now, Excel is easier to clean the relatively small amounts of data that it can handle. Turning on a filter and seeing what irregular or missing values in a column is more familiar. I can see that the skill ceiling, or what is achievable with expertise, is higher with SQL though. It’s just a bit mechanical for me right now – for this exercise I mostly had to rely on modifying queries from the lesson to achieve the intended output. Looking forward to developing an internal understanding of the SQL syntax.