Supported Presto and TD Functions

This article lists all the functions supported by the Presto engine on Treasure Data.

All native Presto functions can also be used on Treasure Data. For a complete list of functions, see Presto Function and Operators pages.

- TD_APPROX_MOST_FREQUENT
- TD_TIME_RANGE
- TD_SCHEDULED_TIME
- TD_INTERVAL
- TD_INTERVAL_RANGE
- TD_TIME_ADD
- TD_TIME_FORMAT
- TD_TIME_PARSE
- TD_TIME_STRING
- TD_DATE_TRUNC
- TD_SESSIONIZE_WINDOW
- TD_PARSE_USER_AGENT
- TD_PARSE_AGENT
- TD_MD5
- TD_URL_DECODE
- SMART_DIGEST
- TD_CURRENCY_CONV
- TD_IP_TO_COUNTRY_CODE
- TD_IP_TO_COUNTRY_NAME
- TD_IP_TO_SUBDIVISION_NAMES
- TD_IP_TO_MOST_SPECIFIC_SUBDIVISION_NAME
- TD_IP_TO_LEAST_SPECIFIC_SUBDIVISION_NAME
- TD_IP_TO_CITY_NAME
- TD_IP_TO_LATITUDE
- TD_IP_TO_LONGITUDE
- TD_IP_TO_METRO_CODE (US Only)
- TD_IP_TO_TIME_ZONE
- TD_IP_TO_POSTAL_CODE
- TD_IP_TO_CONNECTION_TYPE
- TD_IP_TO_DOMAIN
- TD_LAT_LONG_TO_COUNTRY
- SORTED_GROUP_CONCAT
- Notes for Geometry types
- ST_Point
- ST_Polygon
- ST_Intersection
- ST_Intersects

TD_APPROX_MOST_FREQUENT

Signature

```
TD_APPROX_MOST_FREQUENT(long num_buckets, long/varchar values, long capacity)
```

Example

```
SELECT TD_APPROX_MOST_FREQUENT(3, values, 10);
```

Description

This function picks the frequent distinct items from the collection of values. This selection is approximate. The top `num_buckets` elements are obtained `values`. It returns a map whose keys are elements and values are estimated frequencies in the collection.

Unlike a normal histogram, it selects the frequent values online to significantly save memory resources. The error rate is bounded by the capacity parameter controlling the size of the internal data structure.

TD_TIME_RANGE

For convenience, we recommend using TD_INTERVAL instead of TD_TIME_RANGE.

Signature

```
boolean TD_TIME_RANGE(int/long unix_timestamp,
                      int/long/string start_time,
                      int/long/string end_time
                      [, string default_timezone = 'UTC'])
```

Example

This example selects records with timestamps '2013-01-01 00:00:00 PDT' or later. The time of day ('00:00:00') can be omitted. Alternately, the time of day can be specified up to seconds. In general, the time string should be formatted as 'YYYY-MM-DD' or 'YYYY-MM-DD hh:mm:ss'. For example, '2013-01-01' or '1999-01-01 07:00:00'.

```
SELECT ... WHERE TD_TIME_RANGE(time, '2013-01-01 PDT')                # OK
SELECT ... WHERE TD_TIME_RANGE(time, '2013-01-01', '2013-01-02','PDT') # OK
SELECT ... WHERE TD_TIME_RANGE(time, NULL, '2013-01-01', 'PDT')       # OK
SELECT ... WHERE TD_TIME_RANGE(time, '2013-01-01', NULL, 'PDT')       # OK
SELECT ... WHERE TD_TIME_RANGE(time, '2013-01-01', 'PDT')             # NG
```

Description

We strongly recommend that you take advantage of time-based partitioning. Refer to Performance Tuning for more information.

This UDF returns true if the *unix_timestamp* is equal to or later than the *start_time* and older than the *end_time* (start_time <= time && time < end_time). If *end_time* is omitted or NULL, the UDF assumes it's infinite. If *start_time* is NULL, the UDF assumes it's 0.

start_time and *end_time* can be a string that represents a time (e.g. '2012-01-01 00:00:00 +0900') or a UNIX timestamp (e.g. 1325343600). If the format of *start_time* or *end_time* strings is invalid, the UDF returns NULL.

default_timezone is used to interpret the timezone of *start_time* or *end_time*. If *start_time* or *end_time* themselves specify a timezone (e.g. '2012-01-01 +0700'), then the *default_timezone* is ignored. If *default_timezone* is not specified and *start_time* or *end_time* does not indicate a timezone, then the UDF uses 'UTC' as the timezone for *start_time* or *end_time*. A list of supported time zones.

