



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Micro Project – Problem Statement

We have seen a Spark example earlier which gives us the following feel:

1. What do we do in the Spark application?
2. How does a typical Spark application code looks?
3. How to write and execute your Spark application?

Apache Spark is a data processing engine.

In a typical Spark application, we read data from a source, process it, and write or present the results.

And this course is all about learning how to do these three things - Read, Process and Write.

We have seen a sample code of the Spark application, where we learned how to write and execute Spark code.

It's time to start learning Spark Programming and develop your Spark coding skills. We will take a project-driven approach to learning things.

We have a micro project problem statement shown below.

I am giving you the Fire Calls-For-Service dataset.

Problem Statement

Data Set: Fire Department Calls for Service

URL: <https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3>

What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
 1. How many distinct types of calls were made to the fire department?
 2. What are distinct types of calls made to the fire department?
 3. Find out all responses or delayed times greater than 5 mins?
 4. What were the most common call types?
 5. What zip codes accounted for the most common calls?
 6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
 7. What was the sum of all calls, average, min, and max of the call response times?
 8. How many distinct years of data are in the CSV file?
 9. What week of the year in 2018 had the most fire calls?
 10. What neighborhoods in San Francisco had the worst response time in 2018?

Reference: <https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3>

This data set represents a response to calls at fire units. If we go to the above link, we can see the 34 column names available in this data set. You can also see the table preview in this page if you scroll down.

Column Name	Description	Type
Call Number		Plain Text <input type="text"/>
Unit ID		Plain Text <input type="text"/>
Incident Number		Plain Text <input type="text"/>
Call Type		Plain Text <input type="text"/>
Call Date		Date & Time <input type="text"/>
Watch Date		Date & Time <input type="text"/>
Received DtTm		Date & Time <input type="text"/>

[Show All \(34\)](#)

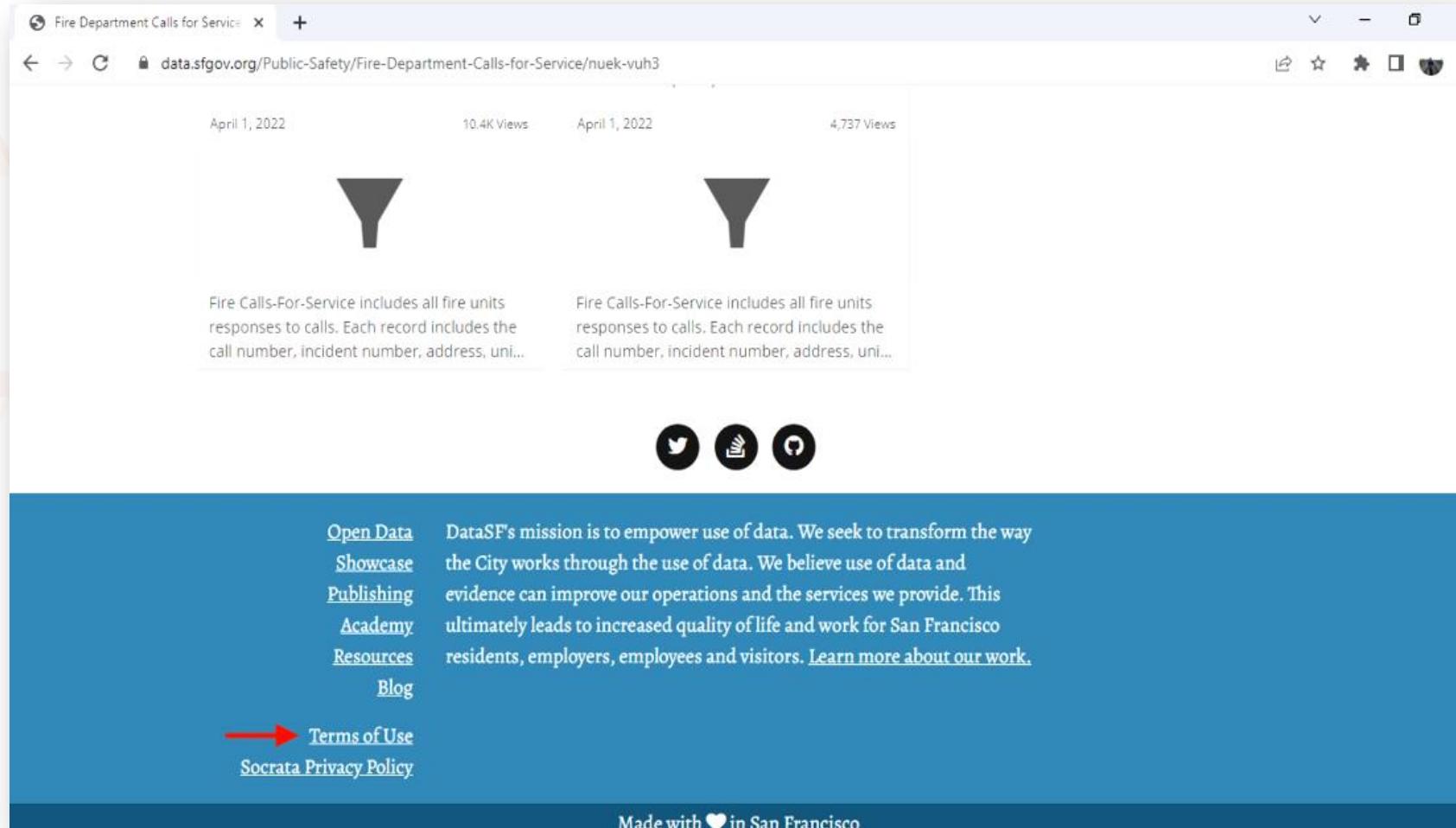
Table Preview

[View Data](#) [Create Visualization](#)

Call ...	Unit ID	Incid...	Call T...	Call ...	Watc...	Recei...	Entry...	Disp...	Resp...	On S...	Tran...	Hos
----------	---------	----------	-----------	----------	---------	----------	----------	---------	---------	---------	---------	-----

Reference: <https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3>

Scroll to the bottom of this page, you will see a link for terms of use, click that.



The data set is available under the PDDL terms. Click the PDDL link you see in the side panel.

A screenshot of a web browser displaying the "Terms of Use" page for DataSF. The page has a dark blue header with the DataSF logo, navigation links for OPEN DATA, SHOWCASE, PUBLISHING, ACADEMY, RESOURCES, and BLOG, and a sign-in button. Below the header, there are links for Explore, Browse Data, Developers, and a search bar. The main content area is titled "Terms of Use" and includes sections for "I. Introduction" and "II. Accepting the Terms of Use". To the right, a sidebar titled "In this article" lists ten sections from I to X. The section "IX. Public Domain Dedication and License (PDDL)" is highlighted with a red border.

Terms of Use

For data.sfgov.org (DataSF)

Last revised April 21, 2017

I. Introduction

As a convenience to potential users, the City and County of San Francisco ("City") makes a variety of datasets ("Data") available for download through this website. Your use of the Data is subject to these terms of use, which constitute a legal agreement between You and the City and County of San Francisco ("City"). This legal agreement is referred to as the "Terms of Use."

II. Accepting the Terms of Use

A. Means of Acceptance. In order to use any of the Data, You must agree to these Terms of Use. You agree to the Terms of Use by either: (1) Clicking to accept the Terms of Use; or (2) Downloading or using any of the Data or any Derivative Work, in which case you understand and agree that the City will treat your download or use of the Data or a Derivative Work as an acceptance of the Terms of Use from that

In this article

- I. Introduction
- II. Accepting the Terms of Use
- III. Definitions
- IV. City's Intellectual Property Rights Not Affected
- V. Exclusion of Warranties
- VI. Confidentiality
- VII. Limitation of Liability and Indemnity
- VIII. Acceptance of Other Conditions
- IX. Public Domain Dedication and License (PDDL)
- X. General Provisions

You will see the open data commons URL. Copy the URL and paste it into a new browser tab.

The screenshot shows a web browser window displaying the DataSF Terms of Use page. The URL in the address bar is datasf.org/opendata/terms-of-use/#toc8. The page header includes the DataSF logo, navigation links for OPEN DATA, SHOWCASE, PUBLISHING, ACADEMY, RESOURCES, and BLOG, and a sign-in button. The main content area features a large heading "IX. Public Domain Dedication and License (PDDL)" in blue. Below it, a paragraph states: "Except where otherwise stated in the file containing such Data or on the page from which such Data is accessed, including its metadata, Data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at <http://www.opendatacommons.org/licenses/pddl/1.0/>". To the right, a sidebar titled "In this article" lists ten sections, with "IX. Public Domain Dedication and License (PDDL)" being the current section. The sidebar also includes a "Sign In" button. The page footer contains links for SFGov, Coordinator's Portal, About, Join Us, and Help.

IX. Public Domain Dedication and License (PDDL)

Except where otherwise stated in the file containing such Data or on the page from which such Data is accessed, including its metadata, Data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at <http://www.opendatacommons.org/licenses/pddl/1.0/>

X. General Provisions

A. These Terms of Use shall be governed by and interpreted under the laws of the State of California without regard to conflict of laws provisions. Any dispute arising out of these Terms of Use shall be subject to the exclusive venue of the state and federal courts within the Northern District of California, and You and the City hereby consent to the venue and jurisdiction of such courts.

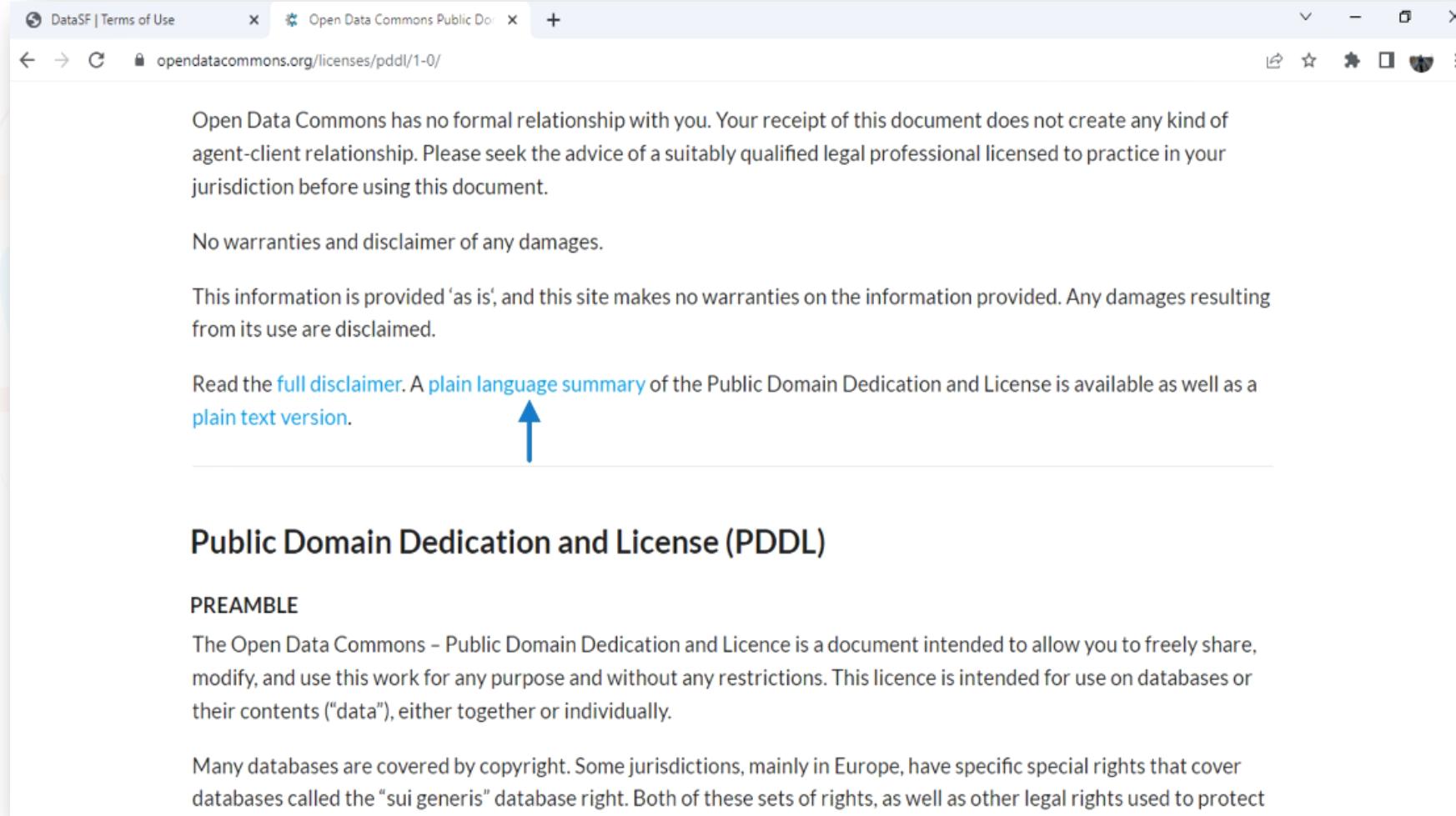
B. No modification to these Terms of Use, nor any waiver of any rights, shall be effective except by an instrument in writing signed by You and the City, and the waiver of any breach or default shall not constitute a waiver of any other right hereunder or any subsequent breach or default.

C. These Terms of Use contain the entire agreement and understanding between You and the City with respect to the subject matter hereof and completely replace and supersede all prior agreements, understandings and representations. In no event will any additional terms or conditions be effective unless expressly accepted by the City in writing.

In this article

- I. Introduction
- II. Accepting the Terms of Use
- III. Definitions
- IV. City's Intellectual Property Rights Not Affected
- V. Exclusion of Warranties
- VI. Confidentiality
- VII. Limitation of Liability and Indemnity
- VIII. Acceptance of Other Conditions
- IX. Public Domain Dedication and License (PDDL)**
- X. General Provisions

If you go to the open data commons URL, and scroll down, you will see a plain-language summary. Click that.



Open Data Commons has no formal relationship with you. Your receipt of this document does not create any kind of agent-client relationship. Please seek the advice of a suitably qualified legal professional licensed to practice in your jurisdiction before using this document.

No warranties and disclaimer of any damages.

This information is provided 'as is', and this site makes no warranties on the information provided. Any damages resulting from its use are disclaimed.

Read the [full disclaimer](#). A [plain language summary](#) of the Public Domain Dedication and License is available as well as a [plain text version](#).

Public Domain Dedication and License (PDDL)

PREAMBLE

The Open Data Commons – Public Domain Dedication and Licence is a document intended to allow you to freely share, modify, and use this work for any purpose and without any restrictions. This licence is intended for use on databases or their contents ("data"), either together or individually.

Many databases are covered by copyright. Some jurisdictions, mainly in Europe, have specific special rights that cover databases called the "sui generis" database right. Both of these sets of rights, as well as other legal rights used to protect

So you are free to share, create and adapt this data set. It is essential to check the terms of use for any data set that you are using for your learning and experiments.
I showed you how to check the license terms for the Fire Calls-For-Service dataset.

The screenshot shows a web browser window with the title bar "DataSF | Terms of Use" and "Open Data Commons Public Do". The address bar displays "opendatacommons.org/licenses/pddl/summary/". The main content area is titled "Open Data Commons Public Domain Dedication and License (PDDL) Summary". It states: "This is a human-readable summary of the [Public Domain Dedication and License 1.0](#). Please see the disclaimer below." Below this, under "You are free:", there is a list:

- *To share*: To copy, distribute and use the database.
- *To create*: To produce works from the database.
- *To adapt*: To modify, transform and build upon the database.

 This list is enclosed in a red-bordered box. Under "As long as you:", there is a single item:

- *Blank*: This section is intentionally left blank. The PDDL imposes no restrictions on your use of the PDDL licensed database.

 Below this, a section titled "Disclaimer" is present with the text: "This is not a license. It is simply a handy reference for understanding the [PDDL 1.0](#) – it is a human-readable expression of some of its key terms. This document has no legal value, and its contents do not appear in the actual license. Read the [full PDDL 1.0 license text](#) for the exact terms that apply."

You are given a San Francisco fire call response data set. And you are asked to fulfil the following set of requirements

Problem Statement

Data Set: Fire Department Calls for Service

URL: <https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3>

What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
 1. How many distinct types of calls were made to the fire department?
 2. What are distinct types of calls made to the fire department?
 3. Find out all responses or delayed times greater than 5 mins?
 4. What were the most common call types?
 5. What zip codes accounted for the most common calls?
 6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
 7. What was the sum of all calls, average, min, and max of the call response times?
 8. How many distinct years of data are in the CSV file?
 9. What week of the year in 2018 had the most fire calls?
 10. What neighborhoods in San Francisco had the worst response time in 2018?

We want to do only two things:

1. Load the data into a Spark Data frame
2. Query it to find the answers to the ten questions shown earlier.

The main objective of the Micro Project is to learn two things:

1. How to load data into Spark
2. How to query data in Spark.

We have two approaches to do it:

1. Using Spark SQL
2. Using Spark Dataframe API

While we work on the solution of the given 10 questions, we will learn the following concepts:

1. Using Spark SQL and DDL statements
2. Spark Data Frame and Data Frame Reader API
3. Working with CSV Files
4. Data Frame Transformations and Actions
5. How to read Spark API Documentation



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Spark Dataframes - Introduction

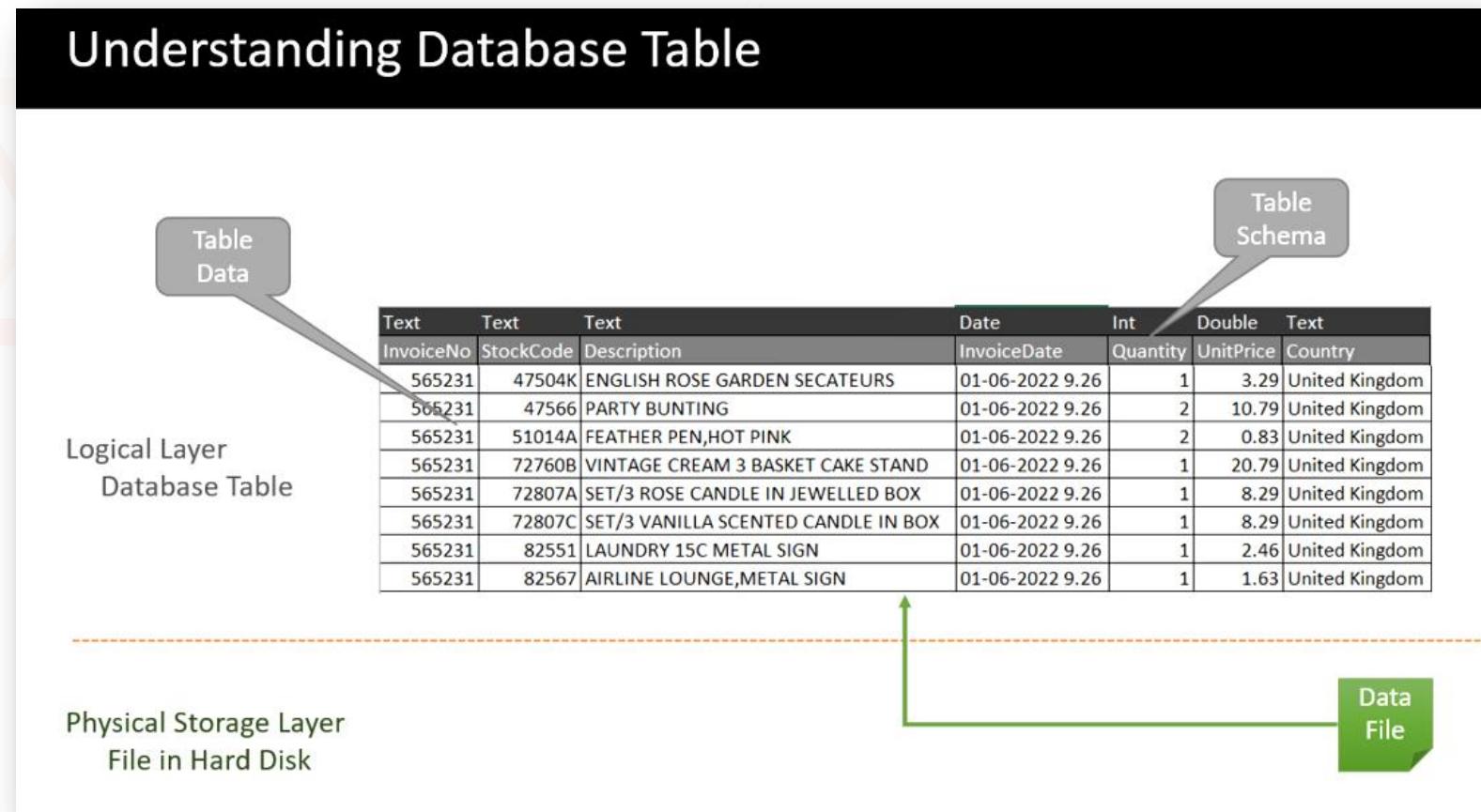
We learned that Spark is a data processing platform.
We also know that Databases are the most popular and
most widely used data processing platforms.
They offer two things at a high level.

