How to Extract Substrings in PySpark Using the `substr` Function

Authored by

stats writer

January 20, 2026

Working with large datasets often requires sophisticated string manipulation, and PySpark provides robust functions for this purpose. To efficiently extract specific sections of text, known as substrings, from columns within a DataFrame, we primarily rely on the `substr` function (or its alias, `substring`).

The `substr` function is highly versatile, requiring three key parameters to perform the extraction: the input column containing the string data, the starting index (which is 1-based in PySpark), and the required length of the substring. For instance, if you have a column named "name" containing values like "John Smith" and the goal is to isolate the first name, you might use an expression such as `df.select(substr(df, 1, 4).alias('first_name'))`. This powerful operation creates a derived column named "first_name" by extracting the initial four characters from the original "name" field. Beyond simple name splitting, these techniques are essential for parsing complex identifiers, dates, or log entries.

To handle diverse data cleaning and transformation needs, PySpark offers multiple approaches for isolating text segments within a column. We will explore five primary methods here, utilizing both fixed-length functions and delimiter-based functions.

Understanding the PySpark `substr` Function

The core of fixed-length string extraction in DataFrames is the `F.substr()` function. It is important to remember that PySpark indices are **1-based**, meaning the first character is at position 1, not 0 (unlike native Python indexing). This function is critical for standardized data parsing where the required substring length is known or predictable.

Method 1: Extract Substring from Beginning of String

This method involves starting the extraction at index 1 and specifying the desired length. It is the most common use case for extracting prefixes or short codes, ensuring consistency when data starts uniformly.

from pyspark.sql import functions as F

#extract first three characters from team column
df_new = df.withColumn('first3', F.substring('team', 1, 3))

Method 2: Extract Substring from Middle of String

To extract content from the middle, you must accurately determine the starting position within the string. For example, if we want four characters starting from the second position, the start index is 2 and the length is 4. This technique is often necessary when dealing with fixed-width data fields

that have been concatenated.

from pyspark.sql import functions as F

#extract four characters starting from position two in team column
df_new = df.withColumn('mid4', F.substring('team', 2, 4))

Method 3: Extract Substring from End of String

PySpark supports **negative indexing** within the `substr` function to facilitate backward traversal. Using a negative starting index allows us to easily isolate suffixes or trailing segments, regardless of the overall string length. For example, using `-3` as the start index will begin counting three characters from the end of the string. The length parameter must still be specified as a positive integer.

from pyspark.sql import functions as F

#extract last three characters from team column
df_new = df.withColumn('last3', F.substring('team', -3, 3))

Delimiter-Based Extraction Using `substring_index`

While `substr` works well for fixed-length or position-based extraction, many real-world strings (like URLs, file paths, or complex identifiers) are structured using delimiters (e.g., periods, slashes, spaces). For these scenarios, PySpark's `substring_index` function provides a more practical solution. This function takes three arguments: the column, the delimiter, and a count (which determines whether to keep content before or after the delimiter).

Method 4: Extract Substring Before Specific Character

To capture all content preceding a specific separator, we use a positive count value (typically `1`). This tells PySpark to return the substring that appears before the Nth occurrence of the specified delimiter. In cases where we want the first word or segment, using a count of 1 is sufficient.

from pyspark.sql import functions as F

#extract all characters before space in team column
df_new = df.withColumn('beforespace', F.substring_index('team', ' ', 1))

Method 5: Extract Substring After Specific Character

Conversely, to isolate content that appears after a delimiter, we utilize a negative count value (typically `-1`). A count of -1 instructs PySpark to return the text following the delimiter, counting backward from the end of the string. This is ideal for extracting suffixes, last names, or specific elements at the end of a path or identifier.

from pyspark.sql import functions as F

#extract all characters after space in team column
df_new = df.withColumn('afterspace', F.substring_index('team', ' ', -1))

Setting Up the Demonstration Environment

To demonstrate these five powerful methods, we will first define a sample DataFrame containing NBA team names and points. This initial setup requires importing the `SparkSession` and defining our data schema to create the environment.

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

#define data
data = ,
,
,
,
,
,
]

#define column names
columns =

#create dataframe using data and column names
df = spark.createDataFrame(data, columns)

#view dataframe
df.show()

+--------------+------+
| team|points|
+--------------+------+
| Dallas Mavs| 18|

| Brooklyn Nets| 33|
| Atlanta Hawks| 12|
|Boston Celtics| 15|
| Miami Heat| 19|
|Cleveland Cavs| 24|
| Orlando Magic| 28|
+--------------+------+

Example 1: Extract Substring from Beginning of String

Applying Fixed-Length Extraction (`substr`)

This first example demonstrates the most basic application of the `substr` function. We instruct PySpark to start at the first position (index 1) and capture a fixed length of 3 characters. This method is highly effective for generating standardized abbreviations or initials from text fields like the team name.