TD_SCHEDULED_TIME

Signature

```
long TD_SCHEDULED_TIME()
```

Description

This UDF returns the exact time when the job was scheduled by the scheduled query feature. The returned value can differ from `NOW()` because the actual query start time might be delayed.

If the query is not a scheduled query, the UDF returns the time when the job was issued. You can use this UDF with `TD_TIME_ADD` for incremental aggregation.

TD_INTERVAL

TD_INTERVAL() is a companion function to TD_TIME_RANGE(). Both are especially useful in WHERE clauses, to make sure that your queries take advantage of time-based partitioning. TD_INTERVAL is used to compute relative time ranges that would otherwise require complex date manipulation. (TD_TIME_RANGE is used for absolute time ranges.)

Signature

TD_INTERVAL(time, interval_string, default_timezone)

```
boolean TD_INTERVAL(int/long time,
                    string interval_string,
                    [, string default_timezone = 'UTC'])
```

Example

These examples assume that the scheduled_time (or query start time) is 2018-08-14 01:23:45 (Tue, UTC):

```
# The last 7 days [2018-08-07 00:00:00, 2018-08-14 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-7d')
# The last week. Monday is the beginning of the week (ISO standard) [2018-08-05 00:00:00, 2018-08-13 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-1w')
# Today [2018-08-14 00:00:00, 2018-08-15 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '1d')
# The last month [2018-07-01 00:00:00, 2018-08-01 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-1M')
# This month [2018-08-01 00:00:00, 2018-09-01 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '1M')
# This year [2018-01-01 00:00:00, 2019-01-01 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '1y')
# The last 15 minutes [2018-08-14 00:08:00, 2018-08-14 01:23:00)
SELECT  ... WHERE TD_INTERVAL(time, '-15m')
# The last 30 seconds [2018-08-14 01:23:15, 2018-08-14 01:23:45)
SELECT  ... WHERE TD_INTERVAL(time, '-30s')
# The last hour [2018-08-14 00:00:00, 2018-08-14 01:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-1h')
# From the last hour to now [2018-08-14 00:00:00, 2018-08-14 01:23:45)
SELECT  ... WHERE TD_INTERVAL(time, '-1h/now')
# The last hour since the beginning of today [2018-08-13 23:00:00, 2018-08-14 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-1h/0d')
# The last 7 days since 2015-12-25 [2015-12-18 00:00:00, 2015-12-25 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-7d/2015-12-25')
# The last 10 days since the beginning of the last month [2018-06-21 00:00:00, 2018-07-01 00:00:00)
SELECT  ... WHERE TD_INTERVAL(time, '-10d/-1M')
# The last 7 days in JST
SELECT  ... WHERE TD_INTERVAL(time, '-7d', 'JST')
```

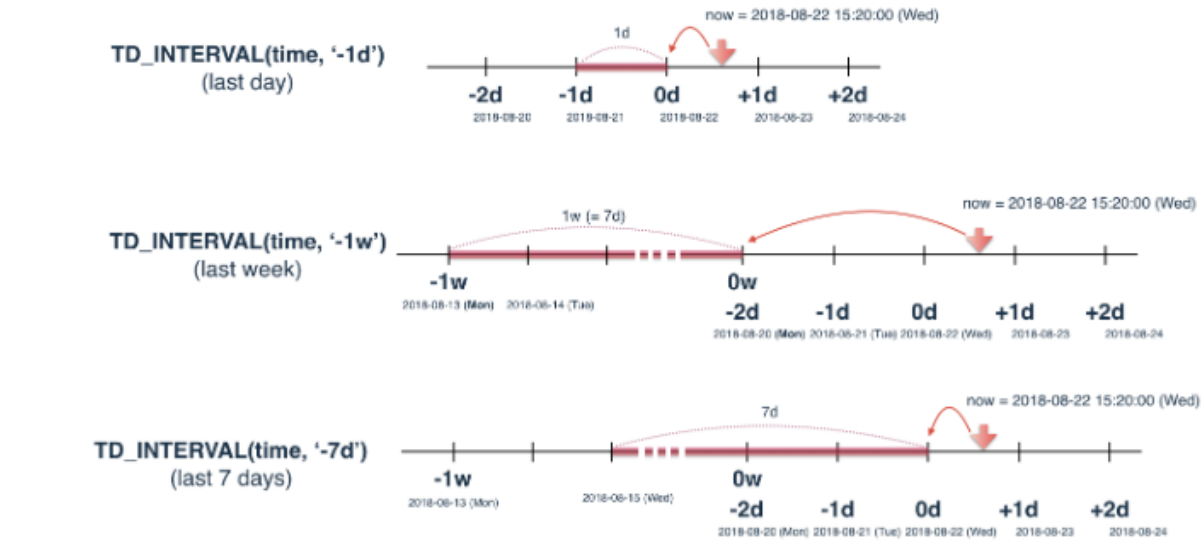
Description

TD_INTERVAL() is a companion function to TD_TIME_RANGE(). Both are especially useful in WHERE clauses, to make sure that your queries take advantage of time-based partitioning. TD_INTERVAL is used to compute relative time ranges that would otherwise require complex date manipulation. (TD_TIME_RANGE is used for absolute time ranges.)

