

3.6: SUMMARIZING & CLEANING DATA IN SQL

STEP 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.

- Check for duplicate data in film and customer tables:

The screenshot shows two SQL queries in a query editor. The first query, for the 'film' table, selects columns: film_id, title, description, special_features, fulltext, and count(*). It groups by film_id and filters for records where the count is greater than 1. The second query, for the 'customer' table, selects columns: customer_id, store_id, first_name, last_name, email, address_id, activebool, and active. It groups by customer_id and filters for records where the count is greater than 1. Below each query is a 'Data Output' table showing the results of the queries.

```
51 SELECT film_id,
52 title,
53 description,
54 special_features,
55 fulltext,
56 COUNT(*)
57 FROM film
58 GROUP BY film_id,
59 title,
60 description,
61 special_features,
62 fulltext
63 HAVING COUNT(*)>1
64
```

film_id	title	description	special_features	fulltext	count
[PK] integer	character varying (255)	text	text[]	tsvector	bigint

```
65 SELECT customer_id,
66 store_id,
67 first_name,
68 last_name,
69 email,
70 address_id,
71 activebool,
72 active,
73 COUNT(*)
74 FROM customer
75 GROUP BY customer_id,
76 store_id,
77 first_name,
78 last_name,
79 email,
80 address_id,
81 activebool,
82 active
83 HAVING COUNT(*)>1
84
```

customer_id	store_id	first_name	last_name	email	address_id	activebool	active	count
[PK] integer	smallint	character varying (40)	character varying (40)	character varying (50)	smallint	boolean	boolean	integer

Both queries select needed columns and count all the rows. HAVING COUNT(*)>1 will filter out records where the count is greater than 1, so the query will return a list of duplicate records.

In case that we find duplicate records, we can either delete them (if we have access to) or create a VIEW table where we select only unique records.

- Check for non-uniform data in film and customer tables:

The screenshot shows two SQL queries in a query editor. The first query, for the 'film' table, selects the 'rating' column and filters for distinct values. The second query, for the 'customer' table, selects the 'store_id' column and filters for distinct values. Below each query is a 'Data Output' table showing the results of the queries.

```
85 SELECT DISTINCT rating
86 FROM film
87
```

rating
mpaa_rating
1 PG
2 R
3 NC-17
4 PG-13
5 G

```
89 SELECT DISTINCT store_id
90 FROM customer
```

store_id
smallint
1 1
2 2

To check for non-uniform data, we can use the DISTINCT command and run query for each column in the table. This way we can check for inconsistencies.

In case of non-uniform data, we update the table by setting the values in one format.

- Check for missing values in film and customer tables:

148	SELECT *
149	FROM film
150	WHERE NOT (film IS NOT NULL)

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating	last
[PK] integer	character varying (255)	text	integer	smallint	smallint	numeric (4,2)	smallint	numeric (5,2)	mpaa_rating	time

152	SELECT *
153	FROM customer
154	WHERE NOT (customer IS NOT NULL)

id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	active
integer	smallint	character varying (45)	character varying (45)	character varying (50)	smallint	boolean	date	timestamp without time zone	integer

In case of missing values, we can impute the empty records with estimates. However, a preferred approach would be to omit the columns with high number of missing values in our analysis.

STEP 2

Summarize your data: Use SQL to calculate descriptive statistics for both the film table and the customer table.

Film Data

Query	Query History	Scratch Pad
91		
92	SELECT	
93	MIN(rental_duration) AS min_rental_duration,	
94	MAX(rental_duration) AS max_rental_duration,	
95	AVG(rental_duration) AS avg_rental_duration,	
96		
97	MIN(rental_rate) AS min_rental_rate,	
98	MAX(rental_rate) AS max_rental_rate,	
99	AVG(rental_rate) AS avg_rental_rate,	
100		
101	MIN(length) AS min_length,	
102	MAX(length) AS max_length,	
103	AVG(length) AS avg_lenth,	
104		
105	MIN(replacement_cost) AS min_replacement_cost,	
106	MAX(replacement_cost) AS max_replacement_cost,	
107	AVG(replacement_cost) AS avg_replacement_cost,	
108		
109	MODE() WITHIN GROUP (ORDER BY release_year) AS modal_release_year,	
110	MODE() WITHIN GROUP (ORDER BY language_id) AS modal_language_id,	
111	MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating	

max_length	avg_lenth	min_replacement_cost	max_replacement_cost	avg_replacement_cost	modal_release_year	modal_language_id	modal_rating
smallint	numeric	numeric	numeric	numeric	integer	smallint	mpaa_rating
1	185	115.272000000000000000	9.99	29.99	19.984000000000000000	2006	1

**The whole query is not visible due to size*

Customer Data

```
114 SELECT
115 MIN(customer_id) AS min_customer_id,
116 MAX(customer_id) AS max_customer_id,
117
118 MODE() WITHIN GROUP (ORDER BY first_name) AS modal_first_name,
119 MODE() WITHIN GROUP (ORDER BY store_id) AS modal_store_id,
120 MODE() WITHIN GROUP (ORDER BY activebool) AS modal_activebool
121 FROM customer;
122
```

Data Output Messages Notifications

	min_customer_id integer	max_customer_id integer	modal_first_name character varying	modal_store_id smallint	modal_activebool boolean
1	1	599	Jamie	1	true

**Not all columns were run in the query*

STEP 3

Main advantage of SQL in comparison with Excel is the simplicity and efficiency when working with larger datasets.

When we have smaller datasets, profiling the data using pivot tables seems to be a preferable option.

Once I am used to writing queries and/or have templates to calculate descriptive statistics, the process is easier than Excel.