Expert Services

Hands On Advanced Analytics with Apache Spark

Curs III

Apache Spark

Not the spark

Până în acest punct – Sesiune Spark și Citire

- Sesiune Spark: Poarta de acces către Spark SQL

```
spark = SparkSession.builder.master('local[*]').config('spark.driver.memory', '3g').getOrCreate()
```

- Crearea unui Data Frame Spark dintr-o listă de Python.

```
data_df = spark.createDataFrame(data)
```

- Citirea fișierelor de tip CSV / JSON / Parquet într-un Data Frame

```
data_df = spark.read.format('csv').option('header', 'true').load('/path/to/folder/or/file')
```

```
data_df = spark.read.format('json').load('/path/to/folder/or/file')
```

```
data_df = spark.read.format('parquet').load('/path/to/folder/or/file')
```

Până în acest punct – Afișare, Colectare și Scriere

- Afișare tipurilor de date la consolă

```
data_df.printSchema()
```

- Afișare datelor la consolă

```
data_df.show()
```

- Colectarea datelor într-o listă de Python

```
data_list = data_df.collect()
```

- Colectarea datelor într-o tabelă de Pandas

```
data_pdf = data_df.toPandas()
```

- Scrierea datelor în fișiere de tip CSV / JSON / Parquet

```
data_df.write.format('csv').option('header', 'true').save('/path/to/save/folder')
```

```
data_df.write.format('json').save('/path/to/save/folder')
```

```
data_df.write.format('parquet').save('/path/to/save/folder')
```

Până în acest punct – Proiecții și Planul de execuție

- Selectarea coloanelor

```
new_df = data_df.select('nume', 'varsta')
```

- Ștergerea coloanelor

```
new_df = data_df.drop('varsta', 'inactiv', 'extra')
```

- Redenumirea coloanelor

```
new_data_df = data_df.withColumnRenamed('ocupatie', 'job')
```

- Afișarea planului de execuție

```
new_df.explain('extended')
```

```
== Parsed Logical Plan ==
Project [inactiv#86, nume#88, ocupatie#89 AS job#103, vechime#91L]
+- Project [inactiv#86, nume#88, ocupatie#89, vechime#91L]
   +- Relation [inactiv#86,extra#87,zona#92,nume#88,ocupatie#89,varsta#90L,vechime#91L]

...
```

Până în acest punct – Construirea de Expresii

- Modulul de expresii și funcții din Spark

```
from pyspark.sql import functions as f
```

- Expresie de bază pentru o valoare constantă

```
expr = f.lit(130)
```

- Expresie de bază pentru valoarea unui coloane

```
expr = f.col('varsta')
```

- Construirea expresiilor folosind operatori

```
expr = (f.col('varsta') * f.lit(2) + f.col('vechime') - f.lit(3)) / f.lit(2)  < (f.lit(2)**6)
```

- Construirea expresiilor prin funcții

```
expr = f.contains(f.trim(f.col('nume')), f.lit('a'))
```

- Parsarea expresiilor dintr-o expresie SQL

```
expr = f.expr('concat("in varsta de ", varsta, " ani")')
```

Până în acest punct – Expresii și Transformări

- Conversia într-un anumit tip de date

```
expr = (f.col('varsta') + f.lit(1)).cast('string')
```

- Specificarea explicită a numelui pentru o expresie

```
expr = f.array_join(',', f.col('extra')).alias('consumatori')
```

- Condiții IF-ELSE

```
f.when(f.col('varsta')<25, f.lit('I')).when(f.col('varsta')<32, f.lit('II')).otherwise(f.lit('III'))
```

- Operația de Transformare – Adăugarea sau Înlocuirea unui coloane

```
new_df = data_df.withColumn('text', f.concat(f.lit('in varsta de '), f.col('varsta'), f.lit(' ani')))
```

- Operația de Selecție – Selecție a coloanelor

```
new_df = data_df.select(
   'nume', 'extra', f.col('varsta') - f.col('vechime'),
   f.concat(f.lit('in varsta de '), f.col('varsta'), f.lit(' ani')).alias('text')
)
```

Până în acest punct –Filtrări, Ordonări și Reutilizarea Operațiilor

- Operația de Filtrare – Păstrarea datelor pe baza unei condiții

```
new_df = data_df.filter(f.col('varsta') < 0)
```

```
new_df = data_df.where(f.col('varsta') < 0)
```

- Operația de Sortare – Ordonarea datelor după unul sau mai multe criterii de ordonare

```
new_df = data_df.sort('nume', f.desc(f.col('varsta')))
```

- Operația de Limitare – Limitarea numărului de rânduri

```
new_df = data_df.limit(2)
```

- Operația de Caching – Stocarea datelor în memorie +/- pe disk până la închiderea sesiunii de Spark

```
new_df = data_df.cache()
```

```
from pyspark.storagelevel import StorageLevel
new_df = data_df.persist(StorageLevel.MEMORY_ONLY)
```

Până în acest punct – Program PySpark

- Pas 1: Create porții de acces către Spark SQL, Sesiune Spark

```
from pyspark.sql import SparkSession, functions as f
spark = SparkSession.builder.master('local[*]').getOrCreate()
```

- Pas 2: Citirea fișierelor de într-un Data Frame

```
data_df = spark.read.format('json').load('/path/to/course/data/folder')
```

- Pas 3: Ștergerea colanelor cu date personale

```
transformed_data_df = data_df.withColumn('varsta_contractare', f.col('varsta') - f.col('vechime'))
```

- Pas 4: Redenumirea coloanei de post

```
final_data_df  = transformed_data_df.filter(f.col('varsta') < 30).select('nume', 'varsta_contractare', 'extra')
```

- Pas 5: Afișarea datelor

```
final_data_df.show()
```

Expresii speciale

Feeling like a special snowflake yet?

Expert Services

Column – Nivelarea datelor

Anumite funcții din Spark SQL pot descompune un singur rând de date în mai multe rânduri. Acestea sunt utilizate pentru a desface datele din liste sau colecții în rânduri separate, pentru a nivela și reduce complexitatea datelor.

- Metoda de nivelare a datelor

```
expr = f.explode('extra')
```

  - Un obiect de tip Column este returnat cu expresia care va împărți valorile coloanei extra în mai multe rânduri

```
data_df.withColumn('consumator', expr).show()
```

