|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Системы управления данными»

**Практическое занятие № 7**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Алексеева Екатерина Сергеевна, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

1. Практическая работа №7
   1. Цель работы

В этой лабораторной мы будем использовать PySpark оболочку, чтобы рассмотреть работу с PySpark DataFrame API. Создадим DataFrame из различных источников данных и произведем базовые трансформации.

## Создание DataFrame из различных источников данных

### Создание DataFrame из файлов JSON

#### Переместитесь в директорию /home/student/Data

#### Скопируйте файл people.json в домашнюю директорию HDFS

#### 

#### Используйте Hue или команды командной строки HDFS, чтобы проверить, что people.json был скопирован в HDFS

#### 

#### Из Jupyter, используйте команду spark.read.json(<file\_path>), чтобы прочитать people.json

#### Выведите схему используя действие .printSchema().

#### Выведите содержимое DataFrame с помощью действия .show().

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Отфильтруйте dataframe по людям возрастом больше 20 лет

#### Используйте преобразование where(<условие>) с "age >=30" в качестве условия.

#### Выведите схему используя printSchema().

#### Выведите содержимое dataframe используя show().

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Создайте новый dataframe, который содержит только столбцы name.

#### Используйте преобразование select()

#### Выведите схему нового dataframe

#### Используйте show(5), чтобы проверить

###### 

### Создайте dataframe из файла CSV

#### Скопируйте файл people.csv в домашнюю директорию HDFS из /home/student/Data



#### Проверьте, что файл был скопирован

#### Используйте spark.read.csv(<path\_to\_file>), чтобы создать dataframe из файла CSV.

workerDF = spark.read.csv("people.csv")

workerDF.printSchema()

workerDF.show()

### Результат получился не совсем таким, как ожидалось. Dataframe из одного столбца \_c0 строкового типа. Посмотрите на исходные данные и устраните проблему.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Есть несколько проблем. Первая заключается в том, что Spark собрал все данные в один столбец. Внимательно посмотрите на данные. Обратите внимание, что разделителем является точка с запятой (;). Разделителем Spark по умолчанию для CSV-файлов является запятая (,). Необходимо указать параметр для указания разделителя, отличного от значения по умолчанию. Это можно сделать с помощью опции "sep". Следующая проблема — это имена столбцов. В настоящее время существует \_c0 для имени одного столбца. Spark по умолчанию использует значения \_c0, \_c1 и т.д., когда он не знает имен столбцов. Однако смотрите внимательно. Первая строка содержит все имена столбцов. Установите для параметра "header" значение "true", чтобы Spark знал, что первая строка является заголовком.

#### Устраните обе проблемы, выведите схему и содержимое, чтобы убедиться, что проблема устранена.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Создайте dataframe из файла Parquet

#### Перейдите в директорию /home/student/Data

#### Используйте parquet-tools, чтобы просмотреть схему файла users.parquet

parquet-tools inspect users.parquet

###### 

#### Используйте команду parquet-tools show, чтобы просмотреть содержимое файла users.parquet

parquet-tools show users.parquet | more

###### 

#### Скопируйте users.parquet в домашнюю директорию HDFS

#### Используйте командную строку Hue или HDFS для проверки того, что файл был скопирован

#### 

#### Создайте dataframe из users.parquet: Формат для dataframe в Spark по умолчанию - это parquet. Кроме того, поскольку файлы parquet имеют свою схему, встроенную в двоичный файл, нет никакой необходимости в опциях. Просто используйте spark.read.load(<path\_to\_file>)

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Создание dataframe из файла Avro Хотя формат файла Avro поддерживается начиная с Spark 2.4, файл JAR должен быть доступен для Spark. При запуске оболочки PySpark можно использовать

### --package org.apache.spark:spark-avro\_2.12:3.1.2. Однако этот метод часто приводит к проблемам с подключением. Более простое решение - просто загрузить файл org.apache.spark:spark-avro\_2.12:3.1.2.jar и поместите его в classpath Spark JAR.

#### Переместитесь в /home/student/Data

#### Скопируйте файл spark-avro\_2.12-3.1.2.jar в $SPARK\_HOME/jars

cp spark-avro\_2.12-3.1.2.jar $SPARK\_HOME/jars/.

#### 

#### Перезапустите PySpark и Jupyter

#### Скопируйте users.avro в домашнюю директорию HDFS

#### Используйте командную строку Hue или hdfs для проверки того, что файл был скопирован

#### 

#### Создайте dataframe из users.avro используя

spark.read.format("avro").load(<path\_to\_file>)

#### Проверьте схему

#### Проверьте содержимое

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

## Сохранение DataFrame

### Сохраните данные о пользователях из формата Avro в формат JSON

#### Используйте spark.write.option(<options>).<format\_shortcut>(<path>), чтобы сохранить dataframe в желаемом формате с использованием указанных опций

avroDF.write.json("/user/student/users\_json")

#### Используйте командную строку HDFS или Hue, чтобы убедиться, что данные были сохранены правильно и в правильном формате

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Сохраните dataframe созданный из people.json как файл CSV

#### Сохраните jsonDF созданный на предыдущем шаге в файловом формате CSV

#### Установите разделитель в "|" используя .option("sep", <delimiter>)

#### Добавьте строку заголовка с именами столбцов, используя .option("header", "true")

#### 

#### Сохраните dataframe в /user/student/people\_csv

#### 

#### Используйте командную строку HDFS или Hue, чтобы убедиться, что Данные сохранены верно и в правильном формате.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Попытайтесь сохранить avroDF как файл CSV. Сами выберите разделитель на свое усмотрение и сохраните в /user/student/users\_csv.

