everis

an NTT DATA Company

ZURICH

everis Bigdata

PySpark Training

06/11/2018

ZURICH

everis
an NTT DATA Company

1. **Get Started with Apache Spark**
    i. **Introduction to Spark**
    ii. **Install pyspark and run the first job**
2. Spark RDD
    i. RDD basics
    ii. Create RDDs
    iii. Transformations vs Actions
    iv. Basic Transformations
    v. Basic Actions
    vi. Summary of RDD operations
3. Spark PairRDD
    i. Introduction to pair RDD
    ii. Create pair RDDs
    iii. Basic transformations
    iv. Join operations
4. SparkSQL
    i. Introduction to Spark SQL
    ii. Dataframe or RDD
    iii. Spark SQL joins
    iv. Home retention example

Getting started with Apache Spark

Introduction to Apache Spark

Apache Spark is a fast, **in-memory** data processing engine which allows data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative, access to datasets.

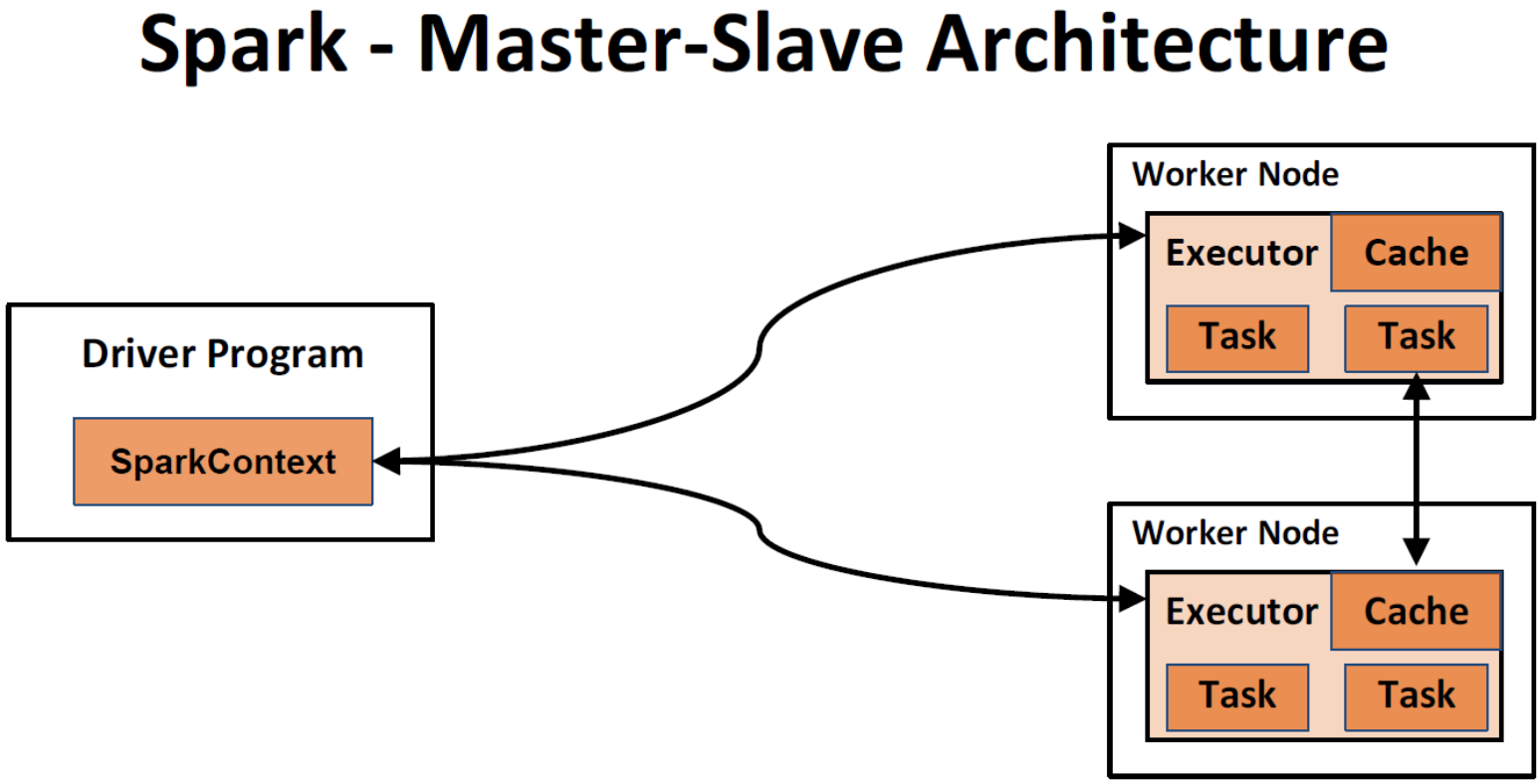
Getting started with Apache Spark

¿Que es Apache Spark?

- Motor de procesamiento
  - Para grandes cantidades de información.
  - Rápido
  - Propósito general
- Programado en Scala
  - Lenguaje de **programación funcional** que se ejecuta en una JVM
- Programable en Scala, Java, Python y R
- Procesamiento en local y en clúster (Standalone, Mesos y YARN)
- Un programa Spark es un programa normal conocido como Driver
  - Se realizan llamadas a Spark a partir de un objeto especial, el Spark Context
  - Bajo determinadas circunstancias invocará una ejecución distribuida

