In this lesson, you will be learning a number of techniques to

1. Clean and re-structure messy data.

2. Convert columns to different data types.

3. Tricks for manipulating NULLs.

This will give you a robust toolkit to get from raw data to clean data that's useful for analysis.

**LEFT & RIGHT**

Three new functions:

**LEFT**

**RIGHT**

**LENGTH**

**LEFT** pulls a specified number of characters for each row in a specified column starting at the beginning (or from the left).

**RIGHT** pulls a specified number of characters for each row in a specified column starting at the end (or from the right).

**LENGTH** provides the number of characters for each row of a specified column.

Example:

SELECT first\_name,

last\_name,

phone\_number,

LEFT(phone\_number, 3) AS area\_code,

RIGHT(phone\_number, 8) AS phone\_number\_only,

RIGHT(phone\_number, LENGTH(phone\_number) - 4) AS phone\_number\_alt

FROM customer\_data;

Exercises:

1. There is much debate about how much the name (or even the first letter of a company name) matters. Use the **accounts** table to pull the first letter of each company name to see the distribution of company names that begin with each letter (or number).

SELECT LEFT(UPPER(name), 1) AS first\_letter, COUNT(\*) num\_companies

FROM accounts

GROUP BY 1

ORDER BY 2 DESC;

2. Use the **accounts** table and a **CASE** statement to create two groups: one group of company names that start with a number and a second group of those company names that start with a letter. What proportion of company names start with a letter?

SELECT SUM(num) nums, SUM(letter) letters

FROM (SELECT name, CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5','6','7','8','9')

THEN 1 ELSE 0 END AS num,

CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5','6','7','8','9')

THEN 0 ELSE 1 END AS letter

FROM accounts) t1;

3. Consider vowels as **a, e, i, o**, and **u**. What proportion of company names start with a vowel, and what percent start with anything else?

SELECT SUM(vowels) vowels, SUM(other) other

FROM (SELECT name, CASE WHEN LEFT(UPPER(name), 1) IN ('A','E','I','O','U')

THEN 1 ELSE 0 END AS vowels,

CASE WHEN LEFT(UPPER(name), 1) IN ('A','E','I','O','U')

THEN 0 ELSE 1 END AS other

FROM accounts) t1;

**POSITION, STRPOS & SUBSTR**

**POSITION** takes a character and a column, and provides the index where that character is for each row. The index of the first position is 1 in SQL. If you come from another programming language, many begin indexing at 0. Here, you saw that you can pull the index of a comma as **POSITION(',' IN city\_state)**.

**STRPOS** provides the same result as **POSITION**, but the syntax for achieving those results is a bit different as shown here: **STRPOS(city\_state, ',')**.

Note, both **POSITION** and **STRPOS** **are case sensitive,** so looking for A is different than looking for a.

Therefore, if you want to pull an index regardless of the case of a letter, you might want to use **LOWER** or **UPPER** to make all of the characters lower or uppercase.

Example:

SELECT first\_name,

last\_name,

city\_state,

POSITION(',' IN city\_state) AS comma\_position,

STRPOS(city\_state, ',') AS substr\_comma\_position,

LOWER(city\_state) AS lowercase,

UPPER(city\_state) AS uppercase,

LEFT(city\_state, POSITION(',' IN city\_state)) AS city

FROM customer\_data;

Exercises:

1. Use the **accounts** table to create **first** and **last** name columns that hold the first and last names for the **primary\_poc**.

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name,

RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name

FROM accounts;

2. Now see if you can do the same thing for every rep **name** in the **sales\_reps** table. Again provide **first** and **last** name columns.

SELECT LEFT(name, STRPOS(name, ' ') -1 ) first\_name,

RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) last\_name

FROM sales\_reps;

**CONCAT**

**CONCAT** combines values from several columns into one column.

In this lesson you will learn about:

1. **CONCAT**

2. **Piping ||**

Each of these will allow you to **combine columns together across rows**. First and last names stored in separate columns could be combined together to create a full name: CONCAT(first\_name, ' ', last\_name) or with piping as first\_name || ' ' || last\_name.

Example:

SELECT first\_name,

last\_name,

CONCAT(first\_name, ' ',last\_name) AS full\_name

first\_name || ' ' || last\_name AS full\_name\_alt

FROM customer\_data;

Exercises:

1. Each company in the **accounts** table wants to create an email address for each **primary\_poc**. The email address should be the first name of the **primary\_poc .** last name **primary\_poc @** company name **.com**.

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', name, '.com')

FROM t1;

2. You may have noticed that in the previous solution some of the company names include spaces, which will certainly not work in an email address. See if you can create an email address that will work by removing all of the spaces in the account **name**, but otherwise your solution should be just as in question **1**.

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', REPLACE(name, ' ', ''), '.com')

FROM t1;

3. We would also like to create an initial password, which they will change after their first log in. The first password will be the first letter of the **primary\_poc**'s first name (lowercase), then the last letter of their first name (lowercase), the first letter of their last name (lowercase), the last letter of their last name (lowercase), the number of letters in their first name, the number of letters in their last name, and then the name of the company they are working with, all capitalized with no spaces.