1. Tables
2. SQL

A database table allows you to load the data in the table. The data in a table is internally stored as a .dbf file, which are stored on the disk. But we don't care about this dbf file and storage layer, we always look at the table, and query the table.

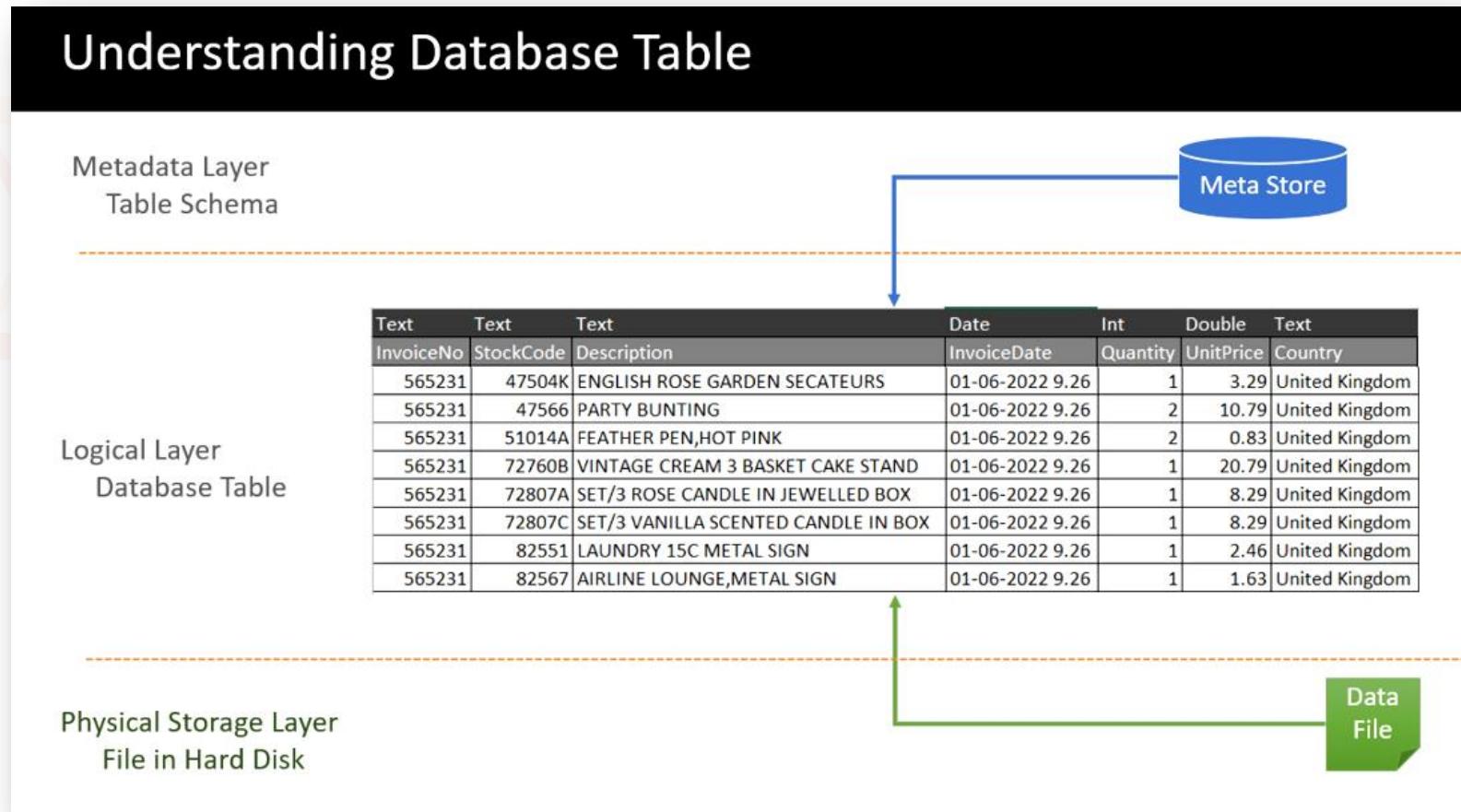
A table contains two things:

1. Table Schema – It is a list of column names and data types.
2. Table Data – It is the data stored inside the table.



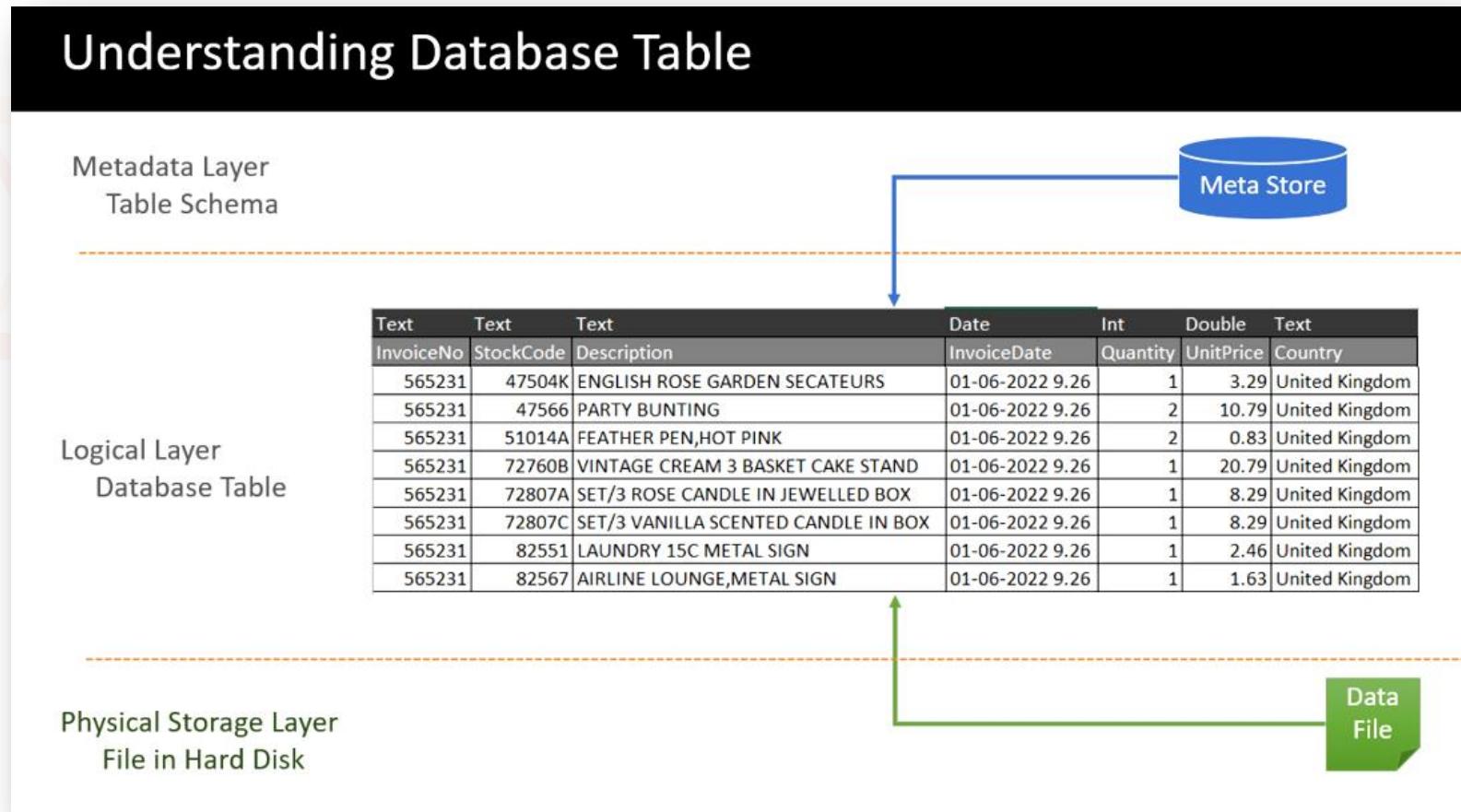
We have three layers to form a table:

1. Storage layer stores the table data in a file
2. Metadata layer stores the table schema and other important information
3. The Logical layer presents you with a database table, and you can execute SQL queries on the logical table.

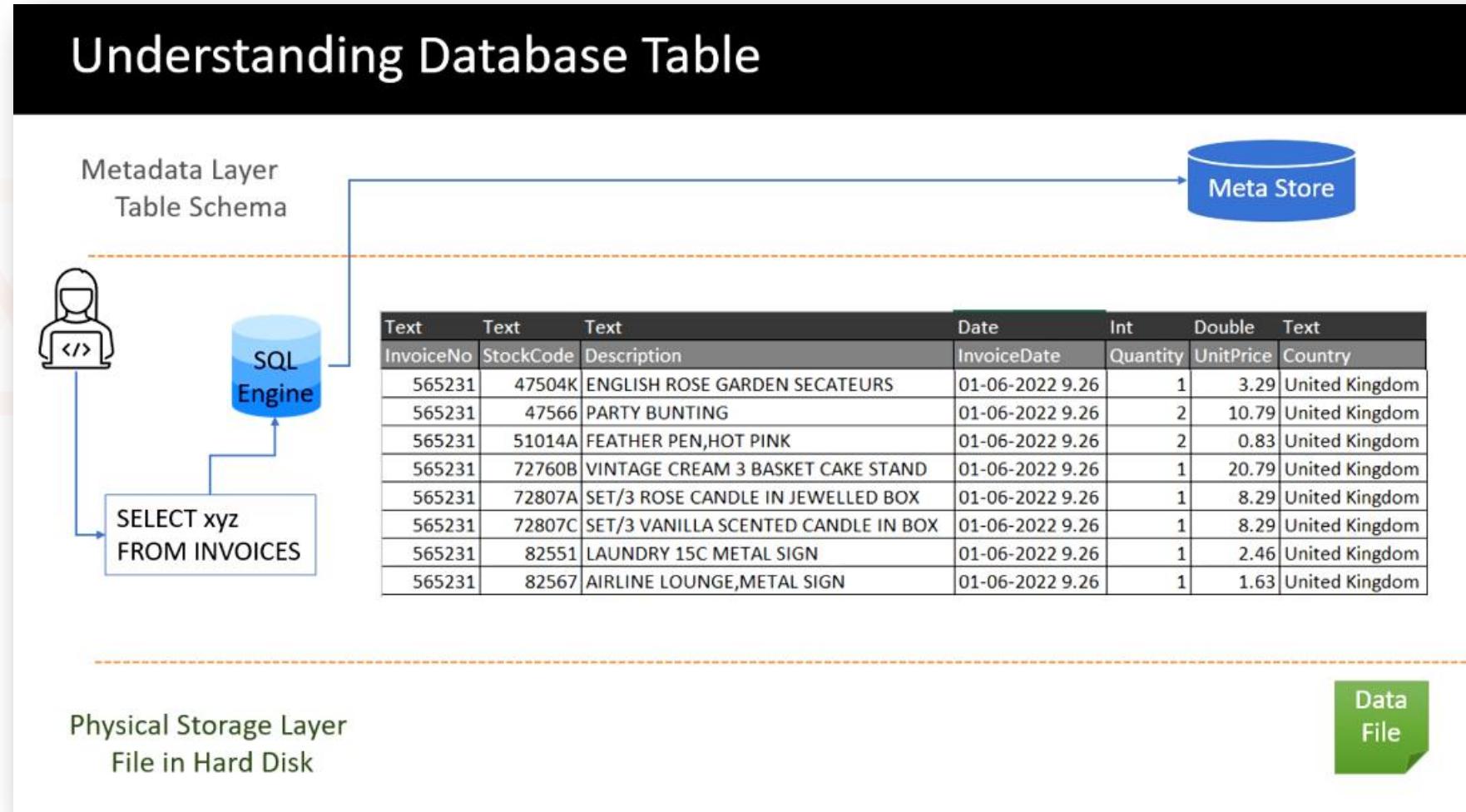


We have three layers to form a table:

1. Storage layer stores the table data in a file
2. Metadata layer stores the table schema and other important information
3. The Logical layer presents you with a database table, and you can execute SQL queries on the logical table.

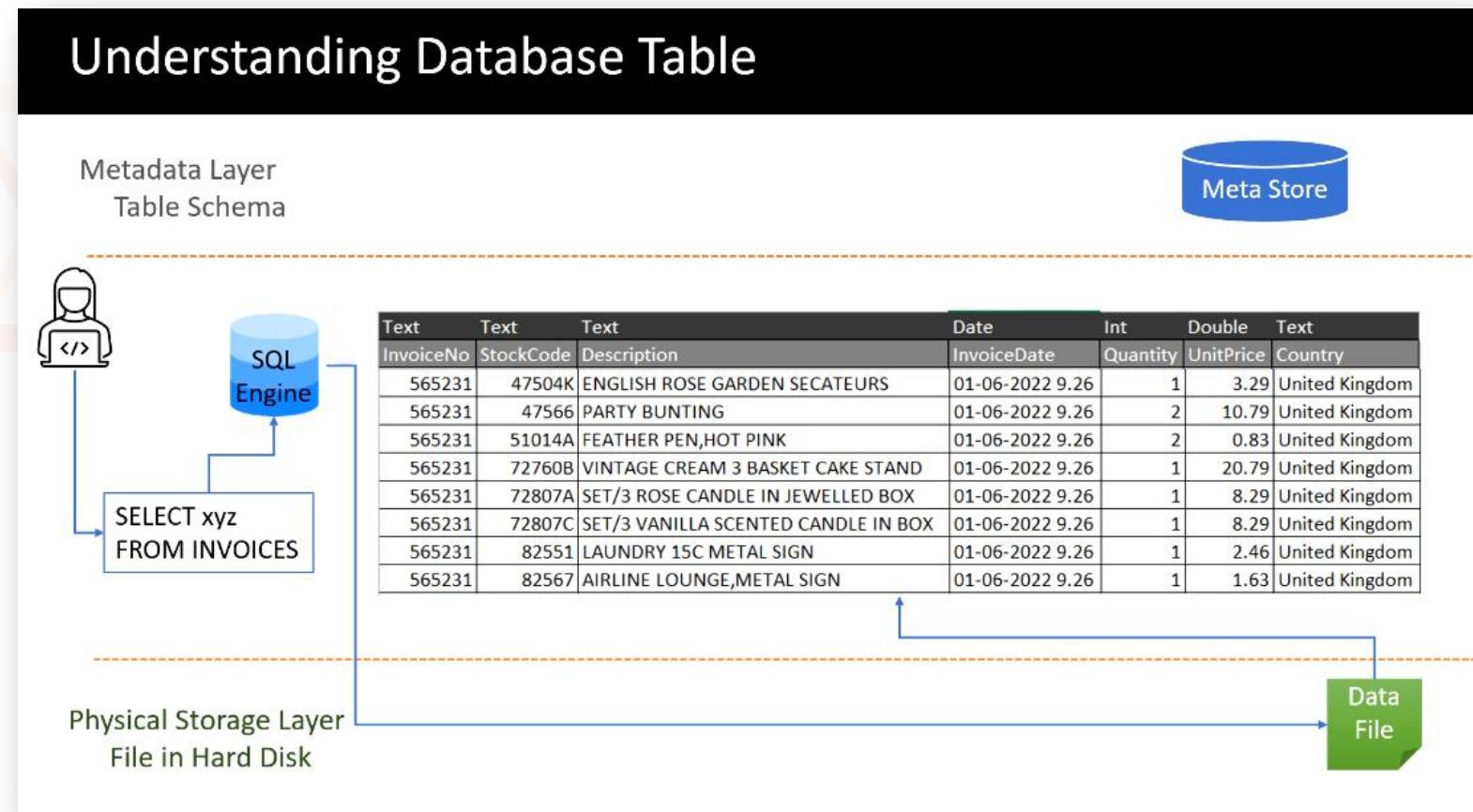


When you submit a SQL query to your database SQL Engine, the database will refer to the metadata store for parsing your SQL queries. And the database will throw a syntax error or an analysis error if you are using a column name in your SQL that does not exist in the metadata store.



The schema is essential for the database table and for the SQL expressions to work correctly. The table data is stored in the dbf files behind the table. So if your SQL query is correct and passes all the schema validation, you will see query results.

The database will read data from the "dbf" file, process it according to your SQL query, and show you the results. And that is all about the data processing in databases.



Apache Spark offers you two ways of data processing:

1. Spark Database and SQL
2. Spark Dataframe and Dataframe API

The first approach is precisely the same as a typical database. You will create table and load data into the table. Spark table data is internally stored in the data files. But these files are not dbf files like in the case of a database.

