Data Immersion Ach 03.06

Ryan Wick – 01/24/2025

Directions

Rockbuster's database engineers have loaded some new data into the database, and your manager has asked you to clean and profile it. Follow the instructions below to complete their request:

- **Check for and clean dirty data:** Find out if the film table and the customer table contain any dirty data, specifically non-uniform or duplicate data, or missing values. Create a new "Answers 3.6" document and copy-paste your queries into it. Next to each query write 2 to 3 sentences explaining how you would clean the data (even if the data is not dirty).
- **Table: film**
  1. **film Duplicate Data:**

Query Query History

```sql
SELECT
       title,
       release_year,
       language_id,
       rental_duration,
       COUNT(*)
FROM film
GROUP BY
       title,
       release_year,
       language_id,
       rental_duration
HAVING COUNT (*) >1
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 59 msec.
0 rows affected.
```

  2. **film Non-Uniform Data:**

Query Query History

```sql
SELECT DISTINCT
       title,
       release_year,
       language_id
       email
FROM film
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 71 msec.
1000 rows affected.
```

3. **film Incorrect Data:**
    - There really isn't much in the way of queries you can do here. Best recommended from the reading is perform queries to find "obvious" issues like someone's age being a negative or in the hundreds. Outside of that you'll need access to the data source.

4. **film Missing Data:**
    - **NOTE**: The query used here is only checking if a field is empty. You could also modify it to very the fields are filled with things like N/A, (a blank space), etc.

Query Query History

```sql
SELECT *
FROM film
WHERE
      film_id IS NULL
      OR title IS NULL
      OR description IS NULL
      OR release_year IS NULL
      OR language_id IS NULL
      OR rental_duration IS NULL
      OR rental_rate IS NULL
      OR length IS NULL
      OR replacement_cost IS NULL
      OR rating IS NULL
      OR last_update IS NULL
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 66 msec.
0 rows affected.
```

- **Table: customer**
    5. **customer Duplicate Data:**

Query Query History

```sql
SELECT
      first_name,
      last_name,
      email,
      COUNT(*)
FROM customer
GROUP BY
      first_name,
      last_name,
      email
HAVING COUNT (*) >1
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 59 msec.
0 rows affected.
```

6. **customer Non-Uniform Data:**

Query Query History

```sql
SELECT DISTINCT
       first_name,
       last_name,
       email
FROM customer
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 78 msec.
599 rows affected.
```

7. **customer Incorrect Data:**
   - There really isn't much in the way of queries you can do here. Best recommended from the reading is perform queries to find "obvious" issues like someone's age being a negative or in the hundreds. Outside of that you'll need access to the data source.

8. **customer Missing Data:**
   - **NOTE:** While I tried to mix it up from the film query be reminded that you can do queries to find if the field is NULL, N/A, etc.

Query Query History

```sql
SELECT *
FROM customer
WHERE
      customer_id = 0
      OR store_id = 0
      OR first_name = ' '
      OR last_name = ' '
      OR email = ' '
      OR address_id = 0
      OR activebool NOT IN ('true', 'false')
      OR create_date = NULL
      OR last_update = NULL
      OR active NOT IN (0,1);
```

Data Output Messages Notifications

```
Successfully run. Total query runtime: 61 msec.
0 rows affected.
```

- **Suggested Cleaning Methods:**
  1. **Duplicate Data:**
     - For this cleaning method, I feel it's straightforward. Should any of the columns come back with a value (meaning there is a duplicate) I would simply create a view to display only unique records for that column.
  2. **Non-Uniform Data:**
     - Utilize the UPDATE statement to help fix non-uniform data.
  3. **Missing Data:**
     - The two main options most people use to correct this issue are:
       1. For rows that have multiple empty fields simply remove them or
       2. Place a zero/random number or 'blank' in empty fields
  4. **Incorrect Data:**
     - Again, outside of really having availability of the source you would at best utilize the UPDATE statement to fix any "obvious" errors.

- **Summarize your data:** Use SQL to calculate descriptive statistics for both the film table and the customer table. For numerical columns, this means finding the minimum, maximum, and average values. For non-numerical columns, calculate the mode value. Copy-paste your SQL queries and their outputs into your answers document.
- **Table: film**
  1. **film Numerical Variables:**

