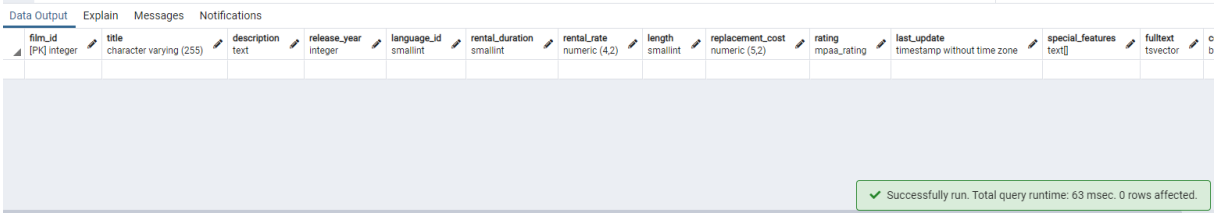
1.1 DUPLICATES

Film

```
SELECT film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext,
       COUNT(*)
FROM film
GROUP BY film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext
HAVING COUNT (*) >1 --no result set means no duplicates
```



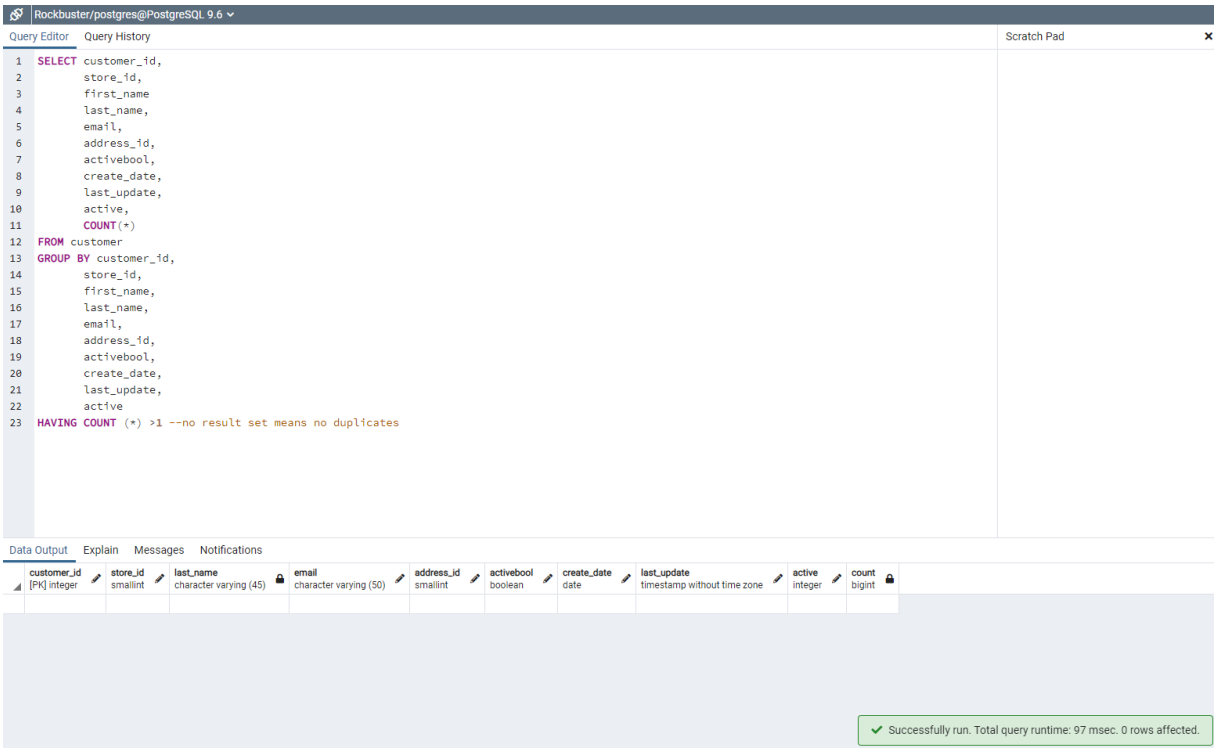
Output: No duplicates

Script

```
SELECT film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext,
       COUNT(*)
FROM film
GROUP BY film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
```

```
        replacement_cost,
        rating,
        last_update,
        special_features,
        fulltext
HAVING COUNT (*) >1 --no result set means no duplicates
```

CUSTOMER



Output: No duplicates

Script

```
SELECT customer_id,
        store_id,
        first_name
        last_name,
        email,
        address_id,
        activebool,
        create_date,
        last_update,
        active,
        COUNT(*)
FROM customer
GROUP BY customer_id,
        store_id,
        first_name,
        last_name,
        email,
        address_id,
```

```
        activebool,
        create_date,
        last_update,
        active
HAVING COUNT (*) >1 --no result set means no duplicates
```