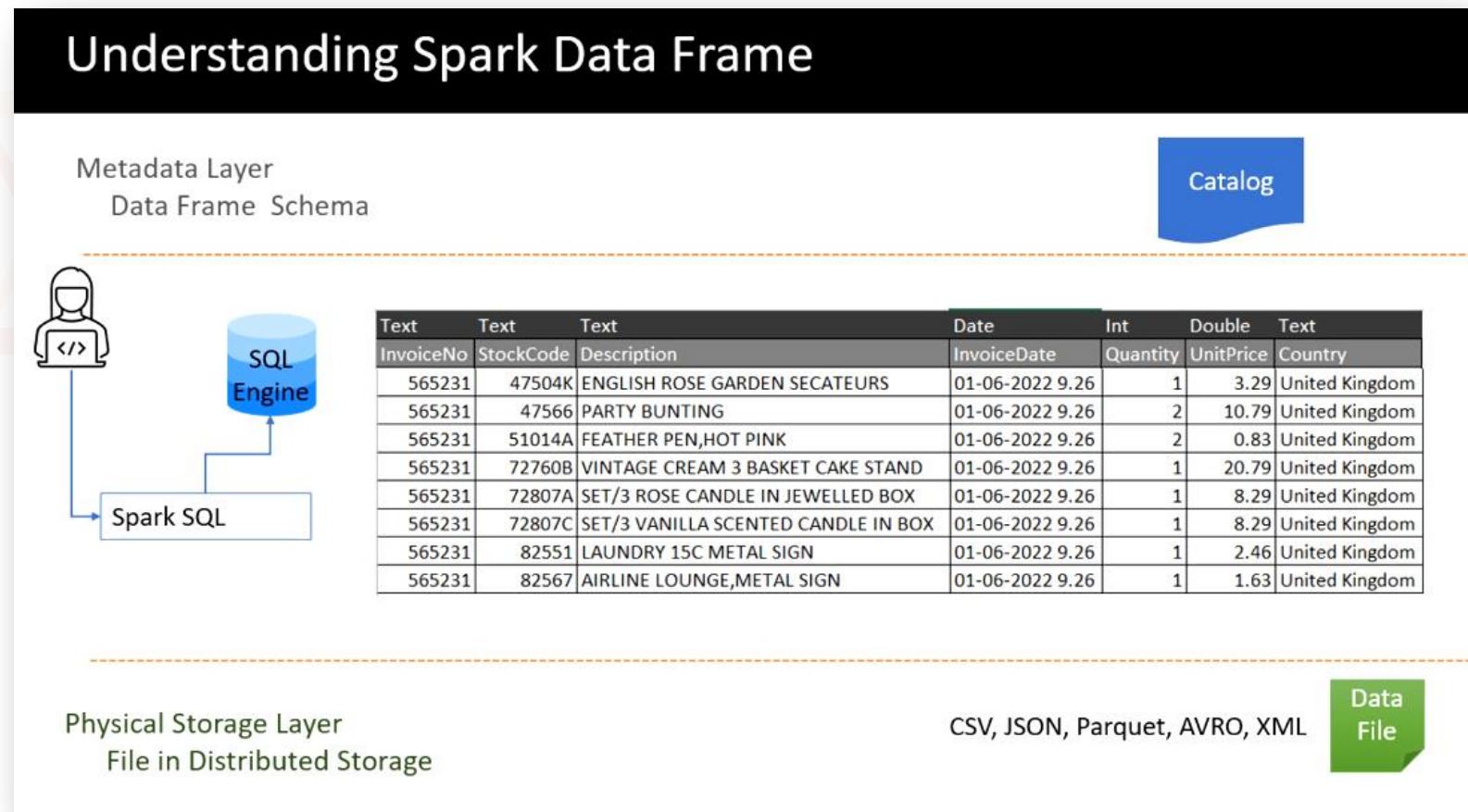
Spark gives you the flexibility to choose the file format and supports many file formats such as CSV, JSON, Parquet, AVRO and XML.

The DBF file format and database engine were designed to support structured data. However, Spark supports structured, semi-structured, and unstructured data.

The spark storage layer also supports distributed storage such as HDFS and Cloud storage such as Amazon S3 and Azure ADLS. So you are not limited to disk storage capacity. You can use distributed storage and store large data files. Spark also has a metadata store for storing table schema information. So that part is similar to the databases.

Then Spark also comes with an SQL query engine and supports standard SQL syntax for processing and querying data from Spark tables.

The second approach to data processing in Apache Spark is **Spark Dataframe and Dataframe API**. Spark Dataframe is structurally the same as the table. However, it does not store any schema information in the metadata store. Instead, we have a runtime metadata catalog to store the Dataframe schema information. It is similar to the metadata store, but Spark will create it at the runtime to store schema information in the catalog.



We have two reasons for storing schema information in the catalog.

1. Stark Dataframe is a runtime object – You can create a Spark Data frame at runtime and keep it in memory until your program terminates. Once your program terminates, your Dataframe is gone. So, it is an in-memory object.
2. Spark Dataframe supports schema-on-read – Dataframe does not have a fixed and predefined schema stored in the metadata store. We load the data into a Dataframe and tell the schema when loading the data. And Spark will read the file, apply the schema at the time of reading, create the Dataframe using the schema and load the data.

We have two reasons for storing schema information in the catalog.

1. Stark Dataframe is a runtime object – You can create a Spark Data frame at runtime and keep it in memory until your program terminates. Once your program terminates, your Dataframe is gone. So, it is an in-memory object.
2. Spark Dataframe supports schema-on-read – Dataframe does not have a fixed and predefined schema stored in the metadata store. We load the data into a Dataframe and tell the schema when loading the data. And Spark will read the file, apply the schema at the time of reading, create the Dataframe using the schema and load the data.

Here is a quick summary discussing the differences between Spark Tables and Dataframes.

Spark Table Vs Data Frame

Spark Table

1. Tables store schema information in metadata store
2. Table and metadata are persistent objects and visible across applications
3. We create tables with a predefined table schema
4. Table supports SQL Expressions and does not support API

Spark Data Frame

1. Data Frame stores schema information in runtime Catalog
2. Data Frame and Catalog are runtime objects and live only during the application runtime. Data Frame is visible to your application only.
3. Data Frame supports schema-on-read
4. Data Frame offers APIs and does not support SQL expressions

Spark Table and a Data Frame are convertible objects



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Creating Spark Dataframe

You have already seen the requirement for creating a Dataframe using the San Francisco fire call response data set. The data set is already available in the Databricks community cloud, and it is made available for you by the Databricks for learning purposes. You can access it from the following location in your Databricks community cloud. There is no need to download the data from the source and bring it to the cloud. That part is already done for you.

Problem Statement

Data Set: Fire Department Calls for Service

Location: </databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv>

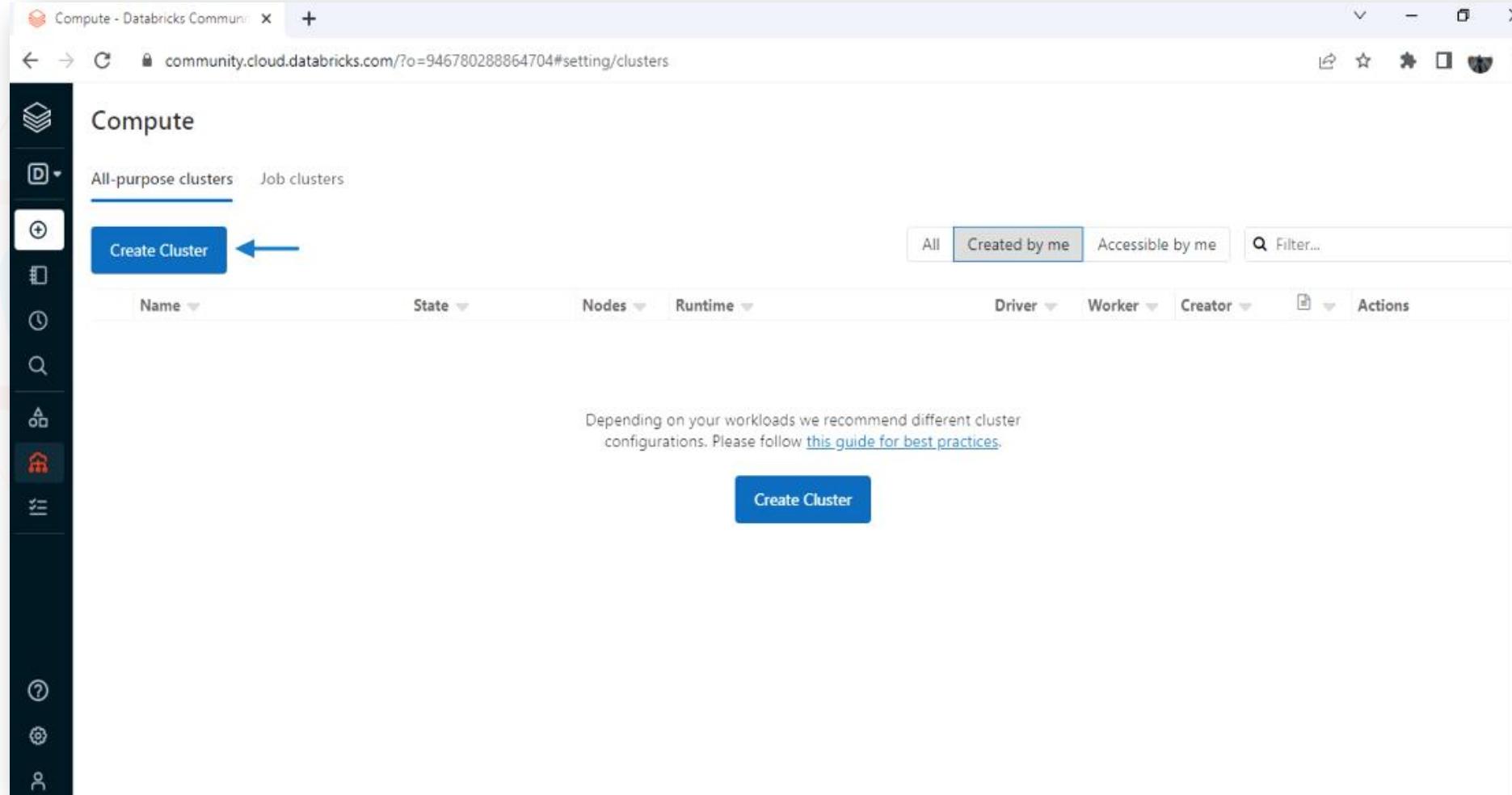
What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
 1. How many distinct types of calls were made to the fire department?
 2. What are distinct types of calls made to the fire department?
 3. Find out all responses or delayed times greater than 5 mins?
 4. What were the most common call types?
 5. What zip codes accounted for the most common calls?
 6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
 7. What was the sum of all calls, average, min, and max of the call response times?
 8. How many distinct years of data are in the CSV file?
 9. What week of the year in 2018 had the most fire calls?
 10. What neighborhoods in San Francisco had the worst response time in 2018?

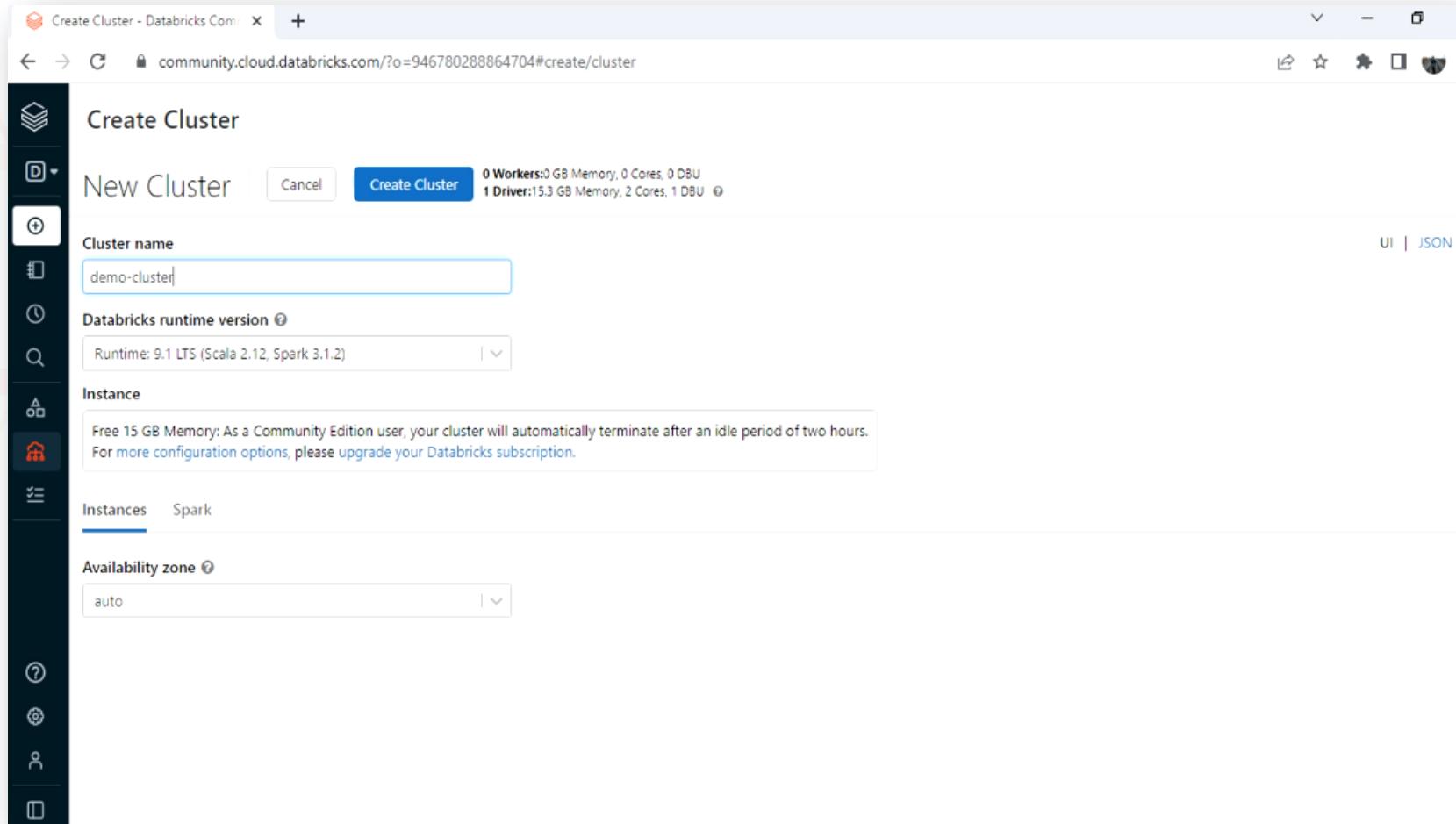
Start your Databricks Workspace.

The screenshot shows the Databricks Community Edition workspace. The left sidebar has a dark theme with icons for Notebook, Data import, Partner Connect, Guide: Quickstart tutorial, Recents, Documentation, and Best practices. The main area is titled "Data Science & Engineering". It features three cards: "Notebook" (Create a new notebook for querying, data processing, and machine learning), "Data import" (Quickly import data, preview its schema, create a table, and query it in a notebook), and "Partner Connect" (Fivetran, dbt, Tableau, Power BI). Below these are sections for "Recents" (listing "01-getting-started" last viewed "a day ago"), "Documentation" (Get started guide, This tutorial gets you going with Databricks Data Science & Engineering; Best practices, Get the best performance when using Databricks), "Release notes" (Runtime release notes, Databricks preview releases, Platform release notes, More release notes), and "Blog posts" (Implementing the GDPR 'Right to be Forgotten' in Delta Lake, March 23, 2022; Structured Streaming: A Year in Review, February 7, 2022).

Go to the Compute menu and create a new cluster.



Give a name to your cluster and create it. Your cluster will start in a few minutes, and Databricks will terminate it if your cluster is idle for 2 hours. Once terminated, you cannot start it again. So if you want to work the next day, you should delete the older terminated cluster and create a new one. You can use your cluster for as long as you want and delete it when you are done.

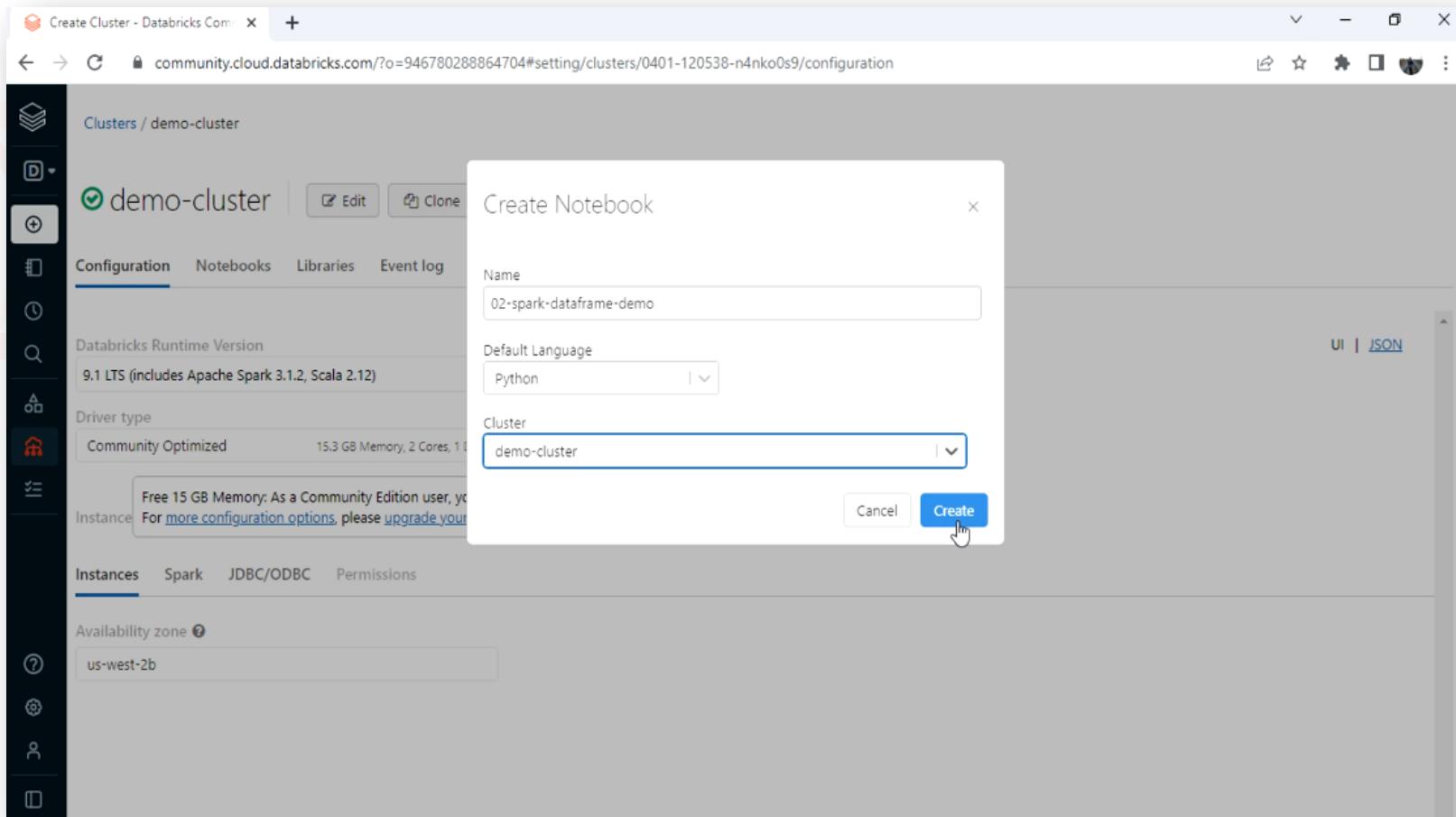


Once your cluster is up and running, you can go ahead to the “*create*” menu options and create a new notebook.

Give a name to the notebook. Every notebook should have a default language.

You have four language options: Python, Scala, SQL and R.