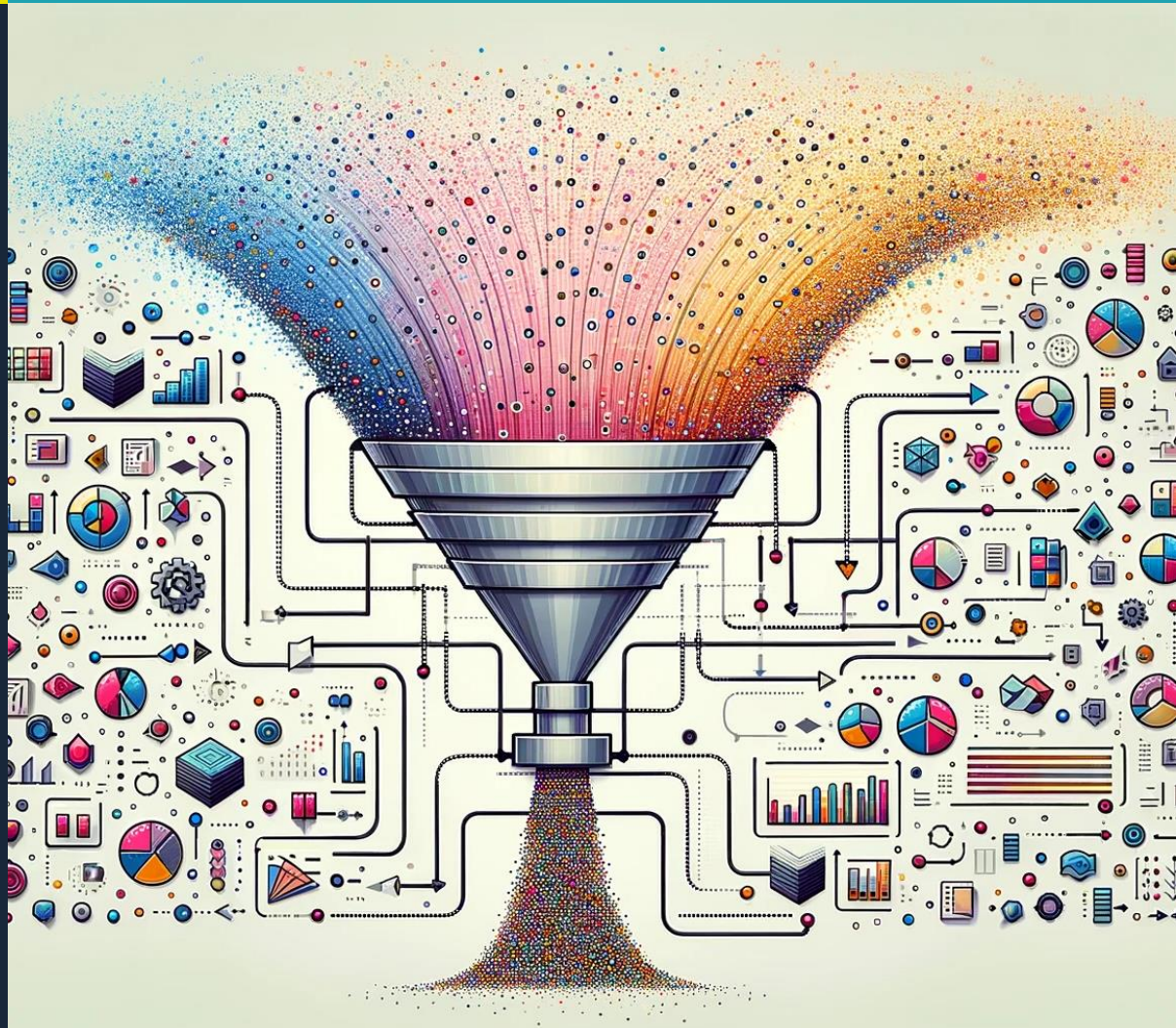
```
+----+------+-----------+-------+-------+----+-----------------+----------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|consumator|
+----+------+-----------+-------+-------+----+-----------------+----------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|3D Printer|
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|      XBOX|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|        EV|
+----+------+-----------+-------+-------+----+-----------------+----------+
```

- ✓ În setul de date nou, se va afla câte un rând de date pentru fiecare element a listei.

Gruparea și Agregarea datelor

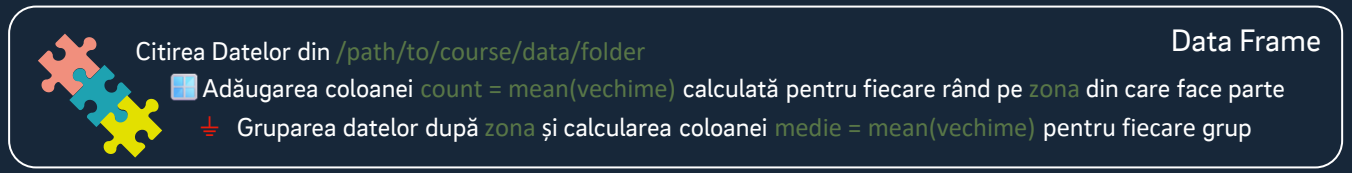
Nu măcinăm carne, ci date

Expert Services



Comenzi de grupare și agregare / transformare

Spark poate transmite și comenzi foarte complexe, comenzi de grupare și agregare / transformare, către executori. Aceste operații permit gruparea datelor, calculul de statistici pe grupuri și fie consolidarea rândurilor dintr-un grup în unul singur, fie adăugarea de noi coloane pe baza agregărilor.

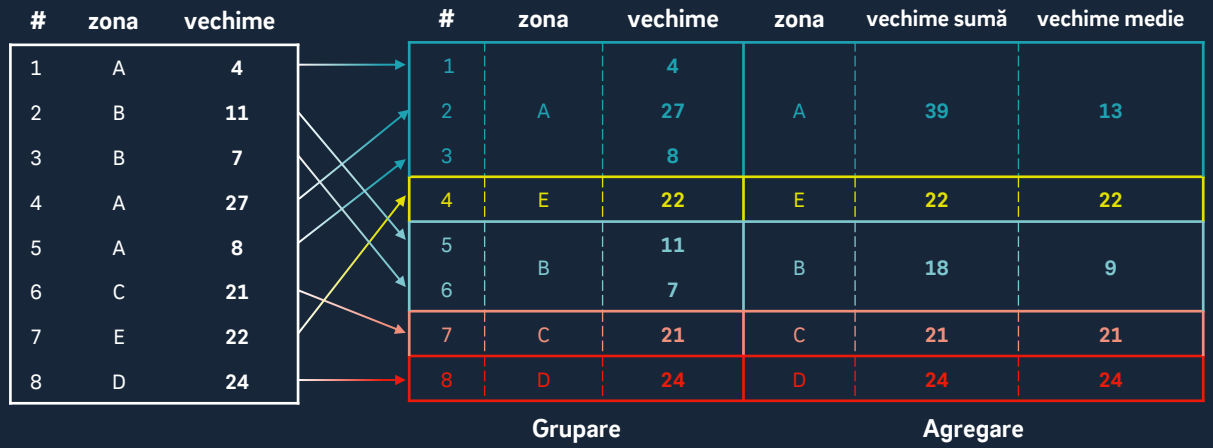


- Comanda de grupare și agregare: se împarte setul de date în grupuri cu date care au aceleași valori pentru coloanele date pentru grupare, iar apoi se calculează, pe baza unei formule de agregare, per grup, coloane noi

- Comanda de grupare și transformare: se calculează coloane noi sau se înlocuiesc cele existente, pe baza unei formule de agregare aplicată pe grupul din care face parte fiecare rând

Comanda de grupare și agregare

În analiza datelor, de multe ori întâlnim nevoia de a împărți un set de date în grupuri mai mici, pentru a procesa împreună datele din fiecare grup, adesea prin metode de agregare pentru a obține perspective noi asupra datelor.