CLEANING DUPLICATES:

After identifying whether or not duplicates exist, I would then create a view to flag the duplicates and be able to delete them like so:

```
CREATE VIEW  AS duplicate_customers
SELECT
  customer_id,
  store_id,
  first_name,
  last_name,
  email,
  address_id,
  activebool,
  create_date,
  last_update,
  active,
  ROW_NUMBER() OVER ( -- ROW_NUMBER creates a column that assigns a number to each row on the view per the established partition, meaning that if there are more than one records of the same, the second record will be labeled as '2' and so forth. Therefore, a record that has no duplicates is only labeled as '1'
    PARTITION BY store_id, first_name, last_name, email  --PARTITION BY would determine how to group the records in the view and single out the duplicates. In this case, the partition would organize the records by store_id, the customer name, and email, thus only focusing on main identifiers of the customer.
    ORDER BY last_update DESC – ORDER BY in this case would order the records in descending order by their last update, meaning that the newest records are at the top and are therefore labeled as '1' per the ROW_NUMBER command. I.e.: the older record will be the duplicate.
  ) AS row_num
FROM customer;
```

After this you would delete the records like so:

```
DELETE FROM duplicate_customers  --this is deleting the records from the view, not the base table
WHERE customer_id IN (
  SELECT customer_id
  FROM duplicate_customers
```

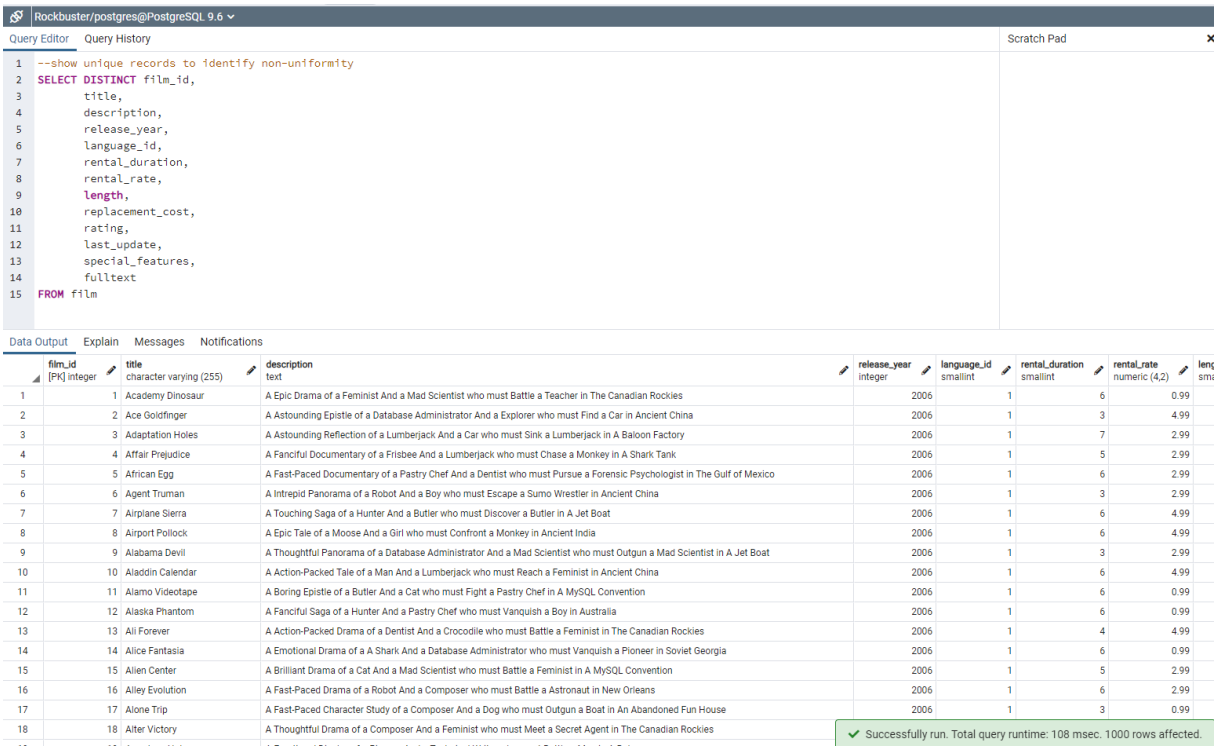
WHERE row_num > 1 --this part of the script is the one that scans for the row_num that is larger than 1, i.e. the duplicates
);

