
3.6: Summarizing & Cleaning Data in SQL

Innocent Bayai

26 May 2024

1. 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**. Next to each query write 2 to 3 sentences explaining how you would clean the data (even if the data is not dirty).

DUPLICATE QUERY

film

```
Query Query History
1 SELECT film_id,
2 title,
3 description,
4 release_year,
5 language_id,
6 rental_duration,
7 rental_rate,
8 length,
9 replacement_cost,
10 rating,
11 last_update,
12 special_features,
13 fulltext,
14 COUNT (*)
15 FROM film
16 GROUP BY film_id,
17 title,
18 description,
19 release_year,
20 language_id,
21 rental_duration,
22 rental_rate,
23 length,
24 replacement_cost,
25 rating,
26 last_update,
27 special_features,
28 fulltext
29 HAVING COUNT(*)>1;
```

customer

```
Query Query History
1 SELECT customer_id,
2 store_id,
3 first_name,
4 last_name,
5 email,
6 address_id,
7 activebool,
8 create_date,
9 last_update,
10 active,
11 COUNT (*)
12 FROM customer
13 GROUP BY customer_id,
14 store_id,
15 first_name,
16 last_name,
17 email,
18 address_id,
19 activebool,
20 create_date,
21 last_update,
22 active
23 HAVING COUNT(*)>1;
```

Data Output Messages Notifications

| customer_id | store_id | first_name | last_name | email |
|--------------|----------|------------------------|------------------------|--------------|
| [PK] integer | smallint | character varying (45) | character varying (45) | character va |

Total rows: 0 of 0 Query complete 00:00:00.078

Both tables have no duplicate records (film and customer).

If I were to clean this data, I would do the following:

- i. If granted the authority, I would delete the duplicate records using the **DELETE FROM WHERE** command.
 - ii. Otherwise, I use the view table which allows me view only unique records as deleting records might not be allowed. I use the **SELECT DISTINCT, FROM** command.
-

NON-UNIFORM DATA

customer table

Query Query History

```
1 SELECT DISTINCT customer_id,
2     store_id,
3     first_name,
4     last_name,
5     email,
6     address_id,
7     activebool,
8     create_date,
9     last_update,
10    active
11 FROM customer
12 ORDER BY customer_id;
```

Data Output Messages Notifications

| customer_id | store_id | first_name | last_name |
|--------------|----------|------------------------|------------------------|
| [PK] integer | smallint | character varying (45) | character varying (45) |
| 598 | 598 | Wade | Weirvaile |
| 599 | 599 | Austin | Cintron |

Total rows: 599 of 599 Query complete 00:00:00.103

film table

Query Query History

```
1 SELECT DISTINCT film_id,
2     title,
3     description,
4     release_year,
5     language_id,
6     rental_duration,
7     rental_rate,
8     length,
9     replacement_cost,
10    rating,
11    last_update,
12    special_features,
13    fulltext
14 FROM film
15 ORDER BY film_id;
```

Data Output Messages Notifications

| film_id | title | description |
|--------------|-------------------------|-----------------------|
| [PK] integer | character varying (255) | text |
| 993 | Wrong Behavior | A Emotional Saga of |
| 994 | Wyoming Storm | A Awe-Inspiring Pan |
| 995 | Yentl Idaho | A Amazing Display o |
| 996 | Young Language | A Unbelievable Yarn |
| 997 | Youth Kick | A Touching Drama o |
| 998 | Zhivago Core | A Fateful Yarn of a C |
| 999 | Zoolander Fiction | A Fateful Reflection |
| 1000 | Zorro Ark | A Intrepid Panorama |

Total rows: 1000 of 1000 Query complete 00:00:00.144

The use of SELECT DISTINCT, FROM, ORDER BY commands selects unique records in both tables (customer and film). The commands do not identify duplicate records but returns only unique records thereby. However, the returned records in both cases are equal to the count of records prior running the SQL commands implying that there are no non-uniform records.










If any non-uniform records existed, I would use the UPDATE, SET, WHERE commands to correct the records.

MISSING VALUES

No missing records were found for both tables (film and customer). See screen shots for SQL commands in the next page.

If I found missing values, the best way to fix this problem is to work with only the available data for my analysis if the count of the missing values is significant. Otherwise, I can impute the missing values using the average.