- **Pasul de „Grupare"** împarte rândurile din setul de date în grupuri, în funcție de valorile coloanelor. Rândurile vor face parte din același grup doar dacă au exact aceeași valoare pentru fiecare coloană de grupare.
- **Pasul de „Agregare"** calculează valori „agregate" pentru fiecare grup în parte, de exemplu suma unei coloane pentru tot grupul, și construiește pentru fiecare grup câte un rând folosind coloanele de grup și valorile agregate.

Gruparea Datelor – Group By

În Spark, comanda de grupare a datelor și agregarea lor se construiește în doi pași, primul fiind instrucțiunile pentru pasul de grupare, urmate de instrucțiunile pentru pasul de agregare.

- Operația de Grupare – Gruparea datelor după valorile coloanelor

```
data_df.groupby('zona')
```

  - Funcția returnează un obiect special de tip Grouped Data care oferă diverse metode pentru a executa operații de agregare, de exemplu metode des întâlnite pentru vizualizare precum metodele max și sum.

```
data_df.groupby('zona').max().show()
```

```
+----+-----------+------------+
|zona|max(varsta)|max(vechime)|
+----+-----------+------------+
|   A|         23|           4|
|   B|         34|          11|
+----+-----------+------------+
```

```
data_df.groupby('zona', 'inactiv').sum().show()
```

```
+----+-------+-----------+------------+
|zona|inactiv|sum(varsta)|sum(vechime)|
+----+-------+-----------+------------+
|   A|   NULL|         23|           4|
|   B|   NULL|         34|          11|
|   B|   true|         29|           7|
+----+-------+-----------+------------+
```

Column - Funcții de agregare

Pentru a construi expresii de agregare programatic, PySpark oferă funcții speciale de agregare în acest scop. Acestea se află tot în modulul de funcții din Spark unde se află și funcțiile de transformare a datelor.

- Funcțiile de agregare fie acceptă direct numele coloanei, fie o expresie de transformare ce va fi executată înainte.

```
f.first('vechime', ignorenulls=True)
```
```
Column<'first(vechime)'>
```

```
f.mean(f.col('vechime') + 1)
```
```
Column<'mean((vechime + 1))'>
```

- Unele funcții returnează lista de valori a grupului pentru coloana respectivă cu toate sau o parte din valori. Putem după să le folosim mai târziu pentru diverse transformări, sau chiar în timpul agregării.

```
f.collect_list('vechime')
```
```
Column<'collect_set(vechime)'>
```

```
f.size(f.collect_set(f.col('vechime') + 1))
```
```
Column<'size(collect_set((vechime + 1)))'>
```

Funcțiile și detaliile lor găsiți la https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html

Gruparea Datelor - Agregare

Pentru a executa diverse și complexe agregări în același comandă, folosind expresiile construite, Spark SQL oferă o metodă generică de agregare. Față de metodele rapide de agregare, numele expresiei este luat în considerare.

- Operația de Grupare și Agregare – Gruparea datelor după valorile coloanelor și calcularea agregărilor

```
new_data_df = data_df.groupby('zona').agg(f.sum('vechime').alias('suma'), f.mean('vechime').alias('medie'))
```

  - Un nou obiect de tip Data Frame este returnat care va avea doar coloanele de grupare, în acest caz zona, și coloanele, după numele expresiilor, cu rezultatul agregărilor, în acest caz suma și medie.

```
new_data_df.show()
```

```
+----+----+-----+
|zona|suma|medie|
+----+----+-----+
|   A|   4|  4.0|
|   B|  18|  9.0|
+----+----+-----+
```

Gruparea Datelor – Deduplicare

Dacă nu trebuie efectuate agregări, ci doar grupări, pentru a elimina duplicatele din setul de date, există și astfel de metode. Față de agregare, este necesar un singur pas.

- Operația de Deduplicare – Deduplicarea datelor după valorile coloanelor

```
distinct_df = data_df.select('nume').distinct()
```

  - Un nou obiect de tip Data Frame este returnat cu valorile distincte din setul de date.

```
dudplicated_data_df = data_df.dropDuplicates(['nume'])
```

  - Un Data Frame nou este returnat cu primul rând întâlnit pentru fiecare combinație unică de valori.

distinct_df.show()

```
+----+
|nume|
+----+
|Vali|
|Vlad|
| Bea|
+----+
```

dudplicated_data_df.show()

```
+----+------+----------+-------+-------+----+-----------------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|
+----+------+----------+-------+-------+----+-----------------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|
+----+------+----------+-------+-------+----+-----------------+
```

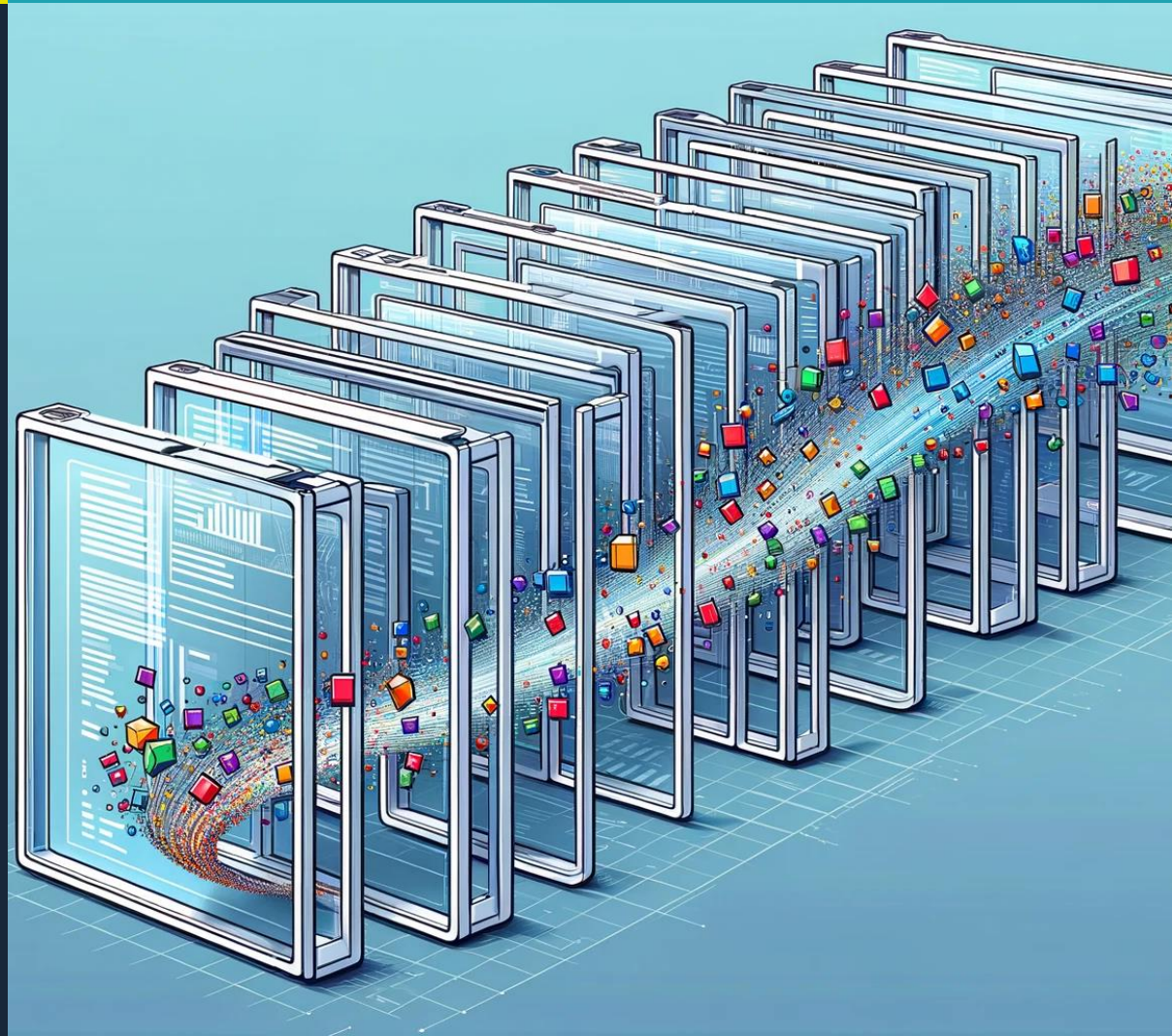
Un program simplu PySpark cu grupări și agregări de date

```
from pyspark.sql import SparkSession, functions as f

spark = SparkSession.builder.master('local[*]').getOrCreate()

data_df = spark.read.format('json').load('/path/to/course/data/folder')

grouped_data_df = (
   data_df
   .withColumn('varsta_contractare', f .col('varsta') - f.col('vechime'))
   .groupBy('varsta_contractare')
   .agg(
      f.sum(f.coalesce(f.col('inactiv'), f.lit(False)).cast('integer')).alias('nr_contracte_inactive'),
      f.mean(f.col('vechime')).alias('vechime_medie')
   )
)

grouped_data_df.write.format('json').save('/path/to/save/folder')
```

Gruparea și Transformarea datelor

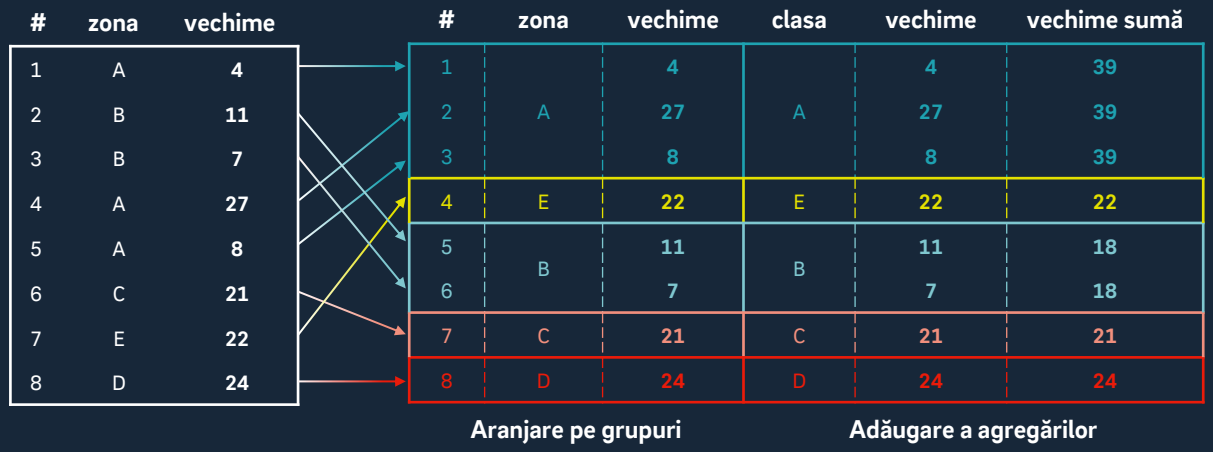
Am amețit un pic ...

Expert Services



Comanda de grupare și transformare

De multe ori ne întâlnim și cu situația de a fi nevoie să împărți un set de date în grupuri mai mici, dar nu pentru a combina împreună datele din fiecare grup, ci pentru a folosi valori agregate ale grupurilor în transformarea datelor.



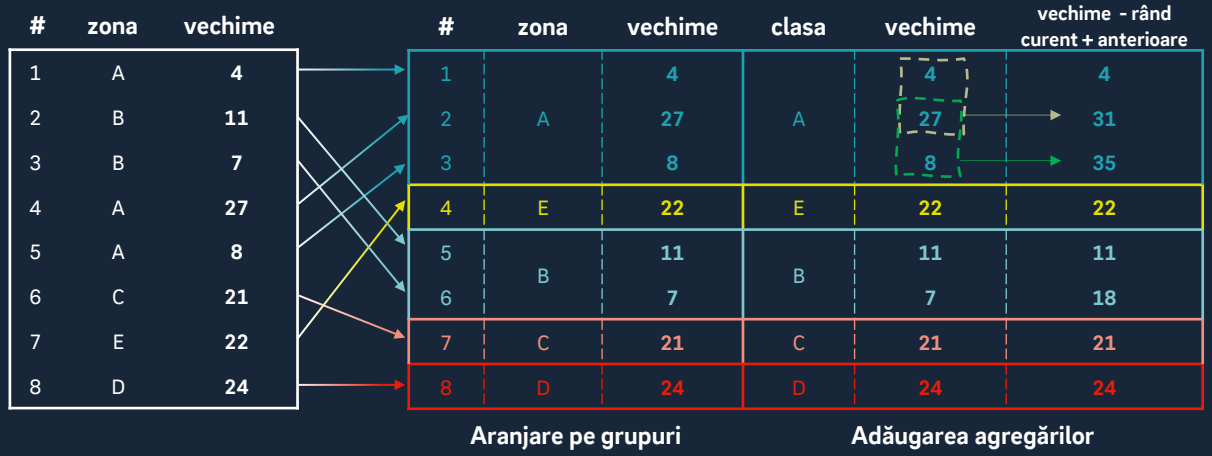
#	zona	vechime
1	A	4
2	B	11
3	B	7
4	A	27
5	A	8
6	C	21
7	E	22
8	D	24

#	zona	vechime	clasa	vechime	vechime sumă
1	A	4	A	4	39
2		27		27	39
3		8		8	39
4	E	22	E	22	22
5	B	11	B	11	18
6		7		7	18
7	C	21	C	21	21
8	D	24	D	24	24

- **Pasul de „Aranjare pe grupuri”** împarte rândurile din setul de date în grupuri, în funcție de valorile coloanelor. Rândurile vor face parte din același grup doar dacă au exact aceeași valoare pentru fiecare coloană de grupare.
- **Pasul de „Adăugare a agregărilor”** calculează valori „agregate” pentru fiecare grup în parte, dar de data aceasta, valorile sunt incorporate sub formă de coloane noi în rândurile existente pentru viitoare transformări.

Comanda de grupare și transformare - Window

Această comandă este mai flexibilă decât comanda standard de grupare și agregare. Este oferită și posibilitatea de a selecta specific, în procesul de agregare a unei coloane pentru fiecare rând dintr-un grup, doar anumite valori.



Această comanda este adesea cunoscută sub numele de Window.

