Cleaning strings

Most of the functions presented in this lesson are specific to certain data types. However, using a particular function will, in many cases, 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.

LEFT, RIGHT, and LENGTH

Let's start with `LEFT`. You can use `LEFT` to pull a certain number of characters from the left side of a string and present them as a separate string. The syntax is `LEFT(string, number of characters)`. As a practical example, we can see that the `date` field in this dataset begins with a 10-digit date, and include the timestamp to the right of it. The following query pulls out only the ogimage: "/images/og-images/sql-facebook.png" date:

```
1. select incidnt_num,
2.        date,
3.        left(date, 10) as cleaned_date
4.   from tutorial.sf_crime_incidents_2014_01
```

`RIGHT` does the same thing, but from the right side:

```
1. select incidnt_num,
2.        date,
3.        left(date, 10) as cleaned_date,
4.        right(date, 17) as cleaned_time
5.   from tutorial.sf_crime_incidents_2014_01
```

`RIGHT` works well in this case because we know that the number of characters will be consistent across the entire `date` field. If it wasn't consistent, it's still possible to pull a string from the right side in a way that makes sense. The `LENGTH` function returns the length of a string. So `LENGTH(date)` will always return `28` in this dataset. Since we know that the first 10 characters will be the date, and they will be followed by a space (total 11 characters), we could represent the `RIGHT` function like this:

```
1. select incidnt_num,
2.        date,
3.        left(date, 10) as cleaned_date,
4.        right(date, length(date) - 11) as cleaned_time
5.   from tutorial.sf_crime_incidents_2014_01
```

When using functions within other functions, it's important to remember that the innermost functions will be evaluated first, followed by the functions that encapsulate them.

TRIM

The `TRIM` function is used to remove characters from the beginning and end of a string. Here's an example:

```
1. select location,
2.        trim(both '()' from location)
3.   from tutorial.sf_crime_incidents_2014_01
```

The `TRIM` function takes 3 arguments. First, you have to specify whether you want to remove characters from the beginning ('leading'), the end ('trailing'), or both ('both', as used above). Next you must specify all characters to be trimmed. Any characters included in the single quotes will be removed from both beginning, end, or both sides of the string. Finally, you must specify the text you want to trim using `FROM`.

POSITION and STRPOS

`POSITION` allows you to specify a substring, then returns a numerical value equal to the character number (counting from left) where that substring first appears in the target string. For example, the following query will return the position of the character 'A' (case-sensitive) where it first appears in the `descript` field:

```
1. select incidnt_num,
2.        descript,
3.        position('a' in descript) as a_position
4.   from tutorial.sf_crime_incidents_2014_01
```

You can also use the `STRPOS` function to achieve the same results—just replace `IN` with a comma and switch the order of the string and substring:

```
1. select incidnt_num,
2.        descript,
3.        strpos(descript, 'a') as a_position
4.   from tutorial.sf_crime_incidents_2014_01
```

Importantly, both the `POSITION` and `STRPOS` functions are case-sensitive. If you want to look for a character regardless of its case, you can make your entire string a single by using the `UPPER` or `LOWER` functions described below.

SUBSTR

`LEFT` and `RIGHT` both create substrings of a specified length, but they only do so starting from the sides of an existing string. If you want to start in the middle of a string, you can use `SUBSTR`. The syntax is `SUBSTR(*string*, *starting character position*, *# of characters*)`:

```
select incidnt_num,
       date,
       substr(date, 4, 2) as day
  from tutorial.sf_crime_incidents_2014_01
```

PROBLEM

Write a query that separates the `location` field into separate fields for latitude and longitude. You can compare your results against the actual `lat` and `lon` fields in the table.

```
select
      trim ('(' from left(location, position(',' in location)-1)) as latitude,
      trim (')' from right (location, (length(location) - position(',' in location)
      ))) as longitude
from tutorial.sf_crime_incidents_2014_01
```

