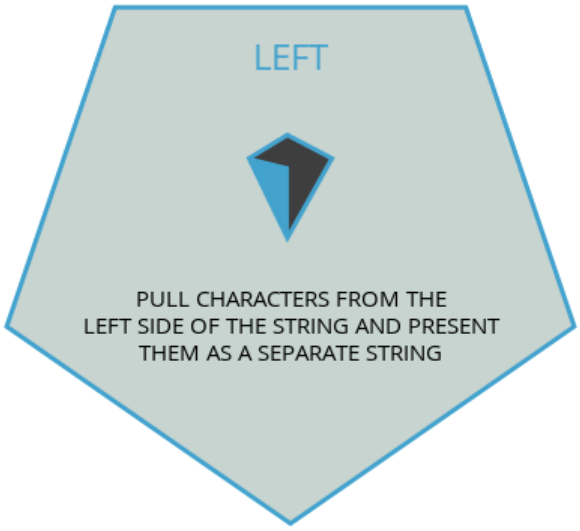
LEFT & RIGHT:-

- **Our goal here is to clean up this data set, to make it more useful for analysis. For text fields, that means making clean groups that will be useful to aggregate across.**

```
SELECT first_name,
       last_name,
       phone_number
FROM demo.customer_data
```

	first_name	last_name	phone_number
1	Alric	Gouny	399-751-5387
2	Thatcher	Buscher	711-549-5882
3	Zak	Gabby	124-829-9663
4	Sheppard	Gatty	216-394-4468
5	Clayborn	Gethyn	769-454-6689
6	Zorina	Goggen	408-635-6341
7	Merrill	Etherington	643-951-8119

- **Let's start by pulling the area code out of the phone number. Since the structure of a phone number in this data set is always the same, we can use that to our advantage. The area code is always the first three characters of a phone number. We can get this using a left function.**

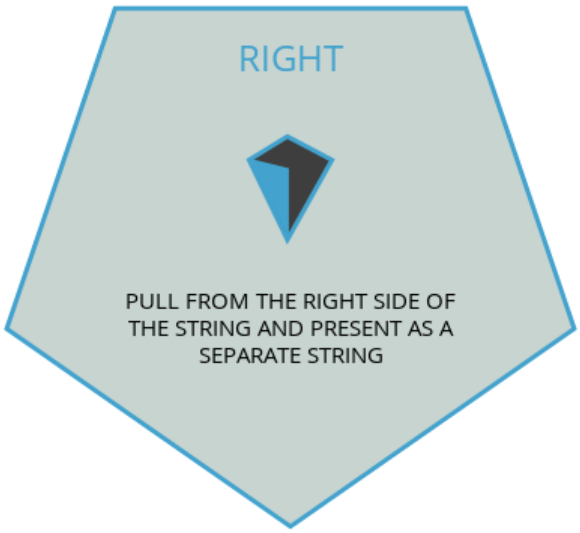


- **You can use left to pull certain number of characters from the left side of the string, and present them as a separate string.**

```
SELECT first_name,
       last_name,
       phone_number,
       LEFT(phone_number, 3) AS area_code
FROM demo.customer_data
```

	first_name	last_name	phone_number	area_code
1	Alric	Gouny	399-751-5387	399
2	Thatcher	Buscher	711-549-5882	711
3	Zak	Gabby	124-829-9663	124
4	Sheppard	Gatty	216-394-4468	216
5	Clayborn	Gethyn	769-454-6689	769
6	Zorina	Goggen	408-635-6341	408
7	Merrill	Etherington	643-951-8119	643

- **If we just want the phone number, we can use right, which does the same thing but from the right side.**