#### Что произошло? Что означает полученное сообщение об ошибке? Не все форматы данных могут быть сохранены в любом формате. В этом случае файл Avro содержит столбец, представляющий собой массив (аналогично списку в Python). К сожалению, формат CSV не может поддерживать сложный столбец, такой как массив.

### Сохраните workerDF в формате CSV format.

#### Сохраните файл CSV в /user/student/workers

#### 

#### Установите параметр, чтобы не включать строку заголовка. Другими словами, установите для параметра заголовка значение false.

#### Убедитесь, что файл был создан, как ожидалось

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

## Режимы записи DataFrame

### Создайте новый dataframe из коллекции

#### Создайте следующий список и назовите его new\_workers

new\_workers = [("Henry", '18', "Mail Clerk"),

("Sharon", '24', "Marketing"),

("Shaun", '32', "Attorney")]

#### Используйте SparkSession.createDataFrame(<collection>), чтобы создать новый dataframe из new\_workers list

newWorkersDF = spark.createDataFrame(new\_workers)

### Сохраните newWorkersDF в /user/student/workers в формате CSV

#### Сначала попробуйте сохранить, используя тот же синтаксис, что и раньше.

newWorkersDF.write \

.option("header", "false") \

.csv("/user/student/workers")

### Возникает сообщение об ОШИБКЕ. Что это за сообщение об ошибке? Почему появилось сообщение об ошибке?

###### 

### Ранее мы сохранили workerDF в указанный каталог на шаге 2.4. Невозможно просто сохранить в один и тот же каталог. Настройка Spark по умолчанию заключается в том, чтобы вызывать ОШИБКУ при попытке записи в существующий каталог. Это хорошая задумка, так как мы не хотим случайно перезаписать какую-то ранее сохраненную работу.

#### Используйте метод .mode(<write\_mode>) чтобы изменить режим записи на append. Используйте следующую команду, на этот раз:

#### newWorkersDF.write \

#### .option("header", "false") \

#### .mode("append") \

#### .csv("/user/student/workers")

#### Проверьте с помощью Hue, что дополнительные рабочие данные были добавлены в указанный каталог.

#### 

#### Наконец, создайте dataframe, прочитав файл CSV в /user/student/workers и используйте show, чтобы отобразите его содержимое. Убедитесь, что дополнительные работники были добавлены.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Новые рабочие были добавлены, но имена столбцов были потеряны. Это происходит потому что мы явно установили для параметра заголовка значение false при сохранении данных. В другой лабораторной работе мы узнаем, как явно создавать информацию о схеме и применять ее к dataframe.

# Работа с Hive в Spark

В этой лабораторной работе мы будем использовать Spark для доступа к таблицам Hive, изменять их и сохранять как управляемые, так и внешние таблицы Hive.

## Создайте dataframe из таблицы Hive

### Проверьте, существует ли таблица авторов в Hive. Если нет импортируйте ее из MySQL.

#### В предыдущей лабораторной работе таблица authors в MySQL была импортирована в Hive в базе данных mydb. Убедитесь, что база данных mydb и таблица авторов существуют: запустите beeline со следующей командой и используйте show databases из нее.

$ beeline -u jdbc:hive2://

###### 

если база данных не существует, используйте следующую команду для создания базы данных mydb

jdbc:hive2://> CREATE DATABASE mydb;

Если mydb не существует, то и таблица авторов, скорее всего, тоже не существует. С другого терминала используйте Sqoop для импорта таблицы авторов в Hive. Используйте следующую команду sqoop.