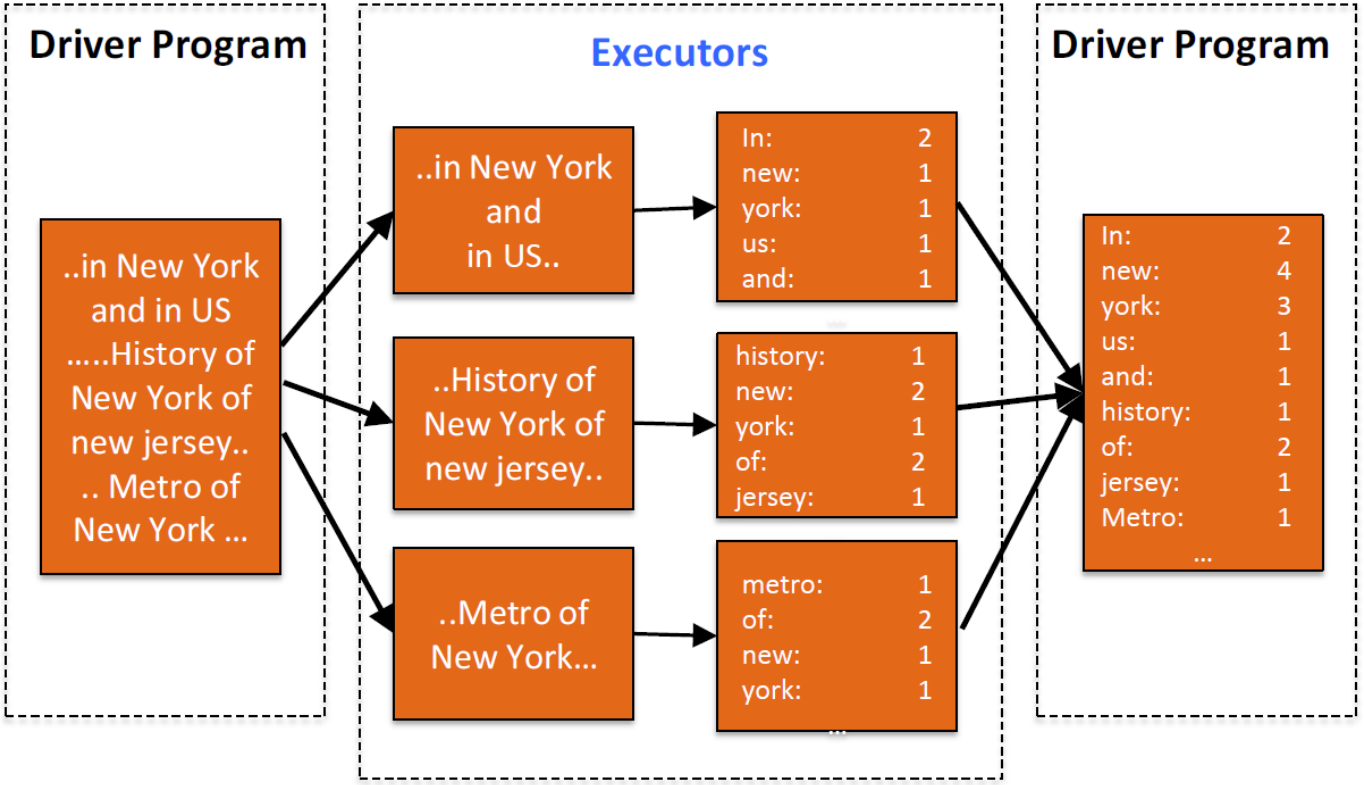
Getting started with Apache Spark

Spark Architecture



Getting started with Apache Spark

Spark Architecture



everis Bigdata

ZURICH

everis an NTT DATA Company

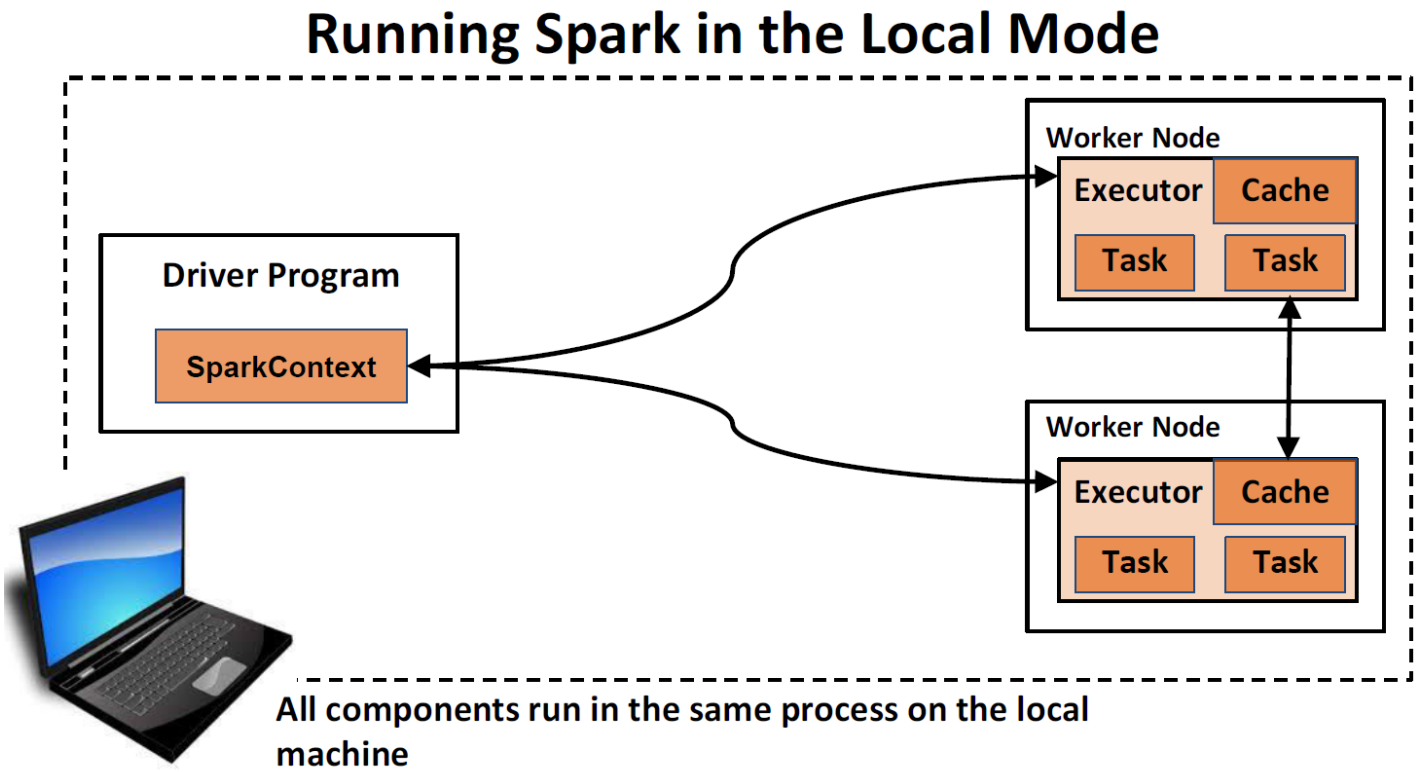
Getting started with Apache Spark

Crear un Driver

```
from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    conf =SparkConf().setAppName("test").setMaster("local[*]")
    sc = SparkContext(conf = conf)
```