> We strongly recommend that you take advantage of time-based partitioning. Refer to the Performance Tuning article for more information. Not using time-based filtering in SQL SELECT statements can cause inefficient full table scans that affect query performance.

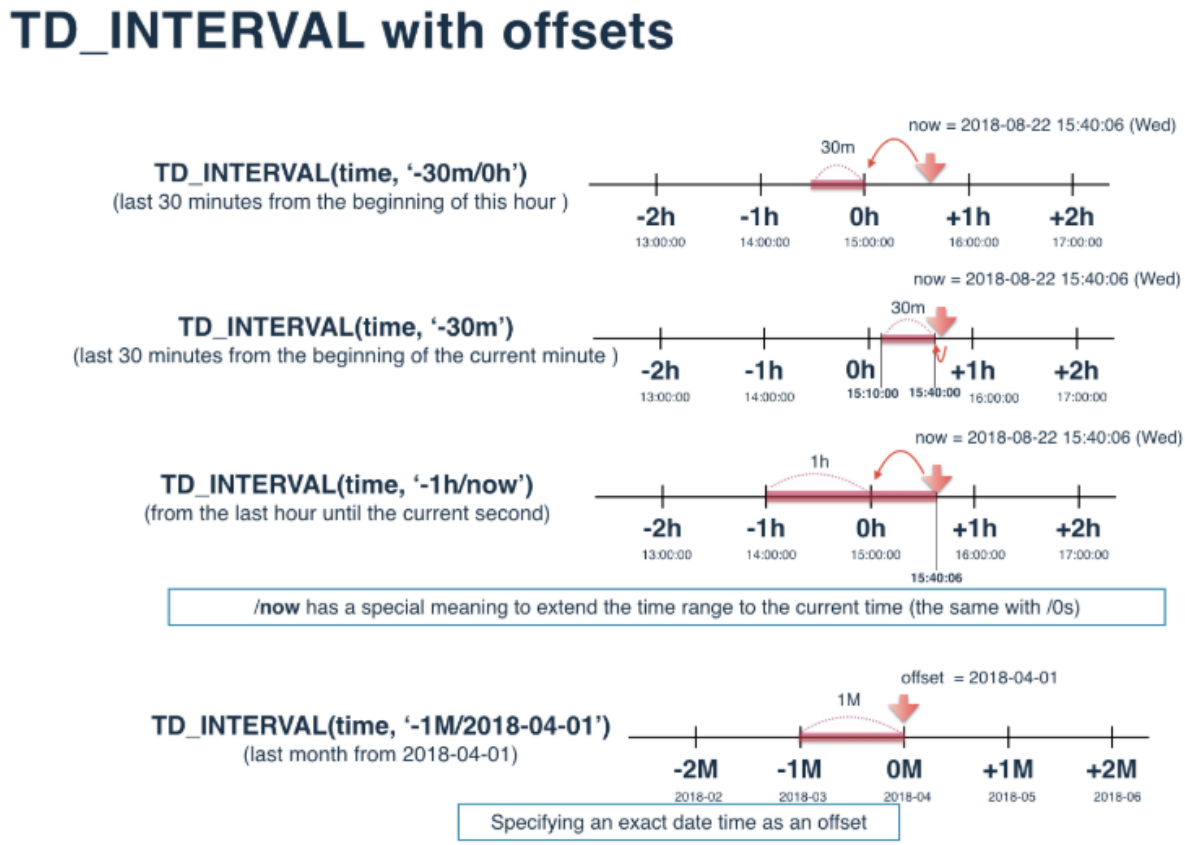
This UDF returns true if *time* value is within the interval which is represented by *interval_string* (state time <= time < end time).

interval_string must be a 'duration/offset' formatted string. The offset is optional and the UDF assumes offset is the current time (the job scheduled time actually) based on your browser timezone, if the offset is omitted. Also, support 'q' for quarters. For example, '-1d' means yesterday and '-3M' means the last 3 months. The interval is calculated in the specified time unit. This means '-30m' Is the last 30 minutes from the beginning of the latest minute, not from just now.



Offset can be specified relatively (e.g. '3d/-1y') and specifically (e.g. '1y/2018-01-01'). For example, '3d/-1y' means the first 3 days of the last year and '-1M /2018-04-01' means the last 1 month before '2018-04-01'. In other words, '2018-03-01' to '2018-03-31'.