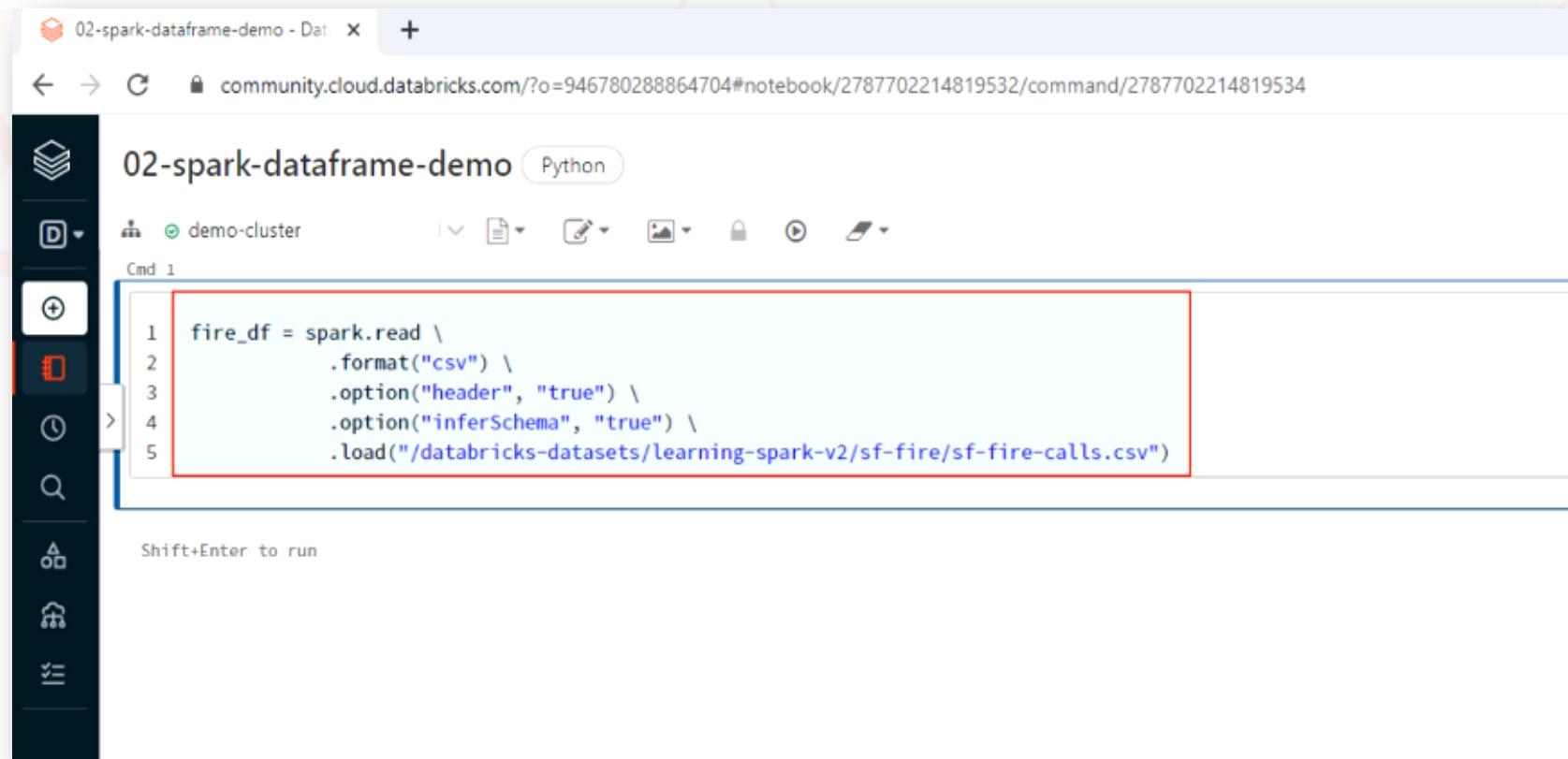
We want Python for Spark programming, so keep the default language Python. Your notebook will not run without a cluster. So you should also attach your cluster with the notebook. And finally, hit the create button.



Once the notebook is created, you can write the code to create a Spark Dataframe using the San Francisco fire call response data set.

And here we have the code for the same shown below. **Reference: (02-spark-dataframe-demo.ipynb)**

I am using spark.read which is a Spark Session object. It is the entry point for the Spark programming APIs. The rest of the code, such as format, option, and load, are the methods of the DataframeReader. I am telling the DataframeReader that the data file is a CSV file. Then I am telling that the CSV file comes with a header row. I am also saying that the DataframeReader should infer schema from the file itself. Finally, I call the load method so the DataframeReader can load the file from the given location. The code will return a dataframe, and I am assigning the dataframe to the fire_df variable.



```
fire_df = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

Visit <https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html> to go to the Spark SQL documentation. You will see that all the things listed in this page belongs to the 14 objects in the pyspark.sql package which are listed in the image below. Each object offers some attributes and methods.

The screenshot shows a web browser displaying the Apache Spark API Reference for PySpark 3.2.1. The URL in the address bar is <https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html>. The page title is "Spark SQL". On the left, there is a sidebar titled "Spark SQL" with a list of classes: `pyspark.sql.SparkSession`, `pyspark.sql.Catalog`, `pyspark.sql.DataFrame`, `pyspark.sql.Column`, `pyspark.sql.Row`, `pyspark.sql.GroupedData`, `pyspark.sql.PandasCogroupedOps`, `pyspark.sql.DataFrameNaFunction`, `pyspark.sql.DataFrameStatFunction`, `pyspark.sql.Window`, `pyspark.sql.SparkSession.builder`, `pyspark.sql.SparkSession.builder.c`, `pyspark.sql.SparkSession.builder.e`, and `pyspark.sql.SparkSession.builder.g`. The main content area is titled "Core Classes" and lists several classes under "SparkSession": `SparkSession(sparkContext[, jsparkSession])`, `Catalog(sparkSession)`, `DataFrame(jdf, sql_ctx)`, `Column(jC)`, `Row`, and `GroupedData(jgd, df)`. A red box highlights a list of 14 objects:

1. `pyspark.sql.SparkSession`
2. `pyspark.sql.DataFrame`
3. `pyspark.sql.DataFrameReader`
4. `pyspark.sql.DataFrameWriter`
5. `pyspark.sql.Row`
6. `pyspark.sql.Column`
7. `pyspark.sql.types`
8. `pyspark.sql.GroupedData`
9. `pyspark.sql.Window`
10. `pyspark.sql.Catalog`
11. `pyspark.sql.conf`
12. `pyspark.sql.functions`
13. `pyspark.sql.DataFrameNaFunctions`
14. `pyspark.sql.DataFrameStatFunctions`

. Below this list, a tooltip for `Column(jC)` states: "A column in a DataFrame.". A tooltip for `Row` states: "A row in DataFrame.". A tooltip for `GroupedData(jgd, df)` states: "A set of methods for aggregations on a DataFrame, created by DataFrame.groupBy()". To the right of the main content, there is a sidebar titled "On this page" with links to "Classes", "Session APIs", "Configuration", "Input and Output", "DataFrame APIs", "Column APIs", "Types", "Window Functions", "Grouping", and "Catalog APIs".

Visit <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html> to go to the DataFrameReader object under the Spark SQL documentation. The DataFrameReader offers eleven methods as shown in the screenshot below.

The screenshot shows a browser window displaying the Apache Spark API Reference for the `pyspark.sql.DataFrameReader` class. The URL in the address bar is <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrameReader.html>. The page has a dark blue header with the Apache logo and navigation links for Getting Started, User Guide, API Reference (which is highlighted), Development, and Migration Guide. A sidebar on the left lists various methods and attributes, such as `version`, `udf`, `table`, `bucketBy`, `csv`, `json`, `parquet`, `orc`, `jdbc`, `table`, `option`, `options`, `schema`, `load`, `format`, `read`, `readStream`, `catalog`, `conf`, `sparkContext`, `streams`, `builder`, and `version`. The `Attributes` section contains detailed descriptions for each. A red box highlights the eleven methods listed under the `read` attribute:

1. `DataFrameReader.csv`
2. `DataFrameReader.json`
3. `DataFrameReader.parquet`
4. `DataFrameReader.orc`
5. `DataFrameReader.jdbc`
6. `DataFrameReader.table`
7. `DataFrameReader.format`
8. `DataFrameReader.option`
9. `DataFrameReader.options`
10. `DataFrameReader.schema`
11. `DataFrameReader.load`

We can also create the same Dataframe using CSV method as shown in the second cell of the screenshot. In the first cell, I am using the format method to tell that I want to load a CSV file and the option method to set other configurations.

In the second cell, we are using the shortcut CSV method. I am not setting the format and options.

The CSV method implicitly tells that the format is CSV, and all other details will go as function arguments. So, both the code are doing the same thing, but I prefer the first one as this code is standardized for loading data from any file format.

The screenshot shows a Databricks notebook interface with the title "02-spark-dataframe-demo" and a Python tab selected. The notebook has two command cells:

Cmd 1:

```
1 fire_df = spark.read \
2     .format("csv") \
3     .option("header", "true") \
4     .option("inferSchema", "true") \
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

Cmd 2:

```
1 fire_df = spark.read \
2     .csv("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv",
3           header="true",
4           inferSchema="true")|
```

A blue box highlights the code in Cmd 2. At the bottom of the notebook, there is a "Shift+Enter to run" button.

I created a DataFrame, and now I want to see some records from this dataframe. You can use the show method. I take this data frame variable and call the Dataframe show method. I want to see ten records, so I use show(10). And you can see the output below the code cell. But it doesn't look nice and easy to read.

The screenshot shows a Jupyter Notebook interface with a single cell containing the Python command `fire_df.show(10)`. The output of this command is a table displaying 10 rows of data from a DataFrame. The columns are labeled as follows:

Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Address	City	Zipcode of		
Incident	Battalion	Station Area	Box	OrigPriority	Priority	Final Priority	ALS Unit	Call Type Group	NumAlarms	UnitType	Unit sequence in call dispatch	
Prevention District	Supervisor District	Neighborhood	Location	RowID	Delay							
20110014	M29	94103	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:...	10TH ST/MARKET ST	SF		
	B02	802	36 2338	1	1	2	true	null	1	MEDIC	1	
2		6	Tenderloin (37.7765408927183...	020110014-M29	5.233333333333333							
3	20110015	M08	94107	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:...	300 Block of 5TH ST	SF	
	B03	803	08 2243	1	1	2	true	null	1	MEDIC	1	
4		6	South of Market (37.7792841462441...	020110015-M08	3.0833333333333335							
5	20110016	B02	94109	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:...	2000 Block of CAL...	SF	
	B04	804	38 3362	3	3	3	false	null	1	CHIEF	6	
6		5	Pacific Heights (37.7895840679362...	020110016-B02	3.05							
7	20110016	B04	94109	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:...	2000 Block of CAL...	SF	
	B04	804	38 3362	3	3	3	false	null	1	CHIEF	3	
8		5	Pacific Heights (37.7895840679362...	020110016-B04	2.3166666666666667							

At the bottom of the cell, the message "Command took 1.11 seconds -- by prashant@scholarnest.com at 4/1/2022, 8:18:39 PM on demo-cluster" is displayed.

So, you can use `display()` function. And now the output looks much better and easy to read. The `display` function is not part of Apache Spark. It is an additional tool by Databricks to format the output of a DataFrame. So it takes a DataFrame and shows the content of the data frame.

The screenshot shows a Databricks notebook interface with the title "02-spark-dataframe-demo" and the language "Python". The notebook contains a single command:

```
1 display(fire_df)
```

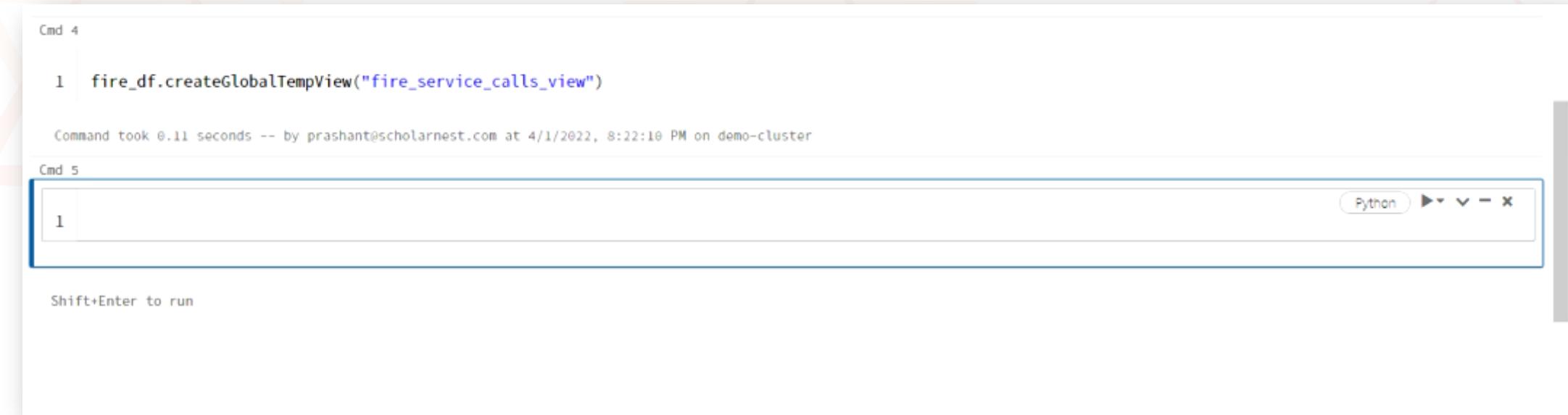
When executed, it displays a table with the following data:

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	200C

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 0.72 seconds -- by prashant@scholarnest.com at 4/1/2022, 8:20:12 PM on demo-cluster

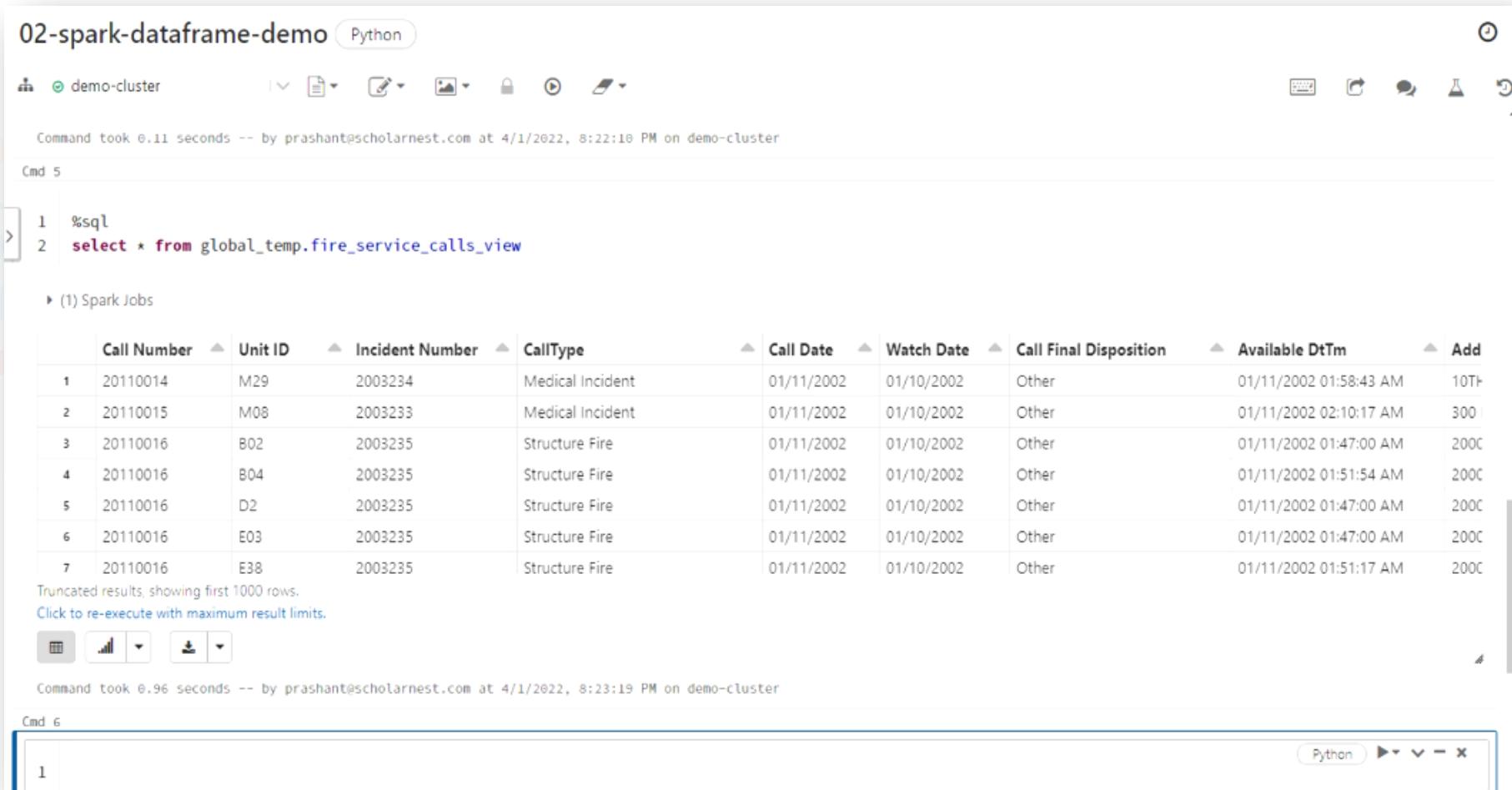
In the code shown below, I am taking my dataframe and creating a global-temporary view for the dataframe. The view name is *fire_service_calls_view*. I am not converting my Dataframe to a table. Instead, I am creating a global-temporary view that is similar to the table.



The screenshot shows a Jupyter Notebook interface with two command cells:

- Cmd 4:** Contains the Python code `fire_df.createGlobalTempView("fire_service_calls_view")`. Below the code, the output shows the command took 0.11 seconds to run, executed by prashant@scholarnest.com at 4/1/2022, 8:22:10 PM on demo-cluster.
- Cmd 5:** A new command cell is being typed, starting with the number 1. The cell has a "Python" tab selected and includes standard Jupyter notebook control icons (run, stop, etc.). A tooltip "Shift+Enter to run" is visible at the bottom left of the cell.

You can run SQL expression on the global-temporary view. But you need to change the cell type from Python to SQL. The `global_temp` is a hidden database, and all global temporary tables are created in this `global_temp` database. So you must use `global_temp` for accessing a global-temporary view.



The screenshot shows a Databricks notebook titled "02-spark-dataframe-demo" running in Python. The notebook contains two commands:

```
1 %sql
2 select * from global_temp.fire_service_calls_view
```

Command 5 displays the results of the SQL query as a Spark DataFrame:

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtM	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	200C

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command 6 is currently empty, showing only the number 1.



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Creating Spark Tables

Start your Databricks Workspace.