$ sqoop import \

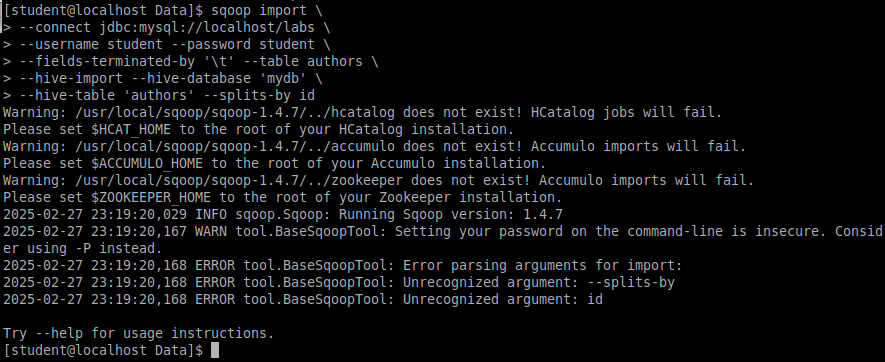
--connect jdbc:mysql://localhost/labs \

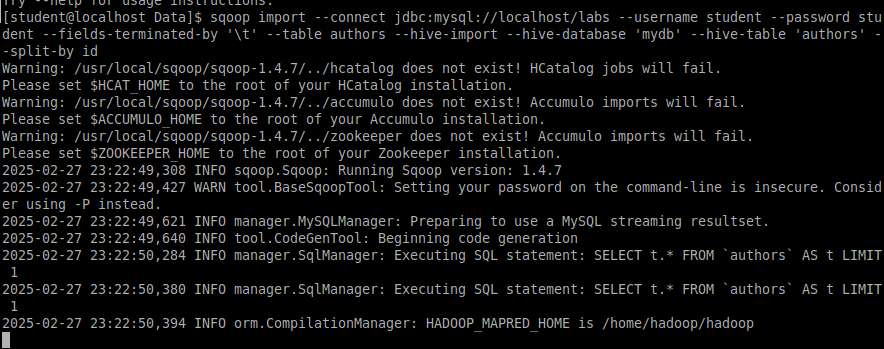
--username student --password student \

--fields-terminated-by '\t' --table authors \

--hive-import --hive-database 'mydb' \

--hive-table 'authors' --split-by id





Sqoop может выдать ошибку если /user/student/authors директория уже существует из предыдущих лабораторных работ. Sqoop импортирует и размещает таблицу авторов в домашнем каталоге пользователя перед вызовом механизма Hive для создания и загрузки данных. Если это так, переименуйте /user/student/authors в /user/student/my\_authors и снова вызовите команду sqoop. Чтобы переименовать директорию HDFS, используйте команду hdfs dfs -mv <original\_name> <new\_name>

$ hdfs dfs -mv authors my\_authors

#### 

#### Убедитесь, что таблица авторов была правильно импортирована либо из предыдущей лабораторной работы, либо из предыдущего шага выше. Выполните следующую команду beeline, чтобы проверить это.

jdbc:hive2://> use mydb;

jdbc:hive2://> show tables;

###### 

### Из Jupyter, прочитайте Hive таблицу authors

#### Используйте команду spark.read.table("<database\_name>.<table\_name>"), чтобы прочитать таблицу и сохранить в authorsDF.

#### Распечатайте схему authorsDF, используя действие printSchema() для dataframe

###### 테이블이(가) 표시된 사진 자동 생성된 설명

#### Выведите первые 5 строк authorsDF с .show(5)

테이블이(가) 표시된 사진

자동 생성된 설명

## Создайте управляемую Hive таблицу из Spark

### Начиная с authorDF, создадим новый dataframe только с "id", "email", и "birthdate"

#### Используйте select("<column\_name1>","<column\_name2>",. . .) и назовите новый dataframe как bdayDF.

#### Выведите схему и первые 5 строк, чтобы проверить преобразование. По умолчанию функция show() выводит 20 строк и усекает столбцы. Чтобы контролировать количество печатаемых строк и не выполнять усечение, используйте следующие параметры: show(<num\_rows>, <truncate = True/False>)

###### 테이블이(가) 표시된 사진 자동 생성된 설명

### Сохраните таблицу Hive в качестве управляемой таблицы

#### Используйте spark.write.saveAsTable(<db\_name.table\_name>), чтобы сохранить bdayDF как новую управляемую Hive таблицу

###### 

### Проверьте author\_bday таблицу Hive

#### Вернитесь к beeline или снова запустите beeline

#### Перейдите к базе данных mydb и выведите все таблицы в базе данных mydb

jdbc:hive2://> USE mydb;

jdbc:hive2://> SHOW TABLES;

#### Выполните следующую команду, чтобы просмотреть первые 5 строк author\_bday

jdbc:hive2://> SELECT \* FROM author\_bday LIMIT 5;

###### 

#### Используйте команду describe formated Hive для просмотра метаданных для таблицы author\_bday

jdbc:hive2://> DESC FORMATTED author\_bday;

### При просмотре выходных данных найдите раздел, в котором показано расположение сохраненных данных и тип таблицы. Обратите внимание, что author\_bday - это управляемая таблица hive, и данные хранятся в каталоге хранилища Hive, как и ожидалось.

###### 

#### Используйте Hue для перехода к местоположению, в котором были сохранены данные. Проверьте местоположение и содержимое.

###### 테이블이(가) 표시된 사진 자동 생성된 설명

### Обратите внимание, что данные были сохранены в виде файла parquet. Формат Spark по умолчанию для сохранения dataframe - Parquet format

#### Используйте hdfs dfs -get подкоманду, чтобы скопировать файл part-00000-xxxxx на локальный диск. Используйте parquet-tools для просмотра схемы и содержимого файла parquet. Чтобы упростить получение файла, используйте подстановочный знак \*, чтобы соответствовать окончанию после начальной части part -00000.

$ hdfs dfs -ls /user/hive/warehouse/mydb.db/author\_bday

$ hdfs dfs -get /user/hive/warehouse/mydb.db/author\_bday/part-00000\*

$ parquet-tools inspect part-xxxxx

$ parquet-tools show part-xxxxx

## 

### 

### 

## Создайте внешнюю таблицу Hive из Spark

### Создайте новый dataframe только с "id", "first\_name", и "last\_name" столбцами

### Используйте printSchema() и show(), чтобы проверить новый dataframe

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Сохраните таблицу Hive как внешнюю таблицу в формате CSV, используя разделитель табуляции, и которая включает строку заголовка. Spark сохраняет таблицы Hive как внешнюю таблицу, если указан явный путь. В противном случае таблица сохраняется как управляемая таблица в каталоге хранилища Hive.