Offset can be specified as '/now' (e.g. '-7d/now'). Note the difference from '/0d' or '/0h' illustrated in the following figure:



default_timezone is used to interpret the timezone of *interval_string*. If *interval_string* specifies a timezone (e.g. '-1h/2017-01-23 01:00:00 +0700'), then the *default_timezone* is ignored. If *default_timezone* is not specified and *interval_string* does not have a timezone, then UDF uses 'UTC' as the timezone. A list of supported time zones can be found here.

TD_INTERVAL_RANGE

Signature

TD_INTERVAL_RANGE('interval string', 'time zone')

Description

TD_INTERVAL_RANGE can be used to confirm the time range of TD_INTERVAL.

TD_INTERVAL_RANGE returns an ARRAY[(start time), (end time)].

interval_string must be a 'duration/offset' formatted string. The offset is optional and the UDF assumes offset is the current time (the job scheduled time actually) based on your browser timezone, if the offset is omitted. Also, support 'q' for quarters.

time zone is used to interpret the timezone of *interval_string*. If *interval_string* specifies a timezone (e.g. '-1h/2017-01-23 01:00:00 +0700'), then *time zone* is ignored. If the *time zone* is not specified and *interval_string* does not have a timezone, then UDF uses 'UTC' as the timezone. A list of supported time zones can be found here.

TD_TIME_ADD

Signature

```
long TD_TIME_ADD(int/long/string time,
                 string duration
                 [, string default_timezone = 'UTC'])
```

Example

This example selects records with timestamps '2013-01-01 00:00:00 UTC' or later but older than '2013-01-02 00:00:00 UTC'.

```
SELECT ... WHERE TD_TIME_RANGE(time,
                               '2013-01-01',
                               TD_TIME_ADD('2013-01-01', '1d'))
```

Description

TD_TIME_ADD returns a timestamp equal to *time* offset by *duration*. The UDF supports the following formats for the *duration*:

- "Nw": after N weeks (e.g. "1w", "2w", "5w")
- "-Nw": before N weeks (e.g. "-1w", "-2w", "-5w")
- 'Nd': after N days (e.g. '1d', '2d', '30d')
- '-Nd': before N days (e.g. '-1d', '-2d', '-30d')
- 'Nh': after N hours (e.g. '1h', '2h', '48h')
- '-Nh': before N hours (e.g. '-1h', '-2h', '-48h')
- 'Nm': after N minutes (e.g. '1m', '2m', '90m')
- '-Nm': before N minutes (e.g. '-1m', '-2m', '-90m')
- 'Ns': after N seconds (e.g. '1s', '2s', '90s')
- '-Ns': before N seconds (e.g. '-1s', '-2s', '-90s')

The formats above can be combined. For example, '1h30m' means 'after 1 hour and 30 minutes'.

default_timezone is used to interpret *time*. If *time* has timezone (e.g. '2012-01-01 +0700'), then *default_timezone* is ignored. If *default_timezone* is not specified and *time* does not specify a timezone, then the UDF uses 'UTC' as the timezone for *time*. A list of supported time zones can be found here.

If the formats of the *time* or *duration* strings are invalid, the UDF returns NULL.

'year' and 'month' durations are **NOT** supported, because to do so would adversely impact performance. A month can be 28, 29, 30, or 31 days, and a year could be 365 or 366 days.

TD_TIME_FORMAT

For convenience, we recommend using TD_TIME_STRING instead of TD_TIME_FORMAT.

Signature

```
string TD_TIME_FORMAT(long unix_timestamp,
                      string format
                      [, string timezone = 'UTC'])
```

Example

This example formats a UNIX timestamp into a date formatted string:

```
SELECT TD_TIME_FORMAT(time, 'yyyy-MM-dd HH:mm:ss z') ... FROM ...
SELECT TD_TIME_FORMAT(time, 'yyyy-MM-dd HH:mm:ss z', 'PST') ... FROM ...
SELECT TD_TIME_FORMAT(time, 'yyyy-MM-dd HH:mm:ss z', 'JST') ... FROM ...
```

Description

TD_TIME_FORMAT converts a UNIX timestamp to a string with the specified format (see the Supported time formats in TD_TIME_FORMAT UDF page for available formats). For example, 'yyyy-MM-dd HH:mm:ss z' converts 1325376000 to '2012-01-01 00:00:00 UTC'. If no timezone is specified, the UDF uses UTC.