CONCAT

You can combine strings from several columns together (and with hard-coded values) using `CONCAT`. Simply order the values you want to concatenate and separate them with commas. If you want to hard-code values, enclose them in single quotes. Here's an example:

```
select incidnt_num,
       day_of_week,
       left(date, 10) as cleaned_date,
       concat(day_of_week, ', ', left(date, 10)) as day_and_date
  from tutorial.sf_crime_incidents_2014_01
```

PROBLEM

Concatenate the `lat` and `lon` fields to form a field that is equivalent to the `location` field. (Note that the answer will have a different decimal precision.)

```
SELECT CONCAT('(', lat, ', ', lon, ')') AS concat_location,
       location
FROM tutorial.sf_crime_incidents_2014_01
```

Alternatively, you can use two pipe characters (`||`) to perform the same concatenation:

```
select incidnt_num,
       day_of_week,
       left(date, 10) as cleaned_date,
       day_of_week || ', ' || left(date, 10) as day_and_date
  from tutorial.sf_crime_incidents_2014_01
```

PROBLEM

Create the same concatenated `location` field, but using the `||` syntax instead of `CONCAT`.

```
select '(' || lat || ',' || lon || ')' as concat_location,
       location
from tutorial.sf_crime_incidents_2014_01
```

PROBLEM

Write a query that creates a date column formatted YYYY-MM-DD.

```
select incidnt_num,
       date,
       substr(date, 7, 4) || '-' ||
       substr(date, 1, 2) || '-' ||
       substr(date, 4, 2) as cleaned_date
from tutorial.sf_crime_incidents_2014_01
```

Changing case with UPPER and LOWER

Sometimes, you just don't want your data to look like it's screaming at you. You can use `LOWER` to force every character in a string to become lower-case. Similarly, you can use `UPPER` to make all the letters appear in upper-case:

```
select incidnt_num,
       address,
       upper(address) as address_upper,
       lower(address) as address_lower
  from tutorial.sf_crime_incidents_2014_01
```

PROBLEM

Write a query that returns the `category` field, but with the first letter capitalized and the rest of the letters in lower-case.

```
select incidnt_num,
       category,
       upper(left (category, 1)) ||
       lower(right (category, length(category)-1)) AS category_cleaned
FROM tutorial.sf_crime_incidents_2014_01
```

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. If you're using Mode's public service as in this tutorial, the Postgres literature contains the related functions.

Turning STRINGS into DATES

Dates are some of the most commonly screwed-up formats in SQL. This can be the result of a few things:

- The data was manipulated in Excel at some point, and the dates were changed to MM/DD/YYYY format or another format that is not compliant with SQL's strict standards.
- The data was manually entered by someone who use whatever formatting convention he/she was most familiar with.
- The date uses text (Jan, Feb, etc.) intsead of numbers to record months.

In order to take advantage of all of the great date functionality (`INTERVAL`, as well as some others you will learn in the next section), you need to have your date field formatted appropriately. This often involves some text manipulation, followed by a `CAST`. Let's revisit the answer to one of the practice problems above:

```
1. select incidnt_num,
2.        date,
3.        (substr(date, 7, 4) || '-' ||
4.        left(date, 2) ||
5.        '-' || substr(date, 4, 2))::date as cleaned_date
6.   from tutorial.sf_crime_incidents_2014_01
```

This example is a little different from the answer above in that we've wrapped the entire set of concatenated substrings in parentheses and cast the result in the `date` format. We could also cast it as `timestamp`, which includes additional precision (hours, minutes, seconds). In this case, we're not pulling the hours out of the original field, so we'll just stick to `date`.