#### Используйте spark.write.format(<format>).option(<options>).saveAsTable(<db>.<table>)

nameDF.write \

.format("csv") \

.option("path", "/user/student/author\_names") \

.option("sep", "\t") \

.option("header", "true") \

.saveAsTable("mydb.author\_names")

### 

### Используйте beeline, чтобы убедиться, что новая внешняя таблица создана. Обязательно используйте команду форматирования desc, чтобы убедиться, что таблица является внешней таблицей и данные находятся в ожидаемом местоположении. Обратите внимание, что Hive хранит "place holder" в каталоге mydb.db, поскольку таблица принадлежит этой базе данных.

###### 

### Однако, когда вы прокручиваете дальше вниз, появляется другая информация о "пути", в которой хранятся фактические данные.

###### 

### Используйте Hue для проверки того, что данные были записаны в правильном формате и в указанном формате CSV.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

# Spark SQL Преобразования

В этой лабораторной работе мы рассмотрим различные преобразования Spark SQL. Мы выйдем за рамки базовых преобразований и начнем работать с объектами столбцов, чтобы создать выражение столбца, которое позволяет выполнять более мощные преобразования.

## Запрос dataframe с помощью преобразования Spark SQL

### Выведите имя, адрес электронной почты и дату рождения 3 самых старых авторов При работе с преобразованиями Spark по началу гораздо проще разработать окончательный код, наблюдая за результатом преобразования на каждом шаге. В Jupyter создайте 2 ячейки. В верхней ячейке добавьте и соедините преобразования в цепочку. Во второй ячейке используйте действие show(), чтобы немедленно просмотреть результат преобразования.

#### Создайте новый dataframe, прочитав таблицу авторов в базе данных mydb в Hive. Используйте show(5) во второй ячейке для просмотра результата.

###### 테이블이(가) 표시된 사진 자동 생성된 설명

#### Преобразуйте dataframe, выбрав только имя пользователя, адрес электронной почты и дату рождения. Соедините это преобразование в цепочку до конца и выполните обе ячейки. При соединении преобразований в цепочку PySpark требуется символ "\", если код переходит на новую строку. На самом деле, хорошей практикой является размещение одного преобразования в каждой строке и использование символа "\" для разделения его на отдельные строки. Это делает код намного более разборчивым и понятным.

###### 

#### Используйте orderBy(<column name>) в столбце "birthdate". Соедините преобразование в цепочку и выполните обе ячейки, чтобы просмотреть результаты

###### Ф

#### Используйте limit(<num>), чтобы ограничить количество строк до <num> = 3. Свяжите это преобразование в цепочку и выполните обе ячейки, чтобы просмотреть результаты.

###### 

### Обратите внимание, что, хотя show(5) был выполнен, отображаются только 3 строки. Это связано с тем, что limit(3) уменьшил фрейм данных до 3 строк. Строк для вывода больше нет, поэтому, несмотря на то что был вызван show(5), Spark вернул максимальное количество доступных строк.

### Выведите имя, адрес электронной почты и дату рождения 3 самых старых авторов. Однако на этот раз измените преобразование сверху таким образом, чтобы запрос выбирал 3 самых старых автора, родившихся после начала нового тысячелетия. То есть мы хотим выбирать только из авторов, родившихся 1 января 2001 года или после этой даты.

#### Используйте where(<condition>), чтобы выбрать только авторов, дата рождения которых больше или равна '2001-01-01'. Попробуйте использовать строку "birthdate >= '2001-01-01'" для применения условия. Преобразование where() должно немедленно следовать за преобразованием select сверху и перед преобразованием OrderBy. Преобразование OrderBy — это дорогостоящая операция, требующая, чтобы разделы обменивались информацией для глобальной сортировки dataframe. Мы хотим уменьшить объем данных, c которыми необходимо работать до, а не после. За счет сокращения набора данных до тех, кто родился в новом тысячелетии, до OrderBy, объем данных, которыми необходимо обработать, значительно сокращается. На данный момент удалите преобразование limit(3). Мы хотим убедиться, что мы просматриваем всех авторов, дата рождения которых удовлетворяет условию, ПРЕЖДЕ чем ограничивать выходные данные, чтобы проверить фильтр. Простой способ сделать это - добавить # перед ним. # указывает на то, что то, что следует за этим, является комментарием. По сути, мы закомментировали преобразование limit().

###### 

#### Теперь, когда код протестирован и проверен, добавьте обратно преобразование limit(3), чтобы отобразить 3 самых старых автора, родившихся после нового тысячелетия.

###### 

## Использование объектов столбцов и выражений столбцов

Можно было выполнить довольно простую операцию сравнения, используя понятие строки в приведенном выше предложении where. Однако для более сложных операций объекты столбцов и выражения столбцов могут использовать богатый набор операторов и функций, включая возможность определения пользовательских функций.

### Создайте dataframe из sales.csv

#### Перейдите в локальную директорию /home/student/Data

#### Изучите sales.csv и определите его схему и разделитель

#### Скопируйте sales.csv в домашнюю директорию HDFS

#### 

#### Создайте dataframe из sales.csv и назовите его salesDF. Есть ли в исходном файле строка заголовка? Что такое разделитель? Является ли это разделителем по умолчанию ","?