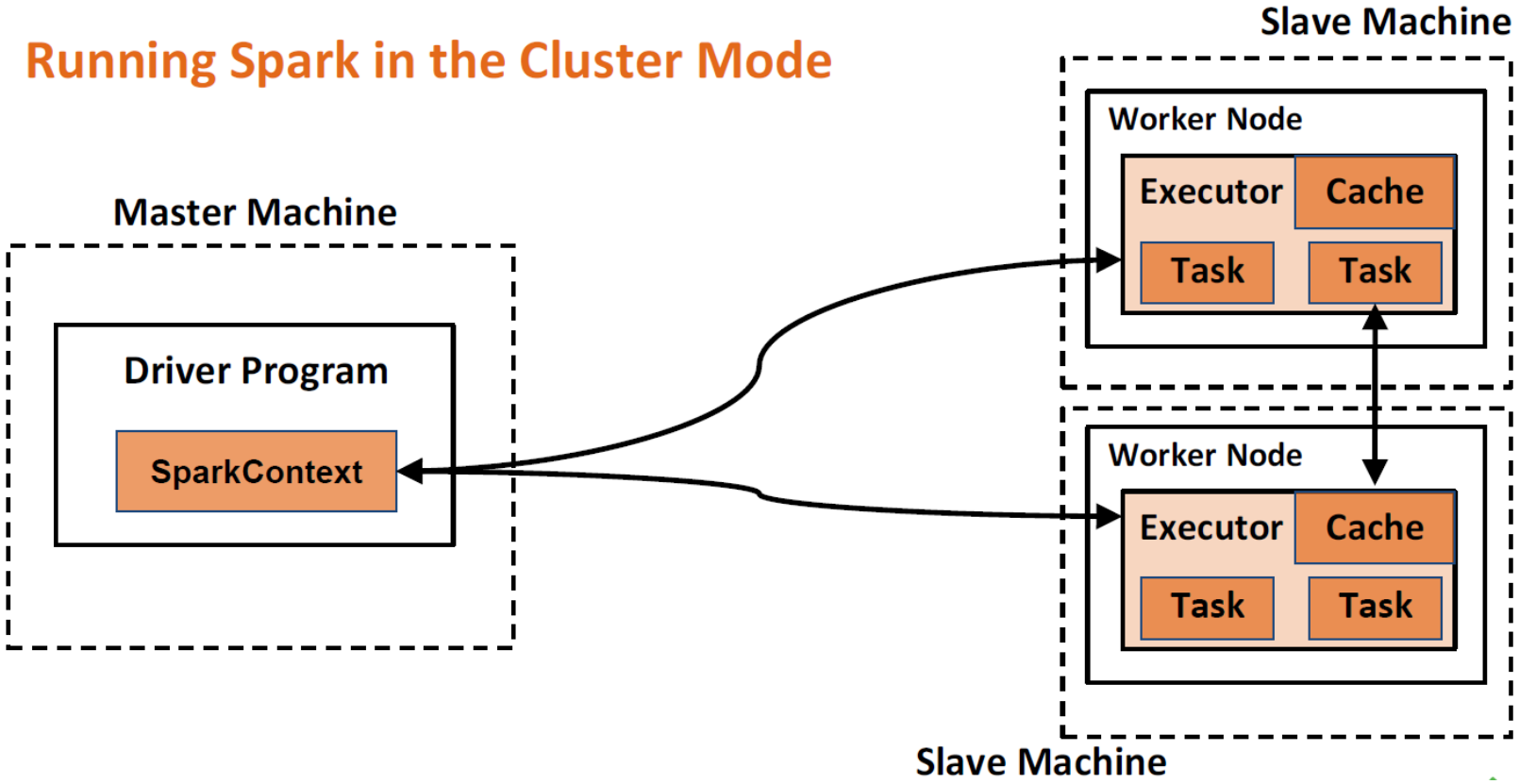
Getting started with Apache Spark

Running Spark in the local mode



Getting started with Apache Spark

Running Spark in the cluster mode

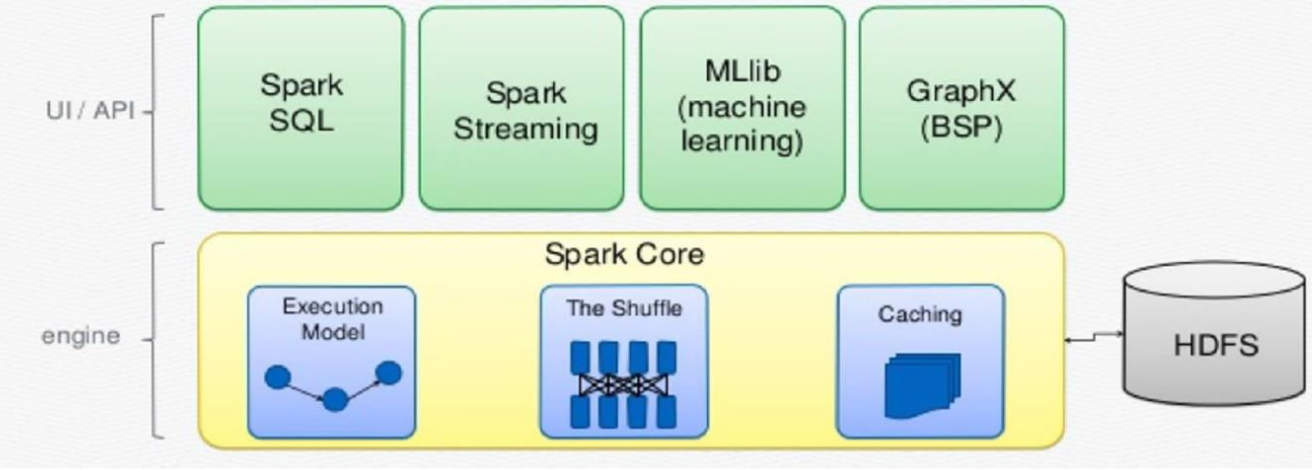


Getting started with Apache Spark

Generality

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

Spark Components



Getting started with Apache Spark

Set up spark_training environment

1. Download *pyspark-training* project:
   https://github.com/albapast/pyspark-training.git

2. Open Pycharm IDE and open the training project downloaded.

3. Go to project settings and create a python virtualenv for this training.

4. Open terminal Windows on Pycharm and install pyspark:
   pip install pyspark

5. Run the WordCount.py example in order to check that the training environment is working.

Getting started with Apache Spark

Datasets (I)

- Un dataset es un conjunto de datos
- Spark ha tenido varias abstracciones para los dataset, cada una con su API
  - RDD
  - DataFrame
  - DataSet
- Un programa Spark consiste en transformar datasets
- Los datasets son inmutables
- Imaginad un dataset como un array gigante, inmutable y distribuido por el clúster.