Finally, you run the WHERE x>1 script again to check for duplicates and make sure you got rid of them all.

```
SELECT *
FROM duplicate_customers
WHERE row_num > 1;
```

1.2 NON-UNIFORM DATA

FILM



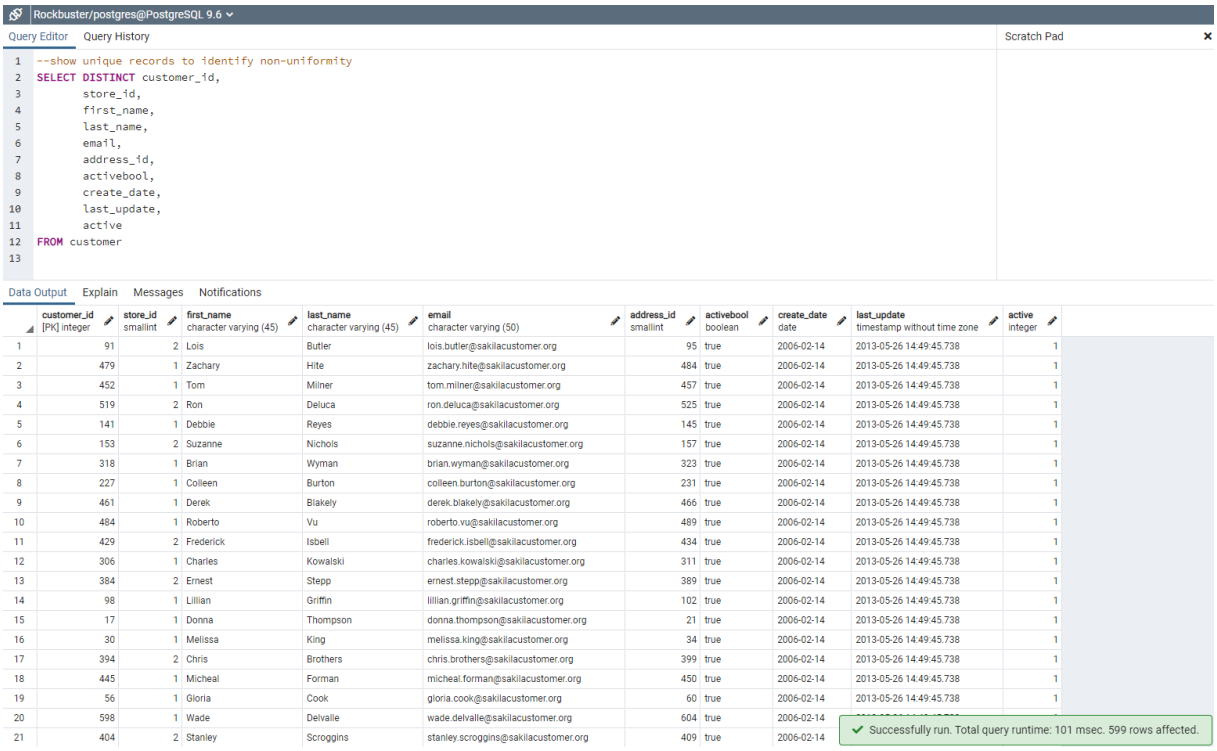
	film_id [PK] integer	title character varying (255)	description text	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)
1	1	Academy Dinosaur	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006	1	6	0.99
2	2	Ace Goldfinger	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006	1	3	4.99
3	3	Adaptation Holes	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	2006	1	7	2.99
4	4	Affair Prejudice	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	2006	1	5	2.99
5	5	African Egg	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico	2006	1	6	2.99
6	6	Agent Truman	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006	1	3	2.99
7	7	Airplane Sierra	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	2006	1	6	4.99
8	8	Airport Pollock	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006	1	6	4.99
9	9	Alabama Devil	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	2006	1	3	2.99
10	10	Aladdin Calendar	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China	2006	1	6	4.99
11	11	Alamo Videotape	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention	2006	1	6	0.99
12	12	Alaska Phantom	A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia	2006	1	6	0.99
13	13	Ali Forever	A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Feminist in The Canadian Rockies	2006	1	4	4.99
14	14	Alice Fantasia	A Emotional Drama of a A Shark And a Database Administrator who must Vanquish a Pioneer in Soviet Georgia	2006	1	6	0.99
15	15	Alien Center	A Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in A MySQL Convention	2006	1	5	2.99
16	16	Alley Evolution	A Fast-Paced Drama of a Robot And a Composer who must Battle a Astronaut in New Orleans	2006	1	6	2.99
17	17	Alone Trip	A Fast-Paced Character Study of a Composer And a Dog who must Outgun a Boat in An Abandoned Fun House	2006	1	3	0.99
18	18	Alter Victory	A Thoughtful Drama of a Composer And a Feminist who must Meet a Secret Agent in The Canadian Rockies				