#### Выведите схему и вызовите show(5), чтобы убедиться, что dataframe создан

###### 

### Используя объекты столбцов, выберите только Country, ItemType, UnitsSold, UnitPrice и UnitCost

#### Используйте salesDF.<column\_name> синтаксис для создания объекта столбца.

#### Выведите схему и используйте show для актуализации преобразования и проверки

###### 

### Используйте функцию приведения объектов столбцов для изменения типа данных и выполнения вычислений

#### Объекты столбцов могут быть приведены к другому типу данных с помощью функции cast(<new\_type>). Используйте cast, чтобы привести типы UnitsSold и UnitPrice к float и рассчитать общую сумму заказа, умножив два значения

###### 

#### Результирующий тип данных после выполнения операции над объектами столбцов является объектом столбца. Объект столбца может быть сохранен в переменной точно так же, как и любой другой объект. На этот раз создайте переменную и сохраните результат операции над объектами столбца. Выведите переменную.

###### 

### Обратите внимание, что выходные данные показывают, что переменная имеет тип Column с фактическими операциями в качестве значения.

#### Вывод шага 2.3.1 показывает имя столбца, которое буквально представляет собой операцию, выполненную для получения результата столбца. Используйте метод объекта столбца псевдонима, чтобы присвоить ему более подходящее имя. Повторите callcRevDF. На этот раз используйте объект столбца calcRevenue, который был сохранен вместе с методом псевдонима, чтобы присвоить имя столбцу "Sales\_Revenue".

###### 

### Обратите внимание, что для одной страны существует несколько записей. Например, в приведенном выше выводе есть по крайней мере два элемента строки для Ливии. Это связано с тем, что каждая продажа дополнительно классифицируется по типу товара. Повторите приведенное выше вычисление, но на этот раз включите ItemType

###### 

### Измените revItemCountryDF. Используйте сравнительный оператор, чтобы проверить, была ли соблюдена квота

#### Создайте новый столбец, который будет иметь значение true или false в зависимости от того, была ли достигнута квота. DataFrame API предоставляет метод withColumn(<column\_name>, <operation to calculate value of column>). Установите для переменной значение квоты в 3 миллиона.

quota = 3000000

#### Создайте объект столбца, который проверяет столбец "Sales\_Revenue" и проверяет, была ли соблюдена квота

quotaMet = (revItemCountryDF.Sales\_Revenue > quota)

#### Используйте функцию withColumn(), создайте новый столбец и назовите его "Quota\_Met". Используйте объект столбца quotaMet для операции

withColumn("Quota\_Met", quotaMet)

#### Выведите схему и вызовите show(5) для проверки преобразований.

###### 

### На этот раз измените определение того, что означает достижение квоты. Квота будет достигнута, если либо рассчитанная сумма превышает квоту, либо количество проданных единиц превышает заданное количество

#### Создайте новый dataframe из sales.csv и выберите "Country", "ItemType", "UnitsSold" и "UnitPrice".

#### Установите переменную amtQuota = 3000000

#### Установите переменную cntQuota = 5000

#### Повторите объект столбца quotaMet. Теперь он должен создать новый объект столбца после проверки как amtQuota, так и cntQuota. Если какая-либо из квот была выполнена, верните значение True

#### Используя withColumn, добавьте столбец "Sales\_Revenue". Значением столбца будет сумма, рассчитанная путем умножения "UnitsSold" на "UnitPrice" На этот раз, не приводите результат к float, как мы делали раньше.

#### Используя withColumn, добавьте столбец "Quota\_met". Значение столбца будет основано на измененном объекте столбца quotaMet

###### 

### Обратите внимание, что Spark был достаточно умен, чтобы понять, что доход от продаж должен быть приведен к некоторому числу. Фактически, он автоматически привел его к double типу данных. Разделение условий и сохранение их в переменной делает преобразование withColumn() намного более читабельным. В PySpark символ ("|") используется для обозначения условий "ИЛИ".

## Использование агрегатных функций с объектами столбцов

### Перечислите все проданные ItemTypes

#### Найдите все типы элементов, сгруппировав их по типу элемента. Для этого воспользуйтесь groupBy(<столбец>).

#### Используйте функцию count() для подсчета количества строк для каждого типа элемента. Результат должен быть похож на приведенный ниже:

###### 

### Можете ли вы догадаться, почему код выбирает только столбец "ItemType" перед группировкой по этому столбцу? groupBy — это очень дорогостоящая операция, которая требует, чтобы все разделы обменивались информацией для создания глобальной группировки. Поэтому важно уменьшить объем данных, которые необходимо перетасовать, прежде чем вызывать преобразование groupBy. Единственным обязательным столбцом является сам "ItemType", поэтому это единственный выбранный столбец.

### Каково среднее количество проданных товаров для каждого ItemType?

#### В salesDF выберите "ItemType" и "UnitsSold". Приведите значение "UnitsSold" к целому числу. Этот столбец должен быть числом, чтобы позже рассчитать среднее значение.

#### Используйте groupBy для каждого ItemType

#### Вычисление среднего значения с использованием преобразования mean (<column>).

#### Выведите схему и вызовите show() для проверки преобразований и выходных данных

###### 

### Каков средний доход от продаж для каждого ItemType?

#### В salesDF выберите "ItemType", "UnitPrice" и "UnitsSold".

#### Используя withColumn, создайте столбец Sales\_Revenue, значение которого равно UnitPrice \* UnitsSold. Результат будет автоматически приведен к типу double.