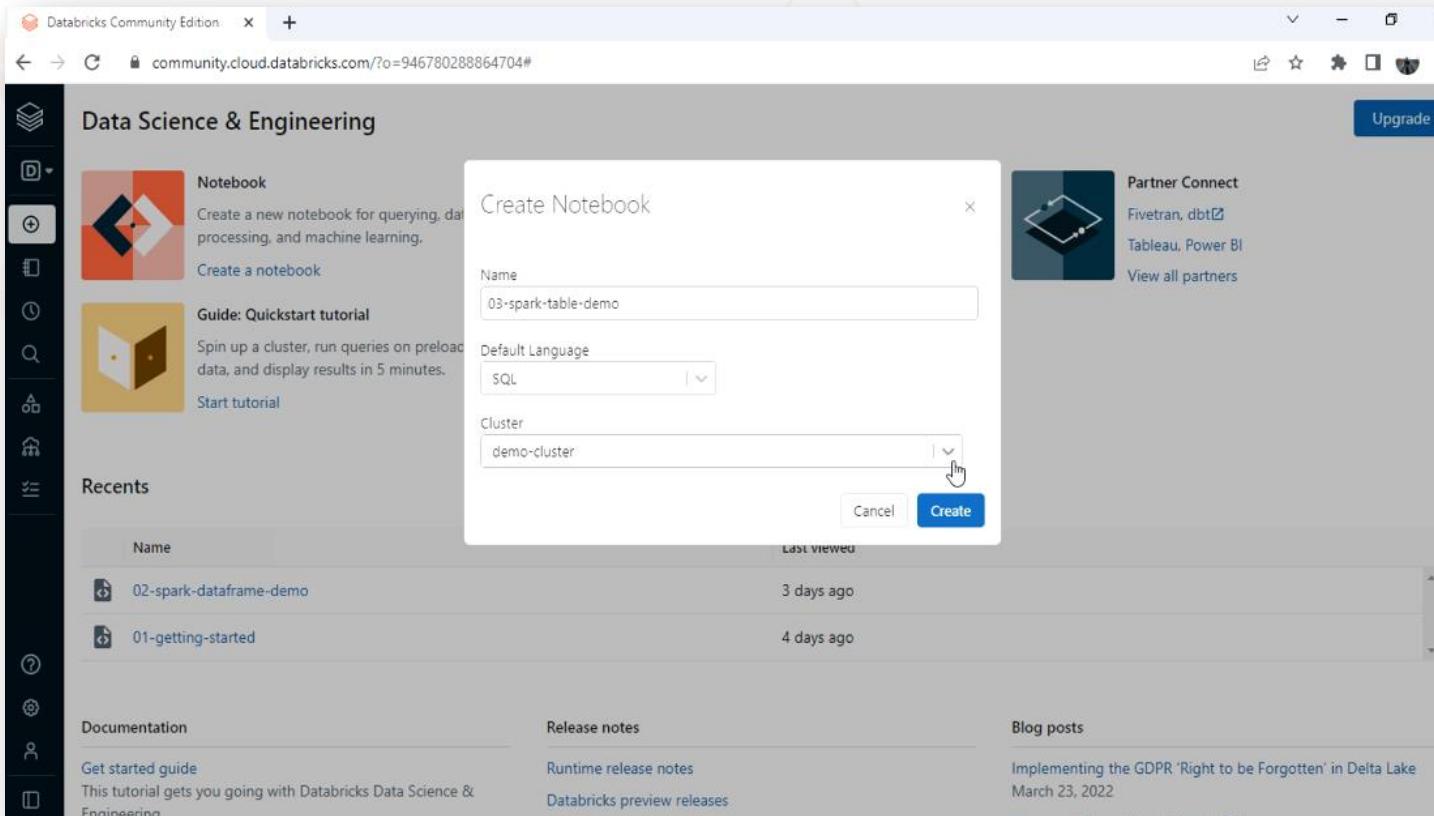
The screenshot shows the Databricks Community Edition workspace. The left sidebar has a dark theme with icons for Notebook, Data import, Partner Connect, Guide: Quickstart tutorial, Recents, Documentation, and Best practices. The main area is titled "Data Science & Engineering". It features three cards: "Notebook" (Create a new notebook for querying, data processing, and machine learning), "Data import" (Quickly import data, preview its schema, create a table, and query it in a notebook), and "Partner Connect" (Fivetran, dbt, Tableau, Power BI). Below these are sections for "Recents" (listing "01-getting-started" last viewed "a day ago"), "Documentation" (Get started guide, This tutorial gets you going with Databricks Data Science & Engineering; Best practices, Get the best performance when using Databricks), "Release notes" (Runtime release notes, Databricks preview releases, Platform release notes, More release notes), and "Blog posts" (Implementing the GDPR 'Right to be Forgotten' in Delta Lake, March 23, 2022; Structured Streaming: A Year in Review, February 7, 2022).

Create a new notebook. (**Reference: 03-spark-table-demo.sql**)

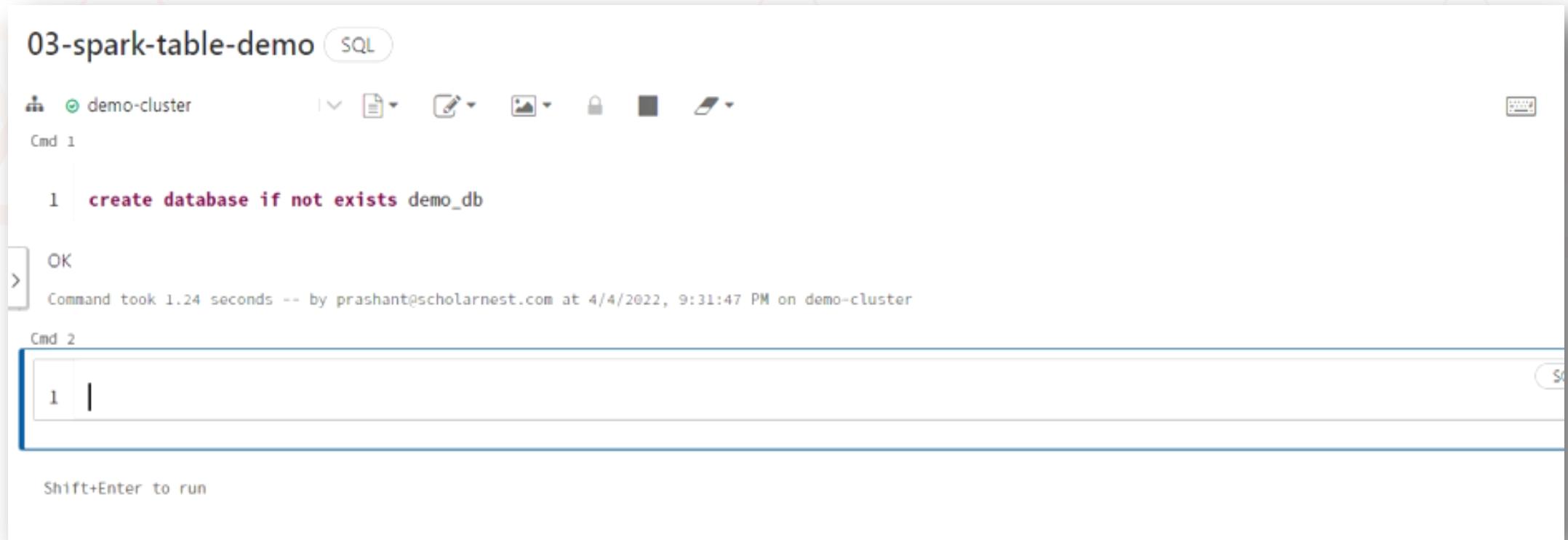
Give a name to your notebook.

Choose your notebook's default language. This time, we want to use Spark SQL, so the default language for the notebook should be SQL.

Attach a cluster to your notebook. If you do not have a running cluster, go back to the Compute menu and create a new cluster.



Create a database as shown below.
The code shown here is purely a SQL DDL Statement. A database should have been created.



The screenshot shows a Jupyter Notebook interface with a single cell containing the following SQL command:

```
1  create database if not exists demo_db
```

The cell output shows the command was successful:

```
OK
```

Below the output, a timestamp indicates the command took 1.24 seconds to run on a demo cluster.

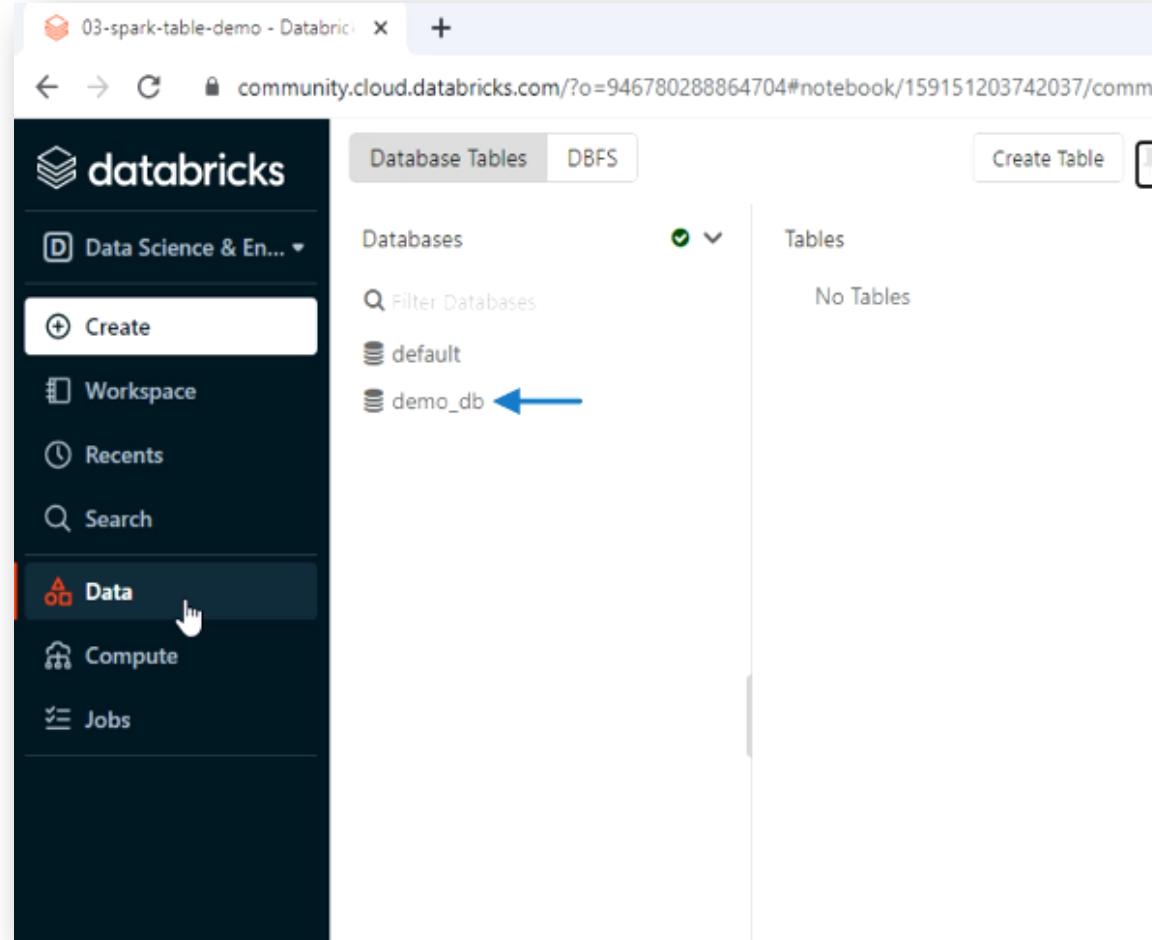
The cell is currently active, as indicated by the cursor in the input field:

```
1 |
```

A note at the bottom of the cell says "Shift+Enter to run".

To see your database, click the Data menu item, and you will see two databases:

1. Spark created one default database for you. So if you create a table, it will be created inside the default database. But we do not recommend using the default database.
2. Instead, you should create a database for your project and keep all your project tables inside your database. So we have already created one database for ourselves.



Now the next aim is to create a table. The creation of table also follows a create table DDL statement as shown below. We have 28 columns in the fire response call dataset. Most of them are string columns, but some are integer and float columns.

```
03-spark-table-demo SQL
demo-cluster
1 create table if not exists demo_db.fire_service_calls_tbl(
2     CallNumber integer,
3     UnitID string,
4     IncidentNumber integer,
5     CallType string,
6     CallDate string,
7     WatchDate string,
8     CallFinalDisposition string,
9     AvailableDtM string,
10    Address string,
11    City string,
12    Zipcode integer,
13    Battalion string,
14    StationArea string,
15    Box string,
16    OriginalPriority string,
17    Priority string,
18    FinalPriority integer,
19    ALSUnit boolean,
20    CallTypeGroup string,
21    NumAlarms integer,
22    UnitType string,
23    UnitSequenceInCallDispatch integer,
24    FirePreventionDistrict string,
25    SupervisorDistrict string,
26    Neighborhood string,
27    Location string,
28    RowID string,
29    Delay float
```

Finally, we will specify the file format for the table. I will be using parquet for the table. Every database table internally stores data in files. So for this table, I want Spark to use parquet file format. And if we run this following code, a table would be created for us.

The screenshot shows a database management system interface with the following details:

- Project:** 03-spark-table-demo
- Cluster:** demo-cluster
- Table Definition:**

```
15 Box string,
16 OriginalPriority string,
17 Priority string,
18 FinalPriority integer,
19 ALSUnit boolean,
20 CallTypeGroup string,
21 NumAlarms integer,
22 UnitType string,
23 UnitSequenceInCallDispatch integer,
24 FirePreventionDistrict string,
25 SupervisorDistrict string,
26 Neighborhood string,
27 Location string,
28 RowID string,
29 Delay float
30 ) using parquet
```
- Status:** OK
- Execution Details:** Command took 1.00 second -- by prashant@scholarnest.com at 4/4/2022, 10:00:36 PM on demo-cluster
- Command Line:** Cmd 3
1 |
Shift+Enter to run

We do not have any data in the table. So we can use the insert-into command to insert some records into the table in a traditional database table, as shown in the image below. I am inserting one row into the *demo_db.fire_service_calls_tbl*. All the column values are null except the first column.

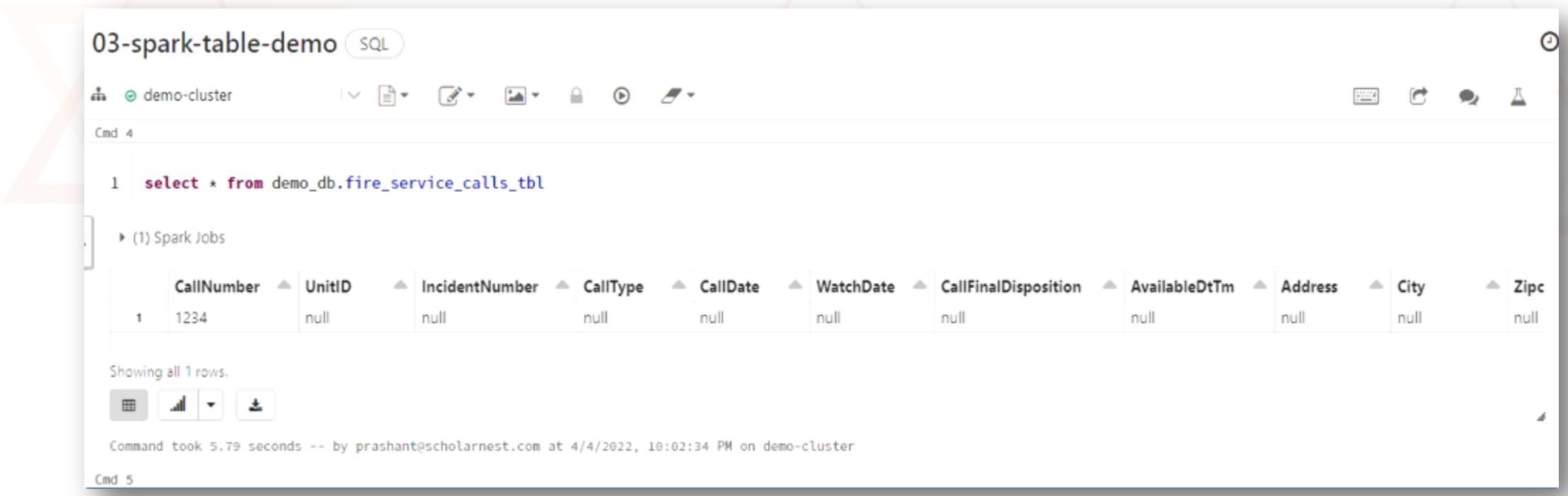
The screenshot shows a Databricks SQL interface with the following session history:

- Cmd 3:** Created a table named *fire_service_calls_tbl* with columns *RowID* (string) and *Delay* (float), using the Parquet format.
- OK**
- Command took 1.00 second -- by prashant@scholarnest.com at 4/4/2022, 10:00:36 PM on demo-cluster**
- Cmd 4:** Executed an *insert into* statement to insert a single row into the *fire_service_calls_tbl* table. The statement is:

```
1 insert into demo_db.fire_service_calls_tbl
2 values(1234, null, null,
3 null, null, null, null, null, null, null, null)
```

A blue arrow points to the *insert into* keyword.
- (1) Spark Jobs**
- OK**
- Command took 8.37 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:01:52 PM on demo-cluster**

You can run a select * expression to see the content of the table. And we can see in the output that data is inserted successfully.



```
03-spark-table-demo SQL
demo-cluster Cmd 4
1 select * from demo_db.fire_service_calls_tbl
▶ (1) Spark Jobs
CallNumber UnitID IncidentNumber CallType CallDate WatchDate CallFinalDisposition AvailableDtNm Address City Zipc
1 1234 null null
```

Showing all 1 rows.

Command took 5.79 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:02:34 PM on demo-cluster

Cmd 5

Spark tables are expected to hold millions and billions of records. And we can't write the insert-into values command to load those records.

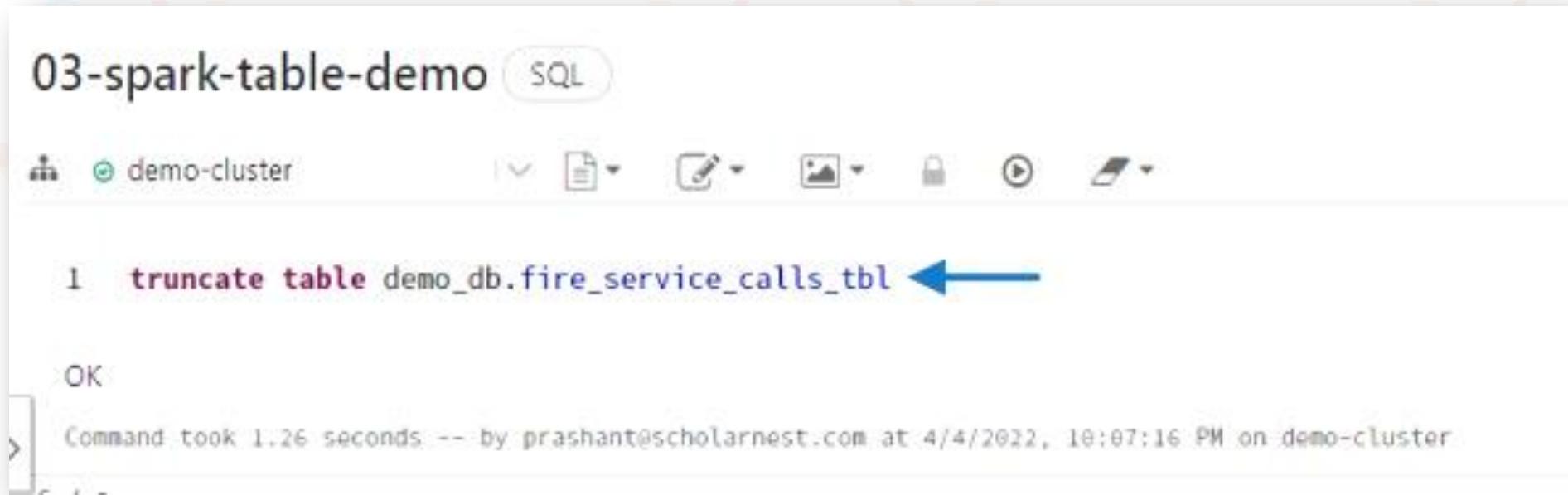
The insert into values expression doesn't make much sense in the big data world.

You will not find a project that uses insert into values to load data into Spark tables.

There are many ways to load data into Spark tables. But for now, we will insert records into the table from another table.

Let us truncate the table, so that we clean the garbage record that we just inserted.

We used the truncate statement because Spark SQL doesn't offer to delete statements.