Getting started with Apache Spark

Datasets (II). Comparativa

- RDD
  - Recomendada para Spark 1.4 y anteriores
  - Cercana a la programación (El compilador ayuda con los errores)
- DataFrame
  - Abstracción recomendada para Spark 1.5 y 1.6 y superiores
  - Cercana al SQL (Cuenta con optimizador de consultas)
- Durante el tema de Spark veremos RDD y PairRDD
- Durante el tema de SparkSQL veremos DataFrame

ZURICH

everis
an NTT DATA Company

1. Get Started with Apache Spark
    i. Introduction to Spark
    ii. Install pyspark and run the first job
2. **Spark RDD**
    i. **RDD basics**
    ii. **Create RDDs**
    iii. **Transformations vs Actions**
    iv. **Basic Transformations**
    v. **Basic Actions**
    vi. **Summary of RDD operations**
3. Spark PairRDD
    i. Introduction to pair RDD
    ii. Create pair RDDs
    iii. Basic transformations
    iv. Join operations
4. SparkSQL
    i. Introduction to Spark SQL
    ii. Dataframe or RDD
    iii. Spark SQL joins
    iv. Home retention example

Spark RDD

RDD (Resilient Distributed Dataset)

- Las RDDs son la unidad de información fundamental de Spark
  - Dataset: Conjunto de datos
  - Distributed: Distribuido entre todo el clúster
  - Resilient: Si una parte de la información se pierde puede ser regenerada

Spark RDD

Crear una RDD

- Hay tres formas de crear una RDD
  - A partir de un **conjunto** de ficheros (ficheros, directorios y comodines)

```
lines = sc.textFile("in/word_count.text")
```

  - A partir de datos en memoria

```
array=[1,2,3,4,5]
rdd= sc.parallelize(array)
```

  - A partir de otra RDD

```
rdd2= rdd.take(2)
```

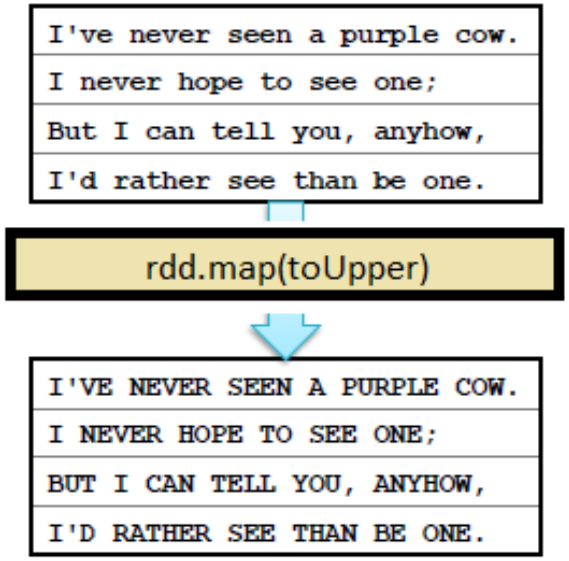
Spark RDD

Transformaciones (1)

- Programar en Spark consiste en aplicar funciones a RDDs para generar nuevos RDDs

- **rdd.map(fn)**: por cada elemento se genera un elemento
  - fn(x) = y
- El tipo que devuelve el resultado de aplicar la función
- map no tiene por qué ser el mismo tipo que el input:

```
lines = sc.textFile("in/uppercase.text")
lengths = lines.map(lambda line: len(line))
```




```
def toUpper(s):
    return s.upper()
```

Spark RDD

Transformaciones (1)

- **rdd.filter(fn)**: devuelve un RDD formado por aquellos elementos que cumplen la función de filtro.
- Suele usarse para eliminar filas no válidas y limpiar el RDD de entrada o tan sólo seleccionar un subset del RDD inicial basándose en la función filtro.

```
cleanedLines = lines.filter(lambda line: line.strip())
```

Spark RDD

Transformaciones (1)

Vamos a practicar:

1. AirportUSAProblem.py

2. AirportByLatitudeProblem.py

Spark RDD

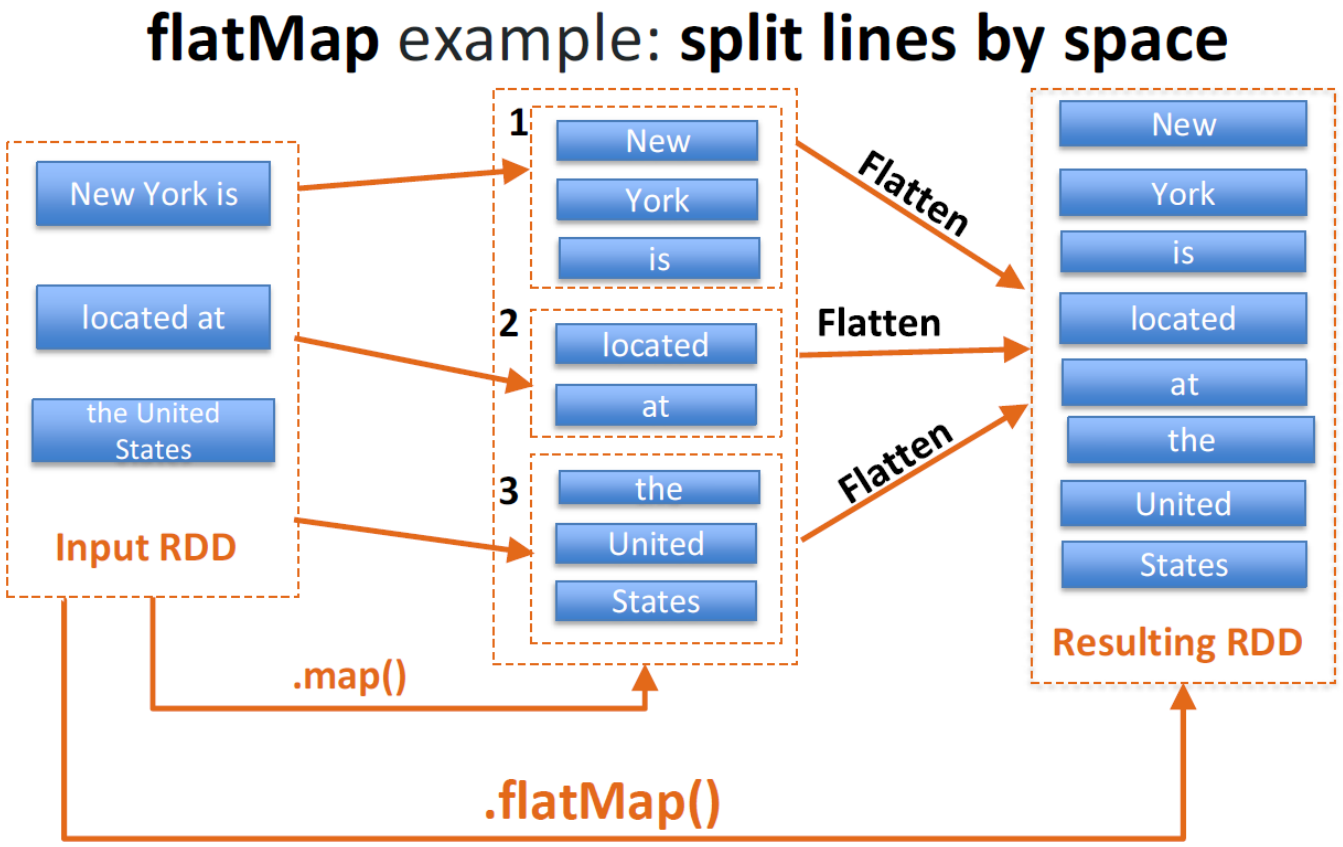
Transformaciones (2)