#### Используйте groupBy для каждого ItemType

#### Рассчитайте средний доход, используя преобразование mean (<column>).

#### Выведите схему и вызовите show() для проверки преобразований и выходных данных

###### 

## Создание пользовательских функций

### Наше текущее условие quotaMet на самом деле не дает объективной информации. Он не учитывает базовый тип элемента при тестировании, если квота была соблюдена. Измените условие, чтобы отразить тип элемента.

#### Создайте словарь Python, как показано ниже:

cntQuotaDict = {"Baby Food" : 5018,

"Cereal" :4908,

"Household" : 5229,

"Vegetables" : 4818,

"Beverages" : 4858,

"Office Supplies" : 4994,

"Cosmetics" : 5680,

"Personal Care" : 5468,

"Fruits" : 5073,

"Snacks" : 4818,

"Clothes" : 4845,}

### Словарь в Python — это элемент Key:Pair. Словари создаются с помощью оператора {item, item,....}. Доступ к значению пары Ключ:Значение осуществляется с помощью метода get(<key>) словаря. Этот словарь представляет собой ItemType в качестве ключа и среднее количество продаж в качестве значения.

#### Создайте определяемую пользователем функцию (UDF), которая принимает два параметра. Первый параметр — это ItemType. Второй параметр — это количество элементов ItemType. UDF будет ссылаться на cntQuotaDict, чтобы проверить, является ли количество элементов больше, чем количество элементов словаря

def cntQuota(item, cnt):

meet\_this = cntQuotaDict.get(item, None)

if meet\_this == None: meet\_this = 0

my\_cnt = int(cnt)

return my\_cnt > meet\_this

#### Для того, чтобы использовать UDF, функция должна быть сначала зарегистрирована. Используйте функцию udf. Тип возвращаемого значения функции также должен быть указан

from pyspark.sql.functions import udf, col

from pyspark.sql.types import BooleanType

quotaUDF = udf(lambda item, cnt: cntQuota(item, cnt), BooleanType())

### 

### Используйте UDF для определения того, была ли достигнута квота

#### Создайте новый dataframe, прочитав файл sales.csv с именем quotaDF.

#### Используйте метод select для включения столбцов "Country" и "ItemType".

#### UDF может быть вызван в инструкции select. Продолжите с преобразованием select из предыдущего шага чтобы включить в результат выполнение UDF. UDF ожидает, что параметры будут переданы как объекты столбца. Используйте метод col для явного создания объектов столбца и передачи его в UDF. Для UDF требуются столбцы ItemType и UnitsSold

#### Используйте функцию псевдонима, чтобы переименовать результат UDF в "Met\_Quota"

#### Выведите схему и вызовите show() для проверки преобразования и вывода

###### 

# Работа со Spark SQL

Spark SQL позволяет разработчикам Spark использовать стандартный SQL ISO для создания запросов из dataframe.

## Использование преобразования SparkSession.sql()

### Создайте запрос с помощью метода sql

#### Выполните следующий код из Jupyter.

authorsDF = spark.sql(" SELECT \* FROM mydb.authors LIMIT 10 ")

authorsDF.printSchema()

authorsDF.show()

### Как можно видеть, spark.sql(<SQL query>) возвращает dataframe. Запрос возвращает все столбцы из таблицы authors в базе данных mydb в Hive.

### Попробуйте другой, более сложный SQL-запрос, где часть запроса включает некоторую строку для фильтрации.

authorsDF = spark.sql(""" SELECT first\_name, last\_name, email

FROM mydb.authors

WHERE email LIKE '%org'

LIMIT 10 """ )

authorsDF.printSchema()

authorsDF.show(truncate=False)

### Этот SQL-запрос возвращает имя, фамилию и адрес электронной почты авторов, адрес электронной почты которых заканчивается на "org". Если запрос содержит такие строки, как '%org', используйте тройные кавычки ("""), чтобы обернуть в них SQL-запрос. Это позволяет легко включать строки в запрос без необходимости экранировать их.

###### 

### Создайте запрос и выведите всех авторов, родившихся в новом тысячелетии.

#### Выберите только first\_name, last\_name и birthdate

#### Отфильтруйте результаты по датам рождения, приходящимся на новое тысячелетие

#### Отсортируйте выходные данных по дате рождения

#### Распечатайте схему и вызовите show() для проверки

###### 

### Как видно, прямое использование SQL может быть гораздо удобнее для тех, кто хорошо знаком с языком структурированных запросов. Однако многие разработчики считают наиболее продуктивным комбинирование выражений столбцов и операторов SQL в своем коде.

## Использование dataframe в качестве таблицы/представления в запросе

До сих пор мы запрашивали существующие таблицы Hive. Однако в SparkSession.sql() можно использовать любой dataframe. Для этого необходимо создать временные представления.

### Создайте dataframe из файла sales.csv в домашнем каталоге HDFS

#### Убедитесь, что файл sales.csv находится в домашнем каталоге HDFS. Если это не так, перейдите в /home/student/Data и скопируйте файл sales.csv в HDFS.