How does TD_TIME_FORMAT handle Leap Second?

```
SELECT
 TD_TIME_FORMAT(1136073600, 'yyyy-MM-dd HH:mm:ss', 'JST') as st,
 TD_TIME_PARSE('2006-01-01 08:59:60', 'JST') as leap,
 TD_TIME_PARSE('2006-01-01 09:00:00', 'JST') as leap2
```

TD_TIME_PARSE

Signature

```
long TD_TIME_PARSE(string time
                   [, string default_timezone = 'UTC'])
```

Description

This UDF converts a time string into a UNIX timestamp.

default_timezone is used to interpret *time*. If *time* has timezone (e.g. '2012-01-01 +0700'), then *default_timezone* is ignored. If *default_timezone* is not specified and *time* does not specify a timezone, then the UDF uses 'UTC' as the timezone for *time*. A list of supported time zones can be found here.

If the format of the *time* string is invalid, the UDF returns NULL.

TD_TIME_STRING

For convenience, we recommend TD_TIME_STRING over TD_TIME_FORMAT.

```
TD_TIME_STRING(time, '(interval string)', time zone?)
```

- time: unix time (bigint)
- interval string:

  ```
  [yqMwdhm](!)?
  ```

If the format string has `!` as the suffix, it truncates the date time string at the specified unit.

format string	format	example
y	yyyy-MM-dd HH:mm:ssZ	2018-01-01 00:00:00+0700
q	yyyy-MM-dd HH:mm:ssZ	2018-04-01 00:00:00+0700
M	yyyy-MM-dd HH:mm:ssZ	2018-09-01 00:00:00+0700
w	yyyy-MM-dd HH:mm:ssZ	2018-09-09 00:00:00+0700
d	yyyy-MM-dd HH:mm:ssZ	2018-09-13 00:00:00+0700
h	yyyy-MM-dd HH:mm:ssZ	2018-09-13 16:00:00+0700
m	yyyy-MM-dd HH:mm:ssZ	2018-09-13 16:45:00+0700
s	yyyy-MM-dd HH:mm:ssZ	2018-09-13 16:45:34+0700
y!	yyyy	2018
q!	yyyy-MM	2018-04
M!	yyyy-MM	2018-09
w!	yyyy-MM-dd	2018-09-09
d!	yyyy-MM-dd	2018-09-13
h!	yyyy-MM-dd HH	2018-09-13 16
m!	yyyy-MM-dd HH:mm	2018-09-13 16:45
s!	yyyy-MM-dd HH:mm:ss	2018-09-13 16:45:34

If there is no `!`, the return value should be the same as:

```
TD_TIME_FORMAT(TD_DATE_TRUNC('(interval unit)', time, timezone), 'yyyy-MM-dd HH:mm:ssZ', timezone)
```

TD_DATE_TRUNC

Signature

```
long TD_DATE_TRUNC(string unit,
                   long time
                   [, string default_timezone = 'UTC'])
```

Description

This UDF performs a timestamp truncation at the level specified by the ‘unit’ parameter. The supported units are:

- ‘minute’
- ‘hour’
- ‘day’

- ‘week’
- ‘month’
- ‘quarter’
- ‘year’

An optional ‘timezone’ parameter can be specified to indicate an alternative reference timezone for the ‘unit’. If the input ‘time’ is in global UNIX time format, in different timezones, the start of a day corresponds to different times.

This function mimics the functionality of native Presto’s `date_trunc` function. However, Presto’s `date_trunc` does not allow specification of the timezone.

Example

```
SELECT TD_DATE_TRUNC('day', time) FROM tbl
```

with `time` equal `1416787667` corresponding to ‘`2014-11-24 00:07:47 UTC`’ returns `1416787200` corresponding to ‘`2014-11-24 00:00:00 UTC`’.

With the same value and timezone ‘PST’ instead,

```
SELECT TD_DATE_TRUNC('day', time, 'PST') FROM tbl
```

the function returns 1416729600 because the start of the day for the ‘PST’ timezone is 8 hours behind the start of the day for ‘UTC’.

TD_SESSIONIZE_WINDOW

Signature

```
string TD_SESSIONIZE_WINDOW(int/long unix_timestamp, int timeout)
```

Description

Sessionization of a table of event data groups a series of event rows associated with users into individual sessions for analysis. The series of events to be grouped into a session must be associated with the same user identifier (typically IP address, email, cookie, or similar identifier) and events are separated by no more than a chosen timeout interval.

TD_SESSIONIZE_WINDOW is a UDF window function used for sessionization. It replaces TD_SESSIONIZE. TD_SESSIONIZE_WINDOW provides consistent results and better performance.

TD_SESSIONIZE_WINDOW takes two arguments:

- The time field specified in the UNIX epoch
- A timeout interval in seconds (when this amount of time elapses between events, it indicates the start of a new session)

Other usage notes:

- Use an OVER clause to partition the input rows
- Partition rows based on the user identifier
- ORDER the rows by the time column passed to TD_SESSIONIZE_WINDOW

Example

The following example is equivalent to the SELECT statement example in the deprecated TD_SESSIONIZE.

```
SELECT
  TD_SESSIONIZE_WINDOW(time, 3600)
    OVER (PARTITION BY ip_address ORDER BY time)
    as session_id,
  time,
  ip_address,
  path
FROM
  web_logs
```

TD_PARSE_USER_AGENT

Signature

```
string TD_PARSE_USER_AGENT(user_agent string [, options string])
```

Description

This UDF returns the result of parsing a user agent string. The user agent is parsed on the basis of rules. Where options are:

Options	Accepts	Returns
os	sting	JSON
os_family	string	string
os_major	string	string
os_minor	string	string
ua	string	JSON
ua_family	string	string
ua_major	string	string
ua_minor	string	string
device	string	string

Example

The example shows the result of parsing user agent from access log.

```
SELECT TD_PARSE_USER_AGENT(agent) AS agent FROM www_access
> {user_agent: {family: "IE", major: "9", minor: "0", patch: null}, os: {family: "Windows 7", major: null,
minor: null, patch: null, patch_minor: null}, device: {family: "Other"}}
SELECT TD_PARSE_USER_AGENT(agent, 'os') AS agent_os FROM www_access
> {family: "Windows 7", major: null, minor: null, patch: null, patch_minor: null}
SELECT TD_PARSE_USER_AGENT(agent, 'os_family') AS agent_os_family FROM www_access
> Windows 7
```

TD_PARSE_AGENT

This UDF returns a Map value of results to parse a user agent string. The UDF is implemented by Woothee.

Signature

```
MAP(varchar,varchar) TD_PARSE_AGENT(user_agent varchar)
```

Example

The example shows the result of parsing the user agent from an access log. If you want to extract a specific ‘key’ from the user agent map. TD recommends using the `element_at` presto function because it is tolerant of non-existent keys. Extracting keys with the `[]` operator (e.g. `TD_PARSE_AGENT(<agent_string>)[‘<keyname>’])` will throw an error if the sought after the key is not present in the map.

```
SELECT TD_PARSE_AGENT(agent) AS parsed_agent, agent FROM www_access
> {"os":"Windows 7","vendor":"Google","os_version":"NT 6.1","name":"Chrome","category":"pc","version":"
16.0.912.77"},
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7

SELECT element_at(TD_PARSE_AGENT(agent), 'os') AS os FROM www_access
> Windows 7 => os from user-agent, or carrier name of mobile phones

SELECT element_at(TD_PARSE_AGENT(agent), 'vendor') AS vendor FROM www_access
> Google // => name of vendor

SELECT element_at(TD_PARSE_AGENT(agent), 'os_version') AS os_version FROM www_access
> NT 6.1 // => "NT 6.3" (for Windows), "10.8.3" (for OSX), "8.0.1" (for iOS), ....

SELECT element_at(TD_PARSE_AGENT(agent), 'name') AS name FROM www_access
> Chrome // => name of browser (or string like name of user-agent)

SELECT element_at(TD_PARSE_AGENT(agent), 'category') AS category FROM www_access
> pc // => "pc", "smartphone", "mobilephone", "appliance", "crawler", "misc", "unknown"

SELECT element_at(TD_PARSE_AGENT(agent), 'version') AS version FROM www_access
> 16.0.912.77 => version of browser, or terminal type name of mobile phones

SELECT TD_PARSE_AGENT(agent)['nonexistentkey'] FROM www_access
! The *query errors out* because the <tt>nonexistentkey</tt> key is not present
! in the map returned by <tt>TD_PARSE_AGENT(agent)</tt>.
```

TD_MD5

Signature

```
string TD_MD5(col)
```

Description

This UDF calculates the MD5 hash digest from a given string.

Example

```
SELECT TD_MD5(column) FROM tbl
```

TD_URL_DECODE

TD_URL_DECODE supports URL decoding for a given string and euc-kr (extended unix code for Korean).

URL Decoding

Signature

```
string TD_URL_DECODE(col)
```

Description

`TD_URL_DECODE` applies URL decoding for a given string. This UDF returns `half-width space` if a character is `\r` or `\n`, or `\t`. This UDF is similar to URL_DECODE(col).

Example

```
SELECT TD_URL_DECODE(column) FROM tbl
```

URL EUC-KR

Signature

```
string TD_URL_DECODE(url [, local])
```

Description

`TD_URL_DECODE` applies URL decoding for a given URL.