- **rdd.flatMap(fn)**: por cada elemento se generan múltiples elementos
- **flatmap vs map**:

```
def map(self, f, preservesPartitioning=False):
    """
    Return a new RDD by applying a function to each element of this RDD.

    >>> rdd = sc.parallelize(["b", "a", "c"])
    >>> sorted(rdd.map(lambda x: (x, 1)).collect())
    [('a', 1), ('b', 1), ('c', 1)]
    """
def flatMap(self, f, preservesPartitioning=False):
    """
    Return a new RDD by first applying a function to all elements of this
    RDD, and then flattening the results.

    >>> rdd = sc.parallelize([2, 3, 4])
    >>> sorted(rdd.flatMap(lambda x: range(1, x)).collect())
    [1, 1, 1, 2, 2, 3]
```

Spark RDD

Transformaciones (2)



Spark RDD

Transformaciones (2)

Vamos a practicar:

- WordCount.py

Spark RDD

Transformaciones (3)

- Algunas transformaciones tienen una lógica interna más compleja
  - **rdd.sortBy(fn):** contendrá los mismos elementos pero ordenados
    - fn(x) = y
- Otras transformaciones no requieren de una función
  - **rdd.distinct():** elimina elementos duplicados.
- Otras transformaciones involucran más de una RDD
  - **rdd.intersection(rdd2):** Sólo contendrá los elementos que aparezcan en ambas.
  - **rdd.subtract(rdd2):** Contendrá los elementos que aparezcan en la primera y no en la segunda
  - **rdd.union(rdd2):** Contendrá todos los elementos de ambas.
- Para todo lo demás …
  - https://spark.apache.org/docs/latest/api/python

Spark RDD

Acciones

- Consiste en extraer información de una RDD
  - Provoca el procesamiento.

- **rdd.collect():** Devuelve al programa driver un array con todos los elementos de la RDD

  - El conjunto de datos a recopilar debe caber en la memoria de una sola máquina, ya que debe estar copiado en el driver cuando se llama a la acción collect(). Es por eso que esta acción no debe usarse en grandes conjuntos de datos.

  - Se usa ampliamente en pruebas unitarias, para comparar el valor de nuestro RDD con nuestro resultado esperado.

`Ejemplo: CollectExample.py`

Spark RDD

Acciones

- **rdd.count():** Devuelve el número de filas de la RDD
- **rdd.countByValue():** Devuelve el número de valores únicos en cada fila de la RDD.

  `Ejemplo: CountExample.py`

- **rdd.take(**n**):** Devuelve al programa driver un array con los n primeros elementos de la RDD

  `Ejemplo: TakeExample.py`

- **rdd.first:** Devuelve al programa driver el primer elemento de la RDD

- **rdd.saveAsTextFile(**directory**):** Guarda en disco.
  Una línea por elemento. N ficheros.

Spark RDD

Acciones

- **rdd.reduce(fn):** Reduce el número de elementos de un RDD usando la función especificada.
  - Con esta operación podemos hacer diferentes tipos de agregaciones.

```
def reduce(self, f):
    """
    Reduces the elements of this RDD using the specified commutative and
    associative binary operator. Currently reduces partitions locally.
product = integerRdd.reduce(lambda x, y: x*y)
```

Ejemplos:

- ReduceExample.py
- SumOfNumbersProblem.py

Spark RDD

General Workflow

- Generate **initial RDDs** from external data.
- Apply **transformations**.
- Launch **actions.**

Spark RDD

Summary

- Crear RDDs
  - rdd = sc.textFile(ficheros y directorios con comodines)
  - rdd = sc.parallelize(collection)
- Transformar
  - rdd2 = rdd.map(fn)
  - rdd2 = rdd.filter(fn)
- Extraer
  - rdd2.count()
  - rdd2.collect()
- Transformaciones siempre devuelven RDDs, mientras que las acciones devuelven otro tipo de datos (int, float, array…)
- Adquirir nuevo conocimiento
  - API: https://spark.apache.org/docs/latest/api/python

ZURICH

everis
an NTT DATA Company

1. Get Started with Apache Spark
    i. Introduction to Spark
    ii. Install pyspark and run the first job
2. Spark RDD
    i. RDD basics
    ii. Create RDDs
    iii. Transformations vs Actions
    iv. Basic Transformations
    v. Basic Actions
    vi. Summary of RDD operations
3. **Spark PairRDD**
    i. Introduction to pair RDD
    ii. Create pair RDDs
    iii. Basic transformations
    iv. Join operations
4. SparkSQL
    i. Introduction to Spark SQL
    ii. Dataframe or RDD
    iii. Spark SQL joins
    iv. Home retention example

Spark PairRDD

PairRDD

- PairRDD es una RDD que permite transformaciones basadas en Map/Reduce
- Los registros de una PairRDD son tuplas de dos elementos
- Cada uno de los dos elementos puede ser cualquier tipo o estructura
- Al **primer** elemento se le conoce como **llave (key)**
- Al **segundo** elemento se le conoce como **valor (value)**