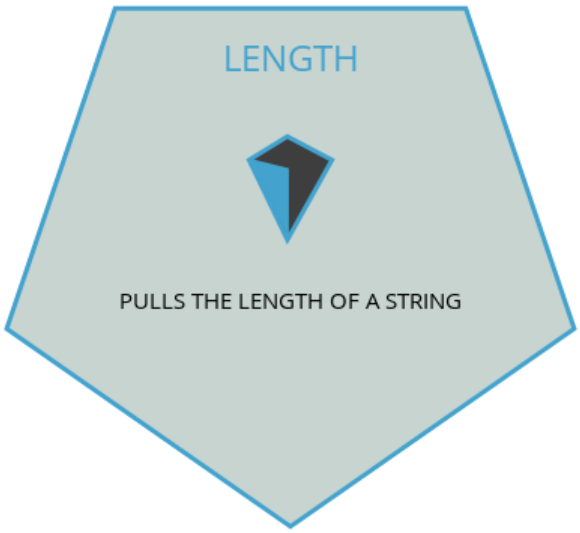


- **Right works well in this case, because we know that the number of characters will be consistent across the entire phone number field. If it wasn't consistent, it's still possible to pull a string from the right side, in a way that makes sense.**

```
SELECT first_name,
       last_name,
       phone_number,
       LEFT(phone_number, 3) AS area_code,
       RIGHT(phone_number, 8) AS phone_number_only
FROM demo.customer_data
```

	first_name	last_name	phone_number	area_code	phone_number_only
1	Alric	Gouny	399-751-5387	399	751-5387
2	Thatcher	Buscher	711-549-5882	711	549-5882
3	Zak	Gabby	124-829-9663	124	829-9663
4	Sheppard	Gatty	216-394-4468	216	394-4468
5	Clayborn	Gethyn	769-454-6689	769	454-6689
6	Zorina	Goggen	408-635-6341	408	635-6341
7	Merrill	Etherington	643-951-8119	643	951-8119

- **The length function returns the length of the string. So, the length of the phone number will always return 12 in this dataset.**



- **Since we know that the first three characters will be the area code, and they'll be followed by a dash. So, total of four characters.**

```
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 demo.customer_data
```

	first_name	last_name	phone_number	area_code	phone_number_only	phone_number_alt
1	Alric	Gouny	399-751-5387	399	751-5387	751-5387
2	Thatcher	Buscher	711-549-5882	711	549-5882	549-5882
3	Zak	Gabby	124-829-9663	124	829-9663	829-9663
4	Sheppard	Gatty	216-394-4468	216	394-4468	394-4468
5	Clayborn	Gethyn	769-454-6689	769	454-6689	454-6689
6	Zorina	Goggen	408-635-6341	408	635-6341	635-6341
7	Merrill	Etherington	643-951-8119	643	951-8119	951-8119

Examples:-

1. **In the accounts table, there is a column holding the website for each company. The last three digits specify what type of web address they are using. A list of extensions (and pricing) is provided here. Pull these extensions and provide how many of each website type exist in the accounts table.**

```
SELECT RIGHT(website, 3) AS domain, COUNT(*) num_companies
FROM accounts
GROUP BY 1
ORDER BY 2 DESC;
```

2. **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;
```

3. **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;
```

4. **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;
```

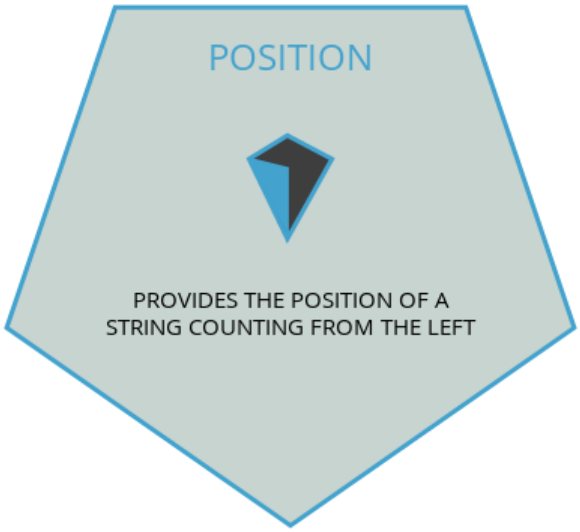
POSITION, STRPOS, & SUBSTR

- **Left and right work pretty well in specific circumstances. When the data is structured very cleanly with a certain number of characters. If you want a separate city and state, you've got to do a little more work. The first thing that will be helpful is figuring out exactly where the city and state split.**