Example

```
SELECT TD_URL_DECODE('%BA%ED%B7%E7%C5%F5%BD%BA', 'ko')
```

SMART_DIGEST

Signature

```
string SMART_DIGEST(col [,weight = 1.0])
```

Description

This UDF calculates the variable-length digest from a given string. It usually generates 6-10 characters of digest from the given string. Due to the higher compression ratio, there is a higher collision ratio, around 5% on average. If you want to avoid the collisions, increase the value of the weight parameter.

Example

```
SELECT SMART_DIGEST(column) FROM tbl
SELECT SMART_DIGEST(column, 1.5) FROM tbl
```

TD_CURRENCY_CONV

Signature

```
string TD_CURRENCY_CONV(string date, string from_currency, string to_currency, float value)
```

Description

This UDF converts currency for the specific date, by accessing the currency exchange rate database.

- List of Supported Currencies

Example

```
SELECT TD_CURRENCY_CONV('2015-01-01', 'USD', 'JPY', 1.0)
```

TD_IP_TO_COUNTRY_CODE

Both Hive and Presto UDFs use a geolocation database supplied by Maxmind. Due to release schedules, the release level of the Maxmind database used by Hive and Presto might be different. This might cause inconsistent results between Hive and Presto geolocation functions.

An example of different results is as follows:

jobid	type	td_ip_to_city_name_v6	td_ip_to_latitude_v6	td_ip_to_longitude_v6	td_ip_to_postal_code_v6
218018944	hive	Tokyo	35.685	139.7514	102-0082
218019099	presto		35.6594	139.8533	134-0087

Signature

```
string TD_IP_TO_COUNTRY_CODE(string ip)
```

Description

This UDF converts IP address to country code. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_COUNTRY_CODE('106.142.252.8') AS ipv4,
    TD_IP_TO_COUNTRY_CODE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

The function returns `JP` in this example.

TD_IP_TO_COUNTRY_NAME

Signature

```
string TD_IP_TO_COUNTRY_NAME(string ip)
```

Description

This UDF converts IP address to country code. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_COUNTRY_NAME('106.142.252.8') AS ipv4,
    TD_IP_TO_COUNTRY_NAME('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

The function returns `Japan` in this example.

TD_IP_TO_SUBDIVISION_NAMES

Signature

```
array<string> TD_IP_TO_SUBDIVISION_NAMES(string ip)
```

Description

This UDF converts IP address to a list of subdivisions (e.g. US states, JP prefectures, etc). This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_SUBDIVISION_NAMES('106.142.252.8') AS ipv4,
    TD_IP_TO_SUBDIVISION_NAMES('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_MOST_SPECIFIC_SUBDIVISION_NAME

Signature

```
string TD_IP_TO_MOST_SPECIFIC_SUBDIVISION_NAME(string ip)
```

Description

This UDF converts IP address to the most specific subdivisions (e.g. US states, JP prefectures, etc). This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_MOST_SPECIFIC_SUBDIVISION_NAME('106.142.252.8') AS ipv4,
    TD_IP_TO_MOST_SPECIFIC_SUBDIVISION_NAME('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_LEAST_SPECIFIC_SUBDIVISION_NAME

Signature

```
string TD_IP_TO_LEAST_SPECIFIC_SUBDIVISION_NAME(string ip)
```

Description

This UDF converts IP address to the least specific subdivisions (e.g. US states, JP prefectures, etc). This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_LEAST_SPECIFIC_SUBDIVISION_NAME('106.142.252.8') AS ipv4,
    TD_IP_TO_LEAST_SPECIFIC_SUBDIVISION_NAME('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_CITY_NAME

Signature

```
string TD_IP_TO_CITY_NAME(string ip)
```

Description

This UDF converts IP address to city name. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_CITY_NAME('106.142.252.8') AS ipv4,
    TD_IP_TO_CITY_NAME('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_LATITUDE

Signature

```
string TD_IP_TO_LATITUDE(string ip)
```

Description

This UDF converts IP address to latitude. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_LATITUDE('106.142.252.8') AS ipv4,
    TD_IP_TO_LATITUDE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_LONGITUDE

Signature

```
string TD_IP_TO_LONGITUDE(string ip)
```

Description

This UDF converts IP address to longitude. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_LONGITUDE('106.142.252.8') AS ipv4,
    TD_IP_TO_LONGITUDE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_METRO_CODE (US Only)

Signature

```
string TD_IP_TO_METRO_CODE(string ip)
```

Description

This UDF converts IP address to metro code (US Only). This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_METRO_CODE('106.142.252.8') AS ipv4,
    TD_IP_TO_METRO_CODE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_TIME_ZONE

Signature

```
string TD_IP_TO_TIME_ZONE(string ip)
```

Description

This UDF converts IP address to time zone. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_TIME_ZONE('106.142.252.8') AS ipv4,
    TD_IP_TO_TIME_ZONE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_POSTAL_CODE

Signature

```
string TD_IP_TO_POSTAL_CODE(string ip)
```

Description

This UDF converts IP address to postal code. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_POSTAL_CODE('106.142.252.8') AS ipv4,
    TD_IP_TO_POSTAL_CODE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_IP_TO_CONNECTION_TYPE

Signature

```
string TD_IP_TO_CONNECTION_TYPE(string ip)
```

Description

This UDF converts IP address to connection type. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_CONNECTION_TYPE('106.142.252.8') AS ipv4,
    TD_IP_TO_CONNECTION_TYPE('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

Possible values are `dial-up`, `cable/DSL`, `corporate` or `cellular`.