Script

```
--show unique records to identify non-uniformity
SELECT DISTINCT film_id,
        title,
        description,
        release_year,
        language_id,
        rental_duration,
        rental_rate,
        length,
        replacement_cost,
```

rating,
last_update,
special_features,
fulltext
FROM film

CUSTOMER



Rockbuster/postgres@PostgreSQL 9.6

Query Editor | Query History | Scratch Pad

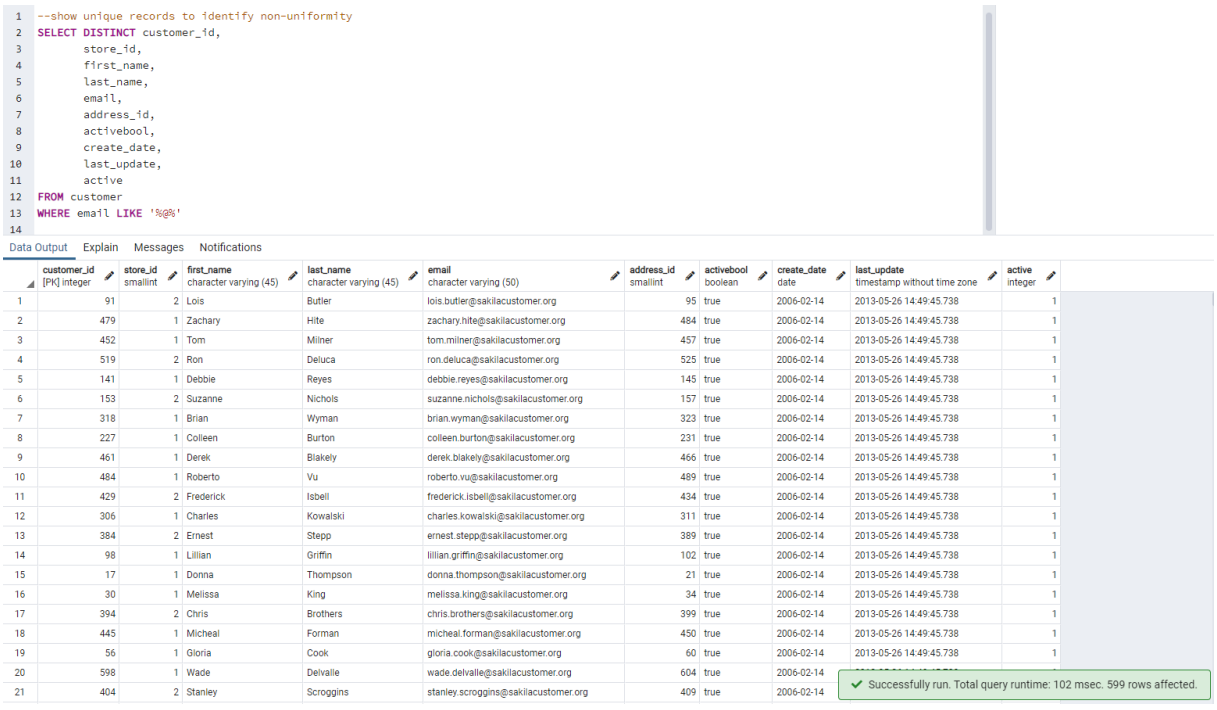
```
--show unique records to identify non-uniformity
SELECT DISTINCT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       create_date,
       last_update,
       active
FROM customer
```

