[Функции даты и отметки времени Spark SQL](https://sparkbyexamples.com/spark/spark-sql-date-and-time-functions/)

Spark SQL предоставляет встроенные стандартные функции даты и времени (включая дату и время), определенные в DataFrame API, они пригодятся, когда нам нужно выполнять операции с датой и временем. Все они принимают входные данные в виде типа даты, типа отметки времени или строки. Если это строка, она должна быть в формате, который может быть преобразован к дате, например yyyy-MM-dd и метка времени в  
 yyyy-MM-dd HH:mm:ss.SSSS и возвращает дату и метку времени соответственно; также возвращает значение NULL, если входные данные представляли собой строку, которую невозможно привести к дате и метке времени.

По возможности старайтесь использовать стандартную библиотеку, поскольку она немного более безопасна во время компиляции, обрабатывает нулевые значения и работает лучше по сравнению с [Spark UDF](https://sparkbyexamples.com/spark/spark-sql-udf/). Если ваше приложение имеет [решающее значение для производительности](https://sparkbyexamples.com/spark/spark-performance-tuning/), постарайтесь любой ценой избегать использования пользовательских UDF, поскольку они не гарантируют производительность.

Для удобства чтения я сгруппировал функции Date и Timestamp следующим образом.

1. [Функции даты Spark SQL](#_Функции_даты_Spark)
2. [Функции временных меток Spark SQL](#_Функции_временных_меток)
3. [Дата и время оконные функции](#_Дата_и_время)

Прежде чем использовать приведенные ниже примеры, убедитесь, что вы создали сессию Spark и импортировали функции SQL.

*import org.apache.spark.sql.SparkSession*

*val spark:SparkSession = SparkSession.builder()*

*.master("local[3]")*

*.appName("SparkByExample")*

*.getOrCreate()*

*spark.sparkContext.setLogLevel("ERROR")*

*import spark.sqlContext.implicits.\_*

*import org.apache.spark.sql.functions.\_*

# Функции даты Spark SQL

Нажмите на каждую ссылку в таблице ниже, чтобы получить дополнительные объяснения и рабочие примеры в Scala.