- Spark permite implementaciones map/reduce flexibles
  - Las funciones map y reduce pueden intercalarse en cualquier orden

Spark PairRDD

Crear PairRDD

- Siempre a partir de otro RDD
- Aplicando una transformación **map**

```
inputStrings = ["Lily 23", "Jack 29", "Mary 29", "James 8"]
regularRDDs = sc.parallelize(inputStrings)

pairRDD = regularRDDs.map(lambda s: (s.split(" ")[0], s.split("")[1]))
pairRDD.coalesce(1).saveAsTextFile("out/pair_rdd_from_regular_rdd")
```

Output:

```
('Lily', '23')
('Jack', '29')
('Mary', '29')
('James', '8')
```

- Transformando tuplas en un PairRDD

```
tuples = [("Lily", 23), ("Jack", 29), ("Mary", 29), ("James", 8)]
pairRDD = sc.parallelize(tuples)

pairRDD.coalesce(1).saveAsTextFile("out/pair_rdd_from_tuple_list")
```

Spark PairRDD

Transformaciones (1)

- Las transformaciones **filter y map** pueden usarse del mismo modo que en los rdd's:

```
airportsRDD = sc.textFile("in/airports.text")

airportPairRDD = airportsRDD.map(lambda line: \
    (Utils.COMMA_DELIMITER.split(line)[1],
     Utils.COMMA_DELIMITER.split(line)[3]))

airportsNotInUSA = airportPairRDD.filter(lambda keyValue:
keyValue[1] != "\"United States\"")

airportsNotInUSA.saveAsTextFile("out/airports_not_in_usa_pair_rdd.
text")
```

Spark PairRDD

Transformaciones (1)

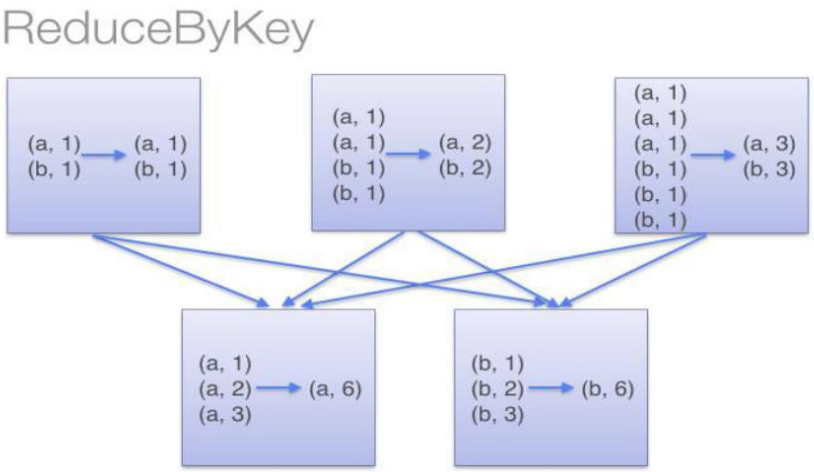
- La mayoría de las veces, cuando trabajamos con pair RDD's, no queremos modificar las llaves, solo queremos tener acceso a los valores.
- **mapValues(fn):** convierte los valores de un pair RDD basándose en la función aplicada.
- Ejemplo: AirportUpperCaseProblem.py

```
airportsRDD = sc.textFile("in/airports.text")

airportPairRDD = airportsRDD.map(lambda line: \
    (Utils.COMMA_DELIMITER.split(line)[1], \
  Utils.COMMA_DELIMITER.split(line)[3]))

upperCase = airportPairRDD.mapValues(lambda countryName:
countryName.upper())

upperCase.saveAsTextFile("out/airports_uppercase.text")
```

Spark PairRDD

Transformaciones Reduce (1)

- Sirven para agrupar los valores con la misma llave y consolidarlos
- Siempre generan un shuffle
  - Tráfico de red
  - Hay que intentar reducir en lo posible los datos involucrados.
- **reduceByKey():** ejecuta varias operaciones de reduce en paralelo, una por cada llave en el dataset, donde cada operación combina valores con la misma llave.



Spark PairRDD

Transformaciones Reduce (1)

Vamos a practicar:

- WordCount.py

Spark PairRDD

Transformaciones Reduce (2)

- **rdd.groupByKey():** Agrupa los valores asociados a la misma llave
  - Los valores luego se pueden recorrer mediante una función map
  - Los valores asociados a una llave deben caber en memoria

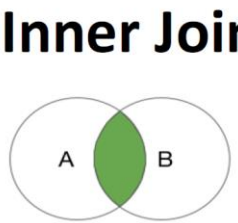
    `Ejemplo: AirportsByCountryProblem.py`

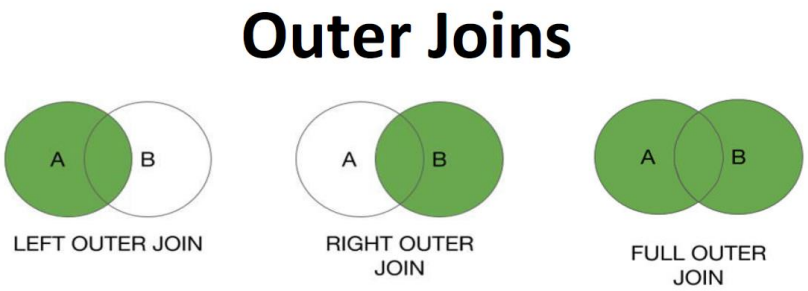
- **rdd.sortByKey()**: Ordena por el orden natural de la llave
  - Lee el dataset dos veces
  - En la primera lee las llaves para conocer la “distribución”
  - En la segunda las llaves se distribuirán y ordenarán en particiones “homogéneas”

Spark PairRDD

Transformaciones Joins

- Las operaciones Join permiten unir 2 rdd's, la cual es la operación mas usada en un pairRDD.
- Tipos de Joins: **leftOuterJoin**, **rightOuterJoin**, **crossJoin**,**innerJoin**, etc.






```
def join(self, other, numPartitions=None):
    """
    Return an RDD containing all pairs of elements with matching keys in
    C{self} and C{other}.
```