Data Output | Explain | Messages | Notifications

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer
1	91	2	Lois	Butler	lois.butler@sakilacustomer.org	95	true	2006-02-14	2013-05-26 14:49:45.738	1
2	479	1	Zachary	Hite	zachary.hite@sakilacustomer.org	484	true	2006-02-14	2013-05-26 14:49:45.738	1
3	452	1	Tom	Milner	tom.milner@sakilacustomer.org	457	true	2006-02-14	2013-05-26 14:49:45.738	1
4	519	2	Ron	Deluca	ron.deluca@sakilacustomer.org	525	true	2006-02-14	2013-05-26 14:49:45.738	1
5	141	1	Debbie	Reyes	debbie.reyes@sakilacustomer.org	145	true	2006-02-14	2013-05-26 14:49:45.738	1
6	153	2	Suzanne	Nichols	suzanne.nichols@sakilacustomer.org	157	true	2006-02-14	2013-05-26 14:49:45.738	1
7	318	1	Brian	Wyman	brian.wyman@sakilacustomer.org	323	true	2006-02-14	2013-05-26 14:49:45.738	1
8	227	1	Colleen	Burton	colleen.burton@sakilacustomer.org	231	true	2006-02-14	2013-05-26 14:49:45.738	1
9	461	1	Derek	Blakely	derek.blakely@sakilacustomer.org	466	true	2006-02-14	2013-05-26 14:49:45.738	1
10	484	1	Roberto	Vu	roberto.vu@sakilacustomer.org	489	true	2006-02-14	2013-05-26 14:49:45.738	1
11	429	2	Frederick	Isbell	frederick.isbell@sakilacustomer.org	434	true	2006-02-14	2013-05-26 14:49:45.738	1
12	306	1	Charles	Kowalski	charles.kowalski@sakilacustomer.org	311	true	2006-02-14	2013-05-26 14:49:45.738	1
13	384	2	Ernest	Stepp	ernest.stepp@sakilacustomer.org	389	true	2006-02-14	2013-05-26 14:49:45.738	1
14	98	1	Lillian	Griffin	lillian.griffin@sakilacustomer.org	102	true	2006-02-14	2013-05-26 14:49:45.738	1
15	17	1	Donna	Thompson	donna.thompson@sakilacustomer.org	21	true	2006-02-14	2013-05-26 14:49:45.738	1
16	30	1	Melissa	King	melissa.king@sakilacustomer.org	34	true	2006-02-14	2013-05-26 14:49:45.738	1
17	394	2	Chris	Brothers	chris.brothers@sakilacustomer.org	399	true	2006-02-14	2013-05-26 14:49:45.738	1
18	445	1	Micheal	Forman	micheal.forman@sakilacustomer.org	450	true	2006-02-14	2013-05-26 14:49:45.738	1
19	56	1	Gloria	Cook	gloria.cook@sakilacustomer.org	60	true	2006-02-14	2013-05-26 14:49:45.738	1
20	598	1	Wade	Delvalle	wade.delvalle@sakilacustomer.org	604	true	2006-02-14		
21	404	2	Stanley	Scroggins	stanley.scroggins@sakilacustomer.org	409	true	2006-02-14		

Successfully run. Total query runtime: 101 msec. 599 rows affected.

ADDING 'LIKE' command to verify format of emails



```
--show unique records to identify non-uniformity
SELECT DISTINCT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       create_date,
       last_update,
       active
FROM customer
WHERE email LIKE '%@%'
```

Data Output | Explain | Messages | Notifications

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer
1	91	2	Lois	Butler	lois.butler@sakilacustomer.org	95	true	2006-02-14	2013-05-26 14:49:45.738	1
2	479	1	Zachary	Hite	zachary.hite@sakilacustomer.org	484	true	2006-02-14	2013-05-26 14:49:45.738	1
3	452	1	Tom	Milner	tom.milner@sakilacustomer.org	457	true	2006-02-14	2013-05-26 14:49:45.738	1
4	519	2	Ron	Deluca	ron.deluca@sakilacustomer.org	525	true	2006-02-14	2013-05-26 14:49:45.738	1
5	141	1	Debbie	Reyes	debbie.reyes@sakilacustomer.org	145	true	2006-02-14	2013-05-26 14:49:45.738	1
6	153	2	Suzanne	Nichols	suzanne.nichols@sakilacustomer.org	157	true	2006-02-14	2013-05-26 14:49:45.738	1
7	318	1	Brian	Wyman	brian.wyman@sakilacustomer.org	323	true	2006-02-14	2013-05-26 14:49:45.738	1
8	227	1	Colleen	Burton	colleen.burton@sakilacustomer.org	231	true	2006-02-14	2013-05-26 14:49:45.738	1
9	461	1	Derek	Blakely	derek.blakely@sakilacustomer.org	466	true	2006-02-14	2013-05-26 14:49:45.738	1
10	484	1	Roberto	Vu	roberto.vu@sakilacustomer.org	489	true	2006-02-14	2013-05-26 14:49:45.738	1
11	429	2	Frederick	Isbell	frederick.isbell@sakilacustomer.org	434	true	2006-02-14	2013-05-26 14:49:45.738	1
12	306	1	Charles	Kowalski	charles.kowalski@sakilacustomer.org	311	true	2006-02-14	2013-05-26 14:49:45.738	1
13	384	2	Ernest	Stepp	ernest.stepp@sakilacustomer.org	389	true	2006-02-14	2013-05-26 14:49:45.738	1
14	98	1	Lillian	Griffin	lillian.griffin@sakilacustomer.org	102	true	2006-02-14	2013-05-26 14:49:45.738	1
15	17	1	Donna	Thompson	donna.thompson@sakilacustomer.org	21	true	2006-02-14	2013-05-26 14:49:45.738	1
16	30	1	Melissa	King	melissa.king@sakilacustomer.org	34	true	2006-02-14	2013-05-26 14:49:45.738	1
17	394	2	Chris	Brothers	chris.brothers@sakilacustomer.org	399	true	2006-02-14	2013-05-26 14:49:45.738	1
18	445	1	Micheal	Forman	micheal.forman@sakilacustomer.org	450	true	2006-02-14	2013-05-26 14:49:45.738	1
19	56	1	Gloria	Cook	gloria.cook@sakilacustomer.org	60	true	2006-02-14	2013-05-26 14:49:45.738	1
20	598	1	Wade	Delvalle	wade.delvalle@sakilacustomer.org	604	true	2006-02-14		
21	404	2	Stanley	Scroggins	stanley.scroggins@sakilacustomer.org	409	true	2006-02-14		

