

# Character data types and common issues

SQL FOR EXPLORATORY DATA ANALYSIS



**Christina Maimone**  
Data Scientist

# PostgreSQL character types

`character(n)` or `char(n)`

- fixed length `n`
- trailing spaces ignored in comparisons

`character varying(n)` or `varchar(n)`

- variable length up to a maximum of `n`

`text` or `varchar`

- unlimited length

# Types of text data

## Categorical

Tues, Tuesday, Mon, TH

shirts, shoes, hats, pants

satisfied, very satisfied, unsatisfied

0349-938, 1254-001, 5477-651

red, blue, green, yellow

## Unstructured Text

I really like this product. I use it every day. It's my favorite color.

We've redesigned your favorite t-shirt to make it even better. You'll love...

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal...

# Grouping and counting

```
SELECT category,          -- categorical variable
       count(*)           -- count rows for each category
FROM product              -- table
GROUP BY category;        -- categorical variable
```

```
category | count
-----+-----
Banana   |      1
Apple    |      4
apple    |      2
apple    |      1
banana   |      3
(5 rows)
```

# Order: most frequent values

```
SELECT category,      -- categorical variable
       count(*)       -- count rows for each category
FROM product          -- table
GROUP BY category     -- categorical variable
ORDER BY count DESC;  -- show most frequent values first
```

```
category | count
-----+-----
Apple    |     4
banana   |     3
apple     |     2
Banana    |     1
apple     |     1
(5 rows)
```

# Order: category value

```
SELECT category,      -- categorical variable
       count(*)       -- count rows for each category
FROM product          -- table
GROUP BY category     -- categorical variable
ORDER BY category;    -- order by categorical variable
```

```
category | count
-----+-----
apple    |      1
Apple    |      4
Banana   |      1
apple    |      2
banana   |      3
(5 rows)
```

# Alphabetical order

```
-- Results
```

category	count
apple	1
Apple	4
Banana	1
apple	2
banana	3

(5 rows)

```
-- Alphabetical Order:
```

```
' ' < 'A' < 'a'
```

```
-- From results
```

```
' ' < 'A' < 'B' < 'a' < 'b'
```

# Common issues

## Case matters

```
'apple' != 'Apple'
```

## Empty strings aren't null

```
'' != NULL
```

## Spaces count

```
' apple' != 'apple'
```

```
'' != ''
```

```
' '
```

## Punctuation differences

```
'to-do' != 'to-do'
```



# Time to examine some text data

SQL FOR EXPLORATORY DATA ANALYSIS

# Cases and Spaces

SQL FOR EXPLORATORY DATA ANALYSIS



**Christina Maimone**  
Data Scientist

# Converting case

```
SELECT lower(' aBc DeFg 7-');
```

```
abc defg 7-
```

```
SELECT upper(' aBc DeFg 7-');
```

```
ABC DEFG 7-
```

# Case insensitive comparisons

```
SELECT *  
FROM fruit;
```

```
customer | fav_fruit  
-----+-----  
349 | apple      <- #1  
874 | Apple      <- #2  
703 | apple      <- #3  
667 | bannana  
622 | banana  
387 | BANANA  
300 | APPLES      <- #4  
313 | apple      <- #5  
499 | banana  
418 | apple      <- #6  
841 | BANANA  
300 | APPLE      <- #7  
754 | apple      <- #8  
(13 rows)
```

```
SELECT *  
FROM fruit  
WHERE lower(fav_fruit)='apple';
```

```
customer | fav_fruit  
-----+-----  
349 | apple  
874 | Apple  
313 | apple  
418 | apple  
300 | APPLE  
(5 rows)
```

# Case insensitive searches

```
-- Using LIKE
```

```
SELECT *  
  FROM fruit  
-- "apple" in value  
WHERE fav_fruit LIKE '%apple%';
```

```
customer | fav_fruit  
-----+-----  
      349 | apple  
      703 | apple  
      313 | apple  
      418 | apple  
      754 | apple  
(5 rows)
```

```
-- Using ILIKE
```

```
SELECT *  
  FROM fruit  
-- ILIKE for case insensitive  
WHERE fav_fruit ILIKE '%apple%';
```

```
customer | fav_fruit  
-----+-----  
      349 | apple  
      874 | Apple  
      703 | apple  
      300 | APPLES  
      313 | apple  
      418 | apple  
      300 | APPLE  
      754 | apple  
(8 rows)
```

# Watch out!

```
SELECT fruit
FROM fruit2;
```

```
fruit
-----
apple
banana
pineapple
grapefruit
grapes
```

```
SELECT fruit
FROM fruit2
WHERE fruit LIKE '%apple%';
```

```
fruit
-----
apple
pineapple
```

# Trimming spaces

```
SELECT trim(' abc ');
```

- `trim` or `btrim` : both ends
  - `trim(' abc ')` = `'abc'`
- `rtrim` : right end
  - `rtrim(' abc ')` = `' abc'`
- `ltrim` : left start
  - `ltrim(' abc ')` = `'abc '`

# Trimming other values

```
SELECT trim('Wow!', '!');
```

Wow

```
SELECT trim('Wow!', '!wW');
```

o



# Combining functions

```
SELECT trim(lower('Wow!'), '!w');
```

0

# Bring order to messy text!

SQL FOR EXPLORATORY DATA ANALYSIS

# Splitting and concatenating text

SQL FOR EXPLORATORY DATA ANALYSIS



**Christina Maimone**  
Data Scientist

# Substring

```
SELECT left('abcde', 2),    -- first 2 characters
       right('abcde', 2);  -- last 2 characters
```

```
left | right
-----+-----
ab   | de
```

```
SELECT left('abc', 10),
       length(left('abc', 10));
```

```
left | length
-----+-----
abc  |      3
```

