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 **NULL**s.

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

LEFT & RIGHT Quizzes

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.

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).

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?

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?

LEFT & RIGHT Solutions

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

1.

```
SELECT LEFT(UPPER(name), 1) AS first_letter, COUNT(*) num_companies
FROM accounts
```

```
GROUP BY 1
ORDER BY 2 DESC;
```
2.

There are 350 company names that start with a letter and 1 that starts with a number. This gives a ratio of 350/351 that are company names that start with a letter or 99.7%.

```
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.

There are 80 company names that start with a vowel and 271 that start with other characters. Therefore 80/351 are vowels or 22.8%. Therefore, 77.2% of company names do not start with vowels.

```
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
```
4.
```
         FROM accounts) t1;
```

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;
```

1.

```
SELECT LEFT(name, STRPOS(name, ' ') -1 ) first_name,

       RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) last_name

FROM sales_reps;
```

2.

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.

CONCAT Solutions

```
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;
```

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;
```

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, '@', 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;
```

3.

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.

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.The Postgres literature contains a lot of the related functions.

CAST Quizzes

For this set of quiz questions, you are going to be working with a single table in the environment below. This is a different dataset than Parch & Posey, as all of the data in that particular dataset were already clean.

Tasks to complete:

Task List

- `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`.
- `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.
- `3.` Look at the `date` column in the **sf_crime_data** table. Notice the date is not in the correct format.
- `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.
- `5.` Once you have created a column in the correct format, use either `CAST` or `::` to convert this to a date.

Note: If the proper tables for this SF Crime Data database do not appear for some reason in the Schema below, you can fix this using the Menu in the lower left of the workspace. Make sure you have first saved any query you have entered, then click on Menu, then choose Reset Data, and type in "Reset Data" as directed. This will definitely bring up the proper schema tables for the SF Crime Data database.

CAST Solutions

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

1.
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.

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

4.

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;
```

5.

COALESCE Quizzes

In this quiz, we will walk through the previous example using the following task list. We will use the COALESCE function to complete the orders record for the row in the table output.

Tasks to complete:

Task List

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

COALESCE Solutions

```
SELECT *
FROM accounts a
LEFT JOIN orders o
```

```
ON a.id = o.account_id
WHERE o.total IS NULL;
```

1.

```
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;
```

2.

```
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, 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.

```
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;
```

4.

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

5.

```
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
```

6. `ON a.id = o.account_id;`

You now have a number of tools to assist in cleaning messy data in SQL. Manually cleaning data is tedious, but you now can clean data at scale using your new skills.

For a reminder on any of the data cleaning functionality, the concepts in this lesson are labeled according to the functions you learned. If you felt uncomfortable with any of these functions at first, that is normal - these take some getting used to. Don't be afraid to take a second pass through the material to sharpen your skills!

Memorizing all of this functionality isn't necessary, but you do need to be able to follow documentation, and learn from what you have done in solving previous problems to solve new problems.

There are a few other functions that work similarly. You can read more about those here. You can also get a walk through of many of the functions you have seen throughout this lesson here.

Nice job on this section!

NEXT