Successfully run. Total query runtime: 102 msec. 599 rows affected.

Output after LIKE statement: Uniform values (same number of records as without LIKE statment)

SCRIPT

```
--show unique records to identify non-uniformity
SELECT DISTINCT customer_id,
        store_id,
        first_name,
        last_name,
        email,
        address_id,
        activebool,
        create_date,
        last_update,
        active
FROM customer
WHERE email LIKE '%@%' --adding like command to verify that all unique values have a correct email format
```

CLEANING NON-UNIFORM DATA

Like in the example in the reading, after identifying all possible values you would have to UPDATE them to only one:

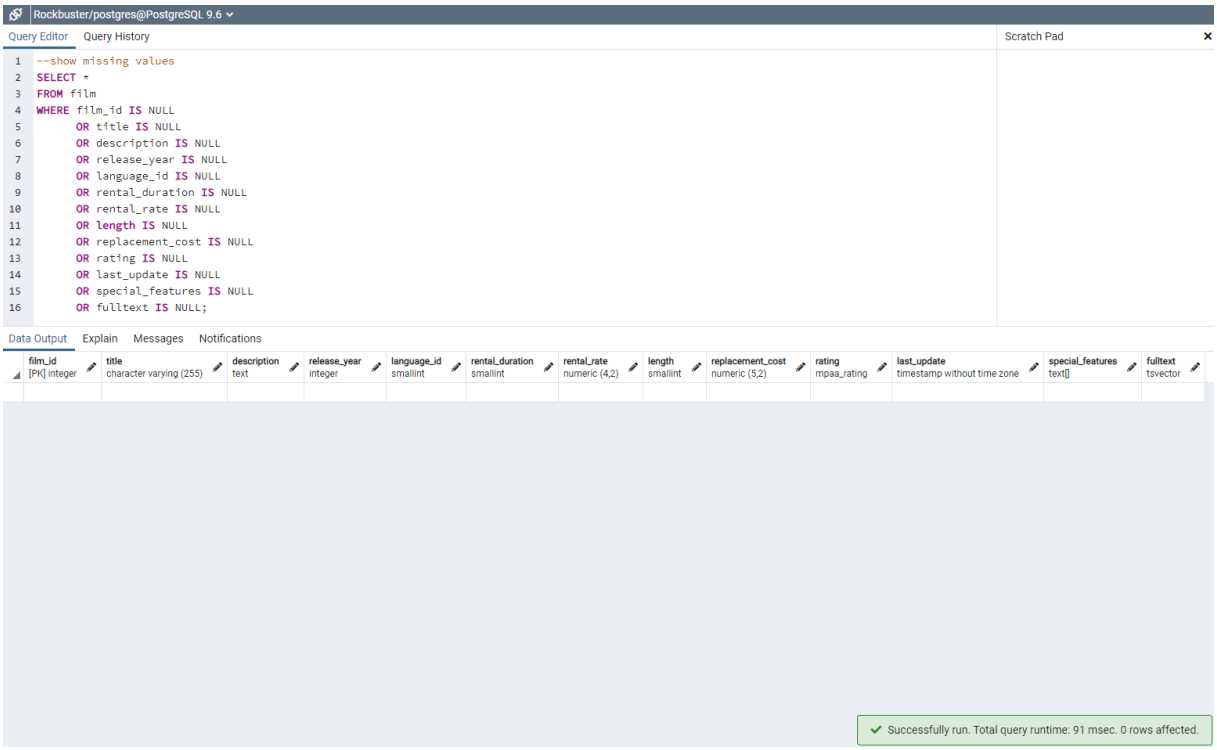
```
UPDATE film
SET rating = 'G'
WHERE rating IN ('gen',
        'g',
        'General')
```

However, you would first need to make sure that the data is inconsistent to begin with using LIKE statements like the one that was used in the script for the customers table. In this table I wanted to make sure that all email addresses were consistent and I tested this by comparing the number of records to the number of records if I applied the LIKE statement searching for the '@' sign.