```
SELECT first_name,
       last_name,
       city_state
FROM demo.customer_data
```

	first_name	last_name	city_state
1	Alric	Gouny	Cincinnati, OH
2	Thatcher	Buscher	Anchorage, AK
3	Zak	Gabby	Saginaw, MI
4	Sheppard	Gatty	Jacksonville, FL
5	Clayborn	Gethyn	San Antonio, TX
6	Zorina	Goggen	Alpharetta, GA
7	Merrill	Etherington	Norfolk, VA

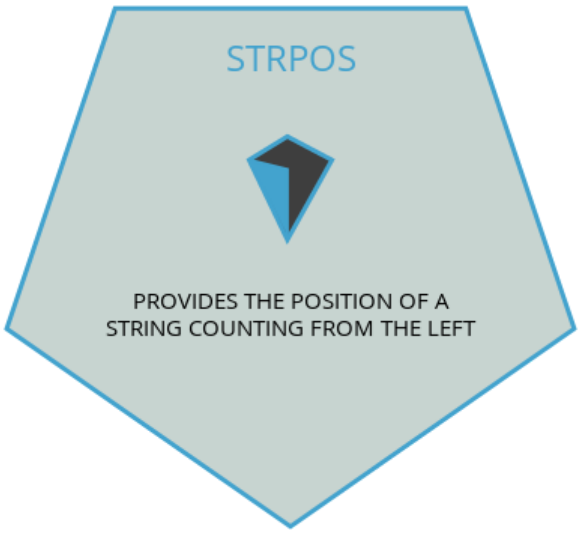
- **Since it will be different for each row, we have to use a function that will find the comma and identify how far into the record it is. Position allows you to specify a sub-string, then it returns a numerical value equal to how far away from the left that particular character appears.**




```
SELECT first_name,
       last_name,
       city_state,
       POSITION(',', IN city_state) AS comma_position
FROM demo.customer_data
```

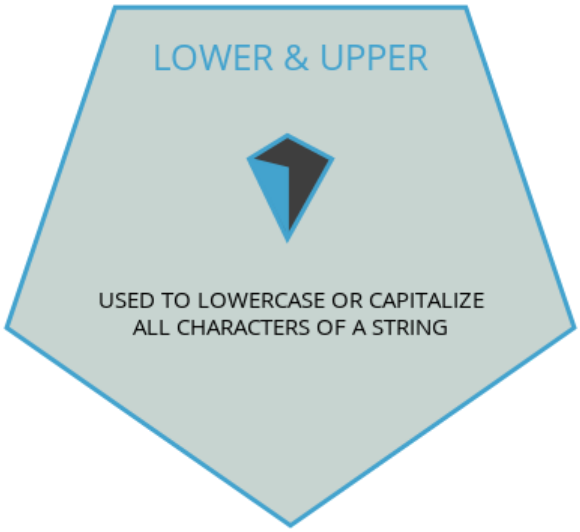
	first_name	last_name	city_state	comma_position
1	Alric	Gouny	Cincinnati, OH	11
2	Thatcher	Buscher	Anchorage, AK	10
3	Zak	Gabby	Saginaw, MI	8
4	Sheppard	Gatty	Jacksonville, FL	13
5	Clayborn	Gethyn	San Antonio, TX	12
6	Zorina	Goggen	Alpharetta, GA	11
7	Merrill	Etherington	Norfolk, VA	8

- **You can also use the string position function which is annotated as as STRPOS to achieve the same results. Just replace in with a comma and switch the order of the string and the sub-string.**




```
SELECT first_name,
       last_name,
       city_state,
       STRPOS(city_state, ',') AS substr_comma_position
FROM demo.customer_data
```

Importantly, both the position and string position functions are case sensitive. If you want to look for a character regardless of its case, you can make the entire string upper or lower case. You can use lower to force every character in a string to become lowercase. Similarly, you can use UPPER to make all the letters appear as uppercase.




