**3.6: Summarizing & Cleaning Data in SQL**
**Prepared by: Rob Rowland**

**Film Table Duplicate Data:**

Query Editor | Query History

```
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 | Explain | Messages | Notifications

| title | release_year | language_id | rental_duration | count |
|---|---|---|---|---|
| character varying (255) | integer | smallint | smallint | bigint |

**Customer Table Duplicate Data:**

Query Editor | Query History

```
SELECT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       COUNT(*)
FROM customer
GROUP BY customer_id,
         store_id,
       first_name,
       last_name,
       email,
       address_id
HAVING COUNT(*) >1;
```

Data Output | Explain | Messages | Notifications

| customer_id | store_id | first_name | last_name | email | address_id | count |
|---|---|---|---|---|---|---|
| [PK] integer | smallint | character varying (45) | character varying (45) | character varying (50) | smallint | bigint |

If either of these tables had duplicate data, I could either a) create a view with unique records using GROUP BY or DISTINCT or b) delete the duplicate record using DELETE. It would likely be most wise to create the view with unique records in case any other users needed to access the duplicate rows for any reason. Permanently deleting a record should be done with care.

**Film Table, Numerical Columns (columns continue to right, not seen on picture):**

Query Editor | Query History

```
SELECT AVG(release_year) AS average_release_year,
       MIN(release_year) AS minimum_release_year,
       MAX(release_year) AS maximum_release_year,
       AVG(rental_duration) AS average_rental_duration,
       MIN(rental_duration) AS minimum_rental_duration,
       MAX(rental_duration) AS maximum_rental_duration,
       AVG(rental_rate) AS average_rental_rate,
       MIN(rental_rate) AS minimum_rental_rate,
       MAX(rental_rate) AS maximum_rental_rate,
       AVG(length) AS average_length,
       MIN(length) AS minimum_length,
       MAX(length) AS maximum_length,
       AVG(replacement_cost) AS average_replacement_cost,
       MIN(replacement_cost) AS minimum_replacement_cost,
       MAX(replacement_cost) AS maximum_replacement_cost
FROM film
```

Data Output | Explain | Messages | Notifications

| | average_release_year numeric | minimum_release_year integer | maximum_release_year integer | average_rental_duration numeric | minimum_rental_duration smallint | maximum_rental_duration smallint | average_rental_rate numeric | minimum_rental_rate numeric | maximum_rental_rate numeric |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2006 | 2006 | 2006 | 4.985 | 3 | 7 | 2.98 | 0.99 | 4.99 |

**Film Table, Non-Numerical Columns:**

```
SELECT mode() WITHIN GROUP (ORDER BY rating) AS modal_rating,
       mode() WITHIN GROUP (ORDER BY title) AS modal_title,
       mode() WITHIN GROUP (ORDER BY language_id) AS modal_language
FROM film;
```

Data Output | Explain | Messages | Notifications

| | modal_rating mpaa_rating | modal_title character varying | modal_language smallint |
|---|---|---|---|
| 1 | PG-13 | Academy Dinosaur | 1 |

**Customer Table, Numerical and Non=Numerical Columns (columns continue to right, not seen on picture):**

Query Editor | Query History

```
1  SELECT MIN(customer_id) AS min_customer_id,
2  MAX(customer_id) AS max_customer_id,
3  AVG(customer_id) AS avg_customer_id,
4  MIN(address_id) AS min_address_id,
5  MAX(address_id) AS max_address_id,
6  AVG(address_id) AS avg_address_id,
7  MIN(store_id) AS min_store_id,
8  MAX(store_id) AS max_store_id,
9  AVG(store_id) AS avg_store_id,
10 MODE() WITHIN GROUP (ORDER BY first_name) AS modal_first_name,
11 MODE() WITHIN GROUP (ORDER BY last_name) AS modal_last_name,
12 MODE() WITHIN GROUP (ORDER BY email) AS modal_email
13 FROM customer;
```

Data Output | Explain | Messages | Notifications

| | min_customer_id integer | max_customer_id integer | avg_customer_id numeric | min_address_id smallint | max_address_id smallint | avg_address_ numeric |
|---|---|---|---|---|---|---|
| 1 | 1 | 599 | 300.0000000000000000 | 5 | 605 | 304.72454 |

I remain more familiar with Excel currently, so I can conduct the data profiling much quicker that way. As data sets get larger, I can certainly see the merit for using SQL. I think that, as I learn and become fluent in SQL, it will be the preferred option for quickly referencing data points from large data sets. However, when trying to build a larger profile involving multiple columns, Excel seems easier to use at this time.