However, this can only be done with certain columns where the data has categories or structure. For example, if it is numerical data, you can make sure by figuring out if all the data falls within the acceptable range by using "BETWEEN" or "IN" statements. For unstructured data like names and addresses it is more complex.

1.3 NULL VALUES

FILM

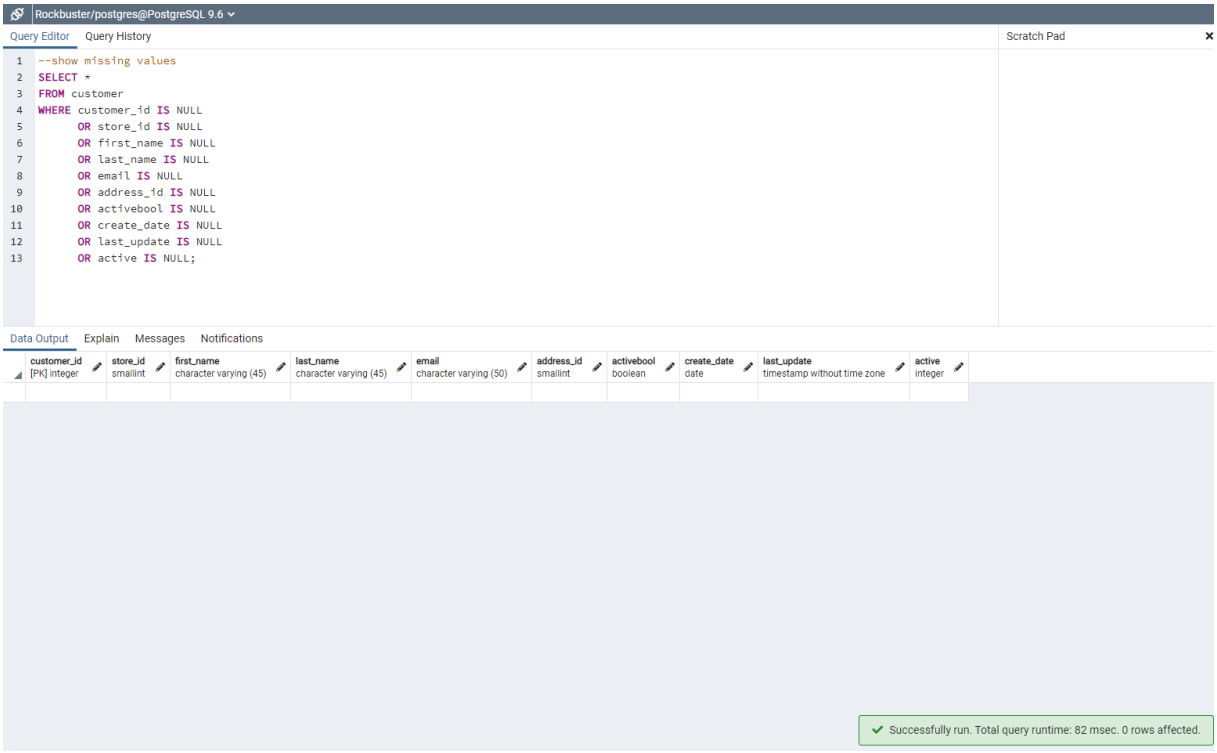


Script

```
--show missing values

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
        OR special_features IS NULL
        OR fulltext IS NULL;
```

CUSTOMER



Script

--show missing values

```
SELECT *
FROM customer
WHERE customer_id IS NULL
        OR store_id IS NULL
        OR first_name IS NULL
        OR last_name IS NULL
        OR email IS NULL
        OR address_id IS NULL
        OR activebool IS NULL
        OR create_date IS NULL
        OR last_update IS NULL
        OR active IS NULL;
```

CLEANING MISSING DATA

Cleaning missing data would be the hardest because we would have to corroborate the data at the source, depending on what type of data is missing. If it is in the customer's table for example, and a name is missing but we have everything else, we can look up in older records with its customer id, or with the email or address. If it's something like the language id in the film table, we can corroborate by consulting the film directly.

Depending on the type of information that is missing the input is either manual or in batches.