Spark oferă posibilitatea de a defini o „fereastră" continuă de rânduri, relativă la rândul curent, suficientă pentru multe situații, și eficient optimizată. Pentru selecții mai elaborate, metoda tradițională, agregarea întregului grup și filtrarea prin condiții IF-ELSE, deși mai lentă și ineficientă, rămâne în continuare singura soluție viabilă.

Window – Definiția selecției de agregare

Selecția care o putem defini în Spark se poate baza fie pe poziția față de rândul curent, prin ordonarea grupului conform unuia sau mai multor criterii, fie pe valoarea unei coloanei, relativ la valoarea rândul curent.

Pentru această funcționalitate, Spark oferă clasa dedicată Window, disponibilă în cadrul librăriei PySpark:

```
from pyspark.sql import Window
```

- Definirea grupării și a selecției pe baza poziției rândului curent, după ordonare

```
window = Window.partitionBy('zona').orderBy('vechime').rowsBetween(-5, 2)
```

  - Selecția cuprinde 5 rânduri precedente, rândul curent și următoarele 2 rânduri, ordonate după vechime

- Definirea grupării și a selecției pe baza valorii rândului curent, după ordonarea pe baza coloanei respective

```
window = Window.partitionBy('zona').orderBy('vechime').rangeBetween(-5, 6)
```

  - Selecția cuprinde rândurile cu valoarea coloanei vechime între valoarea curentă - 5 și valoarea curentă + 6

Window – Selecție parțial limitată

Selecția poate fi și limitată parțial, și pentru selecția pe bază de poziție, si pentru selecția pe bază de valori, prin folosirea valorilor speciale oferite de clasa Window pentru aceste cazuri.

- Definirea grupării și a selecției pe baza poziției rândului curent fără limita inferioară

```
window = Window.partitionBy('zona').orderBy('vechime').rowsBetween(Window.unboundedPreceding, 2)
```

  - Selecția cuprinde toate rândurile precedente, rândul curent și următoarele 2 rânduri, ordonate după vechime

- Definirea grupării și a selecției pe baza valorii rândului curent, fără limita superioară

```
window = Window.partitionBy('zona’).orderBy('vechime').rangeBetween(-5, Window.unboundedFollowing)
```

  - Selecția cuprinde rândurile cu valoarea coloanei vechime mai mare sau egal cu valoarea curentă - 5

Valorile Window.unboundedPreceding și Window.unboundedFollowing merg în ambele cazuri. Clasa Window oferă și valoarea specială Window.currentRow în loc de valoarea 0 pentru a specifica rândul curent.

Window – Selecția întregului grup și restricții

Selecția poate fi și total nelimitată, caz în care toate valorile din grup vor fi folosite la agregare. Această abordare este de evitat deoarece Spark nu o poate optimiza așa de bine precum selecțiile cu măcar o limită.

- Definirea grupării și selecția întregului grup

```
window = Window.partitionBy('zona')
```

  - Selecția cuprinde toate rândurile din grup. Este echivalent cu specificarea Window.unboundedPreceding și Window.unboundedFollowing la metodele rowsBetween sau rangeBetween.

⚠ Dacă nu specificăm nici o ordonare când folosim rowsBetween, ordinea va fi aleatorie

⚠ Dacă folosim rangeBetween, trebuie să furnizăm o singură coloană la ordonare, care și decide rândurile incluse

⚠ Este posibil să nu specificăm nici o coloană pentru grupare, caz în care Spark va agrega întreg setul de date.

Detalii despre Window găsiți la https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/window.html

Column - Window

Pentru a fi folosite în comenzi de grupare și transformare, Spark oferă posibilitatea apelării funcțiilor de agregare peste selecția definită.

- Construirea unei expresii de agregare pe baza unei selecții definite

```
window = Window.partitionBy('zona').orderBy('vechime').rowsBetween(0, 2)
expr = f.sum('vechime').over(window)
```

  - Un nou obiect de tip Column este returnat cu o nouă expresie care calculează suma coloanei vechime folosind valorile de pe rândul curent și următoarele 2 rânduri, ordonate după vechime.

```
print(expr)
```

```
Column<'sum(vechime) OVER (PARTITION BY zona ORDER BY vechime ASC NULLS FIRST ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING)'>
```

    - Toate funcțiile de agregare suportă metoda over. Există și funcții speciale de analiză special conceput și doar compatibile cu „ferestre” de rânduri precum metoda row_number sau rank.

Funcțiile și detaliile lor găsiți la https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html

Gruparea Datelor – Window

La bază fiind o operație de transformare și nu de agregare, ea nemodificând configurația rândurilor, expresiile de agregare pe window pot fi folosite precum și celelalte expresii cu metodele withColumn, select, etc.

- Operația de Agregare pe Window - Exemplu de aplicare folosind metoda withColumn

```
varsta_window = Window.partitionBy('zona')
new_data_df = data_df.withColumn('medie_varsta', f.mean(f.col('varsta')).over(varsta_window))
```

  - Un nou obiect de tip Data Frame este returnat care are adăugată sau actualizată coloana suma cu rezultatul evaluării expresiei. Expresia se analizează și tipul coloanei este dedus automat.

```
new_data_df.show()
```

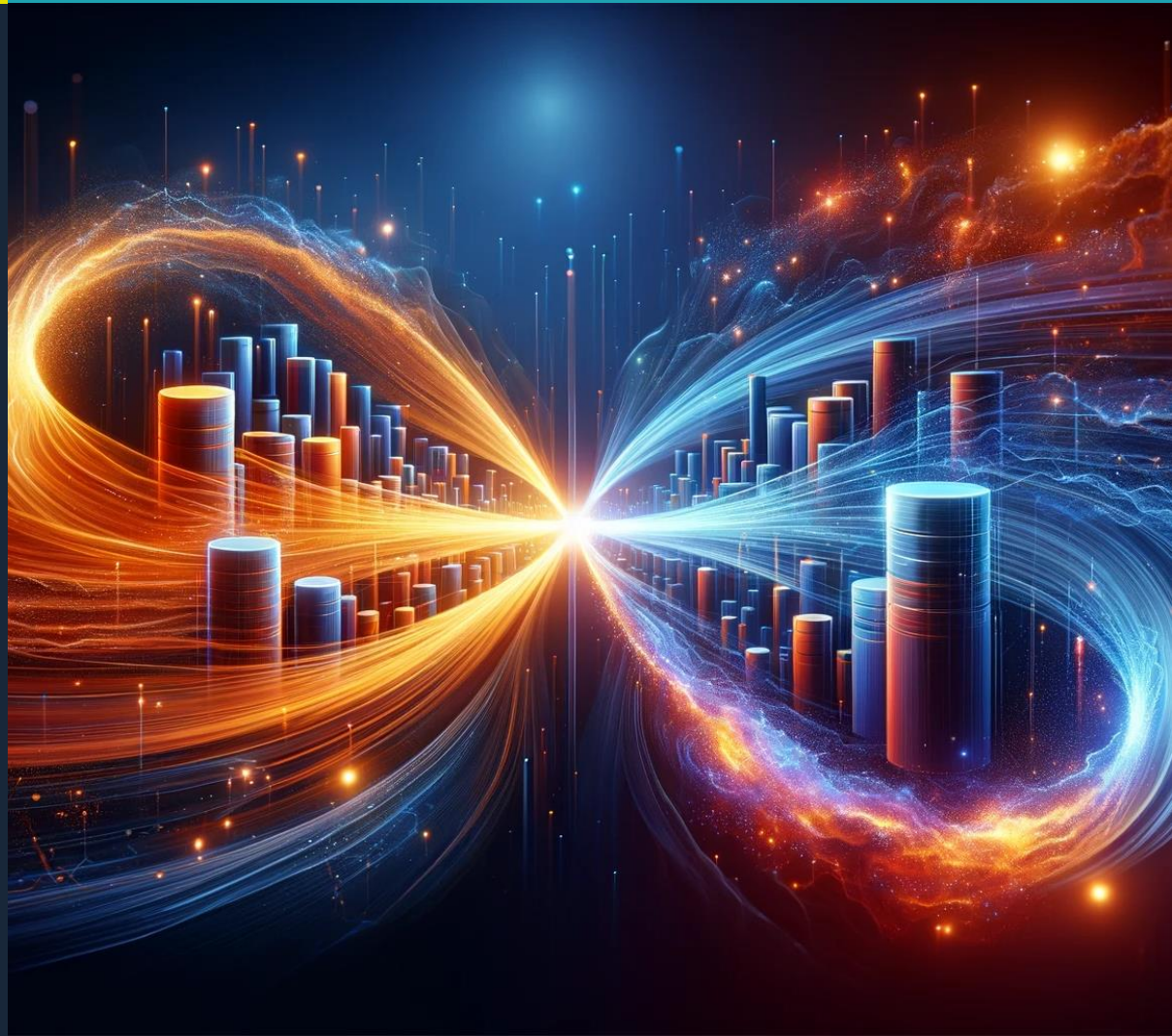
```
+----+------+-----------+-------+-------+----+-----------------+------------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|medie_varsta|
+----+------+-----------+-------+-------+----+-----------------+------------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|        23.0|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|        31.5|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|        31.5|
+----+------+-----------+-------+-------+----+-----------------+------------+
```

