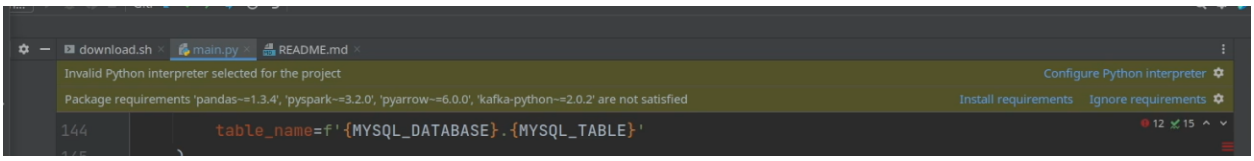
> Конспект > Практика 4 > Основы Apache Spark + Spark SQL. Dataframes

> Оглавление

В этой практике мы напишем Spark приложение, которое будет читать csv файл, формировать на его основе витрину и записывать в базу данных MySQL. В качестве данных служит файлик со статистикой желтого такси, который мы скачаем из AWS. Базу создадим в процессе работы.

> Подготовка

Первым делом необходимо настроить окружение проекта для дальнейшей работы. У вас могут быть такие всплывающие окна:



Для устранения *первого* предупреждения необходимо проделать следующие шаги:

File > Settings > Project: spark > Python Enterpreter > ⚙️ > Add > OK

Этими действиями мы установили виртуальное окружение для нашего проекта.

Для устранения второго предупреждения необходимо установить необходимые библиотеки. Общей и хорошей практикой считается наличие файла requirements.txt в проекте. по которому Pycharm может установить **необходимые**

библиотеки в виртуальное окружение. В нашем проекте такой файлик уже есть, поэтому нам надо кликнуть Install requirements.

В структуре проекта есть необходимый файл download.sh, в котором есть необходимые команды:

`aws s3 cp s3://nyc-tlc/trip\ data/yellow_tripdata_2020-04.csv ./data/ --no-sign-request` - эта команда скачает наш исходный csv файл.

```
mkdir ~/.mysql && \
wget "https://storage.yandexcloud.net/cloud-certs/CA.pem" -O ~/.mysql/root.crt && \
```

`chmod 0600 ~/.mysql/root.crt` - эти команды позволяют скачать сертификат для базы mysql и установить необходимые права.

`wget -O ./jars/mysql-connector-java-8.0.25.jar https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.25/mysql-connector-java-8.0.25.jar` - эта команда позволяет скачать библиотеку для работы с mysql

> Структура нашего пайплайна

Для создания измерений нам необходимо создать list с tuple:

```
vendor_rows = [
    (1, 'Creative Mobile Technologies, LLC'),
    (2, 'VeriFone Inc'),
]

rates_rows = [
    (1, 'Standard rate'),
    (2, 'JFK'),
    (3, 'Newark'),
    (4, 'Nassau or Westchester'),
    (5, 'Negotiated fare'),
    (6, 'Group ride'),
]

payment_rows = [
    (1, 'Credit card'),
    (2, 'Cash'),
    (3, 'No charge'),
    (4, 'Dispute'),
    (5, 'Unknown'),
    (6, 'Voided trip'),
]
```

Также необходимо создать структуру датафрейма - для этого необходимо использовать тип `StructType`. В качестве аргумента необходимо использовать список объектов типа `StructField`. В свою очередь у этого типа 3 аргумента: наименование столбца, тип столбца и необязательный атрибут обязательности.

```
trips_schema = StructType([
    StructField('vendor_id', StringType(), True),
    StructField('tpep_pickup_datetime', TimestampType(), True),
    StructField('tpep_dropoff_datetime', TimestampType(), True),
    StructField('passenger_count', IntegerType(), True),
    StructField('trip_distance', DoubleType(), True),
    StructField('ratecode_id', IntegerType(), True),
    StructField('store_and_fwd_flag', StringType(), True),
    StructField('pulocation_id', IntegerType(), True),
    StructField('dolocation_id', IntegerType(), True),
    StructField('payment_type', IntegerType(), True),
    StructField('fare_amount', DoubleType(), True),
    StructField('extra', DoubleType(), True),
    StructField('mta_tax', DoubleType(), True),
    StructField('tip_amount', DoubleType(), True),
    StructField('tolls_amount', DoubleType(), True),
    StructField('improvement_surcharge', DoubleType(), True),
    StructField('total_amount', DoubleType(), True),
    StructField('congestion_surcharge', DoubleType()),
])
```

