Урок 3. Dataframe API - 2

Union

Нужно следить за порядком колонок

union(*other*) [source]

Return a new **DataFrame** containing union of rows in this and another frame.

This is equivalent to *UNION ALL* in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by a distinct.

Also as standard in SQL, this function resolves columns by position (not by name).

New in version 2.0.

unionAll(*other*) [source]

Return a new **DataFrame** containing union of rows in this and another frame.

This is equivalent to *UNION ALL* in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by a distinct.

Also as standard in SQL, this function resolves columns by position (not by name).

Note: Deprecated in 2.0, use union instead.

Регулярные выражения

```
1 from pyspark.sql.functions import regexp_replace
2 regex_string = "BLACK|WHITE|RED|GREEN|BLUE"
3 df.select(
4     regexp_replace(col("Description"), regex_string, "COLOR")
5         .alias("color_clean"),
6     col("Description")
7 ).show(2)
```

explode & flatten (spark 2.4)

Массив значений в строки

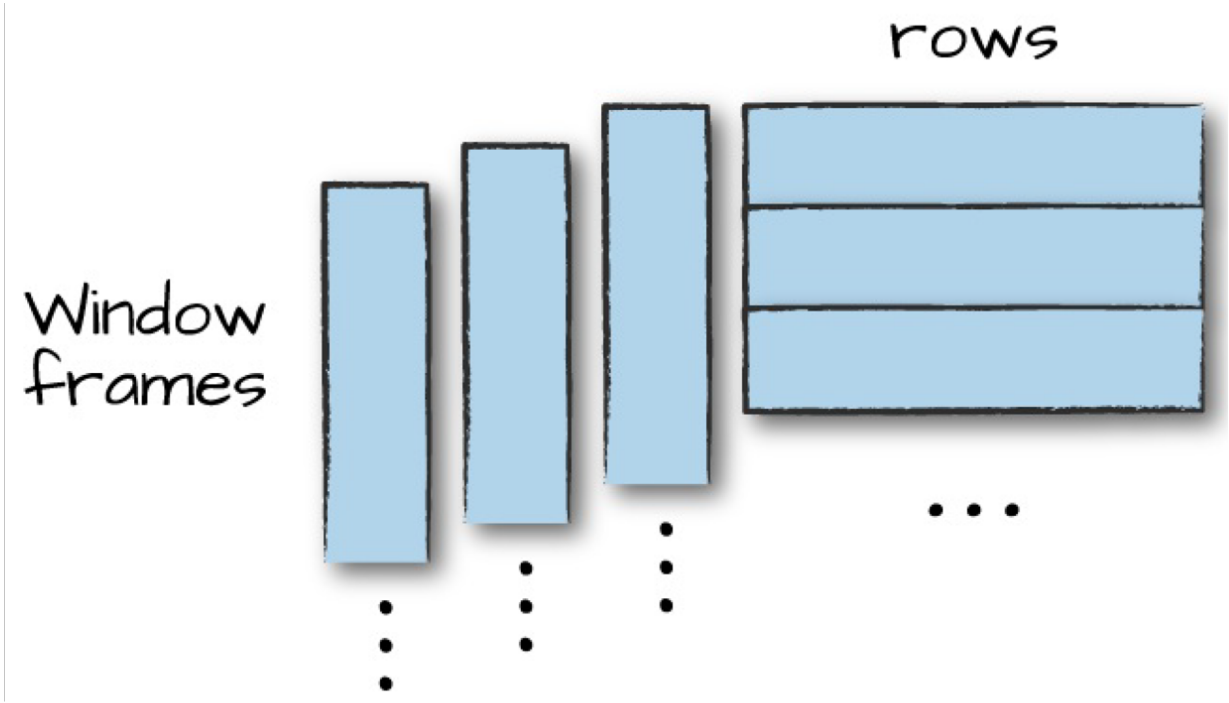
```
arrayArrayData = [
  ("James",[["Java","Scala","C++"],["Spark","Java"]]),
  ("Michael",[["Spark","Java","C++"],["Spark","Java"]]),
  ("Robert",[["CSharp","VB"],["Spark","Python"]])
]

df = spark.createDataFrame(data=arrayArrayData, schema = ['name','subjects'])
```

```
from pyspark.sql.functions import explode
df.select(df.name,explode(df.subjects)).show(truncate=False)

from pyspark.sql.functions import flatten
df.select(df.name,flatten(df.subjects)).show(truncate=False)
```