```
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
FROM demo.customer_data
```

	first_name	last_name	city_state	comma_position	substr_comma_position	lowercase	uppercase
1	Alric	Gouny	Cincinnati, OH	11	11	cincinnati, oh	CINCINNATI, OH
2	Thatcher	Buscher	Anchorage, AK	10	10	anchorage, ak	ANCHORAGE, AK
3	Zak	Gabby	Saginaw, MI	8	8	saginaw, mi	SAGINAW, MI
4	Sheppard	Gatty	Jacksonville, FL	13	13	jacksonville, fl	JACKSONVILLE, FL
5	Clayborn	Gethyn	San Antonio, TX	12	12	san antonio, tx	SAN ANTONIO, TX
6	Zorina	Goggen	Alpharetta, GA	11	11	alpharetta, ga	ALPHARETTA, GA
7	Merrill	Etherington	Norfolk, VA	8	8	norfolk, va	NORFOLK, VA

- **Let's complete the loop and pull just the city out into its own field. We can do this by nesting the position inside a Left function. As you can see, what we really want isn't the full text up to the position of the comma, we want to end one position before the comma so that it's not included in our city column. We can do this by subtracting one within the left function.**

```
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 demo.customer_data
```

	city_state	comma_position	substr_comma_position	lowercase	uppercase	city
1	Cincinnati, OH	11	11	cincinnati, oh	CINCINNATI, OH	Cincinnati,
2	Anchorage, AK	10	10	anchorage, ak	ANCHORAGE, AK	Anchorage,
3	Saginaw, MI	8	8	saginaw, mi	SAGINAW, MI	Saginaw,
4	Jacksonville, FL	13	13	jacksonville, fl	JACKSONVILLE, FL	Jacksonville,
5	San Antonio, TX	12	12	san antonio, tx	SAN ANTONIO, TX	San Antonio,
6	Alpharetta, GA	11	11	alpharetta, ga	ALPHARETTA, GA	Alpharetta,
7	Norfolk, VA	8	8	norfolk, va	NORFOLK, VA	Norfolk,

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

Examples:-

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, LENGTH(primary_poc) - STRPOS(primary_poc, ' ')) AS
first,
       RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' '))
AS last
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, LENGTH(name) - STRPOS(name, ' ')) AS first,
       RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) AS last
FROM sales_reps
```

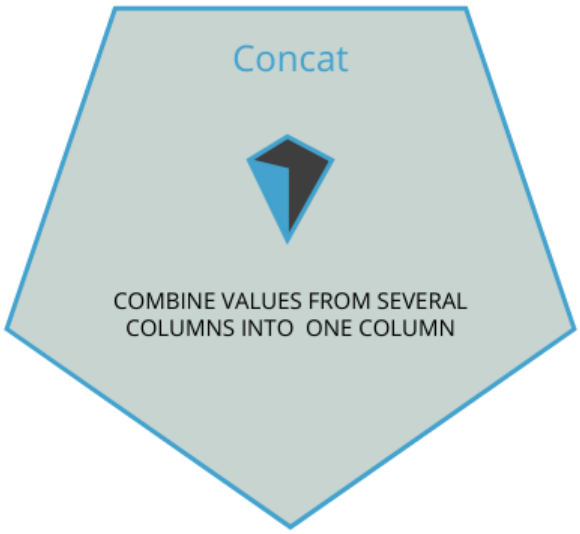
CONCAT

- **Now let's say we want to combine first and last names into a single full name column.**

```
SELECT first_name,
       last_name
FROM demo.customer_data
```

	first_name	last_name
1	Alric	Gouny
2	Thatcher	Buscher
3	Zak	Gabby
4	Sheppard	Gatty
5	Clayborn	Gethyn
6	Zorina	Goggen
7	Merrill	Etherington