Для создания измерения напишем функцию, которая с помощью Spark создает датафрейм:

```
def create_dict(spark: SparkSession, header: list[str], data: list):
    """создание словаря"""
    df = spark.createDataFrame(data=data, schema=header)
    return df
```

> Пайплайн

Сначала вызовем функция agg_calc, результат которой и будет нашей витриной:

```
def agg_calc(spark: SparkSession) -> DataFrame:
    data_path = os.path.join(Path(__name__).parent, './practice4/data', '*.csv')

    trip_fact = spark.read \
        .option("header", "true") \
```

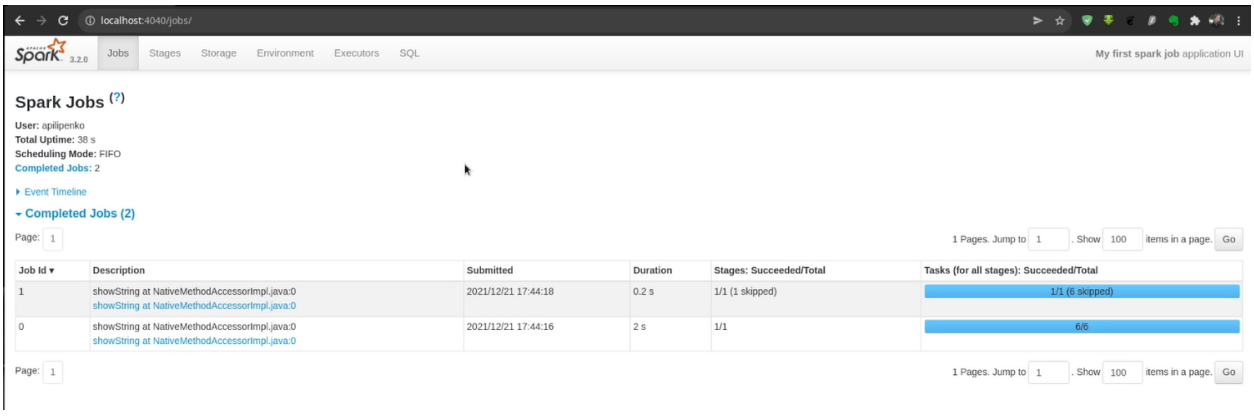
```
        .schema(trips_schema) \
        .csv(data_path)

    datamart = trip_fact \
        .where(trip_fact['vendor_id'].isNotNull()) \
        .groupBy(trip_fact['vendor_id'],
                 trip_fact['payment_type'],
                 trip_fact['ratecode_id'],
                 f.to_date(trip_fact['tpep_pickup_datetime']).alias('dt')
                 ) \
        .agg(f.sum(trip_fact['total_amount']).alias('sum_amount'), f.avg(trip_fact['tip_am
ount']).alias("avg_tips")) \
        .select(f.col('dt'),
                f.col('vendor_id'),
                f.col('payment_type'),
                f.col('ratecode_id'),
                f.col('sum_amount'),
                f.col('avg_tips')) \
        .orderBy(f.col('dt').desc(), f.col('vendor_id'))

    return datamart
```

На 2 строчке вызывается функция os.path.join, которая позволяет получить путь до всех csv файлов в директории ./practise/data