The screenshot shows a Jupyter Notebook cell titled "03-spark-table-demo" with the "SQL" tab selected. The cell contains the following code:

```
1 truncate table demo_db.fire_service_calls_tbl
```

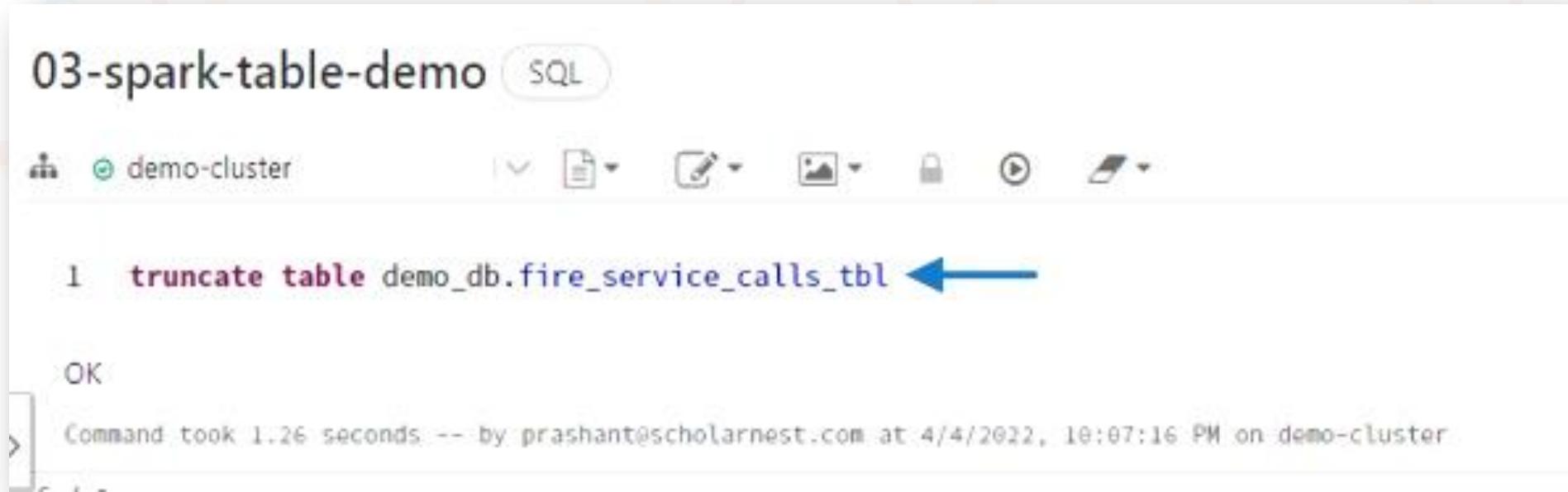
A blue arrow points to the word "truncate". Below the cell, the output shows:

OK

Command took 1.26 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:07:16 PM on demo-cluster

Let us truncate the table, so that we clean the garbage record that we just inserted.

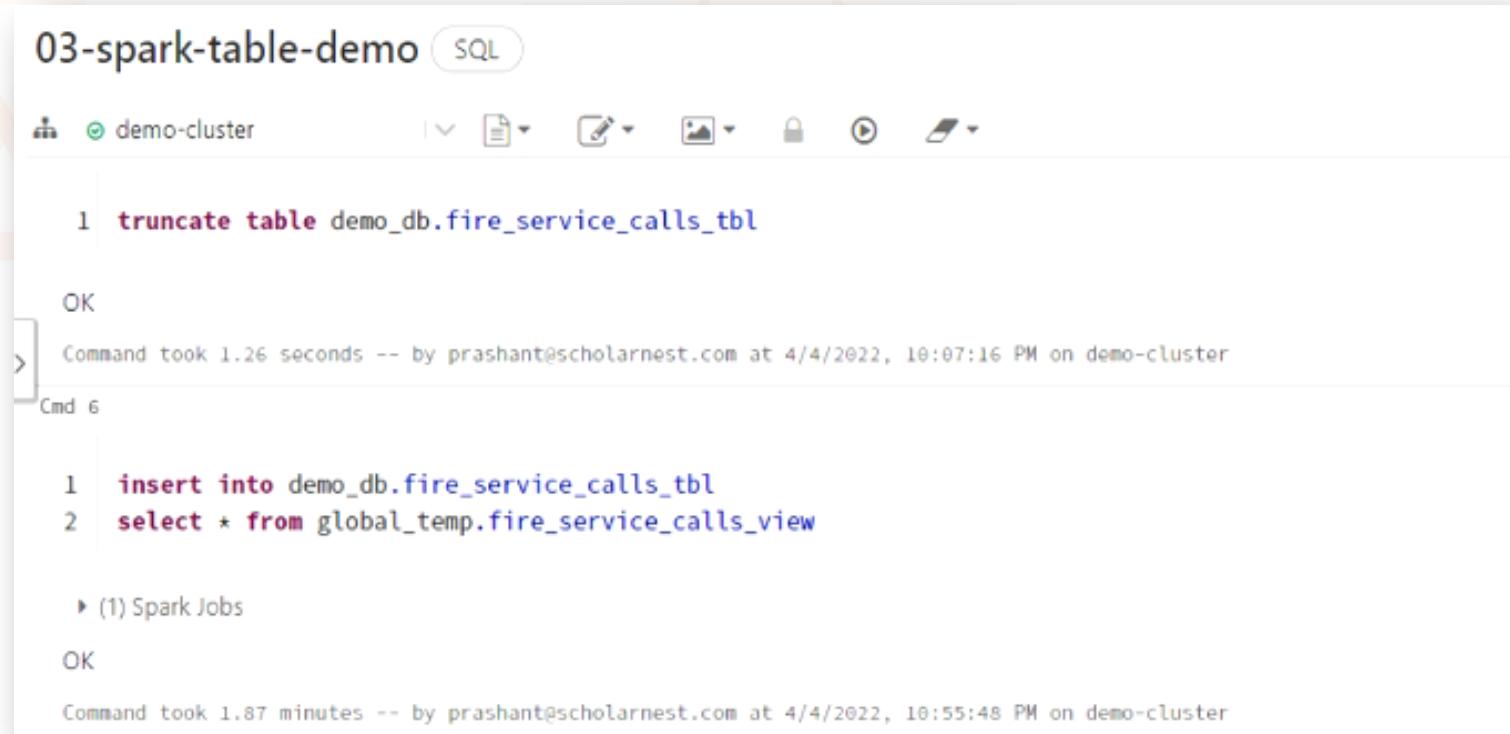
We used the truncate statement because Spark SQL doesn't offer to delete statements. You cannot delete data from a Spark Table using Spark SQL. Databricks does offer a delete expression on Spark tables.



The screenshot shows a Databricks SQL editor interface. The title bar says "03-spark-table-demo" and "SQL". Below the title bar, there's a cluster selection dropdown set to "demo-cluster" and a toolbar with various icons. The main area contains a single line of code: "1 truncate table demo_db.fire_service_calls_tbl". A blue arrow points to the word "truncate". Below the code, the status bar shows "OK" and "Command took 1.26 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:07:16 PM on demo-cluster".

```
1 truncate table demo_db.fire_service_calls_tbl
```

The next target is to insert data into the table after it is truncated. And we have the code for the same in the screenshot shown below. I am inserting records into the *demo_db.fire_service_calls_tbl* from *global_temp.fire_service_calls_view*.



The screenshot shows a Databricks SQL editor window titled "03-spark-table-demo". The interface includes a toolbar with icons for cluster selection, copy, paste, and refresh, and a sidebar showing the current cluster ("demo-cluster").

The SQL history pane displays the following commands:

```
1 truncate table demo_db.fire_service_calls_tbl
OK
Command took 1.26 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:07:16 PM on demo-cluster
Cmd 6

1 insert into demo_db.fire_service_calls_tbl
2 select * from global_temp.fire_service_calls_view

▶ (1) Spark Jobs
OK
Command took 1.87 minutes -- by prashant@scholarnest.com at 4/4/2022, 10:55:48 PM on demo-cluster
```

The first command, `truncate table demo_db.fire_service_calls_tbl`, was executed successfully ("OK") and took 1.26 seconds. The second command, which includes an `insert into` statement followed by a `select * from` statement, was also successful ("OK") and took 1.87 minutes. Both commands were run by user `prashant@scholarnest.com` at 4/4/2022, 10:07:16 PM and 4/4/2022, 10:55:48 PM respectively, on the `demo-cluster`.

Finally, you can see the content of the table using the select expression.

03-spark-table-demo [SQL](#)

demo-cluster [View](#) [Edit](#) [Run](#) [Download](#) [Image](#) [Lock](#) [Help](#) [Edit](#)

Committed: 1.89 minutes -- by prashant@scholarnest.com at 4/4/2022, 10:55:48 PM on demo-cluster

Cmd: 7

```
1 select * from demo_db.fire_service_calls_tbl
```

▶ (1) Spark Jobs

	CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm	Address
1	61160285	E22	6032206	Medical Incident	04/26/2006	04/26/2006	Other	null	1400 B...
2	61160287	93	6032207	Medical Incident	04/26/2006	04/26/2006	Code 2 Transport	04/26/2006 02:32:37 PM	400 Bl...
3	61160287	E13	6032207	Medical Incident	04/26/2006	04/26/2006	Other	04/26/2006 01:40:18 PM	400 Bl...
4	61160287	RC4	6032207	Medical Incident	04/26/2006	04/26/2006	Other	04/26/2006 01:47:22 PM	400 Bl...
5	61160293	B02	6032209	Alarms	04/26/2006	04/26/2006	Other	04/26/2006 01:31:30 PM	0 Block...
6	61160293	E06	6032209	Alarms	04/26/2006	04/26/2006	Other	null	0 Block...
7	61160293	T06	6032209	Alarms	04/26/2006	04/26/2006	Other	04/26/2006 01:31:13 PM	0 Block...

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

[Grid](#) [Table](#) [Download](#) [Copy](#)

Command took 1.89 seconds -- by prashant@scholarnest.com at 4/4/2022, 10:58:31 PM on demo-cluster



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks

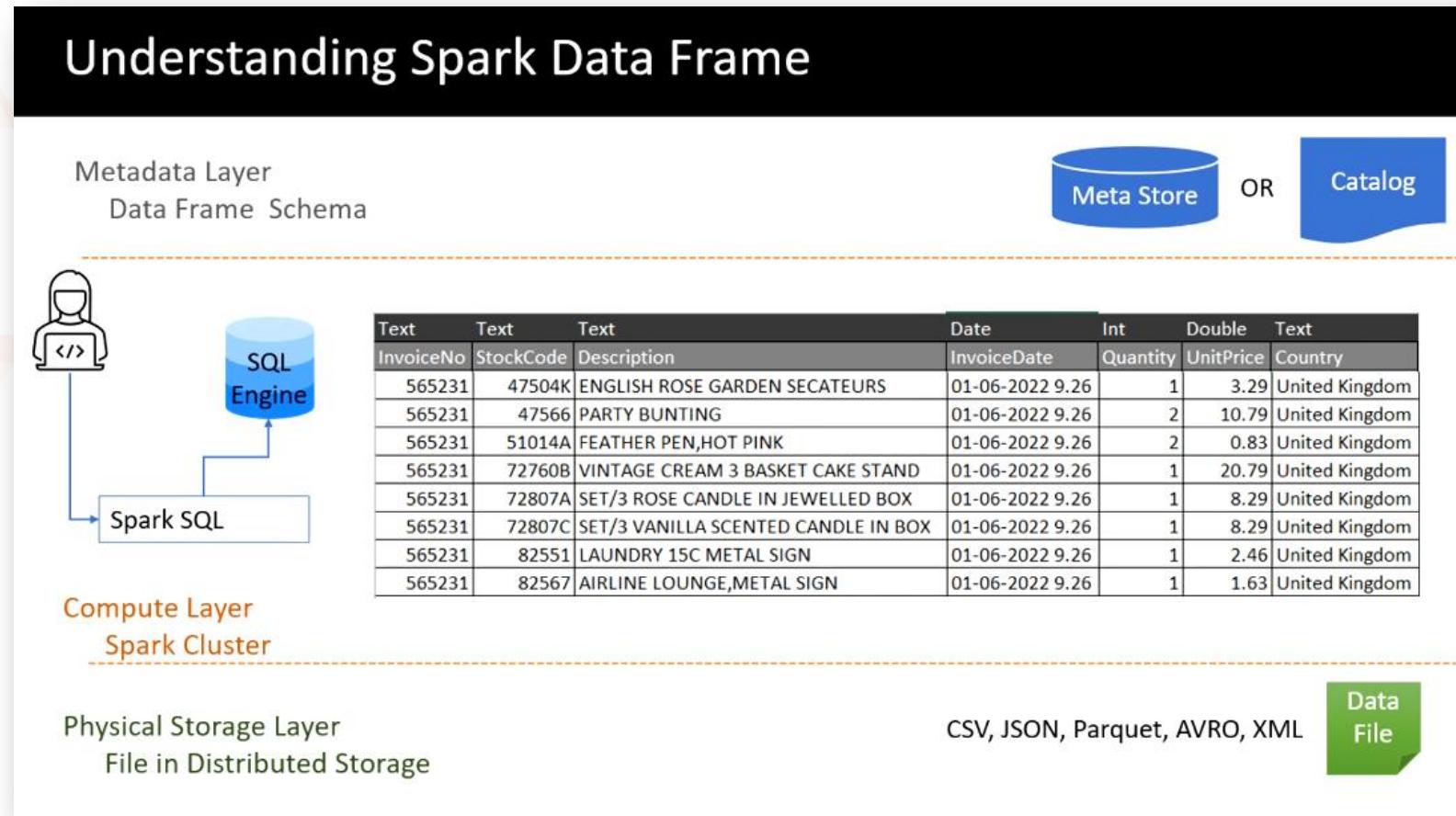




Problems with Databricks Community

Before we proceed further with the micro project, let us have a look at the below diagram once again. Every Spark Table or the Spark Dataframe lives in three layers:

1. Layer – It stores Table or Dataframe definition and the schema information.
2. Compute Layer – It runs your Spark SQL engine.
3. Physical Storage Layer – It stores the spark data in a data file.



When you execute a Spark SQL query, or you run a Dataframe code, the Spark SQL engine will refer to the metadata store. And the metadata will tell the following things:

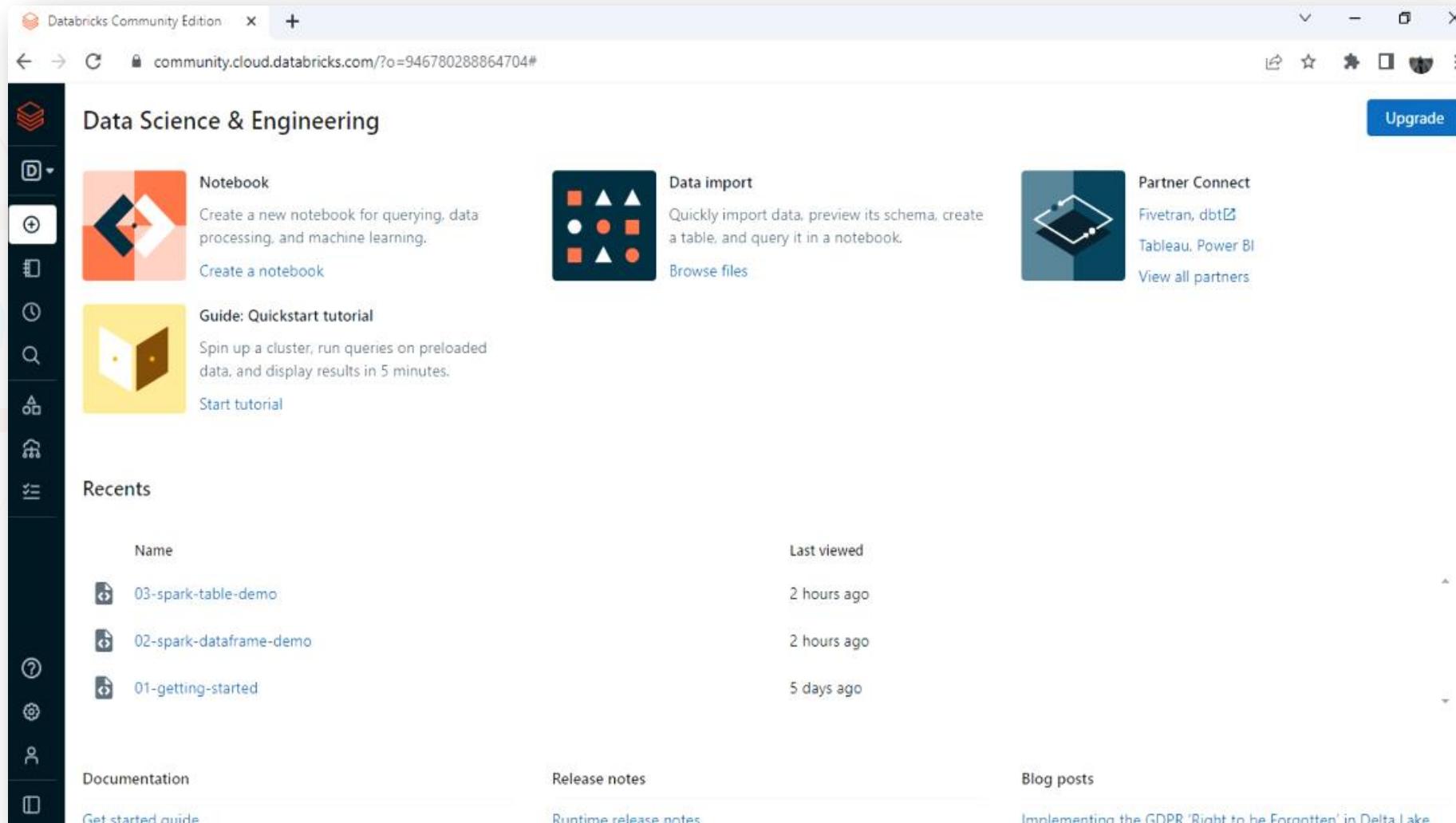
1. Where is the data for this table?
2. What is the schema of the data and table?

If the Spark SQL engine doesn't find metadata data for the table, it throws an analysis exception.

If metadata is there, the SQL engine will read the data from the data file and present it to you like a table or Dataframe. So, the three(Metadata, Compute and Storage) layers of Spark are supercritical.

You will face problems and see errors when any of these three layers are missing or broken.

Go to your Databricks workspace and go to the compute menu.



You need a running cluster for doing anything in the Databricks environment because you need a compute layer. But we have an old cluster here which is in terminated state because I left it idle for more than 2 hours, so Databricks terminated my cluster.

The screenshot shows the Databricks Compute interface. On the left is a dark sidebar with various icons for navigation. The main area has a header "Compute - Databricks Community" and a URL "community.cloud.databricks.com/?o=946780288864704#setting/clusters". Below the header, there are two tabs: "All-purpose clusters" (which is selected) and "Job clusters". A large blue button labeled "Create Cluster" is visible. The main table lists one cluster:

Name	State	Modes	Runtime	Driver	Worker	Creator	Actions
demo-cluster	Terminated	(link)	(link)	(link)	(link)	(link)	(link)

A tooltip over the "Terminated" status cell provides information: "Inactivity: The cluster was automatically terminated after 120 minutes of inactivity." The bottom right corner of the table shows pagination: "1 - 1 of 1" and "20 / Page Go to 1".

I cannot restart this cluster again. I must create a new cluster if I want to work again.