Spark PairRDD

Transformaciones Joins

```
ages = sc.parallelize([("Tom", 29), ("John", 22)])
addresses = sc.parallelize([("James", "USA"), ("John", "UK")])

join = ages.join(addresses)
join.saveAsTextFile("out/age_address_join.text")

leftOuterJoin = ages.leftOuterJoin(addresses)
leftOuterJoin.saveAsTextFile("out/age_address_left_out_join.text")

rightOuterJoin = ages.rightOuterJoin(addresses)
rightOuterJoin.saveAsTextFile("out/age_address_right_out_join.text")

fullOuterJoin = ages.fullOuterJoin(addresses)
fullOuterJoin.saveAsTextFile("out/age_address_full_out_join.text")
```

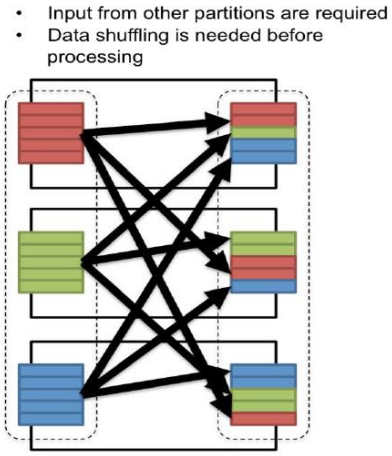
Spark PairRDD

Transformaciones Joins

- Si ambos RDD tienen claves duplicadas, la operación de Join puede expandir el tamaño de los datos. Se recomienda realizar un distinct() o un combineByKey() para reducir el espacio de las llaves si es posible antes de realizar la join.

- La operación Join puede requerir grandes transferencias de red o incluso crear conjuntos de datos más allá de nuestra capacidad de manejar.

- Las Joins, en general, son operaciones con un coste de ejecución muy alto ya que requieren que las llaves (keys) estén en la misma partición para que puedan ser combinados localmente.

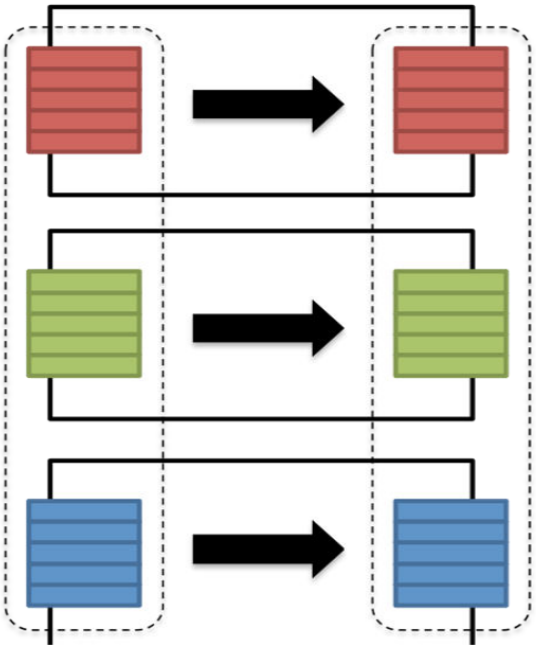


Spark PairRDD

Transformaciones Joins

Avoid Shuffle

- No data movement is needed



- The shuffle can be avoided if both RDDs have a known **partitioner**.
- If they have the same **partitioner**, the data may be colocated; it can prevent network transfer. So it is recommended to call **partitionby** on the two join RDD with the same **partitioner** before joining them.

```
from pyspark.rdd import portable_hash
ages.partitionBy(20, partitionFunc = portable_hash)
addresses.partitionBy(20, partitionFunc = portable_hash)
```

ZURICH

everis
an NTT DATA Company

1. Get Started with Apache Spark
    i. Introduction to Spark
    ii. Install pyspark and run the first job
2. Spark RDD
    i. RDD basics
    ii. Create RDDs
    iii. Transformations vs Actions
    iv. Basic Transformations
    v. Basic Actions
    vi. Summary of RDD operations
3. Spark PairRDD
    i. Introduction to pair RDD
    ii. Create pair RDDs
    iii. Basic transformations
    iv. Join operations
4. **SparkSQL**
    i. **Introduction to Spark SQL**
    ii. **Dataframe or RDD**
    iii. **Spark SQL joins**
    iv. **Home retention example**

SparkSQL

¿Qué es SparkSQL?

- SparkSQL es la interfaz de Spark para trabajar con datos estructurados
- DataFrame API, librería para trabajar con datos como si fueran tablas de BBDD.
  - Define Dataframe con columnas i registros.
- Motor SQL con una interface de comandos.
- Soporta la gran parte del lenguaje SQL-92 standard.
- Puede leer y escribir datos en una variedad de formatos estructurados (por ejemplo, JSON, Hive Tables y Parquet).

SparkSQL

SparkSession

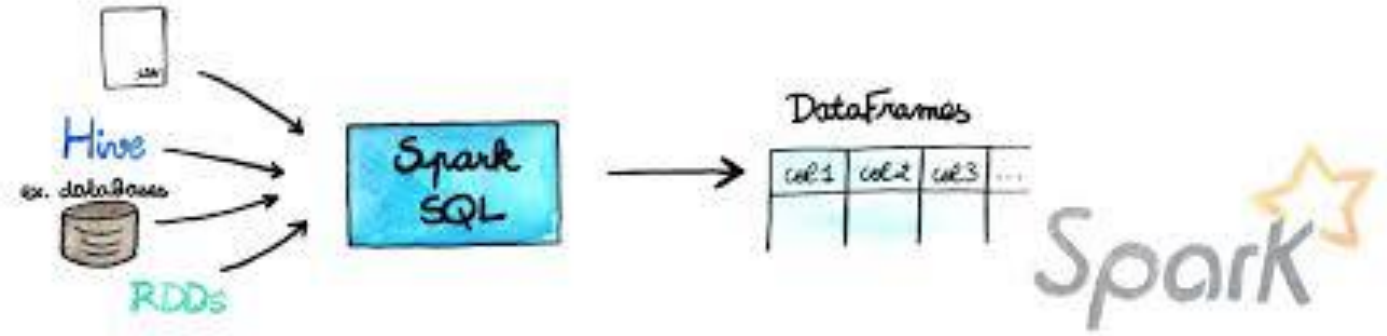
- SparkSession es a SparkSQL lo que el SparkContext es a Spark
- SparkSession es la implementación básica de SparkSQL.

```
session = SparkSession.builder.appName("SessionExample")\
          .master("local[*]")\
          .getOrCreate()
```