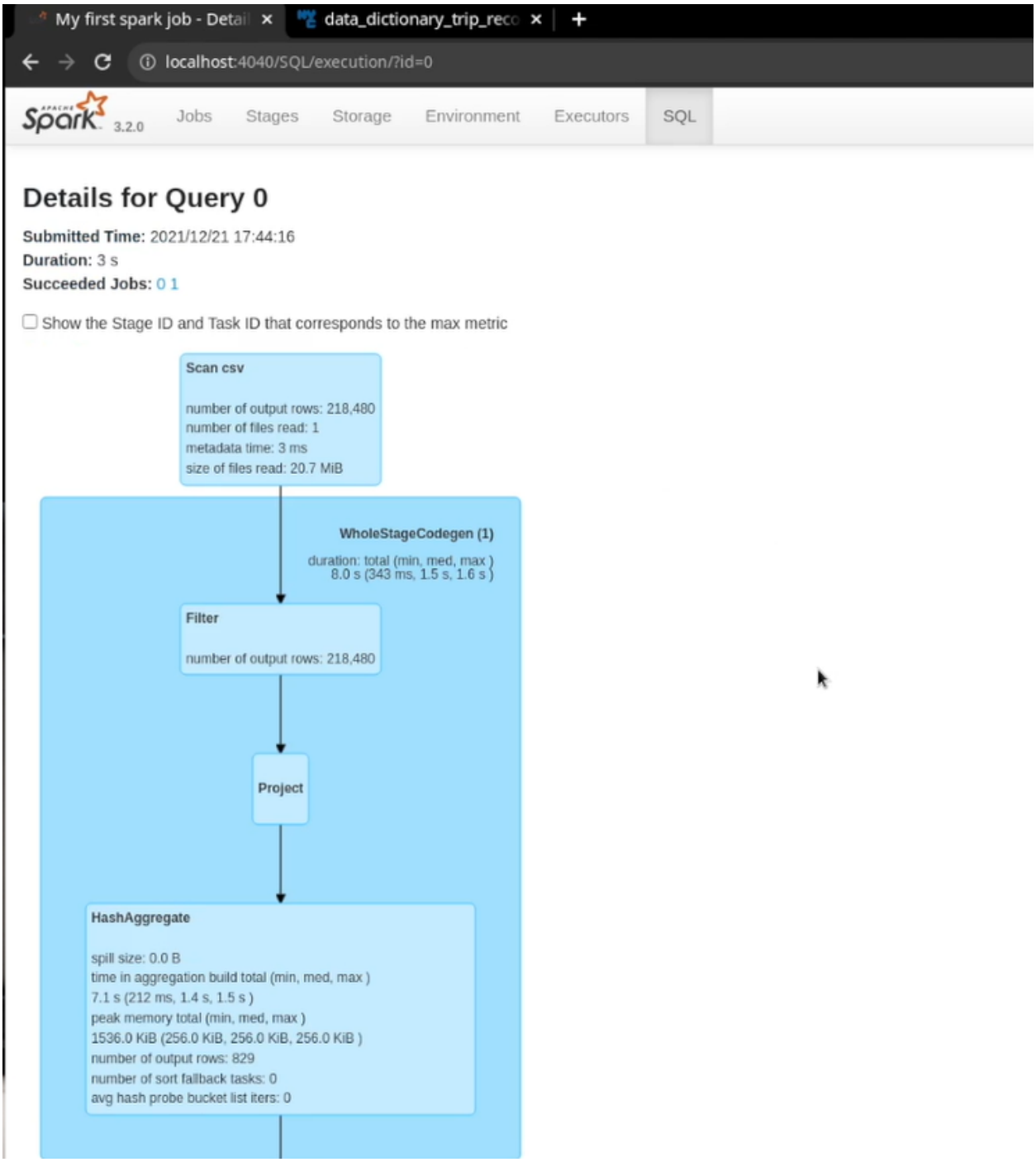
На 4 строчке вызывается метод spark.read, который позволяет считать csv файл в датафрейм. Указав **.option("header", "true")** мы даем указание использовать схему, которую мы предварительно создали.

Далее происходят различные преобразования данных. Сначала фильтрация (where), далее группировка по 4 полям (также используется функция to_date из модуля functions в pyspark.sql). Теперь выбираем аггрегации с помощью agg, а также выполним select. В итоге отсортируем датафрейм с помощью orderBy.