#### Используйте SparkSession.read() для создания dataframe. Включите все столбцы. Назовите фрейм данных salesDF.

### Создайте временное представления

#### Используйте DataFrame.createTempView(<view\_name>) или DataFrame.createOrReplaceTempView(<view\_name>) для создания временного представления. Функция DataFrame.createTempView() выдаст ошибку, если временное представление уже существует. DataFrame.createOrReplaceTempView() является лучшей альтернативой при разработке приложений, поскольку код выполняется повторно, и при использовании DataFrame.createTempView() после создания первого временного представления будет сгенерирована ошибка.

salesDF.createOrReplaceTempView("sales")

### Используя SparkSession.sql(), запросите таблицу продаж и выведите первые 10 строк всех столбцов

### 

### Используйте SparkSession.sql() для создания агрегированных запросов

#### Создайте SQL-запрос, который группирует продажи по каждой Country, ItemType и SalesChannel. Выведите сумму TotalRevenue как Revenue\_Sum и сумму TotalProfit как Profit\_Sum для каждой группы. Выходные данные должны быть упорядочены сначала по наибольшей прибыли и в порядке убывания.

#### 

### Использование полного адреса синтаксиса исходного файла в запросе Иногда желательно просто указать адрес исходного файла для запроса к нему без необходимости создания dataframe, создания временного представления и последующего использования временного представления в запросе.

#### Используйте синтаксис <format>.`<path to source file>` для прямого обращения к исходному файлу внутри SQL-запроса. Создайте запрос для просмотра всех столбцов файла author\_phone.json. Исходный файл окружен символами обратных кавычек. Это клавиша под клавишей ESC и слева от 1 на клавиатуре.

spark.sql(""" SELECT \* FROM

json.`/user/student/author\_phone.json`

LIMIT 10 """).show()

#### При использовании этого параметра нет никакой возможности предоставить какие-либо параметры или преобразовать данные перед их запросом. Этот синтаксис следует использовать только тогда, когда в таких операциях нет необходимости.

#### 

## Использование Spark SQL Magic в Jupyter

### Python предоставляет sparksql-magic, который позволяет удобно использовать ячейки в Jupyter

#### Установите sparksql-magic с помощью следующей команды:

!pip install sparksql-magic

#### 

#### После завершения установки загрузите внешнюю библиотеку

%load\_ext sparksql\_magic

#### Теперь используйте %%sparksql, чтобы указать, что мы будем использовать sparksql-magic для этой ячейки. Выполните это с помощью простого запроса.

###### 

## Объединение DataFrame

### Объединение dataframe с помощью преобразования .join()

#### Создайте dataframe authorsDF из таблицы authors Hive

#### Создайте dataframe authorPhoneDF из author\_phone.файл json в домашнем каталоге HDFS

#### Используйте преобразование dataframe.join(<dataframe>, <join condition>, <join type>) для объединения authorsDF с authorPhoneDF. Используйте внутреннее соединение по умолчанию. Используйте выражения столбцов для <join condition>.

joinDF = authorDF \

.join(authorPhoneDF,

authorDF.id == authorPhoneDF.author\_id)

#### Выведите схему и используйте show() для проверки преобразования и вывода

#### 

### Объединение dataframe с помощью преобразования SparkSession.sql()

#### Создайте временное представление для authorsDF. Назовите его author

#### Создайте временное представление для authorPhoneDF. Назовите представление author\_phone

#### Используйте преобразование sql() для соединения двух таблиц/представлений. Выберите только first\_name, last\_name и phone\_model в качестве выходных столбцов

#### Ограничьте выходные данные 10 строками

###### 

### Используйте sparksql-magic для выполнения запроса, который выводит 10 самых молодых авторов, у которых нет телефонов. Мы проведем розыгрыш призов для счастливых кандидатов.

#### Соедините author и author\_phone. Это имена временного представления, созданного ранее

#### Вызовите select для first\_name, last\_name, phone\_model и birthdate столбцов

#### Отфильтруйте записи с пустой строкой в phone\_model

#### Отсортируйте данные по дате рождения в порядке убывания. Поскольку даты рождения упорядочены в порядке убывания, самые молодые авторы будут отображаться первыми.

#### Ограничьте выходные данные 10 строками

###### 

## Выполнение команд DDL и DML из Spark

В дополнение к SQL разработчики могут запускать DDL (Язык динамического определения) и DML (Язык обработки данных) для определения и создания таблиц Hive, а также их изменения.

### Используйте либо spark.sql(), либо ячейку sparksql-magic для выполнения этих команд DDL

#### Выведите все базы данных в Hive

#### Перейдите к базе данных mydb с помощью команды USE <база данных>

#### Выведите все таблицы в mydb

#### Создайте новую управляемую таблицу и назовите ее spark\_test. Эта таблица будет содержать два столбца: имя в виде строки и возраст в виде целого числа. Оставьте все остальное по умолчанию.

#### Просмотрите подробные сведения о spark\_test с помощью команды DESC FORMATTED

#### Используйте команду INSERT, чтобы добавить несколько строк в spark\_test

#### Распечатайте все столбцы из spark\_test, чтобы убедиться, что строки были вставлены

###### 

###### 

###### 

###### 

### Используйте либо spark.sql(), либо ячейку sparksql-magic для выполнения этих команд DML

#### Добавить еще один столбец: gender строкового типа

spark.sql(""" ALTER TABLE spark\_test

ADD COLUMNS (gender string) """)

#### Протестируйте, добавив еще несколько строк в spark\_test, но теперь включая gender

spark.sql(""" ALTER TABLE spark\_test

ADD COLUMNS (gender string) """)

spark.sql(""" INSERT INTO spark\_test

VALUES ("Shaun", 42, "Male") """)

#### Отобразите всю таблицу spark\_test для проверки результатов

###### 

## Использование Catalog API

### Используйте SparkSession.catalog для доступа к Catalog API. Верните список баз данных, список таблиц, список столбцов. Измените текущую базу данных.