|  |  |
| --- | --- |
| ФУНКЦИЯ ДАТЫ | ФУНКЦИЯ ДАТЫ ОПИСАНИЕ |
| [current\_date () : Column](#_current_date()_and_date_format()) | Возвращает текущую дату в виде столбца даты. |
| [date\_format(dateExpr: Column, format: String): Column](#_current_date()_and_date_format()) | Преобразует дату/метку времени/строку в значение строки в формате, заданном форматом даты, заданным вторым аргументом. |
| [to\_date(e: Column): Column](#_to_date()) | Преобразует столбец в *DateType* путем приведения правил к *DateType*. |
| [to\_date(e: Column, fmt: String): Column](#_to_date()) | Преобразует столбец в *DateType* указанного формата. |
| [add\_months(startDate: Column, numMonths: Int): Column](#_add_months()_,_date_add(),) | Возвращает дату, которая находится через numMonths после startDate. |
| [date\_add(start: Column, days: Int): Column](#_add_months()_,_date_add(),)  [date\_sub(start: Column, days: Int): Column](#_add_months()_,_date_add(),) | Возвращает дату, которая находится через несколько дней после начала (*start*) |
| [datediff(end: Column, start: Column): Column](#_datediff()) | Возвращает количество дней от начала (*start*) до конца (*end*). |
| [months\_between(end: Column, start: Column): Column](#_months_between()) | Возвращает количество месяцев между датами начала (*start*) и окончания (*end*). Возвращается целое число, если оба входных параметра имеют один и тот же день месяца или оба являются последним днем ​​соответствующего месяца. В противном случае разница рассчитывается исходя из 31 дня в месяце. |
| months\_between(end: Column, start: Column, roundOff: Boolean): Column | Возвращает количество месяцев между датами окончания (*end*) и начала (*start*). Если для параметра roundOff установлено значение true, результат округляется до 8 цифр; иначе оно не округляется. |
| next\_day(date: Column, dayOfWeek: String): Column | Возвращает первую дату, которая позже значения столбца даты, приходящегося на указанный день недели. Например, *next\_day('2015-07-27', "Воскресенье")* возвращает 2015-08-02, поскольку это первое воскресенье после 2015-07-27. |
| [trunc(date: Column, format: String): Column](#_trunc()) | Возвращает дату, усеченную до единицы, указанной в формате. Например, trunc(*"2018-11-19 12:01:19", "* *year "*) возвращает 2018-01-01.  формат: “year”, “yyyy”, “yy” для усечения по году,  “month”, “mon”, “mm”’ для усечения по месяцам  “day”, “dd” для усечения по дням Другие варианты: “second”, “minute”, “hour”, “week”, “month”, “quarter” |
| [year(e: Column): Column](#_year(),_month(),_month()) | Извлекает год как целое число из заданной даты/временной метки/строки. |
| quarter(e: Column): Column | Извлекает квартал как целое число из заданной даты/временной метки/строки. |
| [month(e: Column): Column](#_year(),_month(),_month()) | Извлекает месяц как целое число из заданной даты/временной метки/строки. |
| [dayofweek(e: Column): Column](#_year(),_month(),_month()) | Извлекает день недели как целое число из заданной даты/временной метки/строки. Диапазон значений от 1 для воскресенья до 7 для субботы. |
| [dayofmonth(e: Column): Column](#_year(),_month(),_month()) | Извлекает день месяца как целое число из заданной даты/временной метки/строки. |
| [dayofyear(e: Column): Column](#_year(),_month(),_month()) | Извлекает день года как целое число из заданной даты/временной метки/строки. |
| [weekofyear(e: Column): Column](#_year(),_month(),_month()) | Извлекает номер недели как целое число из заданной даты/временной метки/строки. Считается, что неделя начинается в понедельник, а неделя 1 — это первая неделя, в которой более 3 дней, как определено ISO 8601. |
| [last\_day(e: Column): Column](#_year(),_month(),_month()) | Возвращает последний день месяца, к которому принадлежит данная дата. Например, ввод «2015-07-27» возвращает «2015-07-31», поскольку 31 июля — последний день месяца в июле 2015 года. |
| from\_unixtime(ut: Column): Column | Преобразует количество секунд от эпохи Unix (1970-01-01 00:00:00 UTC) в строку, представляющую временную метку этого момента в текущем часовом поясе системы в формате yyyy-MM-dd HH:mm:ss. |
| from\_unixtime(ut: Column, f: String): Column | Преобразует количество секунд из эпохи Unix (1970-01-01 00:00:00 UTC) в строку, представляющую временную метку этого момента в текущем системном часовом поясе в заданном формате. |
| unix\_timestamp(): Column | Возвращает текущую временную метку Unix (в секундах) в виде длинного целого |
| unix\_timestamp(s: Column): Column | Преобразует строку времени в формате yyyy-MM-dd HH:mm:ss в метку времени Unix (в секундах), используя часовой пояс по умолчанию и языковой стандарт по умолчанию. |
| unix\_timestamp(s: Column, p: String): Column | Преобразует строку времени по заданному шаблону в метку времени Unix (в секундах). |

# Функции временных меток Spark SQL

Ниже приведены некоторые функции Spark SQL Timestamp. Эти функции работают как со значениями даты, так и со значениями метки времени. Выберите каждую ссылку, чтобы просмотреть описание и пример каждой функции.

Формат метки времени Spark по умолчанию - *yyyy-MM-dd HH:mm:ss.SSSS*

|  |  |
| --- | --- |
| ФУНКЦИЯ ВРЕМЕНИ | ФУНКЦИЯ ВРЕМЕНИ - ОПИСАНИЕ |
| [current\_timestamp () : Column](#_current_timestamp()) | Возвращает текущую метку времени в виде столбца метки времени. |
| [current\_timestamp () : Column](#_current_timestamp()) | Возвращает текущую метку времени в виде столбца метки времени. |
| [hour(e: Column): Column](#_hour(),_Minute()_and) | Извлекает часы как целое число из заданной даты/временной метки/строки. |
| [minute(e: Column): Column](#_hour(),_Minute()_and) | Извлекает минуты как целое число из заданной даты/временной метки/строки. |
| [second(e: Column): Column](#_hour(),_Minute()_and) | Извлекает секунды как целое число из заданной даты/временной метки/строки. |
| [to\_timestamp(s: Column): Column](#_to_timestamp()) | Преобразуется в метку времени путем приведения правил к *TimestampType*. |
| to\_timestamp(s: Column, fmt: String): Column | Converts time string with the given pattern to timestamp. |

