Dates, Timestamps and Timezones - A Comparative study of Oracle and N1QL support for the Date-Time feature : Part 1

Date and Time formats/types are very different for different databases. In the article we shall compare Couchbase N1QL Date-Time functions with Oracle's Date-Time support.

Oracle contains multiple data types associated with date-time support, namely, DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE. The TIMESTAMP data type is an extension of the DATE type.

Date values can be represented either as literals or as numeric values. The literal needs to be in a specified format. The format for the date times can be set using the NLS_DATE_FORMAT, NLS_TIMESTAMP_FORMAT, NLS_TIMESTAMP_TZ_FORMAT and the NLS_DATE_LANGUAGE parameters. (See table below for detailed comparison with examples)

With Couchbase, Date-Time is done a little differently. All Dates and times should be stored as strings that follow the ISO 8091 Extended date time format. N1QL contains DATETIME functions that can be used to and extract these formatted strings. The DATE and TIME for a specific timezone can also be represented as a Unix timestamp in milliseconds. This essentially means that unlike Oracle, where the format of the input Date and time can change based on the values of the NLS_DATE_FORMAT and NLS_TIMESTAMP_FORMAT, the format for dates in Couchbase follows a strict set. (See table below for detailed comparison with examples)

```
For example in oracle,

'2008-DEC-25 17:30' is a valid date
given the NLS_DATE_FORMAT='YYYY-MON-DD HH24:MI'
```

However to represent the same value in Couchbase the user needs to use one of the given formats (see here).

```
This will be 2008-12-25 17:30:00
```

In oracle *fractional_seconds_precision* is optional and specifies the number of digits in the fractional part of the SECOND datetime field. It can be a number (0 to 9) with the default being 6.

```
For example, In oracle the timestamp format can be given as
TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FFF'

Using this we can define the timestamp to be '2006-01-02
15:04:05.999'
```

N1QL has support for fractional seconds similar to oracle. This is seen when using the format - `"2006-01-02T15:04:05.999"`. However N1QL supports 3 digit precision (nanosecond precision) and oracle supports upto 9 digit fractional second precision.

This means that if we specify the date `"2006-01-02T15:04:05.999123456"`, `n1ql will round off to 3 digits and return "2006-01-02T15:04:05.999"`.

```
For N1QL,

SELECT STR_TO_TZ("2006-01-02T15:04:05.999123456",
'America/Los_Angeles') as west;

    "results": [
        {
            "west": "2006-01-02T15:04:05.999"
        }
    ]

For Oracle,

SELECT TO_TIMESTAMP('25-DEC-2008 01:00:00.336123456', 'dd-mon-yyyy
hh:mi:ss.ff') as D from dual;
```

D
`2008-12-25 01:00:00.336123456`

For N1QL, If we specify a more than 9 digits, the Date-Time function returns null.

```
SELECT STR_TO_TZ("2006-01-02T15:04:05.9991234567",
'America/Los_Angeles') as west;

    "results": [
        {
            "west": null
        }
    ]
```

For oracle, if you give more than 9 digits for the fractional seconds part it throws an error - ORA-01830: date format picture ends before converting entire input string

A comparison of the Couchbase N1QL and Oracles date time support is given is a following table.

Oracle's Date and Time datatypes	Couchbase N1QL's date time format support

DATE (Data type)
 	
Format defined by -
● NLS_DATE_FORMAT
● NLS_DATE_LANGUAGE
 	
Example :
The NLS_DATE_FORMAT parameter needs to be set in the initialization parameter file.
`NLS_DATE_FORMAT='YYYY-MON-DD'`
 	
`Sample date`
`DATE '2008-DEC-25'`
	In N1QL there is no specific date/time/timestamp data type. They are all represented by JSON strings with ISO 8091 extended formats and are manipulated using the date time manipulation and arithmetic functions.
	 - (see documentation for list)
	
	Allowed formats : (The date format) *
	YYYY-MM-DD
	
	Example:
	`{`
	`  "Date" : "2008-12-25"`

To set the NLS_DATE_LANGUAGE parameter we can use the ALTER SESSION statement.
`ALTER SESSION SET NLS_DATE_LANGUAGE = FRENCH;`
 	
`SELECT TO_CHAR(SYSDATE, 'Day:Dd Month yyyy') FROM DUAL;`
 	
`TO_CHAR(SYSDATE,'DAY:DDMONTHYYYY')`
`--------------------------------------`
`Lundi :21 Août      2017`
	`}`
	
	In Couchbase/N1QL dates need to be represented in specific formats with day,month and year in numeric form.
	
