Chapter 3 : Advanced SQL

Aviral Janveja

1 Data Types

Each column in a database table is required to have a name and a data type. The data type of a column defines what value the column can hold : integer, character, date-time, boolean and it also identifies how SQL will interact with the stored data.

It is certainly best for data to be stored in its optimal format from the beginning, but if it isn't, you can always change it in your query. It is particularly common for dates or numbers, for example, to be stored as strings. This becomes problematic when you want to sum a column and you get an error because SQL is reading numbers as strings. When this happens, you can use **CAST** or **CONVERT** to change the data type to a numeric one that will allow you to perform the sum.

You can actually achieve this with two different types of syntax, both of which produce the same result :

```
CAST(column_name AS integer)
or
column_name::integer
```

2 Date Format

Most relational databases format dates as **YYYY-MM-DD**. This format makes a lot of sense because it sorts in chronological order even when the date field is stored as a string.

2.1 Date Subtraction

The following query uses date subtraction to determine how long it took companies to be acquired. Unacquired companies and those without dates entered were filtered out. Note that because the date column is stored as a string, it must be cast as a timestamp before it can be subtracted from another timestamp :

```
SELECT companies_name,
founded_date,
acquired_date,
acquired_date - founded_date::timestamp AS
time_to_acquisition
```

In the example above, the time_to_acquisition column is an interval, not another date. An interval refers to an integer representing a period of time.

2.2 Interval Function

You can introduce intervals using the interval function as well, for example :

```
SELECT companies_name,
founded_date::timestamp + INTERVAL '1 week' AS
plus_one_week
```

The interval is defined using plain-English terms as shown above, like **10 seconds** or **5 months**. Also note that adding or subtracting a date column and an interval column results in another date column.

2.3 Now Function

You can add the current time at which you run the query, into your code using the now function. For example :

```
SELECT companies_name,
   founded_date,
   NOW() - founded_date::timestamp AS founded_time_ago
```

3 Data Cleaning

Real world data is often messy and messy data can lead to flawed conclusions and a waste of resources. Especially in a world of data-driven decision making, it is vital to ensure that data is clean and prepared for analyses so that we get the most accurate insights to base our business decisions on.

Data cleaning is the process of fixing incorrect, incomplete, duplicate or otherwise erroneous data. These issues can arise from human error during data-entry or from combining different data sources that might be using different terminology. Many of these errors can lead to faulty results, skewing our understanding of the data. Let us therefore look at the following data cleaning functions :

3.1 Left, Right and Length

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). **RIGHT** does the same thing, but from the right side. The **LENGTH** function returns the length of a string. For example :

```
SELECT date,
   LENGTH(date),
   LEFT(date, 10) AS cleaned_date,
   RIGHT(date, 17) AS cleaned_time
```

Note : When using functions within other functions, the innermost function will be evaluated first, followed by the encapsulating functions.

3.2 Trim and From

The **TRIM** function is used to remove characters from the beginning and end of a string. It takes 3 arguments. First specify whether you want to remove characters from the beginning : **leading**, the end : **trailing** or both : **both**. Next you must specify all characters to be trimmed. Any characters included within the single quotes will be removed. Finally, you must specify the text you want to trim using **FROM**. For example :

```
SELECT location,
   TRIM(both '()' FROM location)
```

3.3 Position

POSITION allows you to specify a sub-string and find the numerical value equal to the position where that sub-string first appears in the target string, counting from left. For example, the following query will return the position of the character **A** where it first appears in the descript field :

```
SELECT descript,
   POSITION('A' IN descript) AS "A_position"
```

You can also use the **STRPOS** function to achieve the same result. Just replace IN with a comma and switch the order of the string and sub-string :

```
SELECT descript,
   STRPOS(descript, 'A') AS "A_position"
```

Importantly, both the POSITION and STRPOS functions are **case-sensitive**.

3.4 Sub-strings