Window functions

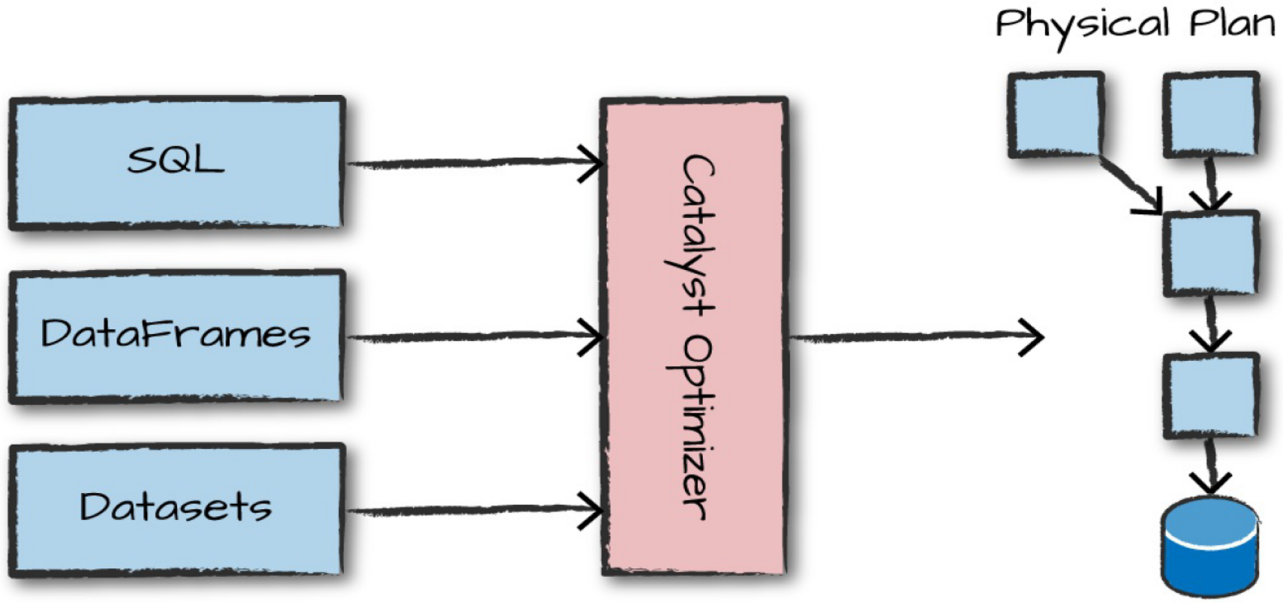


Window functions

WINDOW FUNCTIONS USAGE & SYNTAX	PYSPARK WINDOW FUNCTIONS DESCRIPTION
row_number(): Column	Returns a sequential number starting from 1 within a window partition
rank(): Column	Returns the rank of rows within a window partition, with gaps.
percent_rank(): Column	Returns the percentile rank of rows within a window partition.
dense_rank(): Column	Returns the rank of rows within a window partition without any gaps. Where as Rank() returns rank with gaps.
ntile(n: Int): Column	Returns the ntile id in a window partition
cume_dist(): Column	Returns the cumulative distribution of values within a window partition
lag(e: Column, offset: Int): Column lag(columnName: String, offset: Int): Column lag(columnName: String, offset: Int, defaultValue: Any): Column	returns the value that is `offset` rows before the current row, and `null` if there is less than `offset` rows before the current row.
lead(columnName: String, offset: Int): Column lead(columnName: String, offset: Int): Column lead(columnName: String, offset: Int, defaultValue: Any): Column	returns the value that is `offset` rows after the current row, and `null` if there is less than `offset` rows after the current row.