	For all dates that do not match the input formats, we return the input date in the default format which is YYYY-MM-DDThh:mm:ss.sTZD.
	
	Let us use the function DATE_FORMAT_STR to convert the date format from YYYY-MM-DD to 'YYYY-MON-DD' (which in N1QL is an invalid format)
	
	`select **date_format_str**('2008-12-25', '1111-DEC-12') D;`
	`"results": [ { "D": "2008-12-25T00:00:00-08:00" } ]`
	
	Couchbase supports only ISO Extended date formats. It doesn't support non-numeric dates in multiple languages.

TIMESTAMP (Data Type) Format defined by - ● NLS_TIMESTAMP_FORMAT You can specify the value of NLS_TIMESTAMP_FORMAT by setting it in the initialization parameter file. `NLS_TIMESTAMP_FORMAT  = 'YYYY-MM-DD HH:MI:SS.FF'`	Same functions as Date but with a different input date format (input argument to the function). This needs to be explicitly provided. The timestamp formats are given below - The date format along with timestamp components : ● YYYY-MM-DD hh:mm:ss ● YYYY-MM-DDThh:mm:ss ● YYYY-MM-DD hh:mm:ss.s ● YYYY-MM-DDThh:mm:ss.s Only timestamp based formats - ● hh:mm:ss

We can use the TO_TIMESTAMP function to convert input date-time to timestamp data type.
`SELECT TO_TIMESTAMP('25-DEC-2008 01:00:00.336', 'dd-mon-yyyy hh:mi:ss.ff') from dual;`
 	
`TO_TIMESTAMP('25-DEC-200801:00:00.336','DD-MON-YYYYHH:MI:SS.FF')`
`----------------------------------`
`2008-12-25 01:00:00.336`
	● Hh:mm:ss.s
	
	Couchbase supports nanosecond precision.
	
	For Example:
	`select`
	`date_format_str``('2008-12-25T01:00:00.336', '1111-DEC-12 01:00:00.00') D;`
	`"results": [`
	`    {`
	`     "D": "2008-12-25T01:00:00.336-08:00"`
	`    }`
	`  ]`

TIMESTAMP WITH TIMEZONE
TIMESTAMP WITH LOCAL TIME ZONE
Data type
 	
Format defined by -
● NLS_TIMESTAMP_TZ_FORMAT
 	
You can specify the value of NLS_TIMESTAMP_TZ_FORMAT by setting it in the initialization parameter file.
 	
`NLS_TIMESTAMP_TZ_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF TZH:TZM'`
 	
We can use the TO_TIMESTAMP_TZ function to convert input date-time to timestamp with timezone data type. (It maintains the input timezone)
 	
`SELECT TO_CHAR(TO_TIMESTAMP_TZ('2008-12-25, 01:00:00.336 -08:00', 'yyyy-mm-dd hh:mi:ss.ff TZR')) FROM DUAL;`
	Timezone Manipulation functions - (see documentation for list)
	
	Allowed formats : *
	Including the DateTime components of the format :
	● YYYY-MM-DDThh:mm:ssTZD
	● YYYY-MM-DD hh:mm:ssTZD
	● YYYY-MM-DDThh:mm:ss.sTZD
	● YYYY-MM-DD hh:mm:ss.sTZD
	
	Only timestamp based formats -
	● hh:mm:ss.sTZD
	● hh:mm:ssTZD
	
	With N1QL, in addition to the specific formats we also have specific timezone functions one of which is STR_TO_TZ which converts the input date to the specified timezone.
	
	`SELECT STR_TO_TZ('2008-12-25T01:00:00.336-08:00', 'America/Los_Angeles') as west;`

`TO_CHAR(TO_TIMESTAMP_TZ('2008-12-25 ,01:00:00.336-08:00','YYYY-MM-DDHH:MI:SS.FFTZR'))` ------------------------------------ 25-DEC-08 01.00.00.336000000 AM -08:00	`"results": [ { "west": "2008-12-25T01:00:00.336-08:00" } ]`

Table - Oracle DateTime types VS N1QL ISO Date Formats

* Both Oracle and N1QL automatically determines whether Daylight Saving Time is in effect for a specified time zone and returns the corresponding local time.

** When dealing with the date formats in N1QL, it is important to remember that each ``component of the date time string need to be represented by a valid numeric value. So when passing in the date format to any N1QL functions, we need to pass the date as “2001-12-12” instead of “YYYY-MM-DD”. N1QL only supports the listed formats. Also the Date component of the date-time string has to be separated by “-” and the time components need to be separated by “:”. Otherwise it is not a valid Date.

For any date/time types both Oracle and N1QL store extra information in different fields for the input date. These allow the user to extract specific information about the date.

Oracle’s date-time fields are CENTURY, YEAR, MONTH, DAY, HOUR, MINUTE and SECOND. The TIMESTAMP data types represent seconds as fractional seconds with its precision is determined by the fractional_seconds_precision parameter. It also includes ``the fields TIMEZONE_HOUR, TIMEZONE_MINUTE, TIMEZONE_REGION and TIMEZONE_ABBR. It internally converts the above character values into date values. `The default time for the time component is midnight and the default date for the date component is the first day of the current month. A DATE datatype stores both the date and time information.

In addition to the fields that Oracle supports for its DATE and TIME data-types, N1QL also supports MILLENNIUM, DECADE, QUARTER, WEEK and MILLISECOND. The value of these fields is computed internally using basic arithmetic. N1QL does not support TIMEZONE_REGION and TIMEZONE_ABBR fields.

A detailed comparison for each field is given in the table below.

Let us consider the following sample row for our examples in Oracle :

```
create table t1 (date_purchased  timestamp with time zone );
insert into t1 values (TIMESTAMP '2008-12-25 01:00:00.336 PST');
```

t1

date_purchased
`2008-12-25,01:00:00.336-08:00`

Let us consider the corresponding Couchbase document

Create bucket1 in Couchbase.

```
create primary index on bucket1;
Insert into bucket1 values ("23",
{"date_purchased":"2008-12-25T01:00:00.336-08:00"});
```

Bucket1 - Document id : 23
{
  “date_purchased":"2008-12-25T01:00:00.336-08:00"
}

The TO_CHAR (‘CC’) (with a date as the first arg) and EXTRACT function is used in oracle to retrieve the date-time field values in Oracle. For N1QL there are 2 functions DATE_PART_STR or DATE_PART_MILLIS depending on whether the date is represented as a JSON string or a numeric millisecond. We will use these functions to give examples for each date time field listed below.

Date-Time fields/ parts	Oracle	N1QL
CENTURY	select TO_CHAR(date_purchased, 'CC') from t1;	select DATE_PART_STR(date_purchased,"century") as C from

	`TO_CHAR(DATE_PURCHASED`
	`,'CC')`
	`----------------------`
	`21`
		bucket1;
		
		`"results": [`
		`        {`
		`            "C": 21`
		`        }`
		`    ]`

YEAR	SELECT EXTRACT(year FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`2008`
		select DATE_PART_STR(date_purchased,"year") as C from bucket1;
		
		`"results": [`
		`        {`
		`            "C": 2008`
		`        }`
		`    ]`

MONTH	SELECT EXTRACT(month FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`12`
		select DATE_PART_STR(date_purchased,"month") as C from bucket1;
		
		`"results": [`
		`        {`
		`            "C": 12`
		`        }`
		`    ]`

DAY	SELECT EXTRACT(day FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`25`
		select DATE_PART_STR(date_purchased,"day") as C from bucket1;
		
		`"results": [`
		`        {`
		`            "C": 25`

		`}`
		`]`

HOUR
 	
(different behaviour)
	SELECT EXTRACT(hour FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`9`
	
	Oracle considers the timezone component of the input timestamp.
		select DATE_PART_STR(date_purchased,"hour") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": 1`
		`}`
		`]`
		
		The **difference** in results is because N1QL does not consider the timezone component of the input timestamp.