Query Query History

```sql
SELECT
MIN (release_year) AS earliest_released_year,
MAX (release_year) AS leastest_released_year,
AVG (release_year) AS average_released_year,  -- Min, Max, Avg for release_year
MIN (rental_duration) AS minimum_rental_duration,
MAX (rental_duration) AS maximum_rental_duration,
AVG (rental_duration) AS average_rental_duration, -- Min, Max, Avg for rental_duration
MIN(rental_rate) AS minimum_rental_rate,
MAX (rental_rate) AS maximum_rental_rate,
AVG (rental_rate) AS average_rental_rate, -- Min, Max, Avg for rental_rate
MIN (length) AS minimum_fiml_length,
MAX (length) AS maximum_film_length,
AVG (length) AS average_film_length, -- Min, Max, Avg for length
MIN (replacement_cost) AS minimum_replacement_cost,
MAX (replacement_cost) AS maximum_replacement_cost,
AVG (replacement_cost) AS average_replacement_cost -- Min, Max, Avg for replacement_cost
FROM film;
```

earliest_released_year integer	leastest_released_year integer	average_released_year numeric
2006	2006	2006.0000000000000000

minimum_rental_duration smallint	maximum_rental_duration smallint	average_rental_duration numeric
3	7	4.9850000000000000

minimum_rental_rate numeric	maximum_rental_rate numeric	average_rental_rate numeric
0.99	4.99	2.9800000000000000

minimum_fiml_length smallint	maximum_film_length smallint	average_film_length numeric
46	185	115.2720000000000000

minimum_replacement_cost numeric	maximum_replacement_cost numeric	average_replacement_cost numeric
9.99	29.99	19.9840000000000000

2. **film Non-Numerical Variables:**

Query Query History

```sql
SELECT
    MODE() WITHIN GROUP (ORDER BY title) AS modal_title,
    MODE() WITHIN GROUP (ORDER BY description) AS modal_description,
    MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating,
    MODE() WITHIN GROUP (ORDER BY special_features) AS modal_special_features
FROM film
```

	modal_title character varying	modal_description text
1	Academy Dinosaur	A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convention

modal_rating mpaa_rating	modal_special_features text[]
PG-13	{Trailers,Commentaries,"Behind the Scenes"}

- **Table: customer**
  3. **customer Numerical Variables:**
     - **NOTE**: While likely not that useful I did include address_id and active since they are technically numerical variables.

Query Query History

```sql
SELECT
  MIN(customer_id) AS min_customer_id,
  MAX(customer_id) AS max_customer_id,
  AVG(customer_id) AS avg_customer_id, -- min, max, avg of customer_id
  MIN(store_id) AS min_store_id,
  MAX(store_id) AS max_store_id,
  AVG(store_id) AS avg_store_id, -- min, max, store_id
  MIN(address_id) AS min_address_id,
  MAX(address_id) AS max_address_id,
  AVG(address_id) AS avg_address_id, -- min, max, avg of address_id
  MIN(active) AS min_active,
  MAX(active) AS max_active,
  AVG(active) AS avg_active -- min, max, avg of active
FROM Customer;
```

	min_customer_id integer	max_customer_id integer	avg_customer_id numeric	min_store_id smallint	max_store_id smallint	avg_store_id numeric
1	1	599	300.0000000000000000	1	2	1.4557595993322204

min_address_id smallint	max_address_id smallint	avg_address_id numeric	min_active integer	max_active integer	avg_active numeric
5	605	304.7245409015025042	0	1	0.97495826377295492487

4. **customer Non-Numerical Variables:**
    - **NOTE:** While items like email and activebool likely wouldn't be needed I figured since they are considered "non-numerical" that I would throw them in for fun.

```sql
SELECT
MODE () WITHIN GROUP (ORDER BY first_name) AS mode_first_name,
MODE () WITHIN GROUP (ORDER BY last_name) AS mode_last_name,
MODE () WITHIN GROUP (ORDER BY email) AS mode_email,
MODE () WITHIN GROUP (ORDER BY activebool) AS mode_activebool,
MODE () WITHIN GROUP (ORDER BY create_date) AS mode_create_date,
MODE () WITHIN GROUP (ORDER BY last_update) AS mode_last_update
FROM customer
```

	mode_first_name character varying	mode_last_name character varying	mode_email character varying	mode_activebool boolean	mode_create_date date	mode_last_update timestamp without time zone
1	Jamie	Abney	aaron.selby@sakilacustomer.org	true	2006-02-14	2013-05-26 14:49:45.738

- **Reflect on your work:** Back in Achievement 1 you learned about data profiling in Excel. Based on your previous experience, which tool (Excel or SQL) do you think is more effective for data profiling, and why? Consider their respective functions, ease of use, and speed. Write a short paragraph in the running document that you have started.
    1. When considering the effectiveness of Excel or SQL for data profiling there are stark differences. SQL's best qualities is its capability to handle large datasets, complex databases, and perform complex queries. Now if you look at Excel it's best qualities are its familiar interface, pivot tables, and easy filtering. In the end I personally feel SQL is hands down the better out of the two there are still some use cases that Excel could assist in data analysis.

- Save your "Answers 3.6" document as a PDF and upload it here for your tutor to review.