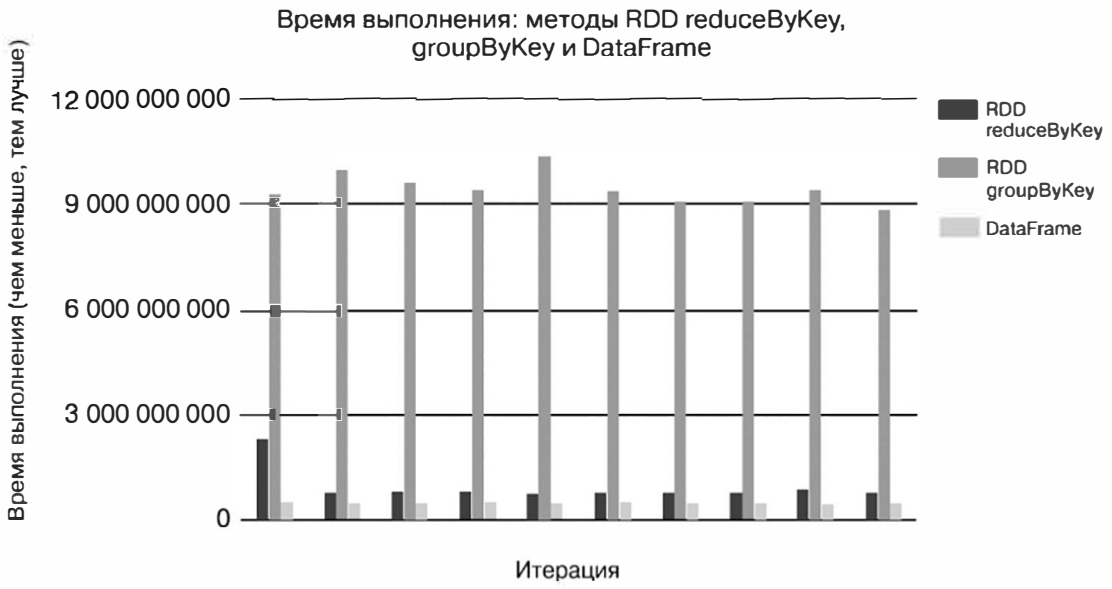
Spark SQL. DataFrame

Spark SQL. DataFrame

- DataFrame – распределенная коллекция данных(RDD) в колончатом(табличном) именованном формате.
- Типы описываются схемой DataFrame с привязкой к типам данных SQL.
- Проверка типов происходит в Runtime.
- DataFrame “alias” для Dataset[Row] (since 2.0.X) или RDD[Row]



Work with DataFrame

Work with DataFrame: SparkSession

- Точкой входа к SQL API является объект SQL/Hive Context (1.6.X) или SparkSession (2.X.X)

- Для версий Spark от 2.X.X SparkSession объединяет SparkContext, StreamingContext, SparkConf, SQL/HiveContext

```
implicit val spark: SparkSession =
SparkSession
    .builder()
    .master("local[*]")
    .appName("AppName")
    .enableHiveSupport()
    .getOrCreate()
```

Work with DataFrame: Чтение

DataFrameReader класс, который позволяет загружать DF из различных источников.

```
val df = spark.read.csv(path)
```

Форматы

- csv
- jdbc
- json
- orc
- parquet
- table
- text
- textfile

Функции

- format(string)
- option("key", value)
- options(map)
- load(path or none)

Work with DataFrame: Создание

Описание схемы для создания DF:

```
val customerSchema = StructType(
  StructField("customer_id", StringType, true) ::
    StructField("customer_name", StringType, false) ::
    StructField("customer_score", LongType, false) ::
    Nil
)
```

```
spark.read.format("csv")
  .option("nullValue", "NULL")
  .option("delimiter", "\t")
  .schema(customerSchema)
  .load("path/customer.csv")
  .createTempView("customer")
```

Work with DataFrame: Запись

Запись данных происходит с помощью DataFrameWriter:

```
df.write.format("orc").save(path)
```