- **You can combine strings from several columns using CONCAT.**



- **Simply order the values you want to concatenate and separate them with commas. If you want a hard code values, enclose them in single quotes the way that I'm doing with this space.**

```
SELECT first_name,
       last_name,
       CONCAT(first_name, ' ', last_name) AS full_name
FROM demo.customer_data
```

	first_name	last_name	full_name
1	Alric	Gouny	Alric Gouny
2	Thatcher	Buscher	Thatcher Buscher
3	Zak	Gabby	Zak Gabby
4	Sheppard	Gatty	Sheppard Gatty
5	Clayborn	Gethyn	Clayborn Gethyn
6	Zorina	Goggen	Zorina Goggen
7	Merrill	Etherington	Merrill Etherington

- **Alternatively, you can use two pipe characters to perform the same concatenation.**

```
SELECT first_name,
       last_name,
       CONCAT(first_name, ' ', last_name) AS full_name,
       first_name || ' ' || last_name AS full_name_alt
FROM demo.customer_data
```

	first_name	last_name	full_name	full_name_alt
1	Alric	Gouny	Alric Gouny	Alric Gouny
2	Thatcher	Buscher	Thatcher Buscher	Thatcher Buscher
3	Zak	Gabby	Zak Gabby	Zak Gabby
4	Sheppard	Gatty	Sheppard Gatty	Sheppard Gatty
5	Clayborn	Gethyn	Clayborn Gethyn	Clayborn Gethyn
6	Zorina	Goggen	Zorina Goggen	Zorina Goggen
7	Merrill	Etherington	Merrill Etherington	Merrill Etherington

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
```

Examples:-

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.**

```
SELECT LEFT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' '))
|| '.' || RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, '
'))
|| '@' || name || '.com'
FROM accounts
```

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.**

```
SELECT LEFT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' '))
|| '.' || RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, '
'))
|| '@' || REPLACE(name, ' ', '') || '.com'
FROM accounts
```

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.**

```
SELECT 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) || UPPER(name) AS password
FROM (SELECT LEFT(primary_poc, STRPOS(primary_poc, ' ') - 1) AS first_name,
             RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, '
')) AS last_name,
             name
      FROM accounts) data
```

CAST

- **We've already explored date functions like date trunc and date part. we'll dig into even more functions for getting value from dates. The vast majority of analysis you perform in a professional setting will have dates attached to it, but you might not be able to get value out of them, if the dates are not formatted correctly.**

```
SELECT *
FROM demo.ad_clicks
```

	month	day	year	clicks
1	January	1	2014	1135
2	January	2	2014	602
3	January	3	2014	3704
4	January	4	2014	8781
5	January	5	2014	1021
6	January	6	2014	1341
7	January	7	2014	3395

- **In order to really maximize value here, you'll need to convert month names into numbers, then concatenate all of these fields together along with hyphens, and then tell the database to understand that the resulting output is a date, which you can do by using the cast function. Let's tackle the first couple steps of this first.**

```
SELECT *,
       DATE_PART('month', TO_DATE(month, 'month')) AS clean_month
FROM demo.ad_clicks
```

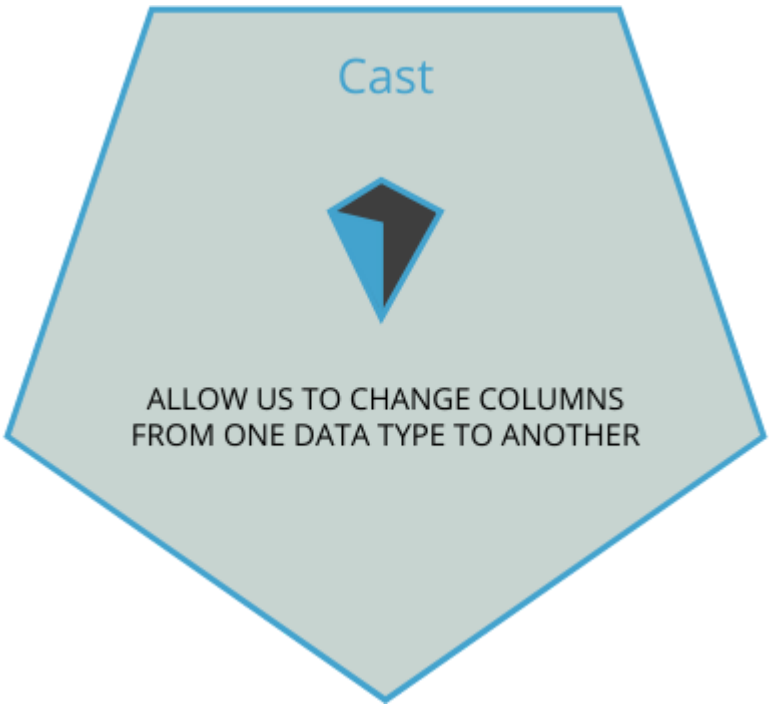
	month	day	year	clicks	clean_month
1	January	1	2014	1135	1
2	January	2	2014	602	1
3	January	3	2014	3704	1
4	January	4	2014	8781	1
5	January	5	2014	1021	1
6	January	6	2014	1341	1
7	January	7	2014	3395	1

- **This first step converted month names into numbers. As you can see, January comes out as one. Now let's concatenate that value together with the year and the day to create something that looks like a real date.**