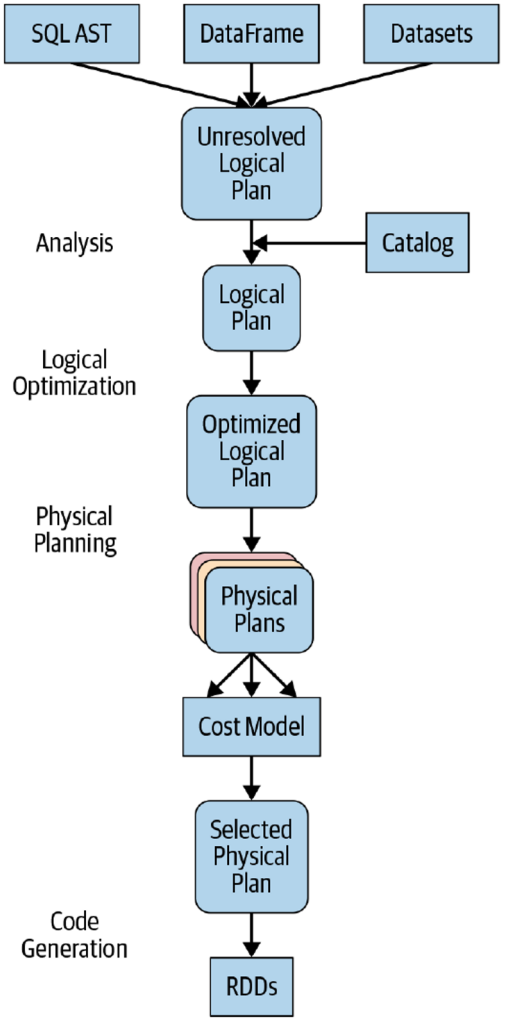
Оптимизатор запросов

Lazy исполнение трансформаций позволяет оптимизировать план



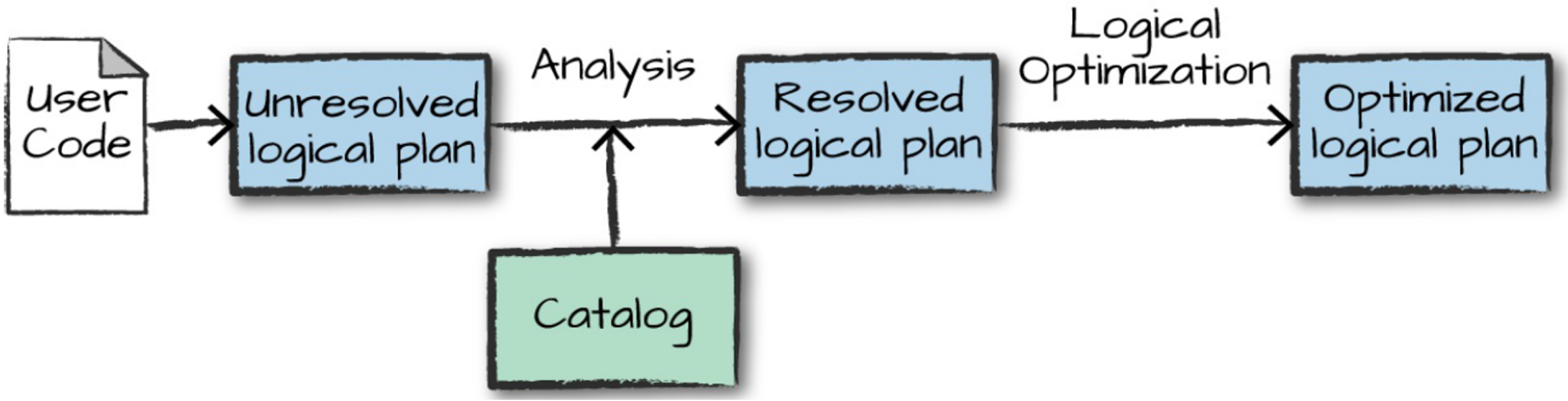
Catalyst Optimizer
transformational phases