Film

| Query | Query History | |
|--|----------------------------------|---------------------|
| 16 | | |
| 17 | SELECT * | |
| 18 | FROM film | |
| 19 | WHERE (film_id, | |
| 20 | title, | |
| 21 | description, | |
| 22 | release_year, | |
| 23 | language_id, | |
| 24 | rental_duration, | |
| 25 | rental_rate, | |
| 26 | length, | |
| 27 | replacement_cost, | |
| 28 | rating, | |
| 29 | last_update, | |
| 30 | special_features, | |
| 31 | fulltext) | |
| 32 | IS NULL | |
| 33 | ORDER BY film_id; | |
| Data Output | Messages | Notifications |
|          | | |
| film_id [PK] integer | title character varying (255) | description text |












customer

```

35  ✓ SELECT*
36  FROM customer
37  WHERE (customer_id,
38         store_id,
39         first_name,
40         last_name,
41         email,
42         address_id,
43         activebool,
44         create_date,
45         last_update,
46         active)
47  IS NULL
48  ORDER BY customer_id

```

Data Output Messages Notifications

| | | | | | | | |
|---|---|--|--|---|---|---|---|
|  |  |  |  |  |  |  |  |
| | customer_id [PK] integer  | store_id smallint  | first_name character varying (45)  | | | | |

2. 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.

Descriptive statistics: film

| Query | Query History | Scratch Pad |
|---------------------------------|--|---------------------------------|
| 48 | ORDER BY customer_id | |
| 49 | | |
| 50 | SELECT | |
| 51 | MIN(rental_rate) AS min_rent, | |
| 52 | MAX(rental_rate) AS max_rent, | |
| 53 | AVG(rental_rate) AS avg_rent, | |
| 54 | MIN(rental_duration) AS min_rental_duration, | |
| 55 | MAX(rental_duration) AS max_rental_duration, | |
| 56 | AVG(rental_duration) AS avg_rental_duration, | |
| 57 | MIN(length) AS min_length, | |
| 58 | MAX(length) AS max_length, | |
| 59 | AVG(length) AS avg_length, | |
| 60 | MIN(replacement_cost) AS min_replacement_cost, | |
| 61 | MAX(replacement_cost) AS max_replacement_cost | |
| 62 | FROM film; | |
| 63 | | |
| 64 | | |
| 65 | | |
| Data Output | Messages | Notifications |
| min_rent numeric | max_rent numeric | avg_rent numeric |
| min_rental_duration smallint | max_rental_duration smallint | avg_rental_duration numeric |
| min_length smallint | max_length smallint | avg_length numeric |
| min_replacement_cost numeric | max_replacement_cost numeric | avg_replacement_cost numeric |
| 1 | 0.99 | 4.99 |
| | | 2.9800000000000000 |
| | 3 | |
| | 7 | 4.9850000000000000 |
| | 46 | 185 |
| | | 115.27200000000000 |

NB: Customer table doesn't have numerical values hence no descriptive statistics are estimated.

Mode for non-numeric variables in film table

```
77 SELECT
78     MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating,
79     MODE() WITHIN GROUP (ORDER BY release_year) AS modal_release_year,
80     MODE() WITHIN GROUP (ORDER BY language_id) AS modal_language
81 FROM film;
82
83
84
85
```

Data Output Messages Notifications

| | modal_rating mpaa_rating | modal_release_year integer | modal_language smallint |
|---|-----------------------------|-------------------------------|----------------------------|
| 1 | PG-13 | 2006 | 1 |

Total rows: 1 of 1 Query complete 00:00:00.093

Mode for non-numeric variables in customer table

```
83 SELECT
84     MODE () WITHIN GROUP (ORDER BY store_id) AS modal_store,
85     MODE () WITHIN GROUP (ORDER BY activebool) AS modal_activebool
86 FROM customer;
```

Data Output Messages Notifications

| | modal_store smallint | modal_activebool boolean |
|---|-------------------------|-----------------------------|
| 1 | 1 | true |

Total rows: 1 of 1 Query complete 00:00:00.057

3. Reflect on your work: 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.

Excel is for basic data profiling tasks but cannot handle the processing of large amounts of data and can become slow and inefficient when handling complex datasets. With excel, one must have the data open and effect the manipulations with it. However, SQL is more powerful for data profiling as it allows more complex queries to be effected on large data sizes. SQL is also faster and efficient than excel. SQL has more flexibility and control over the profiling process. Once a database is loaded into SQL, one only issues commands and get results. This is different in excel as the data has be interfaced with for each processing requirement. Summarily, SQL is more effective and efficient for data profiling due to its advanced functions, speed and flexibility.