# Substring

```
SELECT substring(string FROM start FOR length);
```

```
SELECT substring('abcdef' FROM 2 FOR 3);
```

```
bcd
```

```
SELECT substr('abcdef', 2, 3);
```

# Delimiters

```
some text,more text,still more text
```

^

^

```
delimiter delimiter
```

Fields/chunks:

1. some text
2. more text
3. still more text

# Splitting on a delimiter

```
SELECT split_part(string, delimiter, part);
```

```
SELECT split_part('a, bc, d', ',', 2);
```

```
bc
```

# Splitting on a delimiter

```
SELECT split_part('cats and dogs and fish', ' and ', 1);
```

```
cats
```



# Concatenating text

```
SELECT concat('a', 2, 'cc');
```

```
a2cc
```

```
SELECT 'a' || 2 || 'cc';
```

```
a2cc
```

```
SELECT concat('a', NULL, 'cc');
```

```
acc
```

```
SELECT 'a' || NULL || 'cc';
```

# Manipulate some strings!

SQL FOR EXPLORATORY DATA ANALYSIS

# Strategies for Multiple Transformations

SQL FOR EXPLORATORY DATA ANALYSIS



**Christina Maimone**  
Data Scientist

# Multiple transformations

```
SELECT * FROM naics;
```

id	category	businesses
111110	Agriculture: Soybean Farming	4788
111130	Agriculture   Dry Pea and Bean Farming	3606
111140	Agriculture: Wheat Farming	6393
111150	Agriculture - Corn Farming	26469
111160	Agriculture: Rice Farming	949
111199	Agriculture - All Other Grain Farming	15035
111211	Agriculture   Potato Farming	617
611110	Education - Elementary and Secondary	187859
611210	Education   Junior Colleges	3961
611310	Education: Colleges and Universities	29148

# CASE WHEN

```
-- Case for each of :, -, and |
SELECT CASE WHEN category LIKE '%: %' THEN split_part(category, ': ', 1)
         WHEN category LIKE '% - %' THEN split_part(category, ' - ', 1)
         ELSE split_part(category, ' | ', 1)
       END AS major_category, -- alias the result
       sum(businesses)        -- also select number of businesses

FROM naics

GROUP BY major_category;      -- Group by categories created above
```

```
major_category | sum
-----+-----
Education      | 220968
Agriculture    |  57857
```

# Recoding table

Original values: `fruit` table

```
customer | fav_fruit
-----+-----
349 | apple
874 | Apple
703 | apple
667 | bannana
622 | banana
387 | BANANA
300 | APPLES
313 | apple
499 | banana
418 | apple
841 | BANANA
300 | APPLE
754 | apple
```

Standardized values: `recode` table

```
original | standardized
-----+-----
APPLES | apple
apple | apple
Apple | apple
bannana | banana
apple | apple
banana | banana
banana | banana
APPLE | apple
apple | apple
BANANA | banana
```

# Step 1: CREATE TEMP TABLE

```
CREATE TEMP TABLE recode AS

SELECT DISTINCT fav_fruit AS original, -- original, messy values

               fav_fruit AS standardized -- new standardized values

FROM fruit;
```

# Initial table

```
SELECT *  
FROM recode;
```

```
original | standardized  
-----+-----  
APPLES  | APPLES  
apple   | apple  
Apple   | Apple  
bannana | bannana  
apple   | apple  
banana  | banana  
banana  | banana  
APPLE   | APPLE  
apple   | apple  
BANANA  | BANANA  
(10 rows)
```



# Step 2: UPDATE values

```
UPDATE table_name  
  SET column_name = new_value  
 WHERE condition;
```

# Step 2: UPDATE values

```
-- All rows: lower case, remove white space on ends
```

```
UPDATE recode  
  SET standardized=trim(lower(original));
```

```
-- Specific rows: correct a misspelling
```

```
UPDATE recode  
  SET standardized='banana'  
WHERE standardized LIKE '%nn%';
```

```
-- All rows: remove any s
```

```
UPDATE recode  
  SET standardized=rtrim(standardized, 's');
```

# Resulting recode table

```
SELECT *  
FROM recode;
```

```
original | standardized  
-----+-----  
APPLES   | apple  
apple     | apple  
Apple     | apple  
apple     | apple  
banana    | banana  
banana    | banana  
APPLE     | apple  
apple     | apple  
BANANA    | banana  
bannana   | banana  
(10 rows)
```

# Step 3: JOIN original and recode tables

## Original only

```
SELECT fav_fruit, count(*)
FROM fruit
GROUP BY fav_fruit;
```

fav_fruit	count
APPLES	1
apple	1
apple	3
banana	1
BANANA	2
apple	1
APPLE	1
bannana	1
banana	1
Apple	1

(10 rows)

## With recoded values

```
SELECT standardized,
       count(*)
FROM fruit
LEFT JOIN recode
ON fav_fruit=original
GROUP BY standardized;
```

standardized	count
apple	8
banana	5

(2 rows)

# Recap

1. `CREATE TEMP TABLE` with original values
2. `UPDATE` to create standardized values
3. `JOIN` original data to standardized data

# Clean up the Evanston 311 data!

SQL FOR EXPLORATORY DATA ANALYSIS