1. Analysis
2. Logical optimization
3. Physical planning
4. Code generation



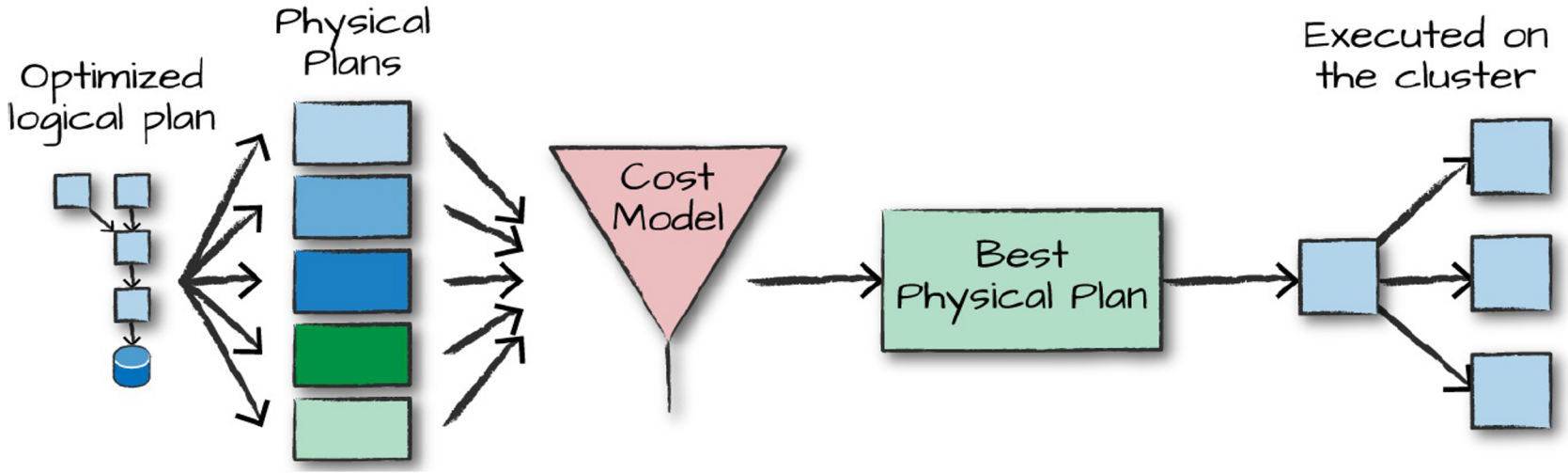
Logical plan

1) проверка наличия таблиц, колонок