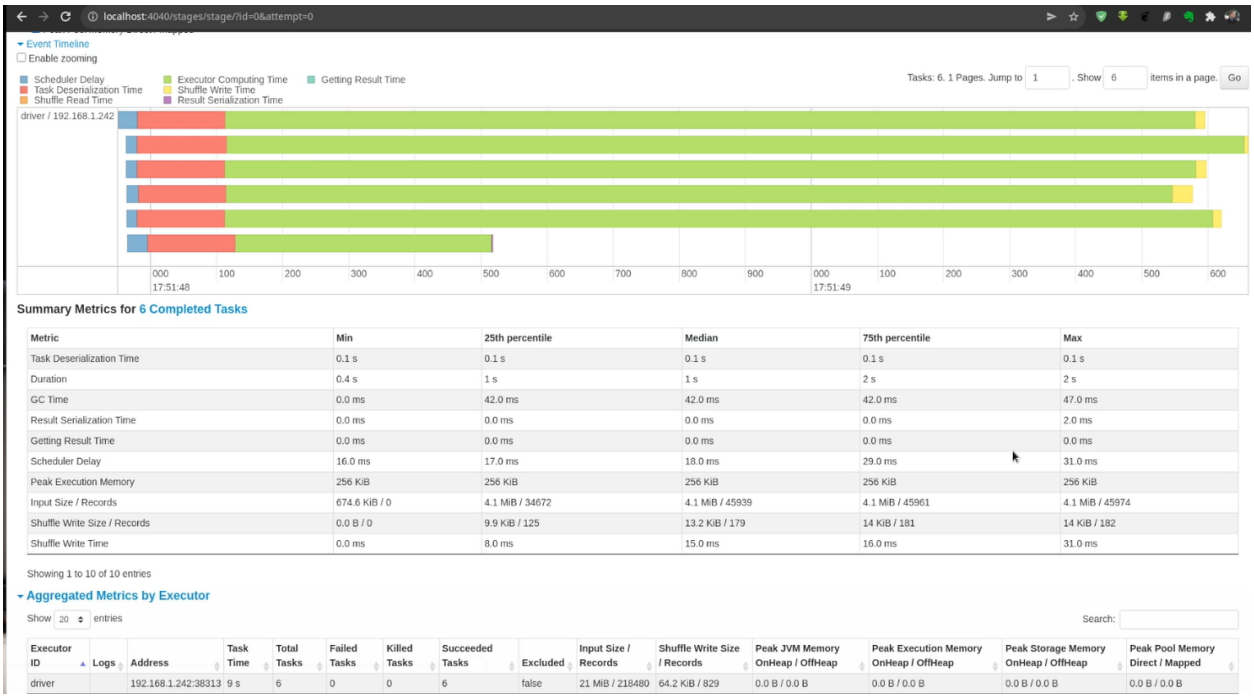
В интерфейсе Spark можно посмотреть запущенные Jobs:



Во вкладке SQL мы можем кликнуть на нашу Job и посмотреть на подробное выполнение команды:



Также можно посмотреть различные метрики по каждому из процессов:



Следующим этапом является соединение нашей витрины с измерениями, которые мы предварительно создали. Для соединения используем функцию join.

```
joined_datamart = datamart \
        .join(other=vendor_dim, on=vendor_dim['id'] == f.col('vendor_id'), how='inner') \
        .join(other=payment_dim, on=payment_dim['id'] == f.col('payment_type'), how='inne
r') \
        .join(other=rates_dim, on=rates_dim['id'] == f.col('ratecode_id'), how='inner') \
        .select(f.col('dt'),
                f.col('vendor_id'), f.col('payment_type'), f.col('ratecode_id'), f.col('su
m_amount'),
                f.col('avg_tips'),
                rates_dim['name'].alias('rate_name'), vendor_dim['name'].alias('vendor_nam
e'),
                payment_dim['name'].alias('payment_name'),
                )
```

Сначала мы джойним 3 измерения, после чего селектим несколько столбцов - таким образом мы сформировали витрину.