# Дата и время оконные функции

Ниже приведены оконные функции даты и времени.

|  |  |
| --- | --- |
| ФУНКЦИЯ | ОПИСАНИЕ |
| window(timeColumn: Column, windowDuration: String,  slideDuration: String, startTime: String): Column | Группируйте строки в одно или несколько временных окон с учетом столбца, определяющего временную метку. Начало окна включает в себя, но конец окна является исключительным, например. 12:05 будет в окне [12:05,12:10), но не в [12:00,12:05). Windows может поддерживать точность до микросекунды. Windows в порядке месяцев не поддерживается. |
| window(timeColumn: Column, windowDuration: String, slideDuration: String): Column | Группируйте строки в одно или несколько временных окон с учетом столбца, определяющего временную метку. Начало окна включает в себя, но конец окна является исключительным, например. 12:05 будет в окне [12:05,12:10), но не в [12:00,12:05). Windows может поддерживать точность до микросекунды. Windows в порядке месяцев не поддерживается. Окна начинаются с 01.01.1970, 00:00:00 UTC. |
| window(timeColumn: Column, windowDuration: String): Column | Генерирует чередующиеся временные окна с учетом столбца, определяющего временную метку. Начало окна включает в себя, но конец окна является исключительным, например. 12:05 будет в окне [12:05,12:10), но не в [12:00,12:05). Windows может поддерживать точность до микросекунды. Windows в порядке месяцев не поддерживается. Окна начинаются с 01.01.1970, 00:00:00 UTC. |

# Примеры функций Spark Date

Ниже приведены наиболее часто используемые примеры функций даты.

## current\_date() and date\_format()

Мы увидим, как получить текущую дату и преобразовать дату в определенный формат даты с помощью date\_format() на примере Scala. В приведенном ниже примере дата анализируется и преобразуется из формата «yyyy-dd-mm» в формат «MM-dd-yyyy».

*import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"))  
 .toDF("Input")  
 .select(   
 current\_date()as("current\_date"),  
 col("Input"),  
 date\_format(col("Input"), "MM-dd-yyyy").as("format")  
 )  
.show()*

|  |  |  |
| --- | --- | --- |
| current\_date | Input | format |
| 2019-07-23 | 2019-01-23 | 01-23-2019 |

## to\_date()

В приведенном ниже примере строка в формате даты «ММ/дд/гггг» преобразуется в DateType «гггг-ММ-дд» с использованием to\_date() в примере Scala.

import org.apache.spark.sql.functions.\_  
Seq(("04/13/2019"))  
 .toDF("Input")  
 .select( col("Input"),  
to\_date(col("Input"), "MM/dd/yyyy").as("to\_date"))  
.show()

|  |  |
| --- | --- |
| Input | to\_date |
| 04/13/2019 | 2019-04-13 |

## datediff()

В приведенном ниже примере возвращается разница между двумя датами с использованием Dateiff() в примере Scala.

import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"),("2019-06-24"),("2019-09-20"))  
 .toDF("input")  
 .select( col("input"), current\_date(),   
 datediff(current\_date(),col("input")).as("diff"))  
.show()

|  |  |  |
| --- | --- | --- |
| input | current\_date() | diff |
| 2019-01-23 | 2019-07-23 | 181 |
| 2019-06-24 | 2019-07-23 | 29 |
| 2019-09-20 | 2019-07-23 | -59 |

## months\_between()

Нижеприведенный пример возвращает месяцы между двумя датами с помощью months\_between() на языке Scala.

