In this lesson we are going to be talking about dates. Dates have always been a problem in databases. There are few reasons for that, the first reason is dates can be formatted in many ways.

For example, the American way of formatting a date would be something like this, 2005-10-10 where 2005 is the year, 10 is the month and 10 is the day. That will be 10th of October 2005.

It can also be formatted as 10-10-2005. Where the first 10 is the day, the second 10 is the month and the last number is the year. It can also be formatted as the same way where the first 10 is the month, the second 10 is the day and the third number is the year.

That is one of the problems. Some of the formats are ambiguous, the second problem is dates can have a time zone attached to them. For example, the date right now is 14th of November 11:00 AM, but for some one reading this in the future, the date for him/her would be different. It could be 15th of October 8:00 PM or it could be different for some one reading it from another time zone. So, the current time is not similar for anyone of the users and that is also a problem.

Our database does not necessarily know which time zone is correct for the user. The other thing that is a big issue is daylight saving and when the clock goes forward or backward for an hour, we gain or lose an hour which is unaccounted for and our database does not know that its going to happen. It does not know what to do with that extra hour or that missing hour. So, during those days where the daylight saving occurs, we are going to suddenly have a gap of an hour in our data or we are going to have double the amount of data for that hour, because the hour is going to appear twice.

These three problems are reasonably large problems and they may date a bit of a pain to deal with in databases.

We recommend that we store dates in our database without time zone information and then when a user retrieves data from the database, the database or our Python application can covert that time and date without a time zone information into a date and time with a time zone information.

Say that you are in British Time, which is an hour ahead of the Universal Time, when we save the date to the database, the date is going to be an hour earlier. If we are to save the current time which is 6:00PM, we would want to save 5:00PM, because that is the date time without a time zone attached to it. When we retrieve the data, it should say 6:00PM, but if we go to Russia and retrieve the data then it should say whatever the Russian time is at that moment. This is way of avoiding the problem of time zones by not attaching a time zone to our date and saving all the data in Universal Time. Depending on where our users are, our applications can convert a naïve date time without time zones into a date time zones.

Postgres has few datatypes that is used to save dates and times. They are things like *timestamps*, *date*, *time* and *interval*. These are the four main datatypes.

Let’s say we want to select a timestamp.

We can do select timestamp and the we should give time which it should convert from.

SELECT timestamp ‘2005-10-10 05:16:45’;

|  |
| --- |
| timestamp  timestamp without time zone |
| 2005-10-10 05:16:45 |

We got the output exactly like the input that we put in. This is the format that we should be using as it is the least confusing, where the user would know that the first number is the year, the second number is the month and the third number is the day and then it is hour, then minute and then seconds. The format is called ISO8601.

If we use this problem, it is going to save us from lot of troubles as Postgres is going to use it by default as well. If we want to use a different format, then we need to do a bunch of changes.

Let’s now see hoe we can convert a timestamp into a more readable string.

The first thing is to select *NOW()* and that gives us the current date and time.

SELECT NOW();

|  |
| --- |
| now  timestamp without time zone |
| 2019-11-14 11:48:32 |

If we want to display this string into a nicer format, for example I want to see the day first. The method we need to use to change the format of a date is called *TO\_CHAR*. So, if we want to display our date as day, month and year format then write *TO\_CHAR* followed by opening parentheses and then *NOW()* followed by comma and the in quotations the format that we want the date to be converted to,

SELECT TO\_CHAR(NOW(), ‘DD-MM-YYYY’);

|  |
| --- |
| to\_char  text |
| 14-11-2019 |

SELECT TO\_CHAR(NOW(), ‘DD-MM-YYYY HH:MI:SS’);

|  |
| --- |
| to\_char  text |
| 14-11-2019 11:48:32 |

We can also covert back this format to TIMESTAMP, and the method that we need to use for that is called *TO\_TIMESTAMP.* Instead of using *NOW(),* this timewe need to use the result that was displayed on the table when we were using *TO\_CHAR* to convert the our readable time, thus 14-11-2019 11:48:32 and then the format we want it to be converted to.

SELECT TO\_TIMESTAMP (‘14-11-2019 11:48:32’, ‘YYYY-MM-DD, HH:MI:SS’);

|  |
| --- |
| to\_timestamp  timestamp with time zone |
| 2019-11-14 11:48:32 |

Now it has selected the appropriate timestamp there. This is how we select the current timestamp.