SparkSQL

DataFrame

- Un DataFrame de Spark es una colección distribuida de datos organizada en columnas con un esquema definido.
- Podemos generar un DataFrame de diferentes fuentes de información:
  - Desde un archivo (Json, Parquet file, AVRO, csv, txt)
  - Desde un RDD.
  - Desde la ejecución de una query a una tabla Hive.
  - Desde una base de datos



SparkSQL

DataFrame desde un archivo json/csv

- Lectura
  - df = session.read.json(“ruta a json files”)
- Escritura
  - df.write.json(“directorio”)
- Ejemplo lectura csv:

```
file_df_spark= session.read.csv(PATH_VAR.path_out + name_file, header=True)
```

SparkSQL

DataFrame con Bases de datos

- Lectura
  - df = session.read.format("jdbc")
    .option("url", valor para conectar)
    .option("dbtable", valor para conectar)
    .option("user", valor para conectar)
    .option("password", valor para conectar)
    .load()
- Escritura
  - df.write.format("jdbc")
    .option("url", valor para conectar)
    .option("dbtable", valor para conectar)
    .option("user", valor para conectar)
    .option("password", valor para conectar)
    .save()

SparkSQL

Running SQL queries programmatically

- **createOrReplaceTempView(table_name):** registra una vista del dataframe para poder ejecutar secuencias de sql.
- **sql(query):** ejecuta queries de sql sobre la vista creada y devuelve un dataframe.

```
# Register the DataFrame as a SQL temporary view

df.createOrReplaceTempView("people") sqlDF = spark.sql("SELECT * FROM people")

sqlDF.show()
# +----+-------+
# | age| name|
# +----+-------+
# |null|Michael|
# | 30| Andy|
# | 19| Justin|
# +----+-------+
```

SparkSQL

Transformaciones (1)

- **select**: aplica clausula select.
- **distinct**: elimina repetidos.
- **join**: realiza un join entre 2 DF.
- **groupBy**: agrupa por una columna
- **limit**: selecciona los primeros n elementos.
- **filter/where**: selecciona los registros por las condiciones deseadas.
- **sort** and **orderBy**: ordenan en orden ascendente o descendente.
- **printschema**: print dataframe schema
- **alias**: return dataframe or column with an alias set.
- Referencia: https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

SparkSQL

Transformaciones (1)

- Example: Sorting in by columns (descending)

```
peopleDF.sort(peopleDF.age.desc())
```

```
peopleDF.sort(peopleDF("age").desc)
```

`.asc` and `.desc` are column expression methods used with `sort`

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age	name	pcode
46	Diana	null
30	Brayden	94304
19	Carla	10036
null	Alice	94304
null	Étienne	94104

SparkSQL

Transformaciones (1)

- **withColumn**: Crea o modifica una columna existente con el resultado de una expresión
- ```
  invoices.withColumn("classify",
       when("amount">10,"cara").otherwise("barata")
  )
  ```
- También se pueden realizar funciones de agrupado
- ```
  df.groupBy($"col1", $"col2")
       .agg(sum($"col3").alias("facturacion"))
  ```

SparkSQL

Acciones

- **show**: muestra por pantalla los primeros n elementos.
- **count**: devuelve el número de filas del DF.
- **write**: Escritura en algún soporte físico

```
> peopleDF.count()
res7: Long = 5

> peopleDF.show(3)
age  name    pcode
null Alice   94304
30   Brayden 94304
19   Carla   10036
```

SparkSQL

Transformaciones y acciones

Vamos a practicar:

- StackOverFlowSurvey.py

SparkSQL

Transformaciones (2): JOIN

- Se pueden unir dos dataframes de forma sencilla
  - `factura.join(cliente, factura(cliente) === cliente(codigo) && …)`
- Se pueden realizar todos los tipos de join
  - `df1.join(df1, joinexpr, "tipo join")`
  - inner, outer, left_outer, right_outer, leftsemi
  - Default es inner

join(*other*, *on=None*, *how=None*) [source

Joins with another **DataFrame**, using the given join expression.

Parameters:
- **other** – Right side of the join
- **on** – a string for the join column name, a list of column names, a join expression (Column), or a list of Columns. If *on* is a string or a list of strings indicating the name of the join column(s), the column(s) must exist on both sides, and this performs an equi-join.
- **how** – str, default `inner`. Must be one of: `inner`, `cross`, `outer`, `full`, `full_outer`, `left`, `left_outer`, `right`, `right_outer`, `left_semi`, and `left_anti`.

SparkSQL

Transformaciones (2): JOIN

Inner join

Name	Age
John	20
Henry	50

Name	Country
Lisa	US
Henry	Korea

Name	Age	Country
Henry	50	Korea

SparkSQL

Transformaciones (2): JOIN

Left outer joins

Name	Age
John	20
Henry	50

Name	Country
Lisa	US
Henry	Korea

Name	Age	Country
Henry	50	Korea
John	20	-

SparkSQL

Transformaciones (2): JOIN

Left semi joins

Name	Age
John	20
Henry	50

Name	Country
Lisa	US
Henry	Korea

Name	Age
Henry	50

SparkSQL

Transformaciones (2): JOIN

Vamos a practicar:

- UkMakerSpaces.py

SparkSQL

Transformaciones (2): JOIN

- UkMakerSpaces.py
  - The postcode in the **maker space** data source is the **full postcode**.
    - **W1T 3AC**
  - The postcode in the **postcode** data source is only the **prefix of the postcode**.
    - **W1T**
  - **Join condition**:
    - If the postcode column in the **maker space** data source starts with the postcode column in the **postcode** data source.
  - **Conner case:**
    - **W14D T2Y** might match both **W14D** and **W14**
  - **Solution:**
    - Append a **space** to the **postcode prefix**
    - Then **W14D T2Y** only matches **"W14D "**, not **"W14 "**

Combinando Spark con SparkSQL

Relación entre RDD y DataFrame

- Un DataFrame no es ni más ni menos que:
  - Unos metadatos con la estructura
  - Un RDD cuyos registros son de tipo Row
  - Los objetos de tipo Row deben cumplir con la estructura
  - https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=column#pyspark.sql.Row
- rdd = df.rdd
- df = rdd.toDF(schema)
  - schema = StructType([
    StructField(“col1”,StringType),
    StructField(“col2”,StringType)
    ])
  - Ejercicio:
    - RddDataframeConversion.py

SparkSQL

Home retention processing example

- From Pandas to PySpark

an **NTT DATA** Company