import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"),("2019-06-24"),("2019-09-20"))  
 .toDF("date")  
 .select( col("date"), current\_date(),   
 datediff(current\_date(),col("date")).as("datediff"),  
 months\_between(current\_date(),col("date")).as("months\_between")  
 ).show()

|  |  |  |  |
| --- | --- | --- | --- |
| input | current\_date() | diff | months\_between |
| 2019-01-23 | 2019-07-23 | 181 | 6.0 |
| 2019-06-24 | 2019-07-23 | 29 | 0.96774194 |
| 2019-09-20 | 2019-07-23 | -59 | -1.90322581 |

## trunc()

В приведенном ниже примере дата усекается в указанной единице измерения с помощью trunc() на языке Scala.

import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"),("2019-06-24"),("2019-09-20"))  
 .toDF("input")  
 .select( col("input"),  
 trunc(col("input"),"Month").as("Month\_Trunc"),  
 trunc(col("input"),"Year").as("Month\_Year"),  
 trunc(col("input"),"Month").as("Month\_Trunc")  
 )  
.show()

|  |  |  |  |
| --- | --- | --- | --- |
| input | Month\_Trunc | Month\_Year | Month\_Trunc |
| 2019-01-23 | 2019-01-01 | 2019-01-01 | 2019-01-01 |
| 2019-06-24 | 2019-06-01 | 2019-01-01 | 2019-06-01 |
| 2019-09-20 | 2019-09-01 | 2019-01-01 | 2019-09-01 |

## add\_months() , date\_add(), date\_sub()

Здесь мы добавляем и вычитаем дату и месяц из заданных входных данных.

import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"),("2019-06-24"),("2019-09-20")).toDF("input")  
 .select( col("input"),  
 add\_months(col("input"),3).as("add\_months"),  
 add\_months(col("input"),-3).as("sub\_months"),  
 date\_add(col("input"),4).as("date\_add"),  
 date\_sub(col("input"),4).as("date\_sub")  
 )  
.show()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| input | add\_months | sub\_months | date\_add | date\_sub |
| 2019-01-23 | 2019-04-23 | 2018-10-23 | 2019-01-27 | 2019-01-19 |
| 2019-06-24 | 2019-09-24 | 2019-03-24 | 2019-06-28 | 2019-06-20 |
| 2019-09-20 | 2019-12-20 | 2019-06-20 | 2019-09-24 | 2019-09-16 |

## year(), month(), month() dayofweek(), dayofmonth(), dayofyear() next\_day(), weekofyear()

import org.apache.spark.sql.functions.\_  
Seq(("2019-01-23"),("2019-06-24"),("2019-09-20"))  
 .toDF("input")  
 .select( col("input"), year(col("input")).as("year"),   
 month(col("input")).as("month"),   
 dayofweek(col("input")).as("dayofweek"),   
 dayofmonth(col("input")).as("dayofmonth"),   
 dayofyear(col("input")).as("dayofyear"),   
 next\_day(col("input"),"Sunday").as("next\_day"),   
 weekofyear(col("input")).as("weekofyear")   
 ).show()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| input | year | month | dayofweek | dayofmonth | dayofyear | next\_day | weekofyear |
| 2019-01-23 | 2019 | 1 | 4 | 23 | 23 | 2019-01-27 | 4 |
| 2019-06-24 | 2019 | 6 | 2 | 24 | 175 | 2019-06-30 | 26 |
| 2019-09-20 | 2019 | 9 | 6 | 20 | 263 | 2019-09-22 | 38 |

# Примеры функций Spark Timestamp

Ниже приведены наиболее часто используемые примеры функций временных меток.