Форматы

- csv
- jdbc(url, table, props)
- json
- orc
- parquet
- text

Функции

- save(path) – Datasource (except Hive)
- saveAsTable(name)
- insertInto(table name)
- createTable()

Work with DataFrame: Запись

Запись данных происходит с помощью DataFrameWriter:

```
Customer.apply(session, properties)
.write.format("orc")
.mode(SaveMode.Overwrite)
.insertInto(“customer")
```

Work with DataFrame: Запись

Варианты SaveMode

Параметр	Описание
Append	Новые данные присоединяются к существующим (до записываются)
Overwrite	Существующий данные перезаписываются новыми
Ignore	Если данные существуют, никаких действий не происходит
ErrorIfExists	Выброс исключения при наличии существующих данных

Spark SQL API

Для выполнения SQL запросов над данными необходимо:

~~df.registerTempTable(“customers_table”)~~

```
df.createTempView(“customers_table”)
```

Далее к данной таблице можно обращаться через ~~sqlContext~~ sparkSession:

```
spark.sql(“SELECT count(*) AS customers_count FROM
   customers_table WHERE customer_score > 100”)
```

Результат можно визуализировать с помощью функций show() и take()

Spark SQL API

```
spark.sql("FROM product")


val orderQuery = "SELECT DISTINCT customer_id, order_date FROM
order WHERE status = 'delivered’»
spark.sql(orderQuery)

spark.table("customer")
```

Work with DataFrame: Функции

Записи внутри DataFrame организованы в виде Dataset[Row]

Структура позволяет выполнять запросы и операции идентичные реляционным базам данных: select , filter, join, group, etc.

```
df.select($"product_name")
  .filter($"count" > 100)

df.groupBy($"customer_name")
  .sum($"score")
```

Work with DataFrame: Распространённые функции

Записи внутри DataFrame организованы в виде Dataset[Row]

Структура позволяет выполнять запросы и операции идентичные реляционным базам данных: select , filter, join, group, etc.

```
df.select($"product_name")
  .filter($"count" > 100)

df.groupBy($"customer_name")
  .sum($"score")
```

Work with DataFrame: Распространённые функции

Преобразования

- map()
- flatMap ()
- filter ()
- distinct ()

Преобразования пар

- union ()
- join()

Действия

- collect ()
- count ()
- sum()
- take (num)
- map (func)
- reduce (func)
- agg ()
- foreach (func)
- reduceByKey()
- groupByKey()

Work with DataFrame: Создание колонки

```
//1)
.withColumn("date", parameters.DATE_1D_ddMMyyyy)

//2)
.withColumn("type", getTypeDevice(col("name")))
```

Work with DataFrame: UDF

```
private val getTypeDevice = udf((nameDevice: String) => {
  val lcaseDeviceName = nameDevice.toLowerCase()
  if(lcaseDeviceName.contains("iphone"))
    "SMARTPHONE"
  else if (lcaseDeviceName.contains("ipad"))
    "TABLET"
  else if (lcaseDeviceName.contains("macbook"))
    "LAPTOP"
  else
   "ACCESSORY"
})
```

Work with DataFrame: Работа с парами DF

```
//1)
val result = order
.join(
    stg,
    $"customer_id" === $"product_id",
    "inner"
  )

//2)
val result = order
  .join(
    stg,
    Seq("customer_id", "product_id"),
    "inner"
  )
```

Work with DataFrame: Агрегация

```
.groupBy('customer_name, 'product_name)
.agg(
  sum('number_of_product * 'price) alias
"sum_price_product",
  max('number_of_product) alias "max_num_of_product",
  min('number_of_product * 'price) alias
"mim_sum_product",
  avg('number_of_product * 'price) alias
"avg_price_product"
)
```

Work with DataFrame: Оконная функция

```
val windowBySum =
Window.partitionBy('customer_id).orderBy('sum_num_of_product.d
esc)

val order = spark.table(Parameters.table_order)
  .groupBy('customer_id, 'product_id)
  .agg(sum('number_of_product) alias "sum_num_of_product")
  .select(
    'customer_id,
    'product_id,
    row_number().over(windowBySum) alias "rn"
  )
  .filter('rn === 1)
```