SQL data cleaning

Here we looked at three new functions:

1. **LEFT**
2. **RIGHT**
3. **LENGTH**

LEFT pulls a specified number of characters for each row in a specified column starting at the beginning (or from the left). As you saw here, you can pull the first three digits of a phone number using **LEFT(phone_number, 3)**.

RIGHT pulls a specified number of characters for each row in a specified column starting at the end (or from the right). As you saw here, you can pull the last eight digits of a phone number using **RIGHT(phone_number, 8)**.

LENGTH provides the number of characters for each row of a specified column. Here, you saw that we could use this to get the length of each phone number as **LENGTH(phone_number)**.

In this lesson, you learned about:

1. **POSITION**
2. **STRPOS**
3. **LOWER**
4. **UPPER**

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.

Quizzes POSITION & STRPOS

You will need to use what you have learned about **LEFT** & **RIGHT**, as well as what you know about **POSITION** or **STRPOS** to do the following quizzes.

1. Use the `accounts` table to create **first** and **last** name columns that hold the first and last names for the `primary_poc`.

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.

POSITION, STRPOS, & SUBSTR Solutions

```
SELECT LEFT(primary_poc, STRPOS(primary_poc, ' ') -1 ) first_name,
RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' ')) last_name
FROM accounts;
```

```
SELECT LEFT(name, STRPOS(name, ' ') -1 ) first_name,
       RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) last_name
FROM sales_reps;
```

In this lesson you learned about:

1. **CONCAT**
2. Piping `||`

Each of these will allow you to combine columns together across rows. In this video, you saw how 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**.

Quizzes CONCAT

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`.

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`. Some helpful documentation is here.

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.

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, '@', name,
'.com')
FROM t1;
```

2.
```
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.
```
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 function

In this video, you saw additional functionality for working with dates including:

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.

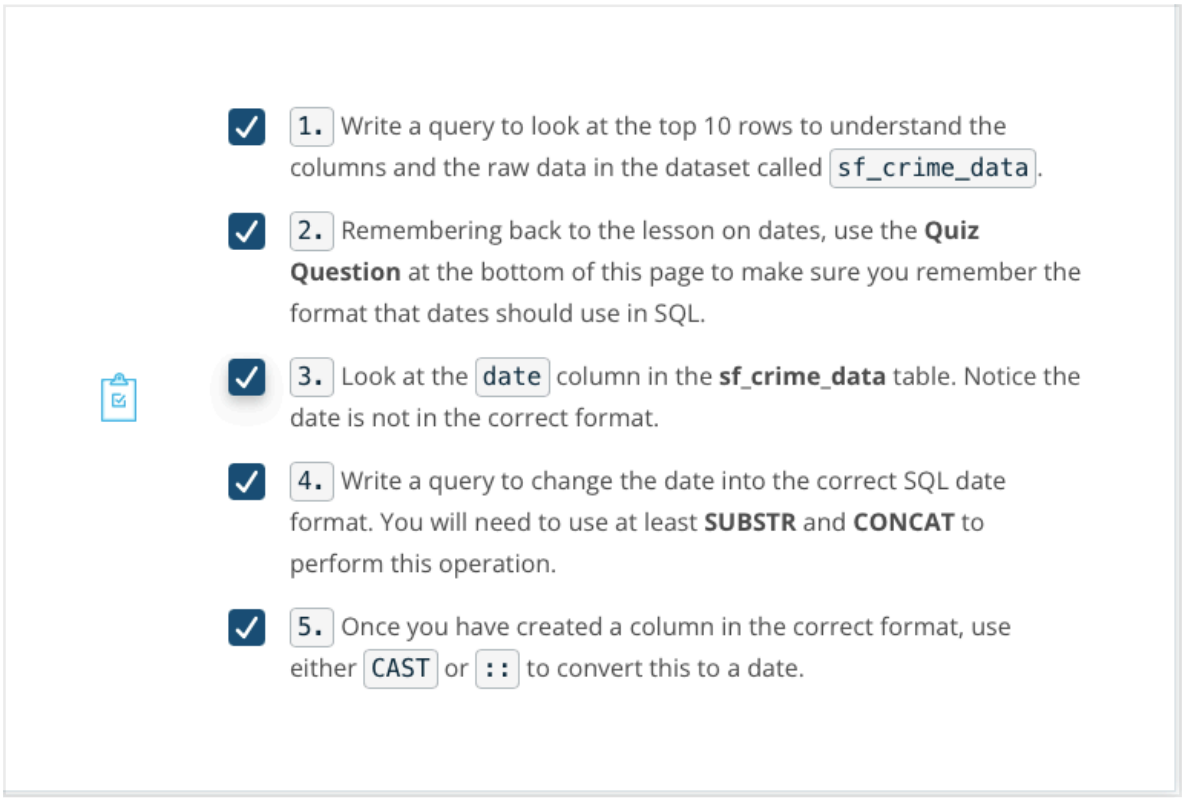
Then you can change a string to a date using **CAST**. **CAST** is actually useful to change lots of column types. Commonly you might be doing as you saw here, where you change a `string` to a `date` using **CAST(date_column AS DATE)**. However, you might want to make other changes to your columns in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



3. **SELECT** date orig_date, (**SUBSTR**(date, 7, 4) || '-' || **LEFT**(date, 2) || '-' || **SUBSTR**(date, 4, 2)) new_date
FROM sf_crime_data;

4. **SELECT** date orig_date, (**SUBSTR**(date, 7, 4) || '-' || **LEFT**(date, 2) || '-' || **SUBSTR**(date, 4, 2))::DATE new_date
FROM sf_crime_data;