Un program simplu PySpark folosind metode Window

```
from pyspark.sql import SparkSession, functions as f, Window

spark = SparkSession.builder.master('local[*]').getOrCreate()

data_df = spark.read.format('json').load('/path/to/course/data/folder’)

window = Window.partitionBy('unitate_varsta').orderBy('vechime').rangeBetween(-2, 2)
processed_data_df = (
   data_df
   .withColumn('inactiv', f.coalesce(f.col('inactiv'), f.lit(False)))
   .withColumn('unitate_varsta', f.floor(f.col('varsta') / 5))
   .withColumn('nr_contracte_similare_inactive', f.sum(f.col('inactiv').cast('integer')).over(window))
   .withColumn('nr_contracte_similare', f.count(f.col('inactiv')).over(window))
   .withColumn('probabilitate_inactivare', f .col('nr_contracte_similare_inactive') / f.col('nr_contracte_similare'))
)

processed_data_df.write.format('json').save('/path/to/save/folder')
```

Uniunea și asocierea datelor

If you can't beat them, join them



Expert Services

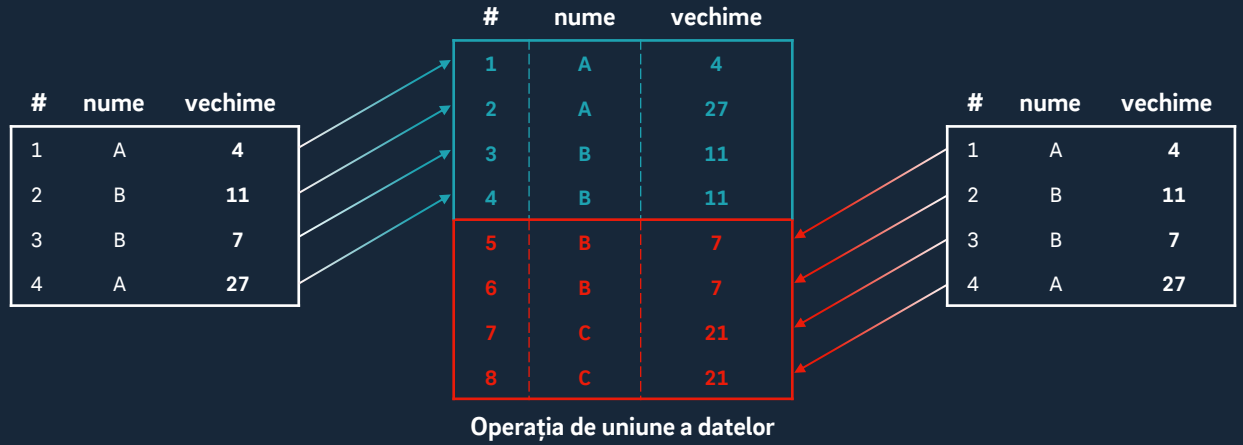
Comenzi de uniune sau asociere a datelor

Cele mai speciale comenzi pe care Spark le poate transmite către executori sunt cele de combinare a datelor. Ele sunt speciale deoarece ele unesc două lanțuri de execuție separate întrucât se îmbină datele.

- Comenzile de uniune: la rândurile primului set de date se adaugă rândurile celui de al doilea set de date
- Comenzile de asociere: rândurile din primul set de date sunt împerecheate cu cele din al doilea, după o regulă definită, creându-se un nou rând prin unirea coloanelor

Comanda de uniune a datelor

Se întâmplă des să se interacționeze cu mai multe seturi de date, fiecare în propria sa formă, iar obiectivul este de a le aduce la un format comun. Într-un final, este necesar să uniunea lor pentru a forma un set de date complet.



În această situație, după transformarea datelor la un format comun, folosim comanda de uniune a datelor, comandă care concatenează rândurile din primul set de date cu rândurile celui de al doilea set de date.

Uniunea Datelor - Union

Data Frame-urile oferă metode de uniune a rândurilor din două seturi de date diferite. Aceste metode se folosesc în general atunci când citim date similare din mai multe locații și se dorește unificarea lor.

- Operația de Uniune – Concatenarea a două seturi de date pe baza ordinii coloanelor

```
new_data_df = data_df.union(data_df)
```

  - Un nou Data Frame cu datele concatenate este returnat. Este folosită ordinea coloanelor din schema datelor, nu a numelui, pentru a efectua operația. Numele coloanelor din primul set sunt utilizate.

```
new_data_df.show()
```

```
+----+------+-----------+-------+-------+----+-----------------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|
+----+------+-----------+-------+-------+----+-----------------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|
+----+------+-----------+-------+-------+----+-----------------+
```

Uniunea Datelor – Union by Name

Există și o metodă de uniune a rândurilor din două seturi de date diferite pe baze numelui coloanelor, nu pe ordinea coloanelor în Data Frame. Această metodă este cel mai des utilizată.

- Operația de Uniune – Concatenarea a două seturi de date pe baza numelui coloanelor

```
new_data_df = data_df.unionByName(data_df)
```

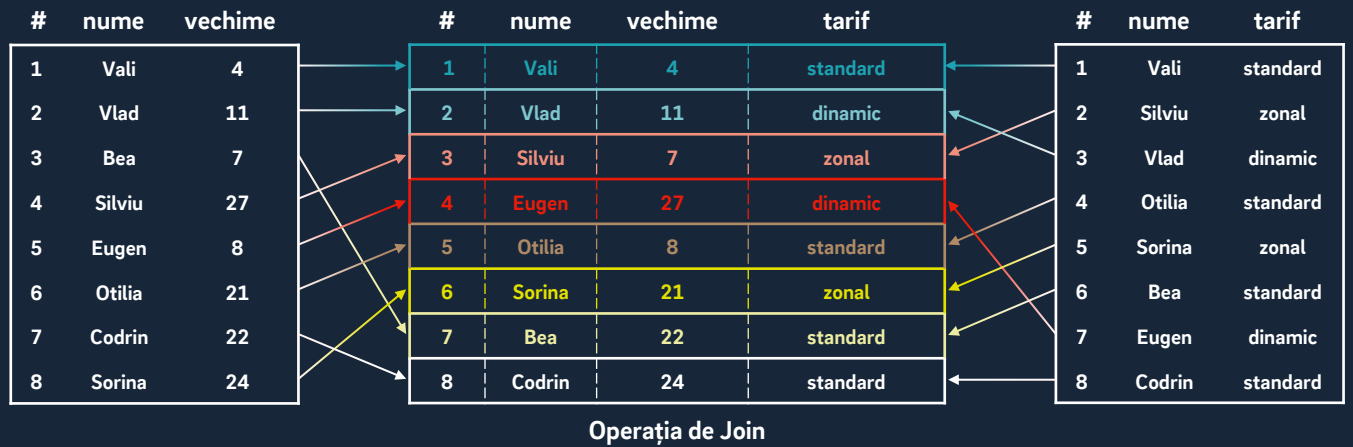
  - Un Data Frame nou cu datele concatenate pe baza numelui coloanelor este returnat. Numele coloanelor și tipurile de date din ambele seturi de date trebuie să fie identice pentru această operație.

```
new_data_df.show()
```

```
+----+------+-----------+-------+-------+----+-----------------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|
+----+------+-----------+-------+-------+----+-----------------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|
|Vlad|    34| Instalator|     11|   NULL|   B|             [EV]|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|
+----+------+-----------+-------+-------+----+-----------------+
```

Comanda de asociere – Join

În toate mediile de lucru, întâlnim de asemenea situația în care avem de integrat mai multe surse de date care oferă informații total diferite și dorim să combinăm coloanele pe baza informațiilor comune dintre ele.



#	nume	vechime
1	Vali	4
2	Vlad	11
3	Bea	7
4	Silviu	27
5	Eugen	8
6	Otilia	21
7	Codrin	22
8	Sorina	24

#	nume	vechime	tarif
1	Vali	4	standard
2	Vlad	11	dinamic
3	Silviu	7	zonal
4	Eugen	27	dinamic
5	Otilia	8	standard
6	Sorina	21	zonal
7	Bea	22	standard
8	Codrin	24	standard

#	nume	tarif
1	Vali	standard
2	Silviu	zonal
3	Vlad	dinamic
4	Otilia	standard
5	Sorina	zonal
6	Bea	standard
7	Eugen	dinamic
8	Codrin	standard

Această comanda este adesea cunoscută sub numele de Join.

În acest caz folosim comanda de Join, comandă care împerechează rândurile din primul set de date cu cele din al doilea, după o regulă definită de dezvoltator, creându-se un nou rând prin unirea coloanelor în noul set de date.

Comanda de asociere – Join Many:1

La împerecherea rândurilor se poate întâlni situația în care un rând se potrivește, după regula definită, cu mai multe alte rânduri din celălalt set de date, fie intenționat, fie neintenționat.

Operația de Join

Comanda de Join va împerechea de mai multe ori rândul, rezultând mai multe intrări în setul de date combinat. Dezvoltatorul fie elimină duplicatele ulterior, fie restrânge regula folosită, dacă acest lucru nu este de dorit.

Comanda de asociere – Join Many:Many

De asemenea, se poate întâlni situația în care mai multe rânduri se potrivesc, după regula definită, cu mai multe alte rânduri din celălalt set de date, fie intenționat, fie neintenționat.

Operația de Join

Comanda de Join va împerechea fiecare rând cu fiecare rând, rezultând mai multe intrări în setul de date combinat. Dezvoltatorul fie elimină duplicatele ulterior / anterior, fie restrânge regula folosită, dacă acest lucru nu este de dorit.

Asocierea Datelor - Join

Data Frame-urile oferă și metode pentru îmbinarea datelor, pe lângă cele de transformare, scriere, colectare și afișare în același stil de apelare prin înlănțuire. Aceste metode se folosesc și ele de expresiile de calcul.

- Operația de Asociere – Asocierea a două seturi de date pe baza unei reguli

```
new_data_df = data_df.join(tariff_df, on='nume')
```

  - Un nou obiect de tip Data Frame este returnat care asociază rândurile din primul set de date cu cele din al doilea atunci când ambele au valoarea din coloana "nume" egală, excluzând valorile NULL. Întrucât coloana nume este specificată în regulă explicit, va apărea o singură data în tabela unită.

```
new_data_df.show()
```

```
+----+------+----------+-------+-------+----+----------------+--------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|           extra|   tarif|
+----+------+----------+-------+-------+----+----------------+--------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|standard|
|Vlad|    34| Instalator|     11|   NULL|   B|            [EV]| dinamic|
| Bea|    29|   Reporter|      7|   true|   B|            NULL|standard|
+----+------+----------+-------+-------+----+----------------+--------+
```

Îmbinarea Datelor – Definirea Regulii de Join

Metoda Join acceptă și reguli bazate pe mai multe coloane, numele lor specificat în text direct, ori una sau mai multe expresii de tip Column, chiar și combinații intre numele coloanelor și expresii de tip Column.

- Operația de Asociere – Asocierea a două seturi de date pe baza unei reguli formată din mai multe criterii

```
new_data_df = data_df.join(zone_source_df, on=['zona', f.col('vechime') < f.col('ani')])
```

  - În acest caz, un nou obiect de tip Data Frame este returnat care împerechează rândurile din primul set de date cu cele din al doilea atunci când ambele au valoarea din coloana "zona" egală, excluzând valorile NULL, și coloana "varsta" din primul set de date este mai mică decât coloana "ani" din al doilea.

```
new_data_df.show()
```

```
+----+------+----------+-------+-------+----+----------------+--------+---+
|nume|varsta|   ocupatie|vechime|inactiv|zona|           extra|   sursa|ani|
+----+------+----------+-------+-------+----+----------------+--------+---+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|   hidro| 10|
|Vlad|    34| Instalator|     11|   NULL|   B|            [EV]|nucleara| 16|
| Bea|    29|   Reporter|      7|   true|   B|            NULL|nucleara| 16|
+----+------+----------+-------+-------+----+----------------+--------+---+
```

Îmbinarea Datelor – Coliziuni în numele coloanelor

Când Data Frame-urile au coloane cu nume identice care nu sunt furnizate în lista de reguli în text, recomandarea este redenumirea acestora. Spark nu poate determina ce valoare să atribuie acestor coloane în setul de date rezultat.