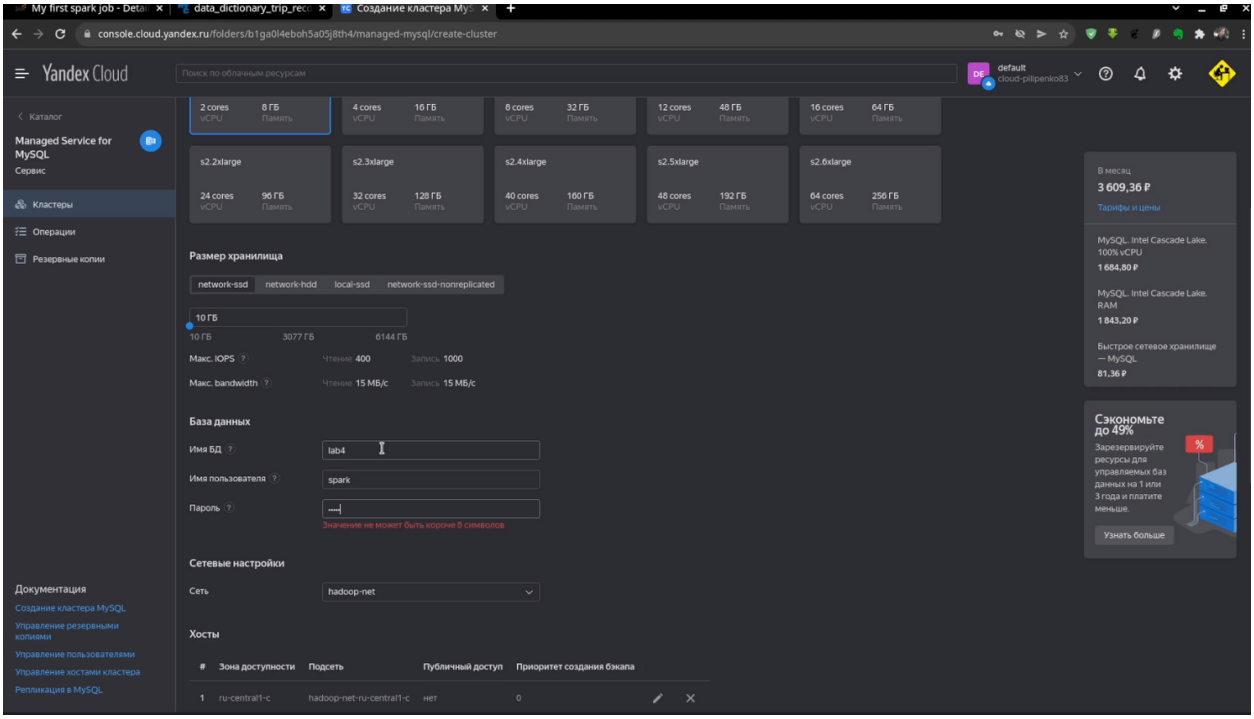
Датафреймы можно также кэшировать, для этого используется Storage в Spark. Чтобы закэшировать датафрейм, нужно вызвать функцию cache(). Кэширование позволяет оптимизировать работу нашего пайплайна, что можно увидеть во вкладке SQL.

> Результат пайплайна

Мы можем сохранить нашу витрину в файлик csv. Для этого нужно использовать функцию write:

`joined_datamart.write.mode('overwrite').csv('output')`

Создадим предварительно базу MySQL:



Теперь мы можем записать наш датафрейм в созданную ранее базу данных MySQL. Для этого используется функция:

```
def save_to_mysql(host: str, port: int, db_name: str, username: str, password: str, df: Da
taFrame, table_name: str):
    props = {
        'user': f'{username}',
        'password': f'{password}',
        'driver': 'com.mysql.cj.jdbc.Driver',
        "ssl": "true",
        "sslmode": "none",
    }

    df.write.mode("append").jdbc(
        url=f'jdbc:mysql://{host}:{port}/{db_name}',
        table=table_name,
        properties=props)
```

Аргументами метода являются данные для подключения к базе данных. В спарке для записи датафрейма в базу используется jdbc драйвер, который мы предварительно скачали.

Таким образом мы написали наше Spark приложение, которое скачивает csv файлик и загружает его в базу MySQL.