```
SELECT *,
       DATE_PART('month', TO_DATE(month, 'month')) AS clean_month,
       year || '-' || DATE_PART('month', TO_DATE(month, 'month')) || '-' ||
day AS concatenated_data
FROM demo.ad_clicks
```

	month	day	year	clicks	clean_month	concatenated_date
1	January	1	2014	1135	1	2014-1-1
2	January	2	2014	602	1	2014-1-2
3	January	3	2014	3704	1	2014-1-3
4	January	4	2014	8781	1	2014-1-4
5	January	5	2014	1021	1	2014-1-5
6	January	6	2014	1341	1	2014-1-6
7	January	7	2014	3395	1	2014-1-7

- **For the last step, we'll cast this new field as a date format. You can see here that the database now understands this as a date and has attached the appropriate zeros to these records so that it reads properly.**




```
SELECT *,
       DATE_PART('month', TO_DATE(month, 'month')) AS clean_month,
       year || '-' || DATE_PART('month', TO_DATE(month, 'month')) || '-' ||
day AS concatenated_data,
       CAST(year || '-' || DATE_PART('month', TO_DATE(month, 'month')) ||
'-' || day AS DATE) AS formatted_date
FROM demo.ad_clicks
```

	month	day	year	clicks	clean_month	concatenated_date	formatted_date
1	January	1	2014	1135	1	2014-1-1	2014-01-01
2	January	2	2014	602	1	2014-1-2	2014-01-02
3	January	3	2014	3704	1	2014-1-3	2014-01-03
4	January	4	2014	8781	1	2014-1-4	2014-01-04
5	January	5	2014	1021	1	2014-1-5	2014-01-05
6	January	6	2014	1341	1	2014-1-6	2014-01-06
7	January	7	2014	3395	1	2014-1-7	2014-01-07

- **The cast function can be a little tricky to read, but luckily, there's also a shorthand. Rather than wrapping all of this in a cast function, we can simply add two colons and then the data type we'd like to cast at the end of this section. The cast function is most useful for turning strings into numbers or dates.**