2) pushing down predicates or selections

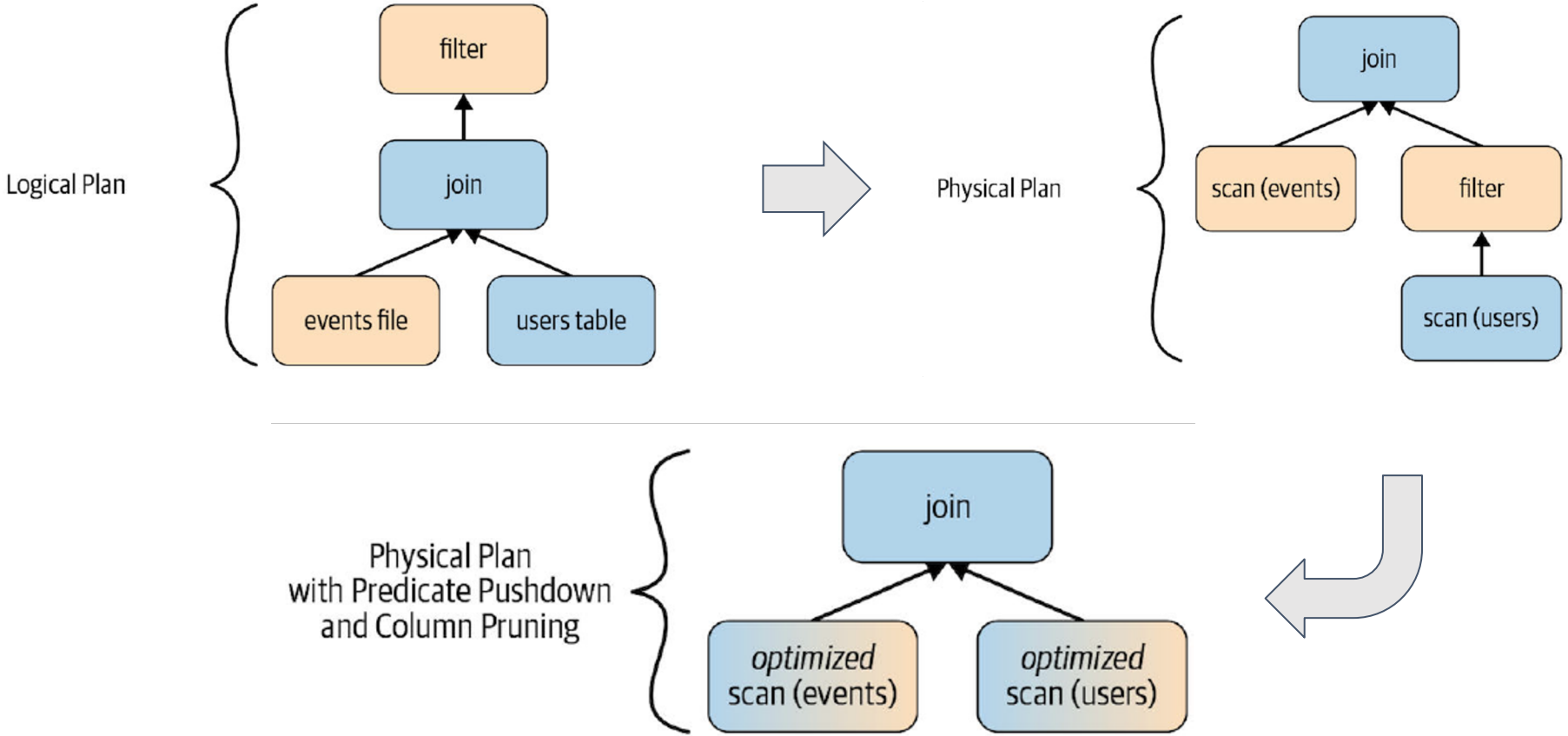


Physical Planning

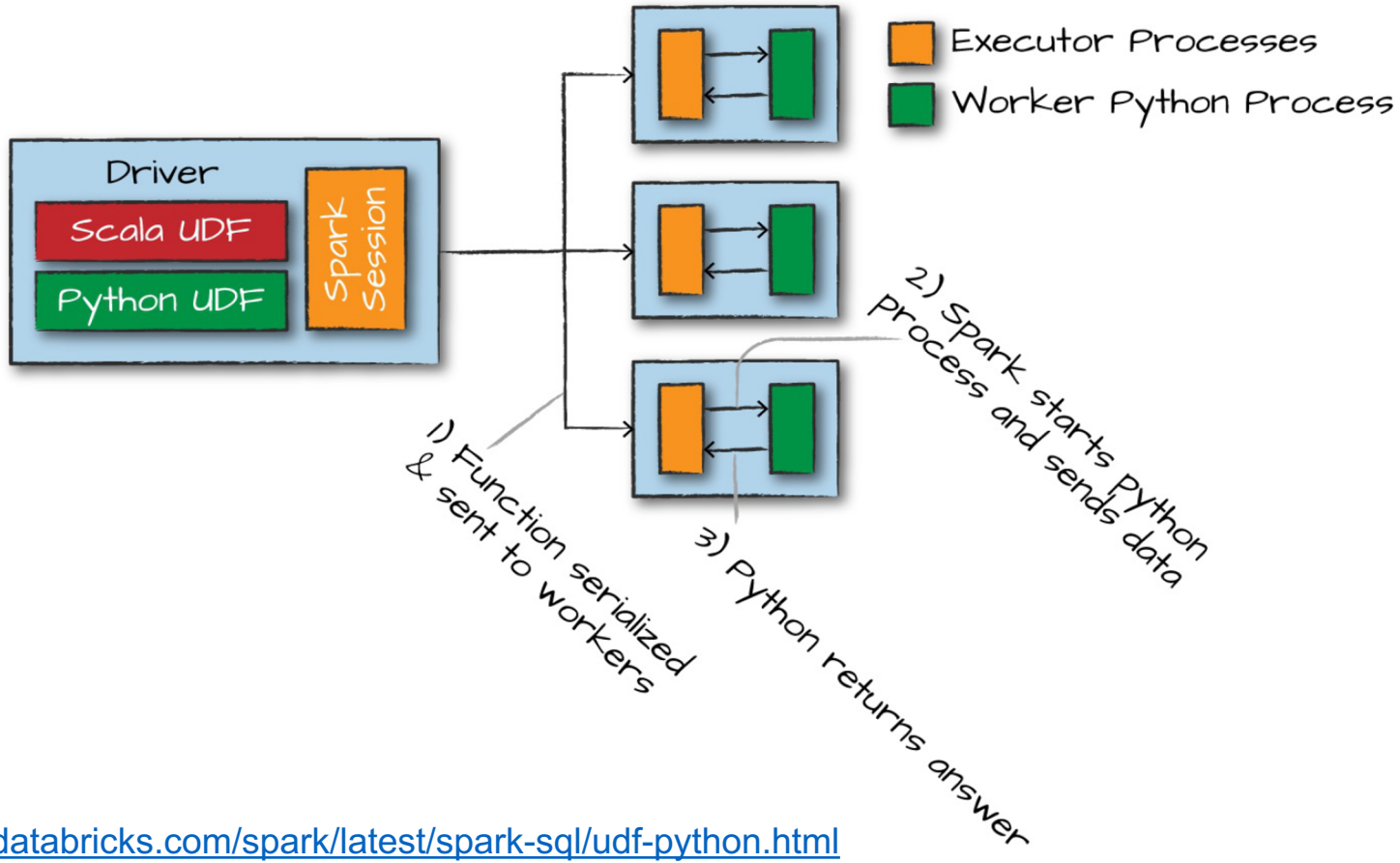
Использует знание о размере и распределении партиций