WITH t1 AS (

SELECT LEFT(primary\_poc, STRPOS(primary\_poc, ' ') -1 ) first\_name, RIGHT(primary\_poc, LENGTH(primary\_poc) - STRPOS(primary\_poc, ' ')) last\_name, name

FROM accounts)

SELECT first\_name, last\_name, CONCAT(first\_name, '.', last\_name, '@', name, '.com'), LEFT(LOWER(first\_name), 1) || RIGHT(LOWER(first\_name), 1) || LEFT(LOWER(last\_name), 1) || RIGHT(LOWER(last\_name), 1) || LENGTH(first\_name) || LENGTH(last\_name) || REPLACE(UPPER(name), ' ', '')

FROM t1;

**CAST**

1. **TO\_DATE**

2. **CAST**

3. Casting with **::**

**DATE\_PART('month', TO\_DATE(month, 'month'))** here changed a month name into the number associated with that particular month.

Then you can change a string to a date using **CAST**. **CAST** is actually useful to change lots of column types. Commonly you might be doing as you saw here, where you **change a string to a date** using **CAST(date\_column AS DATE)**. However, you might want to make other changes to your columns in terms of their data types. You can see other examples here.

In this example, you also saw that instead of **CAST(date\_column AS DATE)**, you can use **date\_column::DATE**.

Expert Tip

Most of the functions presented in this lesson are specific to strings. They won’t work with dates, integers or floating-point numbers. However, using any of these functions will automatically change the data to the appropriate type.

LEFT, RIGHT, and TRIM are all used to select only certain elements of strings, but using them to select elements of a number or date will treat them as strings for the purpose of the function. Though we didn't cover TRIM in this lesson explicitly, it can be used to remove characters from the beginning and end of a string. This can remove unwanted spaces at the beginning or end of a row that often happen with data being moved from Excel or other storage systems.

There are a number of variations of these functions, as well as several other string functions not covered here. Different databases use subtle variations on these functions, so be sure to look up the appropriate database’s syntax if you’re connected to a private database.

Exercises:

1. Write a query to look at the top 10 rows to understand the columns and the raw data in the dataset called **sf\_crime\_data**.

SELECT \*

FROM sf\_crime\_data

LIMIT 10;

2. Look at the **date** column in the **sf\_crime\_data** table. Notice the date is not in the correct format.

SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2)) new\_date

FROM sf\_crime\_data;

3. Write a qury to change the date into the correct SQL date format. You will need to use at least **SUBSTR** and **CONCAT** to perform this operation.

SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2)) new\_date

FROM sf\_crime\_data;

4. Once you have created a column in the correct format, use either **CAST** or **: :** to convert this to a date.

SELECT date orig\_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' || SUBSTR(date, 4, 2))::DATE new\_date

FROM sf\_crime\_data;

Reminder:

**THE CORRECT FORMAT OF DATA IN SQL IS YYY-MM-DD.**

**COALESCE**

In general, COALESCE returns the first non-NULL value passed for each row.

Exercises:

1. Use **COALESCE** to fill in the **accounts.id** column with the **account.id** for the NULL value for the table in 1.

SELECT COALESCE(o.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, o.\*

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

2. Use **COALESCE** to fill in the **orders.account\_id** column with the **account.id** for the NULL value for the table in 1.

SELECT COALESCE(o.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, o.standard\_qty, o.gloss\_qty, o.poster\_qty, o.total, o.standard\_amt\_usd, o.gloss\_amt\_usd, o.poster\_amt\_usd, o.total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

3. Use **COALESCE** to fill in each of the **qty** and **usd** columns with 0 for the table in 1.

SELECT COALESCE(o.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, COALESCE(o.standard\_qty, 0) standard\_qty, COALESCE(o.gloss\_qty,0) gloss\_qty, COALESCE(o.poster\_qty,0) poster\_qty, COALESCE(o.total,0) total, COALESCE(o.standard\_amt\_usd,0) standard\_amt\_usd, COALESCE(o.gloss\_amt\_usd,0) gloss\_amt\_usd, COALESCE(o.poster\_amt\_usd,0) poster\_amt\_usd, COALESCE(o.total\_amt\_usd,0) total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id

WHERE o.total IS NULL;

4. Run the query in 1 with the **WHERE** removed and **COUNT** the number of **id**s.

SELECT COUNT(\*)

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id;

5. Run the query in 5, but with the **COALESCE** function used in question 2 through 4.

SELECT COALESCE(o.id, a.id) filled\_id, a.name, a.website, a.lat, a.long, a.primary\_poc, a.sales\_rep\_id, COALESCE(o.account\_id, a.id) account\_id, o.occurred\_at, COALESCE(o.standard\_qty, 0) standard\_qty, COALESCE(o.gloss\_qty,0) gloss\_qty, COALESCE(o.poster\_qty,0) poster\_qty, COALESCE(o.total,0) total, COALESCE(o.standard\_amt\_usd,0) standard\_amt\_usd, COALESCE(o.gloss\_amt\_usd,0) gloss\_amt\_usd, COALESCE(o.poster\_amt\_usd,0) poster\_amt\_usd, COALESCE(o.total\_amt\_usd,0) total\_amt\_usd

FROM accounts a

LEFT JOIN orders o

ON a.id = o.account\_id;