TD_IP_TO_DOMAIN

Signature

```
string TD_IP_TO_DOMAIN(string ip)
```

Description

This UDF converts IP address to domain. This UDF supports IPv4 and IPv6.

Example

```
SELECT
    TD_IP_TO_DOMAIN('106.142.252.8') AS ipv4,
    TD_IP_TO_DOMAIN('2001:268:d005:f5be:c43e:af35:81f:8f60') AS ipv6
```

TD_LAT_LONG_TO_COUNTRY

Signature

```
string TD_LAT_LONG_TO_COUNTRY(string type, double latitude, double longitude)
```

Description

This UDF converts geolocation information (latitude/longitude) to the country name.

Example

```
SELECT
  TD_LAT_LONG_TO_COUNTRY('FULL_NAME',                 37, -122)
  TD_LAT_LONG_TO_COUNTRY('THREE_LETTER_ABBREVIATION', 37, -122)
  TD_LAT_LONG_TO_COUNTRY('POSTAL_ABBREVIATION',       37, -122)
  TD_LAT_LONG_TO_COUNTRY('SORTABLE_NAME',             37, -122)
```

SORTED_GROUP_CONCAT

This function is not supported.

Signature

```
string SORTED_GROUP_CONCAT(column, delimiter, orderkey)
```

Description

This UDF returns the concatenation of `column` with `delimiter` ordered by `orderkey`in a group of values.

Example

```
SELECT groupkey, SORTED_GROUP_CONCAT(column, '.', time)
FROM table
GROUP BY groupkey
```

Notes for Geometry types

Geometry types are the building blocks of geospatial queries and calculations. They are used as arguments for geospatial functions. Geometry type is the product of a constructor function.

ST_Point and ST_Polygon are examples of geospatial functions used to obtain binary representations of a point, `line, or polygon. You can also use them to convert a geometry data type to text.

ST_Point

Signature

```
point ST_Point(double, double)
```

Description

A constructor function that returns a geometry type point object with the given coordinate values.

Example

```
SELECT s.school_Name as School, s.phoneNumber as Phone
FROM schools as s
JOIN counties as c
ON st_contains(c.geo_shape, st_point(s.loc_lng, s.loc_lang))
```

The above query returns a list of schools that are within a geographic area stored in the counties table. The ‘st_point’ constructor is used to create the point that is being used.

RESULT: School Phone Pelham H.S. (212) 948 5300 Midrand M.S. (212) 233 3587

ST_Polygon

Signature

```
polygon ST_Polygon(varchar)
```

Description

Returns a geometry type polygon object from WKT representation.

Example

```
SELECT r.resevoirName as Name, r.area as Area
FROM resevoirs as r
JOIN parks as p
ON st_contains(p.geo_shape, ST_Polygon(r.geo_shape))
```

The above query returns a list of reservoirs that are within a geographic area of a park in the parks table. The ‘ST_Polygon’ constructor is used to create the polygon that is being used.

RESULT: Park Area Hope Fountain 4000 Hot Springs 5156

ST_Intersection and ST_Intersects are examples of Geospatial Relationship Functions. Relationship functions allow you to find relationships between two different geometric inputs. They return a boolean result type.

ST_Intersection

Signature

```
Geometry ST_Intersection(Geometry, Geometry)
```

Description

Returns the geometry value that represents the point set intersection of two geometries.

Example

```
SELECT ST_AsText(ST_INTERSECTION(ST_POINT(1,1), ST_POINT(1,1))
FROM tbl1
```

RESULT: POINT(1 0) (1 row)

Returns the intersection of 2 points as text coordinates.

ST_Intersects

Signature

```
boolean ST_Intersects(Geometry, Geometry)
```

RESULT: true (1 row)

Description

Returns true if the given geometries spatially intersect in two dimensions (share any portion of space) and false if they do not (they are disjoint).

Example

```
SELECT ST_INTERSECTS(ST_LINE('linestring(8 7, 7 8)'), ST_POLYGON('polygon((1 1, 4 1, 4 4, 1 4))'))
FROM tbl1
```

RESULT: true (1 row)