from pyspark.sql import functions as F

#extract first three characters from team column
df_new = df.withColumn('first3', F.substring('team', 1, 3))

#view updated DataFrame
df_new.show()

+--------------+------+------+
| team|points|first3|
+--------------+------+------+
Dallas Mavs	18	Dal
Brooklyn Nets	33	Bro
Atlanta Hawks	12	Atl
Boston Celtics	15	Bos
Miami Heat	19	Mia
Cleveland Cavs	24	Cle
Orlando Magic	28	Orl
+--------------+------+------+

Example 2: Extract Substring from Middle of String

Targeting Internal Characters

When the desired data segment is located internally, precise index specification is mandatory. In this instance, we are extracting four characters, beginning at the second position of the string (start index 2). This operation allows data engineers to isolate specific embedded codes or sequences that are not located at either end of the string.

from pyspark.sql import functions as F

#extract four characters starting from position two in team column
df_new = df.withColumn('mid4', F.substring('team', 2, 4))

#view updated DataFrame
df_new.show()

+--------------+------+----+
| team|points|mid4|
+--------------+------+----+
Dallas Mavs	18	alla
Brooklyn Nets	33	rook
Atlanta Hawks	12	tlan
Boston Celtics	15	osto
Miami Heat	19	iami
Cleveland Cavs	24	leve
Orlando Magic	28	rlan
+--------------+------+----+

Example 3: Extract Substring from End of String

Utilizing Negative Indexing for Suffixes

This technique leverages negative indexing to extract the final three characters from the **team** column. By setting the starting index to `-3`, we ensure that the extraction automatically adjusts for strings of varying lengths, focusing only on the trailing characters. This is often used for isolating file extensions, country codes, or team suffixes.

from pyspark.sql import functions as F

```
#extract last three characters from team column
df_new = df.withColumn('last3', F.substring('team', -3, 3))

#view updated DataFrame
df_new.show()

+--------------+------+-----+
| team|points|last3|
+--------------+------+-----+
| Dallas Mavs| 18| avs|
| Brooklyn Nets| 33| ets|
| Atlanta Hawks| 12| wks|
|Boston Celtics| 15| ics|
| Miami Heat| 19| eat|
|Cleveland Cavs| 24| avs|
| Orlando Magic| 28| gic|
+--------------+------+-----+
```

Example 4: Extract Substring Before Specific Character

Using `substring_index` for First Segment Isolation

This example showcases the power of `F.substring_index` when dealing with delimited data. By specifying the space character (`' '`) as the delimiter and setting the count parameter to `1`, we accurately extract the first word (the city name) from the **team** column. The positive count signifies that we want everything occurring before the first instance of the delimiter.

```
from pyspark.sql import functions as F

#extract all characters before space in team column
df_new = df.withColumn('beforespace', F.substring_index('team', ' ', 1))

#view updated DataFrame
df_new.show()

+--------------+------+-----------+
| team|points|beforespace|
+--------------+------+-----------+
| Dallas Mavs| 18| Dallas|
| Brooklyn Nets| 33| Brooklyn|
```

| Atlanta Hawks| 12| Atlanta|
|Boston Celtics| 15| Boston|
| Miami Heat| 19| Miami|
|Cleveland Cavs| 24| Cleveland|
| Orlando Magic| 28| Orlando|
+--------------+------+-----------+

Example 5: Extract Substring After Specific Character

Using `substring_index` for Last Segment Isolation

To complete the splitting operation, we utilize `F.substring_index` with a negative count. By using `-1`, we tell PySpark to start counting from the right side of the string, ensuring that all characters after the last delimiter (the team nickname) are captured. This is a robust method for separating compound strings based on the final delimiter.

from pyspark.sql import functions as F

#extract all characters after space in team column
df_new = df.withColumn('afterspace', F.substring_index('team', ' ', -1))

#view updated DataFrame
df_new.show()

+--------------+------+----------+
| team|points|afterspace|
+--------------+------+----------+
Dallas Mavs	18	Mavs
Brooklyn Nets	33	Nets
Atlanta Hawks	12	Hawks
Boston Celtics	15	Celtics
Miami Heat	19	Heat
Cleveland Cavs	24	Cavs
Orlando Magic	28	Magic
+--------------+------+----------+

Further PySpark Data Manipulation

Summary of String Extraction Methods

Effectively extracting substrings in PySpark is a foundational skill for data preparation. Whether you require fixed-length segmentation using `substr` (ideal for standardizing codes and abbreviations) or dynamic delimiter-based segmentation using `substring_index` (essential for parsing complex, variable-length fields), mastering these functions enables high-throughput data cleaning and feature engineering within the distributed computing environment of Apache Spark.

The following tutorials explain how to perform other common tasks in PySpark: