**Настройки Spark Session**

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| spark.blacklist.enabled | Блокирует хосты, если происходят ошибки. На крупных задачах и загруженном кластере может привести к тому, что задача отвалится. |
| spark.executor.memory | Кол-во памяти на один исполнитель |
| spark.executor.cores | Кол-во ядер на один исполнитель. |
| spark.driver.memory | Сколько памяти выделить на драйвер. |
| spark.driver.maxResultSize | Судя по названию - максимальный рамер резулдьтата, который можно отправить на драйвер. |
| spark.executor.instances | Число исполнителей |
| spark.default.parallelism | Число секций, которые будут созданы при перетасовке и некоторых других операциях при работе через Spark Core (RDD). |
| spark.sql.shuffle.partitions | Число секций, которые будут созданы при перетасовке и некоторых других операциях при работе через Spark SQL. |
| spark.yarn.executor.memoryOverhead |  |
| spark.blacklist.timeout |  |
| spark.blacklist.task.maxTaskAttemptsPerNode |  |

**Создать Dataframe**

val df=spark.sql("""select 2+2 as nmbr, 'abracadabra' as txt\_field, current\_timestamp as CURRENT\_DATE""")

**Вывести результаты**

df.show()

**Нахождение различий в датафреймах**

df2.except(df).show

**Нахождение максимального значения**

df2.select(max($"col1")).show

**Для удаления дублей**

val df\_new=df.dropDuplicates

**Для сравнения схем (точнее нахождение тех, которые есть в df2 и нет в df1)**

df1.schema.diff(df2.schema)

**Датасет из таблицы**

val ds= spark.read.table("полное имя таблицы в Hive")

**Сохранение датасета**

ds.write.parquet("путь к папке с .parquet")

**Список всех схем (в формате pd.Series):**

spark.sql("show databases").toPandas()['databaseName']

**Список всех таблиц в схеме (тоже в формате pd.Series):**

spark.sql("show tables in ***name***").toPandas()['tableName']

**Размер таблицы (в байтах, строка):**

spark.sql("DESCRIBE EXTENDED ***name***").filter(f.col("col\_name") == "Statistics").select("data\_type").collect()[0]['data\_type']

**Выборка из датасета**

ds.select("col1", "col2"...) // если колонки известны статически  
   
ds.select(columns: \_\*) // если у нас есть коллекция типа Seq[Column]  
   
ds.select(columns.map(col): \_\*) // если у нас есть коллекция типа Seq[String]  
   
col - это org.apache.spark.sql.functions.col, функция String => Column, у нее есть еще и короткий вариант $"имя".  
   
**Кроме колонок, взятых из таблицы, можно добавлять, например, константы. Ровно так же, как мы делаем для SQL-запросов:**  
   
ds.select(lit("abc")) // мы выбрали колонку, содержащую строковую константу "abc"  
   
**А еще можно применять к колонкам функции**  
   
ds.select(upper(col("abc"))) // колонк "abc" в upper case  
   
   
**Получить схему**  
ds.schema  
   
 **Схему можно, например, сравнить с другой**  
ds.schema.diff(ds1.schema)  
   
**Или удалить из нее колонки**  
ds.drop(index)

**Переменные в Spark**

def jdbcOptions(query: String) = Map[String,String](

"driver" -> config.getString("sqlserver.db.driver"),

"url" -> config.getString("sqlserver.db.url"),

"dbtable" -> s"(select \* from TestAllData where update\_database\_time >= '2019-03-19 12:30:00.003') as subq,

"user" -> config.getString("sqlserver.db.user"),

"password" -> config.getString("sqlserver.db.password"),

"customSchema" -> config.getString("sqlserver.db.custom\_schema")

)

val testDataDF = sparkSession

.read

.format("jdbc")

.options(jdbcOptions())

.load()

val dayColumn = "update\_database\_time"

val dayValue = "2019-03-19 12:30:00.003"

s"(select \* from TestAllData where $dayColumn > '$dayValue') as subq"

id = "1"

query = "SELECT count from mytable WHERE id='{}'".format(id)

sqlContext.sql(query)

bkt = 1

prime = spark.sql(f"SELECT ((year(fdr\_date))\*100)+month(fdr\_date) as fdr\_year, count(\*) as counts\

FROM pwrcrv\_tmp\

where EXTR\_CURR\_NUM\_CYC\_DLQ={bkt}\

and EXTR\_ACCOUNT\_TYPE in('PS','PT','PD','PC','HV','PA')\

group by ((year(fdr\_date))\*100)+month(fdr\_date)\

order by ((year(fdr\_date))\*100)+month(fdr\_date)")

**Загрузить данные из HDFS в SparkDataframe**

spark\_df = spark.read.csv("/user/me/test.csv", sep = ';')

**Выгрузить из HDFS в CSV**

data = spark.sql("select \* from sbx\_t\_team\_business\_services.pprb\_epk\_3")

data.write.mode("overwrite").format("csv").mode("overwrite").save("/user/ilina5-mv\_ca-sbrf-ru/result")

**Конвертировать spark\_df >> pandas\_df (предварительно импортируйте pandas)**

pandas\_df = spark\_df.toPandas()

**Конвертировать   pandas\_df >> spark\_df**

spark\_new\_df = spark.createDataFrame(pandas\_df)

**Важно!**

При чтении и сохранении датафреймов помните, что Spark работает с файловой системой HDFS, а Python (Scala, etc) с локальной файловой системой.

При работе с датафреймами помните, что SparkDataframe хранится и обрабатывается в оперативной памяти всего кластера, а PandasDataframe - в оперативной памяти сервера.

|  |
| --- |
| //first u need to convert pandas dataframe to spark dataframe:    from pyspark.sql import HiveContext    hive\_context = HiveContext(sc)    df = hive\_context.createDataFrame(pd\_df)    //then u can create a temptable which is in memory:    df.registerTempTable('tmp')    //now,u can use hive ql to save data into hive:    hive\_context.sql("""insert overwrite table target partition(p='p') select a,b from tmp'''    //note than:the hive\_context must be keep to the same one! |

# Spark 2.0(spark2-shell):

Можно сразу выполнять запросы:

**Создать Dataframe:**

val df=spark.sql("""select 2+2 as nmbr, 'abracadabra' as txt\_field, current\_timestamp as CURRENT\_DATE""")

**Вывести результаты:**

df.show()

**Можно всё сразу:**  
val df=spark.sql("""select 2+2 as nmbr, 'abracadabra' as txt\_field, current\_timestamp as CURRENT\_DATE""").show()

**Создать таблицу из запроса:**  
spark.sql("""create table custom\_risk\_limk.aag\_test2 select 2+2 as nmbr, 'abracadabra' as txt\_field, current\_timestamp as CURRENT\_DATE""")

**Записать DataFrame в таблицу:**  
df.write.format("parquet").mode("overwrite").saveAsTable(s"custom\_risk\_limk.aag\_test2")  
df.write.format("parquet").mode("append").saveAsTable(s"custom\_risk\_limk.aag\_test2")

**Очистить или удалить таблицу:**  
spark.sql("""truncate table custom\_risk\_limk.aag\_test""")  
spark.sql("""drop table custom\_risk\_limk.aag\_test""")

**Собрать статистику:**  
spark.sql("""analyze table ess\_ini\_tsm\_retail.custrec\_other compute statistics""")

**Выйти из spark-shell:** System.exit(0)

**Выгрузка запроса через Spark на шару:**

Создаем скрипт с расширением **название.scala**, пример ниже:

|  |
| --- |
| import java.io.\_    //функция записи в файл  def printToFile(f: java.io.File)(op: java.io.PrintWriter => Unit) {  val p = new java.io.PrintWriter(f)  try { op(p) } finally { p.close() }  }      //DTM\_EKP\_SCORING\_LOANS\_AGR  val sel = spark.sql("""СЮДА КИДАЕМ SQL ЗАПРОС""")  val ls=sel.collectAsList() //преобразование датафрейма в список  val ar=ls.toArray() //преобразование списка в массив  printToFile(new File("НАЗВАНИЕ\_ТЕСТОВОГО\_ФАЙЛА.txt")) { p => ar.foreach(p.println)} //запись |

**Запуск:**

spark2-shell  -i **название.scala** --num-executors 100 --executor-cores 2 --executor-memory 16g --driver-memory 32g --conf spark.yarn.executor.memoryOverhead=8g --queue root.вашаочередь