If it is in batches, it would be like it was illustrated in the reading:

--imputing missing values with the AVG value

UPDATE tablename
SET = AVG(col1)
WHERE col1 IS NULL

If it were manual and one by one (as with a name, for example), you would have to do like so

--imputing missing value manually

UPDATE customer
SET first_name = 'Emma'
SET last_name = 'Ratri'
WHERE customer_id = 23;

2. SUMMARIZING DATA

2.1 FILM
NUMERICAL VALUES

```sql
SELECT MIN(release_year) AS min_release_year,
       MAX(release_year) AS max_release_year,
       AVG(release_year) AS avg_realease_year,
       COUNT(release_year) AS count_release_year_values,
       MIN(rental_duration) AS min_rental_duration,
       MAX(rental_duration) AS max_rental_duration,
       AVG(rental_duration) AS avg_rental_duration,
       COUNT(rental_duration) AS count_rental_duration_values,
       MIN(rental_rate) AS min_rent,
       MAX(rental_rate) AS max_rent,
       AVG(rental_rate) AS avg_rent,
       COUNT(rental_rate) AS count_rent_values,
       MIN(length) AS min_length,
       MAX(length) AS max_length,
       AVG(length) AS avg_length,
       COUNT(length) AS count_length_values,
       MIN(replacement_cost) AS min_replacement_cost,
       MAX(replacement_cost) AS max_replacement_cost,
       AVG(replacement_cost) AS avg_replacenemt_cost,
       COUNT(replacement_cost) AS count_replacement_cost_values,
       COUNT(*) AS count_rows
FROM film;
```

min_release_year	max_release_year	avg_realease_year	count_release_year_values	min_rental_duration	max_rental_duration	avg_rental_duration
2006	2006	2006.0	1000	3	7	4.985

count_rental_duration_values	min_rent	max_rent	avg_rent	count_rent_values	min_length	max_length
1000	0.99	4.99	2.98	1000	46	185

avg_length	count_length_values	min_replacement_cost	max_replacement_cost	avg_replacenemt_cost	count_replacement_cost_values	count_rows
115.272	1000	9.99	29.99	19.984	1000	1000

NON-NUMERICAL VALUES

```sql
SELECT MODE() WITHIN GROUP (ORDER BY title)
       AS mode_title,
       MODE() WITHIN GROUP (ORDER BY description)
       AS mode_description,
       MODE() WITHIN GROUP (ORDER BY rating)
       AS mode_rating,
       MODE() WITHIN GROUP (ORDER BY last_update)
       AS mode_last_update,
       MODE() WITHIN GROUP (ORDER BY special_features)
       AS mode_special_features,
       MODE() WITHIN GROUP (ORDER BY fulltext)
       AS mode_fulltext
FROM film;
```

mode_title	mode_description	mode_rating	mode_last_update	mode_special_features	mode_fulltext
Academy Dinosaur	A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convention	PG-13	2013-05-26 14:50:58.951	{Trailers,Commentaries,"Behind the Scenes"}	'baloon':19 'confront':14 'documentari':5 'feminist':8,11,16 'mile':2 'must':13 'spi':1 'thrill':4

2.2 CUSTOMER

NUMERICAL

```
1  SELECT MIN(active) AS min_active,
2         MAX(active) AS max_active,
3         AVG(active) AS avg_active,
4         COUNT(active) AS count_active,
5         COUNT(*) AS count_rows
6  FROM customer;
```

min_active	max_active	avg_active	count_active	count_rows
0	1	0.97495826377295492487	599	599

NOTE: There is one non-active customer, meaning a record is probably flagged as deleted.

NON NUMERICAL

```
SELECT MODE() WITHIN GROUP (ORDER BY first_name)
       AS mode_firstname,
       MODE() WITHIN GROUP (ORDER BY last_name)
       AS mode_lastname,
       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	mode_last name	mode_email	mode_activ ebool	mode_create _date	mode_last_u pdate
Jamie	Abney	aaron.selby@sakilacus tomer.org	TRUE	14/02/2006	2013-05-26 14:49:45.738

3. 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.

I believe SQL is a faster and more convenient way of quickly summarizing the data. However, SQL requires that the analyst knows a priori what they want to know—They need to know what to ask. You cannot see the dataset as it comes, so there is no way to make empirical observations before you start programming other than what you may observe at random by running SELECT * FROM table.

This lack of immediate interaction with the data can create a certain bias for whatever the results of this first summarization are.

In brief, SQL is faster and more convenient, but you need to spend more time getting acquainted with the dataset and cleaning it to make sure you are not getting bad results. There is less of an opportunity to randomly catch a mistake without having to troubleshoot the whole process.

Excel, on the other hand, is very interactive and one of its greatest advantages is that you can already visualize some of the summarization with the app's graphs and pivot tables.

However, it takes considerably more time to arrive at the same results. You still spend a lot of time cleaning data as well.

In this sense, both programs are useful for certain purposes, but I think that once you overcome the learning curve, SQL is more convenient.