```
SELECT *,
       DATE_PART('month', TO_DATE(month, 'month')) AS clean_month,
       year || '-' || DATE_PART('month', TO_DATE(month, 'month')) || '-' ||
day AS concatenated_data,
       CAST(year || '-' || DATE_PART('month', TO_DATE(month, 'month')) ||
'-' || day AS DATE) AS formatted_date,
       (year || '-' || DATE_PART('month', TO_DATE(month, 'month')) || '-'
|| day)::DATE AS foramtted_date_alt
FROM demo.ad_clicks
```

	month	day	year	clicks	clean_month	concatenated_date	formatted_date	formatted_date_alt
1	January	1	2014	1135	1	2014-1-1	2014-01-01	2014-01-01
2	January	2	2014	602	1	2014-1-2	2014-01-02	2014-01-02
3	January	3	2014	3704	1	2014-1-3	2014-01-03	2014-01-03
4	January	4	2014	8781	1	2014-1-4	2014-01-04	2014-01-04
5	January	5	2014	1021	1	2014-1-5	2014-01-05	2014-01-05
6	January	6	2014	1341	1	2014-1-6	2014-01-06	2014-01-06
7	January	7	2014	3395	1	2014-1-7	2014-01-07	2014-01-07

Examples:-

- [x] `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`.
- [x] `2.` Remembering back to the lesson on dates, use the **Quiz Question** at the bottom of this page to make sure you remember the format that dates should use in SQL.
- [x] `3.` Look at the `date` column in the **sf_crime_data** table. Notice the date is not in the correct format.
- [x] `4.` Write a query to change the date into the correct SQL date format. You will need to use at least **SUBSTR** and **CONCAT** to perform this operation.
- [x] `5.` Once you have created a column in the correct format, use either `CAST` or `::` to convert this to a date.

1.
```
SELECT *
FROM sf_crime_data
LIMIT 10;
```

2. **yyyy-mm-dd**

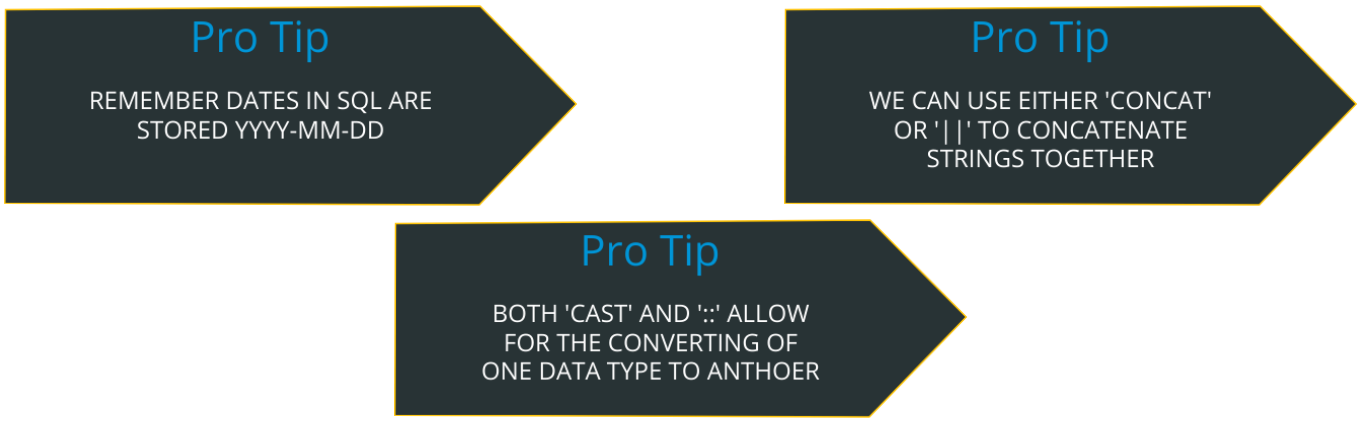
3. **The format of the date column is mm/dd/yyyy with times that are not correct also at the end of the date.**

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

5. **Notice, this new date can be operated on using DATE_TRUNC and DATE_PART in the same way as earlier lessons.**

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

Pro Tips:-



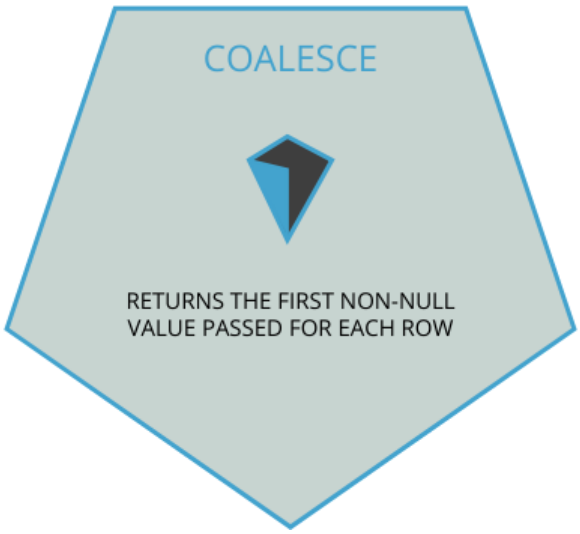
COALESCE

- **Occasionally, you'll end up with a data set that has some nulls that you'd prefer to contain actual values.**