#### Используйте spark.catalog.listDatabases() для возврата списка баз данных. Используйте цикл for для печати каждой из баз данных в вывод

#### Используйте spark.sql("show databases").show() для отображения всех баз данных. Обратите внимание, что функция show() использовалась для отображения dataframe, содержащего базы данных.

#### Используйте spark.catalog.setCurrentDatabase(<database>) для изменения текущей базы данных. Измените базу данных на mydb

#### Используйте spark.sql("use mydb") для изменения базы данных на mydb

#### Используйте spark.catalog.listTables() для перечисления всех таблиц в текущей базе данных. Используйте цикл for для вывода всех таблиц.

#### Используйте spark.catalog.listColumns(<table>) для перечисления всех столбцов в таблице <table>

###### 

###### 

###### 

# Преобразования RDD в DataFrames

В этой главе мы начнем с полуструктурированного источника данных, преобразуем его, чтобы придать ему структуру, и преобразуем его в dataframe для операций запроса.

## Создайте фрейм данных из источника данных JSON

### Импортируйте json и создайте функцию, которая будет извлекать имя телефона и преобразовывать его в понятный текст, при необходимости извлекая несколько телефонов.

Import json and use the following function to set the phone name

import json

def setPhoneName(s):

if s == "": return "Unknown"

elif "," in s: return s.replace(",", " or")

else: return s

#### Создайте RDD с помощью wholeTextFiles() из /user/student/author\_phone.json

#### Используйте flatMap для создания отдельных строк при разборе записей json. Используйте json.load() для содержимого файла, чтобы проанализировать записи JSON. Напомним, что wholeTextFiles создает кортеж формы (<path>,<content>). Передайте <content> в json.load

#### Функция json.load() создает коллекцию типа словаря. flatMap помещает каждую запись словаря в качестве элемента строки. Используйте dictionary.get(<key>), чтобы получить значения author\_id и phone\_model. Создайте кортеж формы (author\_id, phone\_model)

#### Преобразуйте часть phone\_model кортежа ключ:значение, применив функцию setPhoneName, определенную выше, к части значений пары rdd

### Создание схемы

#### Существует несколько способов создания определения схемы. Быстрый и грязный метод заключается в создании строки со следующим форматом: " <col name> <col type>, <col name> <col type>,<col name> <col type>,…"

#### Создайте строковую переменную и назовите ее схемой. Введите следующее значение для строки:

### Создайте dataframe со схемой

#### Используйте spark.createDataFrame(<rdd>, <schema>) для создания dataframe phoneDF

#### Выведите схему и вызовите show(5, truncate=False), чтобы убедиться, что фрейм данных был создан, как ожидалось

###### 

## Создайте dataframe из источника данных XML

### Импортируйте xml.etree.ElementTree и создайте 3 функции. getPosts() извлечет XML-строки и создаст коллекцию XML-записей. getPostID() примет XML-запись и вернет post\_id в виде текста. getPostLocataion() считает XML-запись и вернет местоположение в виде текста. Местоположение будет состоять из строки широты и долготы, разделенной запятой (","). Используйте следующие функции:

import xml.etree.ElementTree as ET

def getPosts(s):

posts = ET.fromstring(s)

return posts.iter("record")

def getPostID(elem):

return elem.find("post\_id").text

def getPostLocation(elem):

return elem.find("location").text

### Используйте wholeTextFiles() для чтения всех XML-файлов в каталоге /user/student/post\_records/.

### Используйте flatMap для создания строки для каждой записи XML. Передайте часть <content> кортежа (<path>, <content>), полученную из wholeTextFiles, в getPosts(<content>). Напомним, что getPosts возвращает коллекцию XML-записей. flatMap() затем разбивает коллекцию на отдельные элементы и создает новую строку для каждой записи XML.

### Используйте преобразование map() для создания списка формы [<post\_id>, [<latitude>, <longitude>]]. Используйте getPostID для <post\_id>. Используйте getPostLocation для возврата строки, разделенной запятой (","). Используйте функцию String.split(<разделитель>), чтобы разобрать строку в список из 2 элементов широты и долготы.

### 2.5 Используйте преобразование map(), чтобы разбить вложенный список, приведенный выше на простой список, состоящий из [<post\_id>, <latitude>, <longitude>]. Используйте функцию float(), чтобы преобразовать строку latitude и longitude в тип данных с плавающей запятой.

### Создайте схему с помощью StructType() и StructField().

from pyspark.sql.types import \*

schema = StructType([

StructField("post\_id", StringType(), True),

StructField("lat", DoubleType(), True),

StructField("lon", DoubleType(), True)])

### Создайте dataframe со схемой

#### Используйте spark.createDataFrame(<rdd>, <schema>) для создания dataframe latlonDF

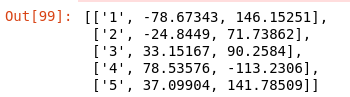
#### Распечатайте схему и вызовите show(5, truncate=False), чтобы убедиться, что фрейм данных был создан, как ожидалось

### Сохраните dataframe latlonDF как управляемую таблицу Hive в базе данных mydb. Назовите таблицу post\_latlon

### Используйте либо Catalog API, либо SparkSession.sql(), чтобы убедиться, что таблица создана

###### 





###### 

###### 