Dar, dacă redenumirea lor anterioară nu este posibilă sau vrem să folosim o regulă de comparație, alta decât cea de egalitate, pentru o coloană comună, similar cu tipul Column, putem seta un nume pentru un Data Frame.

- Operația de Numire – Setarea unui nume pentru Data Frame

```
new_data_df = data_df.alias('date_intrare')
```

  - Un nou obiect de tip Data Frame este returnat care are un nume setat. În operațiile următoare putem să opțional să ne folosim și de numele setat când facem referire la o coloană în expresii.

```
print( f.col('date_intrare.varsta') + f.col('varsta') )
```

```
Column<'(date_intrare.varsta + varsta)'>
```

  - ✓ Putem să folosim numele setat pentru a accesa coloana atâta timp cât nu o suprascriem în orice mod

Îmbinarea Datelor – Definirea Regulii de Join

Folosind metoda alias putem acum specifica reguli de comparație, altele decât cea de egalitate, pentru coloane care se află în ambele Data Frame-ui sub același nume și putem să folosim coloane cu nume identice ulterior. Această metodă este ideală pentru operația de self-join, unde numele coloanelor va fi întotdeauna la fel.

- Operația de Îmbinare – Self-Join

```
new_data_df = data_df.alias('left').join(data_df.alias('right'), on=[f.col('left.varsta') < f.col('right.varsta')]) \
                .withColumn('diferenta_varsta', f.col('right.varsta') - f.col('left.varsta'))
```

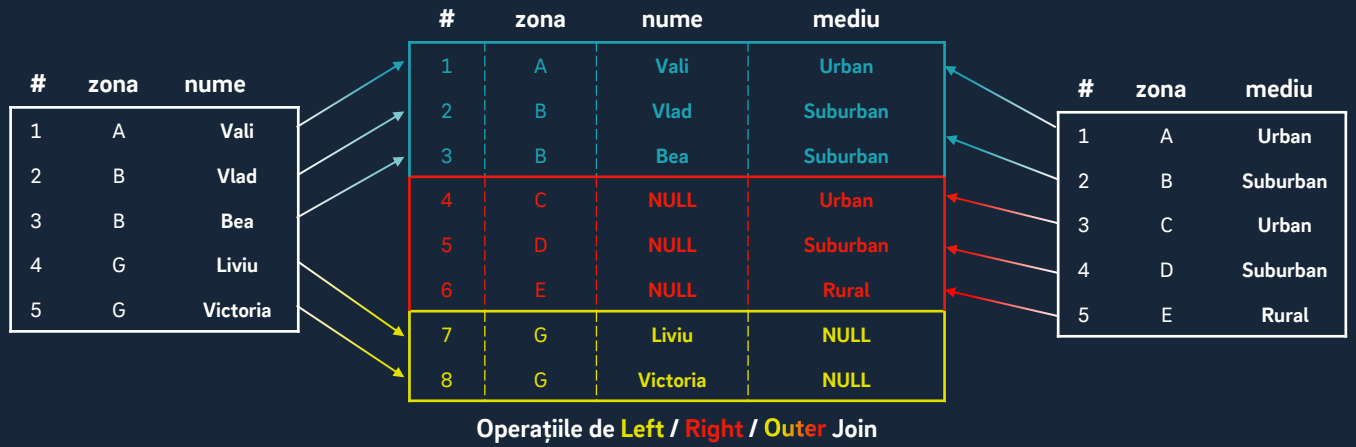
  - În acest caz, un nou obiect de tip Data Frame este returnat care împerechează rândurile din primul set de date cu cele din al doilea atunci când valoarea coloanei "varsta" din primul set de date este mai mică decât valoarea coloanei "varsta" din al doilea.

```
new_data_df.show()
```

```
+----+------+-----------+-------+-------+----+-----------------+----+------+----------+-------+-------+----+-----+----------------+
|nume|varsta|   ocupatie|vechime|inactiv|zona|            extra|nume|varsta|  ocupatie|vechime|inactiv|zona|extra|diferenta_varsta|
+----+------+-----------+-------+-------+----+-----------------+----+------+----------+-------+-------+----+-----+----------------+
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]|Vlad|    34|Instalator|     11|   NULL|   B| [EV]|              11|
|Vali|    23|Programator|      4|   NULL|   A|[3D Printer, XBOX]| Bea|    29|  Reporter|      7|   true|   B| NULL|               6|
| Bea|    29|   Reporter|      7|   true|   B|             NULL|Vlad|    34|Instalator|     11|   NULL|   B| [EV]|               5|
+----+------+-----------+-------+-------+----+-----------------+----+------+----------+-------+-------+----+-----+----------------+
```

Comanda de asociere – Left / Right / Outer Join

De asemenea, se poate întâlni situația în care pentru anumite rânduri nu se găsește o potrivire, după regula definită, cu alte rânduri din celălalt set de date, pentru că pentru că datele ori lipsesc ori nu au cum să existe.



Operațiile de Left / Right / Outer Join

Comanda de Join, în mod normal, va ignora aceste rânduri și ele nu vor apărea în setul de date combinat. Pentru această situație folosim comenzile de Left / Right / Outer Join ce păstrează, în setul de date combinat, rândurile din primul / al doilea / ambele seturi de date care nu se potrivesc după regula definită.

Comanda de asociere – Semi / Anti Join

Mai este și situația în care nu vrem să combinăm datele din seturile de date, ci doar să păstrăm sau să scoatem rândurile dintr-un set de date care au o potrivire în al doilea set de date.

Pentru această situație folosim comenzile de Semi / Anti Join, comenzi care păstrează / elimină rândurile din primul set de date care se potrivesc cu rânduri din al doilea set de date, după o regula definită, fără a mai combina datele.

Asocierea Datelor – Diferite tipuri de Join

În Spark SQL putem specifica tipul de asociere la apelarea metodei de join.

- Operația de Asociere – Asociere de stânga

```
new_data_df = data_df.join(zone_source_df, on='zona', how='left')
```

  - Un nou obiect de tip Data Frame este returnat cu setul de date rezultat din asocierea celor două Data Frame-uri, după tipul de asociere specificat, în acest caz asociere de stânga.

- Operația de Asociere – Anti-Asociere

```
new_data_df = data_df.join(zone_type_df, on='zona', how='anti')
```

  - Un nou obiect de tip Data Frame este returnat cu setul de date rezultat din asocierea celor două Data Frame-uri, după tipul de asociere specificat, în acest caz anti-asociere.

Detalii la https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.join.html

Un program simplu PySpark cu asocieri de date

```
from pyspark.sql import SparkSession, functions as f

spark = SparkSession.builder.master('local[*]').getOrCreate()

data_df = spark.read.format('json').load('/path/to/course/data/folder’)
zone_type_df = spark.read.format('json').load('/path/to/course/zone/type/data/folder’)

statistics_data_df = (
   data_df
   .join(zone_type_df, 'zona')
   .groupby('zona')
   .agg(f.count('*').alias('numar_clienti'))
)

statistics_data_df.write.format('json').save('/path/to/save/folder')
```