## current\_timestamp()

Возвращает текущую метку времени в формате Spark по умолчанию yyyy-MM-dd HH:mm:ss.

import org.apache.spark.sql.functions.\_  
val df = Seq((1)).toDF("seq")  
val curDate = df.withColumn("current\_date",current\_date().as("current\_date"))  
 .withColumn("current\_timestamp",current\_timestamp().as("current\_timestamp"))  
curDate.show(false)

Выдача меньше кода

|  |  |  |
| --- | --- | --- |
| seq | current\_date | current\_timestamp |
| 1 | 2019-11-16 | 2019-11-16 21:00:55.349 |

## to\_timestamp()

Преобразует строковую метку времени в формат типа отметки времени.

import org.apache.spark.sql.functions.\_  
 val dfDate = Seq(("07-01-2019 12 01 19 406"),  
 ("06-24-2019 12 01 19 406"),  
 ("11-16-2019 16 44 55 406"),  
 ("11-16-2019 16 50 59 406")).toDF("input\_timestamp")  
 dfDate.withColumn("datetype\_timestamp",  
 to\_timestamp(col("input\_timestamp"),"MM-dd-yyyy HH mm ss SSS"))  
 .show(false)

|  |  |
| --- | --- |
| input\_timestamp | datetype\_timestamp |
| 07-01-2019 12 01 19 406 | 2019-07-01 12:01:19 |
| 06-24-2019 12 01 19 406 | 2019-06-24 12:01:19 |
| 11-16-2019 16 44 55 406 | 2019-11-16 16:44:55 |
| 11-16-2019 16 50 59 406 | 2019-11-16 16:50:59 |

## hour(), Minute() and second()

import org.apache.spark.sql.functions.\_  
 val df = Seq(("2019-07-01 12:01:19.000"),  
 ("2019-06-24 12:01:19.000"),  
 ("2019-11-16 16:44:55.406"),  
 ("2019-11-16 16:50:59.406")).toDF("input\_timestamp")  
 df.withColumn("hour", hour(col("input\_timestamp")))  
 .withColumn("minute", minute(col("input\_timestamp")))  
 .withColumn("second", second(col("input\_timestamp")))  
 .show(false)

|  |  |  |  |
| --- | --- | --- | --- |
| input\_timestamp | hour | minute | second |
| 2019-07-01 12:01:19.000 | 12 | 1 | 19 |
| 2019-06-24 12:01:19.000 | 12 | 1 | 19 |
| 2019-11-16 16:44:55.406 | 16 | 44 | 55 |
| 2019-11-16 16:50:59.406 | 16 | 50 | 59 |

# Заключение:

В этом посте я объединил полный список функций Spark Date и Timestamp с описанием и примерами некоторых часто используемых. Дополнительную информацию об этом можно найти [в следующем блоге](https://databricks.com/blog/2015/09/16/apache-spark-1-5-dataframe-api-highlights.html).

# Related Articles

* [Spark – получите день года и неделю года](https://sparkbyexamples.com/spark/spark-how-to-get-a-day-and-week-of-year/)
* [Spark – Как получить текущую дату и метку времени](https://sparkbyexamples.com/spark/spark-current-date-and-timestamp/)
* [Spark временная метка — извлечение часа, минуты и секунды.](https://sparkbyexamples.com/spark/spark-extract-hour-minute-and-second-from-timestamp/)
* [Spark SQL – работа с меткой времени Unix](https://sparkbyexamples.com/spark/spark-sql-unix-timestamp/)
* [Разница в временной метке искры в секундах, минутах и часах](https://sparkbyexamples.com/spark/spark-difference-between-two-timestamps-in-seconds-minutes-and-hours/)
* [Параллельное чтение Spark JDBC](https://sparkbyexamples.com/spark/spark-jdbc-parallel-read/)
* [Spark SQL datediff()](https://sparkbyexamples.com/spark/spark-sql-datediff/)
* [Spark Получите текущие настройки SparkContext](https://sparkbyexamples.com/spark/spark-get-the-current-sparkcontext-settings/)
* [Невозможно вызвать методы в остановленном SparkContext в Spark.](https://sparkbyexamples.com/spark/cannot-call-methods-on-a-stopped-sparkcontext/)
* [Настройте количество исполнителей Spark, ядра и память](https://sparkbyexamples.com/spark/spark-tune-executor-number-cores-and-memory/)
* [Рассчитать разницу между двумя датами в днях, месяцах и годах](https://sparkbyexamples.com/spark/spark-calculate-difference-between-two-dates-in-days-months-and-years/)