PROBLEM

Write a query that creates an accurate timestamp using the `date` and `time` columns in `tutorial.sf_crime_incidents_2014_01`. Include a field that is exactly 1 week later as well.

```
1. SELECT incidnt_num, date, time,
2.        (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) ||
3.         '-' || SUBSTR(date, 4, 2) || ' ' || time || ':00')::timestamp AS timestamp,
4.        (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) ||
5.         '-' || SUBSTR(date, 4, 2) || ' ' || time || ':00')::timestamp
6.         + INTERVAL '1 week' AS timestamp_plus_interval
7.   FROM tutorial.sf_crime_incidents_2014_01
```

Turning DATES into more useful DATES

Once you've got a well-formatted date field, you can manipulate in all sorts of interesting ways. To make the lesson a little cleaner, we'll use a different version of the crime incidents dataset that already has a nicely-formatted date field:

You've learned how to construct a date field, but what if you want to deconstruct one? You can use `EXTRACT` to pull the pieces apart one-by-one:

```
select cleaned_date,
       extract('year'   from cleaned_date) as year,
       extract('month'  from cleaned_date) as month,
       extract('day'    from cleaned_date) as day,
       extract('hour'   from cleaned_date) as hour,
       extract('minute' from cleaned_date) as minute,
       extract('second' from cleaned_date) as second,
       extract('decade' from cleaned_date) as decade,
       extract('dow'    from cleaned_date) as day_of_week
        from tutorial.sf_crime_incidents_cleandate
```

You can also round dates to the nearest unit of measurement. This is particularly useful if you don't care about an individual date, but do care about the week (or month, or quarter) that it occurred in. The `DATE_TRUNC` function rounds a date to whatever precision you specify. The value displayed is the first value in that period. So when you `DATE_TRUNC` by year, any value in that year will be listed as January 1st of that year:

```
select cleaned_date,
       date_trunc('year'   , cleaned_date) as year,
       date_trunc('month'  , cleaned_date) as month,
       date_trunc('week'   , cleaned_date) as week,
       date_trunc('day'    , cleaned_date) as day,
       date_trunc('hour'   , cleaned_date) as hour,
       date_trunc('minute' , cleaned_date) as minute,
       date_trunc('second' , cleaned_date) as second,
       date_trunc('decade' , cleaned_date) as decade
     from tutorial.sf_crime_incidents_cleandate
```

PROBLEM

Write a query that counts the number of incidents reported by week. Cast the week as a date to get rid of the hours/minutes/seconds.

```
select date_trunc('week', cleaned_date)::date as week_beginning,
       count(*) as incidents
  from tutorial.sf_crime_incidents_cleandate
 group by 1
 order by 1
```

What if you want to include today's date or time? You can instruct your query to pull the local date and time at the time the query is run using any number of functions. Interestingly, you can run them without a `FROM` clause:

```
select current_date as date,
       current_time as time,
       current_timestamp as timestamp,
       localtime as localtime,
       localtimestamp as localtimestamp,
       now() as now
```

As you can see, the different options vary in precision. You might notice that these times probably aren't actually your local time. Mode's database is set to Coordinated Universal Time (UTC), which is basically the same as GMT. If you run a current time function against a connected database, you might get a result in a different time zone.

You can make a time appear in a different time zone using `AT TIME ZONE`:

```
select current_time as time,
       current_time at time zone 'pst' as time_pst
```

For a complete list of timezones, look here. This functionality is pretty complex because timestamps can be stored with or without timezone metadata. For a better understanding of the exact syntax, we recommend checking out the Postgres documentation.

PROBLEM

Write a query that shows exactly how long ago each indicent was reported. Assume that the dataset is in Pacific Standard Time (UTC - 8).

```
select incidnt_num,
       cleaned_date,
       now() at time zone 'pst' as now,
       now() at time zone 'pst' - cleaned_date as time_ago
  from tutorial.sf_crime_incidents_cleandate
```

COALESCE

Occasionally, you will end up with a dataset that has some nulls that you'd prefer to contain actual values. This happens frequently in numerical data (displaying nulls as 0 is often preferable), and when performing outer joins that result in some unmatched rows. In cases like this, you can use `COALESCE` to replace the null values:

```
select incidnt_num,
       descript,
       coalesce(descript, 'no description')
  from tutorial.sf_crime_incidents_cleandate
 order by descript desc
```

Mode Report Link: https://app.mode.com/sum14/reports/a56e3103b10c