MINUTE	SELECT EXTRACT(minute FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`0`
		select DATE_PART_STR(date_purchased,"minute") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": 0`
		`}`
		`]`

SECOND	SELECT EXTRACT(second FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`0,3360`
		select DATE_PART_STR(date_purchased,"second") as C from bucket1;
		
		`"results": [`
		`{`

		"C": 0 }] ãkz¥ k¥Yb¨nzxYv ¢Y¥¨ ¦ff wnwn¦fbzxdä

TIMEZONE_HOUR	✓
	
	SELECT EXTRACT(timezone_hour FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`-8`
		✓
		
		select DATE_PART_STR(date_purchased,"timezone_hour") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": -8`
		`}`
		`]`

TIMEZONE_MINUTE	✓
	
	SELECT EXTRACT(timezone_minute FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`0`
		✓
		
		select DATE_PART_STR(date_purchased,"timezone_minute") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": 0`
		`}`
		`]`

TIMEZONE_REGION	✓
	
	SELECT EXTRACT(timezone_region FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`PST`
		✗

TIMEZONE_ABBR	✓	✗

	SELECT EXTRACT(timezone_abbr FROM date_purchased) "D" from t1;
	
	`D`
	`--------`
	`PST`	

TIMEZONE (offset from UTC)	ã , Y¨Y ¨±¢f n¨¦ fvkË xz¨ Y knfvdä	✓
		
		select DATE_PART_STR(date_purchased,"timezone") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": -28800`
		`}`
		`]`
		
		The number here represents the timezone in seconds.

MILLENNIUM	✗	✓
		Millennium = (Year / 1000) + 1
		
		select DATE_PART_STR(date_purchased,"millennium") as D from bucket1;
		
		`"results": [`
		`{`
		`"D": 3`
		`}`
		`]`

DECADE	✗	✓ **Decade** = Year / 10

		select DATE_PART_STR(date_purchased,"decade") as D from bucket1;
		
		`"results": [`
		`{`
		`"D": 200`
		`}`
		`]`

QUARTER	✗	✓
		Quarter = (Month + 2) / 3
		
		select DATE_PART_STR(date_purchased,"quarter") as D from bucket1;
		
		`"results": [`
		`{`
		`"D": 4`
		`}`
		`]`

WEEK	✗	✓ **Week** = int(math.Ceil(float64(YearDay) / 7.0)) YearDay returns the day of the year specified by the time, in the range 1 to 365 for non-leap years, and 1 to 366 in leap years. (see golang time package) select DATE_PART_STR(date_purchased,"week") as D from bucket1;

		`"results": [`
		`{`
		`"D": 52`
		`}`
		`]`

MILLISECOND	✗	✓
		
		Millisecond = Nanosecond / 10^6 (as an integer)
		
		select DATE_PART_STR(date_purchased,"millisecond") as D from bucket1;
		
		`"results": [`
		`{`
		`"D": 336`
		`}`
		`]`

ISO_YEAR	✗	✓
		
		Iso_year = ISO 8091 year for the input timestamp.
		
		select DATE_PART_STR(date_purchased,"iso_year") as C from bucket1;
		
		`"results": [`
		`{`
		`"C": 2008`
		`}`
		`]`

ISO_WEEK	✗	✓

		Iso_week = ISO 8091 week for the input timestamp. The week usually ranges from 1 to 53. For example, Jan 1 to Jan 3 of year n might belong to Week 52 or 53 of year n-1, and Dec 29 to 31 might belong to week 1 of year n+1.
		
		select DATE_PART_STR(date_purchased,"iso_week") as D from bucket1;
		
		`"results": [`
		`        {`
		`            "D": 52`
		`        }`
		`    ]`

DAY_OF_YEAR (DOY)	✗	✓
		
		Day_of_year or doy = YearDay
		(see golang time package)
		
		YearDay returns the day of the year specified by the time, in the range 1 to 365 for non-leap years, and 1 to 366 in leap years.
		
		select DATE_PART_STR(date_purchased,"doy") as D from bucket1;
		
		`"results": [`

		`{`
		`"D": 360`
		`}`
		`]`

DAY_OF_WEEK (DOW)	✗	✓
		
		Day_of_week or dow = Weekday function, which returns the day of the week for the given time. (see golang time package)
		
		select DATE_PART_STR(date_purchased,"dow") as D from bucket1;
		
		`"results": [`
		`{`
		`"D": 4`
		`}`
		`]`

For N1QL, within the date time format, TIMEZONE_REGION and TIMEZONE_ABBR are not supported. (But these are passed into the timezone specific N1QL functions which we shall see in Part 2 of this series).