```
SELECT *
FROM demo.accounts
WHERE primary_poc IS NULL
```

	id	name	website	lat	long	primary_poc	sales_rep_id
1	1501	Intel	www.intel.com	41.03153857	-74.66846407		321580
2	1671	Delta Air Lines	www.delta.com	40.75860903	-73.99067048		321510
3	1951	Twenty-First Century Fox	www.21cf.com	42.35467661	-71.05476697		321560
4	2131	USAA	www.usaa.com	41.87745439	-87.62793161		321780
5	2141	Duke Energy	www.duke-energy.com	41.87750558	-87.62754203		321790
6	2151	Time Warner Cable	www.twc.com	41.87655888	-87.63267458		321710
7	2611	Southern	www.southerncompany.com	33.79734976	-84.48783333		321870

- **Looking at the accounts table, you might want to clearly label a no primary point of contact as no POC, so the results will be easily understandable. In cases like this, you can use coalesce to replace the null values.**



- **This is something you may want to do frequently when using numerical data where you might want to display nulls as zero.**

```
SELECT *,
       COALESCE(primary_poc, 'no POC') AS primary_poc_modified
FROM demo.accounts
WHERE primary_poc IS NULL
```

	id	name	website	lat	long	primary_poc	sales_rep_id	primary_poc_modified
1	1501	Intel	www.intel.com	41.03153857	-74.66846407		321580	no POC
2	1671	Delta Air Lines	www.delta.com	40.75860903	-73.99067048		321510	no POC
3	1951	Twenty-First Century Fox	www.21cf.com	42.35467661	-71.05476697		321560	no POC
4	2131	USAA	www.usaa.com	41.87745439	-87.62793161		321780	no POC
5	2141	Duke Energy	www.duke-energy.com	41.87750558	-87.62754203		321790	no POC
6	2151	Time Warner Cable	www.twc.com	41.87655888	-87.63267458		321710	no POC
7	2611	Southern	www.southerncompany.com	33.79734976	-84.48783333		321870	no POC

- **Also, when performing outer joins that result in some unmatched rows, you may want those unmatched rows to display something other than a null value. Of course, this is most valuable when working with a function that treats nulls differently from zero such as a count or an average. We can demonstrate this by wrapping the coalesce in a count function and counting the primary POC column without the coalesce, as well.**

```
SELECT COUNT(primary_poc) AS regular_count,
       COUNT(COALESCE(primary_poc, 'no POC')) AS modified_count
FROM demo.accounts
```

	regular_count	modified_count
1	345	354

Examples:-

- [x] `1.` Run the query entered below in the SQL workspace to notice the row with missing data.
- [x] `2.` Use **COALESCE** to fill in the `accounts.id` column with the `account.id` for the NULL value for the table in `1`.
- [x] `3.` Use **COALESCE** to fill in the `orders.account_id` column with the `account.id` for the NULL value for the table in `1`.
- [x] `4.` Use **COALESCE** to fill in each of the **qty** and **usd** columns with 0 for the table in `1`.
- [x] `5.` Run the query in `1` with the **WHERE** removed and **COUNT** the number of `id`s .
- [x] `6.` Run the query in `5`, but with the **COALESCE** function used in questions `2` through `4`.

1.
```
SELECT *
FROM accounts a
LEFT JOIN orders o
ON a.id = o.account_id
WHERE o.total IS NULL;
```

2.
```
SELECT COALESCE(a.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;
```

3.
```
SELECT COALESCE(a.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,
.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;
```

4.
```
SELECT COALESCE(a.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;
```

5.
```
SELECT COUNT(*)
FROM accounts a
LEFT JOIN orders o
ON a.id = o.account_id;
```

6.
```
SELECT COALESCE(a.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;
```