Планы запроса из примера



User-Defined Functions



https://docs.databricks.com/spark/latest/spark-sql/udf-python.html

Performance concerns with UDFs

- UDFs are black-box to Spark optimizations.
- UDFs block many spark optimizations like
  - WholeStageCodegen
  - Null Optimizations
  - Predicate Pushdown
  - More optimizations from Catalyst Optimizer
- String Handling within UDFs
  - UTF-8 to UTF-16 conversion. Spark maintains string in UTF-8 encoding versus Java runtime encodes in UTF-16.
  - Any String input to UDF requires UTF-8 to UTF-16 conversion.
  - Conversely, a String output requires a UTF-16 to UTF-8 conversion.

df.cache(), df.unpersist()

store as many of the partitions in memory across Spark executors as memory allows

When to Cache and Persist:

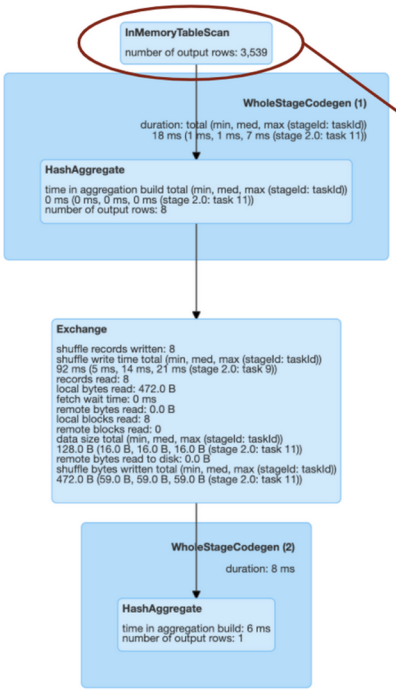
- DataFrames commonly used during iterative machine learning training
- DataFrames accessed commonly for doing frequent transformations during ETL or building data pipelines

When Not to Cache and Persist:

- DataFrames that are too big to fit in memory
- An inexpensive transformation on a DataFrame not requiring frequent use, regardless of size
  https://stackoverflow.com/questions/26870537/what-is-the-difference-between-cache-and-persist

```
df = spark.table("users").filter(col(col_name) > x).cache()

df.count() # now check the query plan in Spark UI
```




```
== Physical Plan ==
*(2) HashAggregate(keys=[], functions=[count(1)]
+- Exchange SinglePartition, true, [id=#59]
   +- *(1) HashAggregate(keys=[], functions=[par
      +- InMemoryTableScan
            +- InMemoryRelation [user_id#0L, disp
                  +- *(1) Project [user_id#0L, d
                     +- *(1) Filter (isnotnull(v
                        +- *(1) ColumnarToRow
                           +- FileScan parquet d
```

This computation is cached.

Ссылки

Introducing Window Functions in Spark SQL - The Databricks Blog

https://www.youtube.com/watch?v=2QNk-bcjN-I

https://sparkbyexamples.com/spark/spark-dataframe-cache-and-persist-explained/

https://towardsdatascience.com/best-practices-for-caching-in-spark-sql-b22fb0f02d34