As we can see above when it comes to representing the TIMESTAMP within N1QL dates, there are additional fields supported. These are ISO_YEAR, ISO_WEEK, DAY_OF_YEAR (DOY), DAY_OF_WEEK (DOW) and TIMEZONE which is the offset from UTC.

In the absence of a time zone indicator, the current local timezone is used (where the Couchbase server is installed).

Let us delve a little deeper into the TIMEZONE comparisons between N1QL and Oracle. The TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE data types in Oracle are variants of the TIMESTAMP data type. The former includes the time zone information, which is the time zone offset which is the time relative to UTC or time zone region name in its value, and the later includes the current session timezone. TIMESTAMP WITH LOCAL TIME ZONE does not store time zone information internally, but you can see local time zone information in SQL output if the TZH:TZM or TZR TZD format elements are specified. (see here for more details).

```
Oracle :
TIMESTAMP '2017-01-31 03:49:30.612 -08:00'
```

```
Couchbase :
"2017-01-31T03:49:30.612-08:00"
```

For Oracle, if two dates being compared represent the same value in UTC, then they are equal.

```
Oracle :
TIMESTAMP '2017-01-15 8:00:00 -8:00' == TIMESTAMP '2017-01-15
10:00:00 -6:00'
```

In N1QL currently in order to compare full date values we need to convert them to milliseconds.

```
N1QL :

STR_TO_MILLIS("2017-01-31T05:49:30.612-06:00") ==
STR_TO_MILLIS("2017-01-31T03:49:30.612-08:00")
Value : 1485863370612
```

For Oracle, we can replace this offset with the Time zone region (TZR) and the abbreviation. The abbreviation (TZD) is used in the event the region value is ambiguous (when the US switches to daylight saving time).

However in N1QL the timezone component of the date is always represented as a UTC offset.

```
For example
TIMESTAMP '2017-01-15 8:00:00 -8:00' can also be TIMESTAMP
'2017-01-15 8:00:00 US/Pacific PDT'
```

Oracle also supports interval data types INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. These store time durations. The former stores the duration using year and month fields and the latter using the days, hours, minutes and second fields. With N1QL, computing an interval is made easy using date time functions and the “part” component. (These functions will be explored more in Part 2).

There are multiple ways to insert date or time data into Oracle. The user can insert a formatted string based on the NLS format value, or a literal with explicit conversion using the TO_DATE / TO_TIMESTAMP or TO_TIMESTAMP_TZ functions or implicit conversion.

For N1QL, all dates are added to a document as a string in the format specified above, or as a number representing a Unix timestamp in milliseconds. This makes handling dates very easy since the functions perform all the necessary arithmetic for the user. One drawback though is that the date has to exactly match one of the formats in the Date formats table. This restricts the user to use only a subset of available formats.

One workaround for this limitation with N1QL is to use the string functions and massage the input date to reflect the format you want.

```
For example

Convert 2016-09-23T18:48:11.000+00:00 into "YYYY/MM/DD"

SELECT  to_string(date_part_str("2016-09-23T18:48:11.000+00:00",
"year" )) || "/" ||
to_string(date_part_str("2016-09-23T18:48:11.000+00:00", "month"
)) || "/" ||
to_string(date_part_str("2016-09-23T18:48:11.000+00:00", "day"
));


 "results": [
        {
            "$1": "2016/9/23"
        }
    ]
```

As we can see, N1QL simplifies manipulating Dates and Timestamps by representing it as a string or a number when compared to Oracle. But this means that the user is restricted to use only specified date time formats and does not have the freedom to manipulate these formats, which Oracle does very easily using its Format Parameters.

Coming up in the Date Time article series -

1. Datetime and Interval Arithmetic
2. Conversion between different date time formats/data types.
3. How Oracle and N1QL handle daylight savings
4. Timezone related functions and how n1ql expects its timezone strings.
5. General SQL/N1QL functions to retrieve timestamps.