Databricks community edition will terminate your cluster after 2 hours of idle time, and once your cluster is terminated, Databricks will clean up the following things:

1. Cluster VM (Compute Layer) – You cannot restart your cluster again.
2. Spark Metadata Store (Metadata Layer) – You will lose your databases and tables.

The screenshot shows the Databricks Compute interface. At the top, there are tabs for 'All-purpose clusters' and 'Job clusters', with 'All-purpose clusters' being the active tab. Below the tabs is a blue button labeled 'Create Cluster'. On the right side of the header, there is a message: 'Could not start cluster demo-cluster. Cluster Start feature is currently disabled.' A red box highlights this message. The main area is a table with the following columns: Name, State, Nodes, Runtime, Driver, Worker, Creator, and Actions. There is one row in the table, representing a cluster named 'demo-cluster' which is 'Terminated'. The 'Actions' column for this row contains three dots (...). At the bottom of the table, there is a pagination control showing '1 - 1 of 1' and a 'Page' dropdown set to '20'.

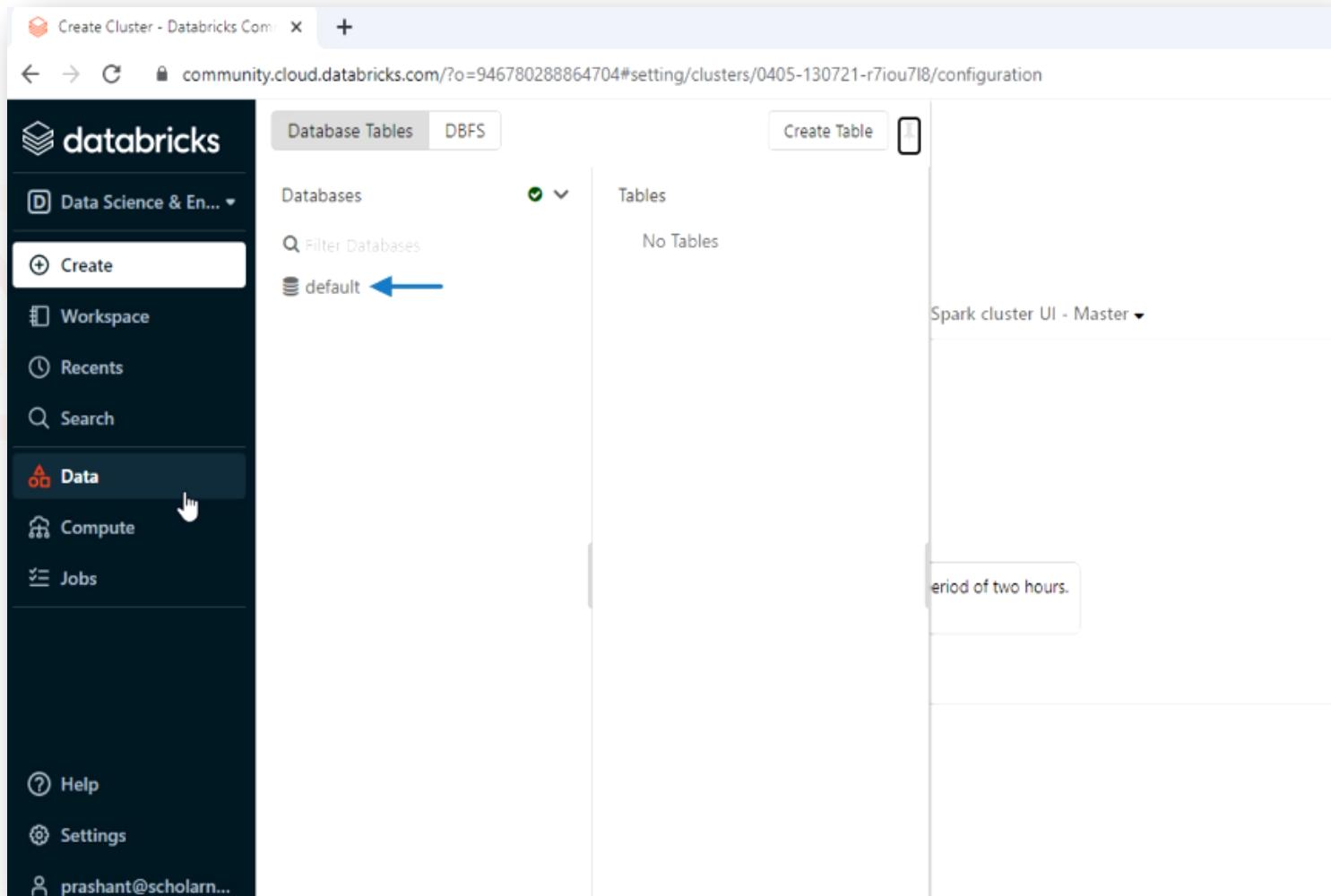
Name	State	Nodes	Runtime	Driver	Worker	Creator	Actions
demo-cluster	Terminated	-	9.1 LTS (includes Apache Spark 3.1.2, Scala 2.12)	Community...	Community...	prashant@sc...	...

You can delete the old cluster, and create a new one with same name. Creating a cluster takes about 5-10 minutes.

The screenshot shows the Databricks Cluster Configuration page for a cluster named 'demo-cluster'. The page includes a sidebar with navigation icons and a main content area with tabs for Configuration, Notebooks, Libraries, Event log, Spark UI, Driver logs, Metrics, Apps, and Spark cluster UI - Master. The Configuration tab is selected. Key details shown include:

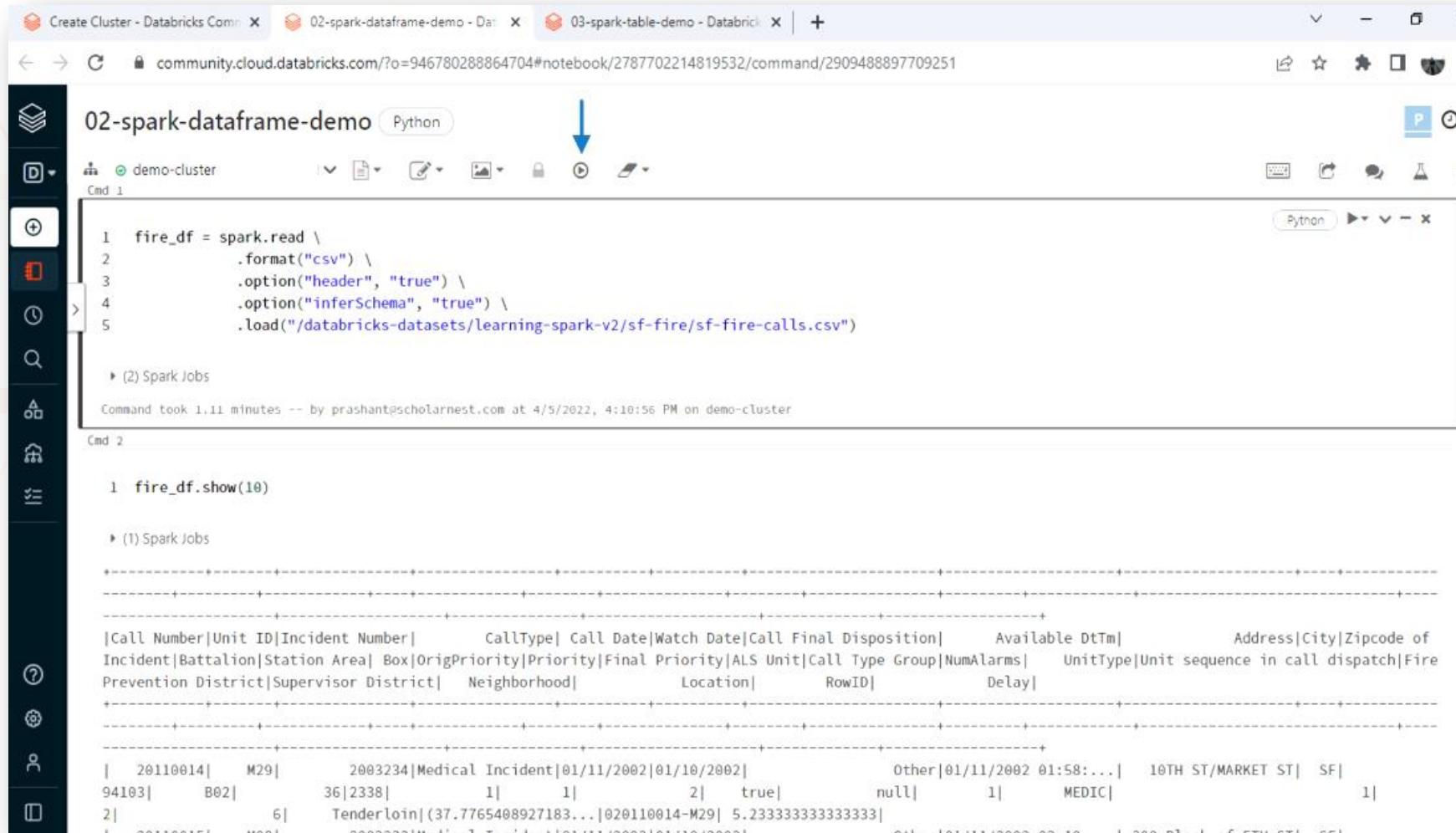
- Databricks Runtime Version: 10.4 LTS (includes Apache Spark 3.2.1, Scala 2.12)
- Driver type: Community Optimized (15.3 GB Memory, 2 Cores, 1 DBU)
- A message box states: "Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For [more configuration options](#), please [upgrade your Databricks subscription](#)."
- Availability zone: us-west-2a

You can click the data menu, and you can see the database and tables. We only have a default database the *demo_db* database that we created yesterday is gone. Because Databricks cleaned my Spark metadata store as my cluster was idle for more than 2 hours. So I lost my database and table definition. This problem does not exist for the full licensed version of the Databricks Cloud.



The community edition will clean up the VM and the metadata store. But it doesn't clean up the directories and files. So you do not lose your storage layer. Your data files remain in place even if your compute and metadata layer is gone. Directories are created in an external distributed storage, and files are stored in these directories on external storage.

You can go the notebooks you created earlier (02-spark-dataframe-demo.ipynb and 03-spark-table-demo.sql) from the workspace menu option, and click the run all option shown below to run the entire notebook at once. Make sure your cluster is attached to both the notebooks.



The screenshot shows the Databricks Notebook interface with two cells visible:

- Cmd 1:** Contains Python code to read a CSV file into a DataFrame:

```
1 fire_df = spark.read \
2     .format("csv") \
3     .option("header", "true") \
4     .option("inferSchema", "true") \
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```
- Cmd 2:** Contains Python code to show the first 10 rows of the DataFrame:

```
1 fire_df.show(10)
```

A blue arrow points to the "Run All" button located at the top of the notebook editor, between the tabs and the toolbar.

Output for Cmd 1:

```
(2) Spark Jobs  
Command took 1.11 minutes -- by prashant@scholarnest.com at 4/5/2022, 4:10:56 PM on demo-cluster
```

Output for Cmd 2:

```
(1) Spark Jobs
```

Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Address	City	Zipcode of								
Incident	Battalion	Station Area	Box	OrigPriority	Priority	Final Priority	ALS Unit	Call Type Group	NumAlarms	UnitType	Unit sequence in call dispatch	Fire Prevention District	Supervisor District	Neighborhood	Location	RowID	Delay	
94103	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:...		10TH ST/MARKET ST	SF	1	6	Tenderloin	(37.7765408927183... 020110014-M29 5.233333333333333			
2	20110015	M29	362338	1	1	2	true	null	1	MEDIC				Medical	94103 01/11/2002 01/10/2002	Other 01/11/2002 02:10:...	1 200 Block of 5TH ST	SF

We ran the *02-spark-dataframe-demo.ipynb* successfully but encountered an error in the *03-spark-table-demo.sql* notebook. The error says that “can not create the managed table. The associated location already exists.” We didn't face any errors yesterday when we created this table. But we are getting an error today because Databricks didn't clean the directory and data files. My storage layer is still there. So my data directory is not cleaned.

The screenshot shows the Databricks SQL interface with three tabs at the top: "Create Cluster - Databricks Com" (closed), "02-spark-dataframe-demo - Data" (closed), and "03-spark-table-demo - Databricks" (active). The URL in the address bar is community.cloud.databricks.com/?o=946780288864704#notebook/159151203742037/command/461968312746697. The sidebar on the left has a dark theme with icons for clusters, databases, tables, search, and more. The main area contains the following SQL code:

```
16 OriginalPriority string,  
17 Priority string,  
18 FinalPriority integer,  
19 ALSUnit boolean,  
20 CallTypeGroup string,  
21 NumAlarms integer,  
22 UnitType string,  
23 UnitSequenceInCallDispatch integer,  
24 FirePreventionDistrict string,  
25 SupervisorDistrict string,  
26 Neighborhood string,  
27 Location string,  
28 RowID string,  
29 Delay float  
30 ) using parquet
```

An error message is displayed in the bottom right corner:

Error running command 2. Go to command
[Clear all notifications](#) | [Settings](#)

A red box highlights the error message in the command history:

④ Error in SQL statement: AnalysisException: Can not create the managed table(``demo_db``.`fire_service_calls_tbl``). The associated location(`dbfs:/user/hive/warehouse/demo_db.db/fire_service_calls_tbl`) already exists.

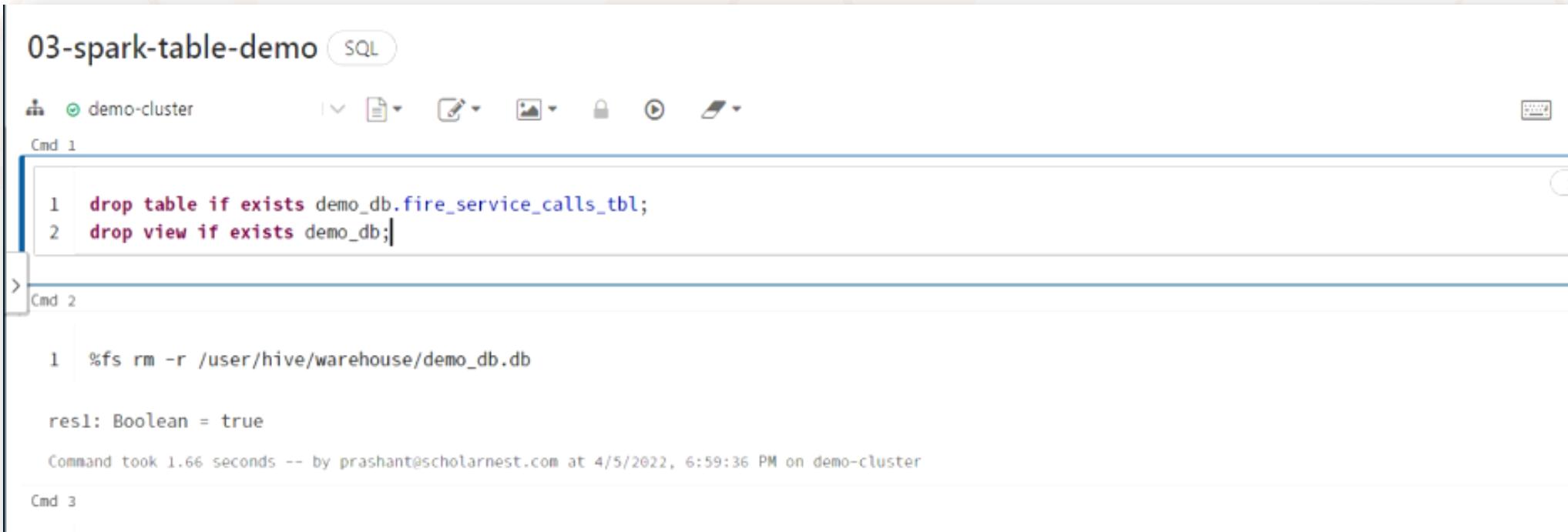
Command took 2.16 seconds -- by prashant@scholarnest.com at 4/5/2022, 6:53:30 PM on demo-cluster

Cmd 3

In order to fix the Analysis Exception, we can add a new cell at the top of the notebook and type the filesystem rm command to clean the directory. Copy the directory location from the error message.

A screenshot of a Jupyter Notebook interface titled "03-spark-table-demo". The notebook has a "SQL" tab selected. The top navigation bar includes icons for cluster selection ("demo-cluster"), file operations (File, Edit, View, Insert, Cell, Kernel), and help. Below the title, there is a toolbar with icons for cluster selection, file operations, and cell execution. A command cell labeled "Cmd 1" contains the command "%fs rm -r /user/hive/warehouse/demo_db.db". The output cell shows the result "res1: Boolean = true". At the bottom, a footer indicates the command took 1.66 seconds and was run by prashant@scholarnest.com at 4/5/2022, 6:59:36 PM on demo-cluster.

I also recommend adding a drop table and drop database command at the top. You can add another cell at the top of the notebook and run two DDL statements to drop your table and database. Do not forget to add a semicolon at the end of the SQL expression.



The screenshot shows a Jupyter Notebook cell titled "03-spark-table-demo" in "SQL" mode. The cell contains two commands:

```
1 drop table if exists demo_db.fire_service_calls_tbl;
2 drop view if exists demo_db;
```

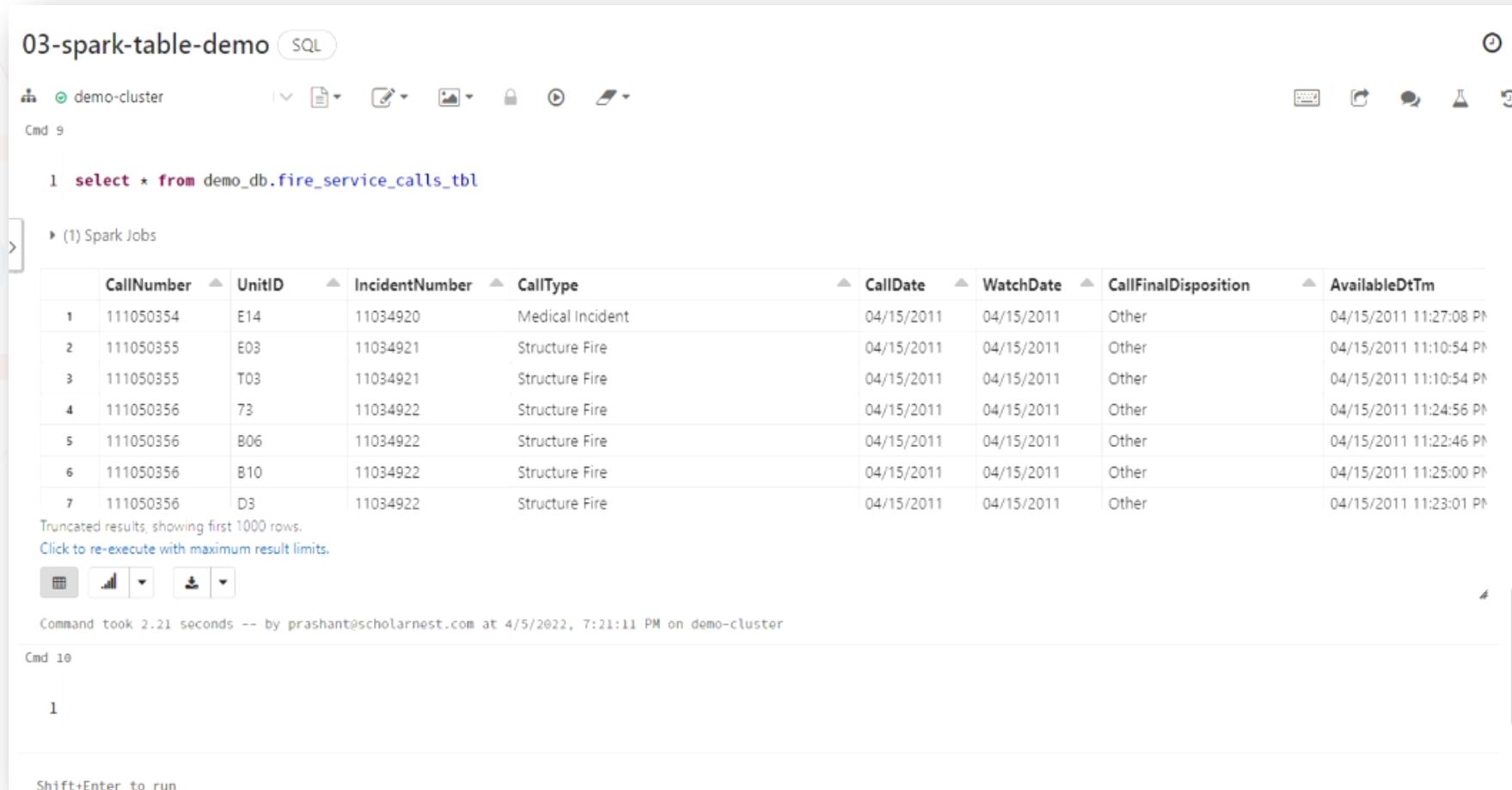
Below the cell, the output shows:

```
1 %fs rm -r /user/hive/warehouse/demo_db.db

res1: Boolean = true
```

At the bottom, the message "Command took 1.66 seconds -- by prashant@scholarnest.com at 4/5/2022, 6:59:36 PM on demo-cluster" is displayed.

Now, you can go ahead and click the run all button for the *03-spark-table-demo.sql* notebook and it should work perfectly fine without giving any errors.



The screenshot shows a Jupyter Notebook interface titled "03-spark-table-demo". The notebook is set to "SQL" mode. The code cell contains the following SQL query:

```
1 select * from demo_db.fire_service_calls_tbl
```

Below the code cell, there is a section titled "(1) Spark Jobs" which displays a table of data. The table has the following columns:

	CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm
1	111050354	E14	11034920	Medical Incident	04/15/2011	04/15/2011	Other	04/15/2011 11:27:08 PM
2	111050355	E03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM
3	111050355	T03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM
4	111050356	73	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:24:56 PM
5	111050356	B06	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:22:46 PM
6	111050356	B10	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:25:00 PM
7	111050356	D3	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:23:01 PM

Below the table, a message states: "Truncated results, showing first 1000 rows. Click to re-execute with maximum result limits."

At the bottom of the notebook, the command took 2.21 seconds and was run by prashant@scholarnest.com at 4/5/2022, 7:21:11 PM on demo-cluster.

The notebook has 10 commands in total.



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks

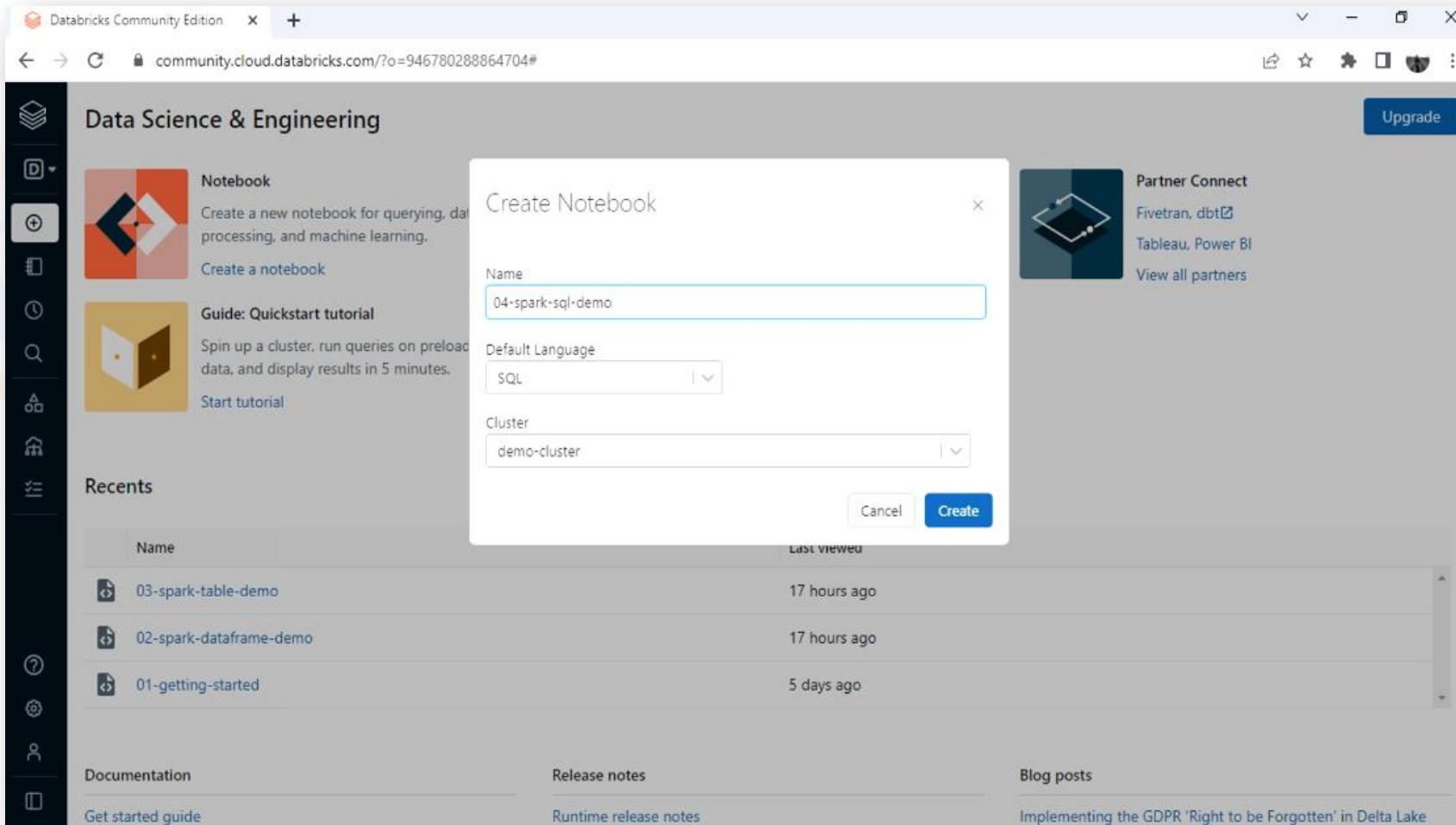


Data Caching Essentials

Go to your Databricks workspace.

Create a notebook. (**Reference: 04-spark-sql-demo.sql**)

We want to use Spark SQL for data analysis. So choose SQL as the default language of your notebook. Make sure to attach a cluster to your notebook.



We created a table for the San Fransisco fire call response dataset. Now let me type the code to select hundred records from the table.

The screenshot shows a Databricks SQL notebook interface. The top bar displays the URL `community.cloud.databricks.com/?o=946780288864704#notebook/1805343142881956`. The sidebar on the left contains various icons for navigation and cluster management. The main area shows a command cell labeled "Cmd 1" containing the SQL query:

```
1 select * from demo_db.fire_service_calls_tbl limit 100
```

Below the query is a table titled "(1) Spark Jobs" showing the results of the query. The table has the following columns:

	CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm	Addl
1	111050354	E14	11034920	Medical Incident	04/15/2011	04/15/2011	Other	04/15/2011 11:27:08 PM	500 I
2	111050355	E03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM	HYD
3	111050355	T03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM	HYD
4	111050356	73	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:24:56 PM	1000
5	111050356	B06	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:22:46 PM	1000
6	111050356	B10	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:25:00 PM	1000
7	111050356	D3	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:23:01 PM	1000

A blue arrow points to the text "Showing all 100 rows." at the bottom of the table. Below the table, a message states "Command took 1.28 seconds -- by prashant@scholarnest.com at 4/5/2022, 8:56:27 PM on demo-cluster".

The bottom section of the notebook shows a second command cell labeled "Cmd 2" with the number "1" entered. A note below says "Shift+Enter to run".

I want to cache this table into memory to speed up my queries. I want to run ten queries on the same table. So it makes sense that I cache this table into the memory and avoid reading data from the disk for every query. And we have the code for the same in the snapshot below.

The screenshot shows a Databricks SQL notebook interface. The title bar reads "04-spark-sql-demo - Databricks". The main area displays a table named "fire_service_calls_tbl" with 100 rows. The columns are: ID, LocationID, CallType, CallNumber, Cause, Date, Time, and Duration. The data shows multiple entries for the same location (11034922) and cause (Structure Fire). Below the table, a message indicates the command took 1.28 seconds. The notebook history shows two previous commands:

```
1 cache lazy table fire_service_calls_tbl_cache as
2 select * from demo_db.fire_service_calls_tbl
```

The third command entry field contains "1 |".

ID	LocationID	CallType	CallNumber	Cause	Date	Time	Duration
4	111050356	73	11034922	Structure Fire	04/15/2011	04/15/2011	Other
5	111050356	B06	11034922	Structure Fire	04/15/2011	04/15/2011	Other
6	111050356	B10	11034922	Structure Fire	04/15/2011	04/15/2011	Other
7	111050356	D3	11034922	Structure Fire	04/15/2011	04/15/2011	Other

Command took 1.28 seconds -- by prashant@scholarnest.com at 4/5/2022, 8:56:27 PM on demo-cluster

Cmd 2

```
1 cache lazy table fire_service_calls_tbl_cache as
2 select * from demo_db.fire_service_calls_tbl
```

OK

Command took 0.40 seconds -- by prashant@scholarnest.com at 4/5/2022, 8:58:52 PM on demo-cluster

Cmd 3

```
1 |
```

Shift+Enter to run

You must learn the following things about the Spark SQL cache table.

1. How does the Spark cache table works?

The cache-table-as command will create a temporary view using the Select expression. So you already have a table. But when you cache it, you will create a temporary view on the table. Once your table is cached as a temporary view, you can use the view or the table. Once cached, Spark will always bring data from the cache.

2. How to make it re-executable?

The cache-table-as command creates a temporary view, so you cannot rerun it. And if you try re-running the cache command again, you will see a “temporary view already exists” error. You can fix this issue by dropping the temporary view before rerunning the cache-table-as command.

3. What is Lazy?

The lazy is to make sure we do not cache the table until it is used. It is optional. You can also remove it, but it is recommended to keep it as the execution will be very fast because it simply marks the table to be cached on its first use.

Caching the table consumes a lot of memory. So you should make use of this feature carefully. We have some guidelines:

1. Do not cache tiny tables because tiny table will anyway work faster.
2. Do not cache huge tables that cannot fit in the Spark memory because there is no point in caching a table into memory if it cannot fit in the memory.

It is recommended to cache medium-sized frequently used tables.

So, you should cache a table if you meet the following criteria:

1. Table data is a reasonable size and fits into the memory.
2. Table is frequently used. You should not cache a table that you do not want to use again and again.



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Working with Spark SQL

Go back to your notebook you created in the previous lecture. (Reference: 04-spark-sql-demo.sql)

We created a table and also cached it into the memory. Now we are ready to start writing some SQL expressions to answer the 10 micro project questions.

```
04-spark-sql-demo - Databricks
```

```
community.cloud.databricks.com/?o=946780288864704#notebook/1805343142881956/command/1805343142881961
```

04-spark-sql-demo SQL

demo-cluster

Cmd 1

```
1 select * from demo_db.fire_service_calls_tbl limit 100
```

(1) Spark Jobs

CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm	Addl
1	E14	11034920	Medical Incident	04/15/2011	04/15/2011	Other	04/15/2011 11:27:08 PM	500 I
2	E03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM	HYD
3	T03	11034921	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:10:54 PM	HYD
4	73	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:24:56 PM	1000
5	B06	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:22:46 PM	1000
6	B10	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:25:00 PM	1000
7	D8	11034922	Structure Fire	04/15/2011	04/15/2011	Other	04/15/2011 11:23:01 PM	1000

Showing all 100 rows.

Command took 1.84 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:40:10 PM on demo-cluster

Cmd 2

```
1 drop view if exists fire_service_calls_tbl_cache;
```

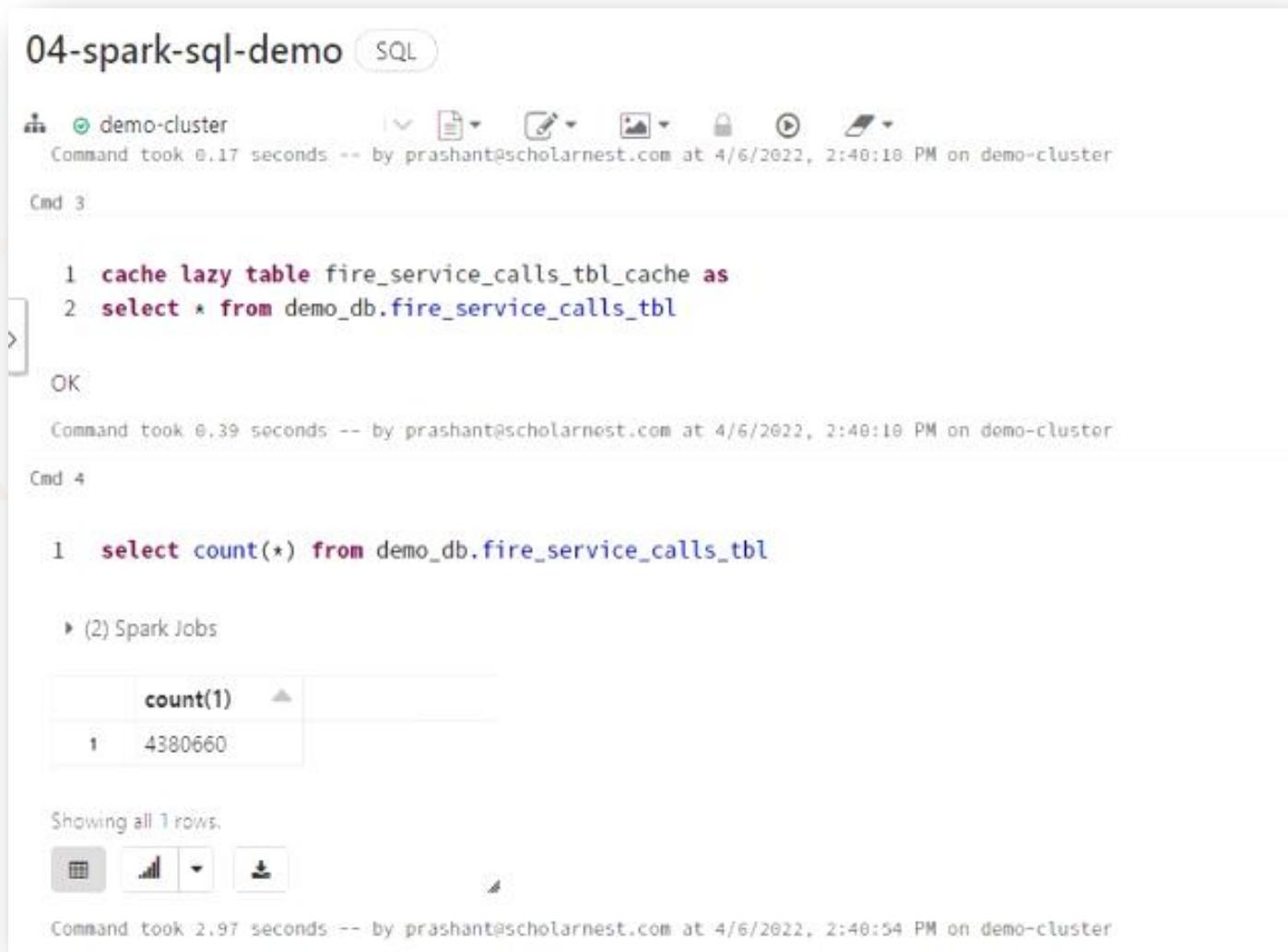
OK

Command took 0.17 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:40:10 PM on demo-cluster

12:51

Cmd 3

Let us start by taking a simple count from the table. We can use the select count(*) query. If we run it, we can see there are more than 4.3 million records in the table.



The screenshot shows a Jupyter Notebook cell titled "04-spark-sql-demo" in "SQL" mode. The cell contains the following code:

```
1 cache lazy table fire_service_calls_tbl_cache as
2 select * from demo_db.fire_service_calls_tbl
```

The output of the first command is "OK". The second command is:

```
1 select count(*) from demo_db.fire_service_calls_tbl
```

This command triggers "(2) Spark Jobs". The resulting table is:

count(1)
1 4380660

Below the table, it says "Showing all 1 rows." and provides icons for table, chart, and download. The command took 2.97 seconds.

Here are ten questions we want to answer:

1. How many distinct types of calls were made to the fire department?
2. What are distinct types of calls made to the fire department?
3. Find out all responses or delayed times greater than 5 mins?
4. What were the most common call types?
5. What zip codes accounted for the most common calls?
6. What San Francisco neighbourhoods are in the zip codes 94102 and 94103
7. What was the sum of all calls, average, min, and max of the call response times?
8. How many distinct years of data are in the CSV file?
9. What week of the year in 2018 had the most fire calls?
10. What neighbourhoods in San Francisco had the worst response time in 2018?

Databricks Notebook allows us to document things using markdown language.

You can use the %md command to create a documentation cell, as shown in the image given below.

The screenshot shows a Databricks Notebook interface. At the top, it says "04-spark-sql-demo" and "SQL". Below that is a table output from a query:

	demo-cluster
count(1)	4380660

Below the table, it says "Showing all 1 rows." There are three small icons below the table: a bar chart, a line graph, and a download icon. At the bottom of the table area, it says "Command took 2.97 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:40:54 PM on demo-cluster".

In the main notebook area, there is a cell labeled "Cmd 5" containing the following text:

```
1 %md
2 ##### Q1. How many distinct types of calls were made to the Fire Department?
```

At the bottom of the cell, it says "Shift+Enter to run". The cell has a "Markdown" tab selected in the top right corner.

Let us look at the first question.

They are asking for how many distinct types of calls were made. So the solution is to take count of distinct call types. And we have the SQL code which says *select distinct call type* and then take a *count*. I added one extra condition to eliminate null values in the cell type.

And we got the expected output below the cell. We have 32 unique call types made to the San Francisco Fire station.

The screenshot shows the Databricks SQL interface. The top navigation bar includes '04-spark-sql-demo' and a 'SQL' tab. Below the navigation is a toolbar with various icons. The main area has a tree view on the left with 'Cmd 5' expanded, showing 'Q1. How many distinct types of calls were made to the Fire Department?'. Under 'Cmd 6', the query is displayed:

```
1 select count(distinct callType) as distinct_call_type_count
2 from demo_db.fire_service_calls_tbl
3 where callType is not null
```

Below the query, it says '(3) Spark Jobs'. A table titled 'distinct_call_type_count' shows one row with value 32. At the bottom, it says 'Showing all 1 rows.' and provides download and copy options. The footer indicates the command took 6.65 seconds and was run by prashant@scholarnest.com on 4/6/2022, 2:53:34 PM.

```
1 select count(distinct callType) as distinct_call_type_count
2 from demo_db.fire_service_calls_tbl
3 where callType is not null
```

Showing all 1 rows.

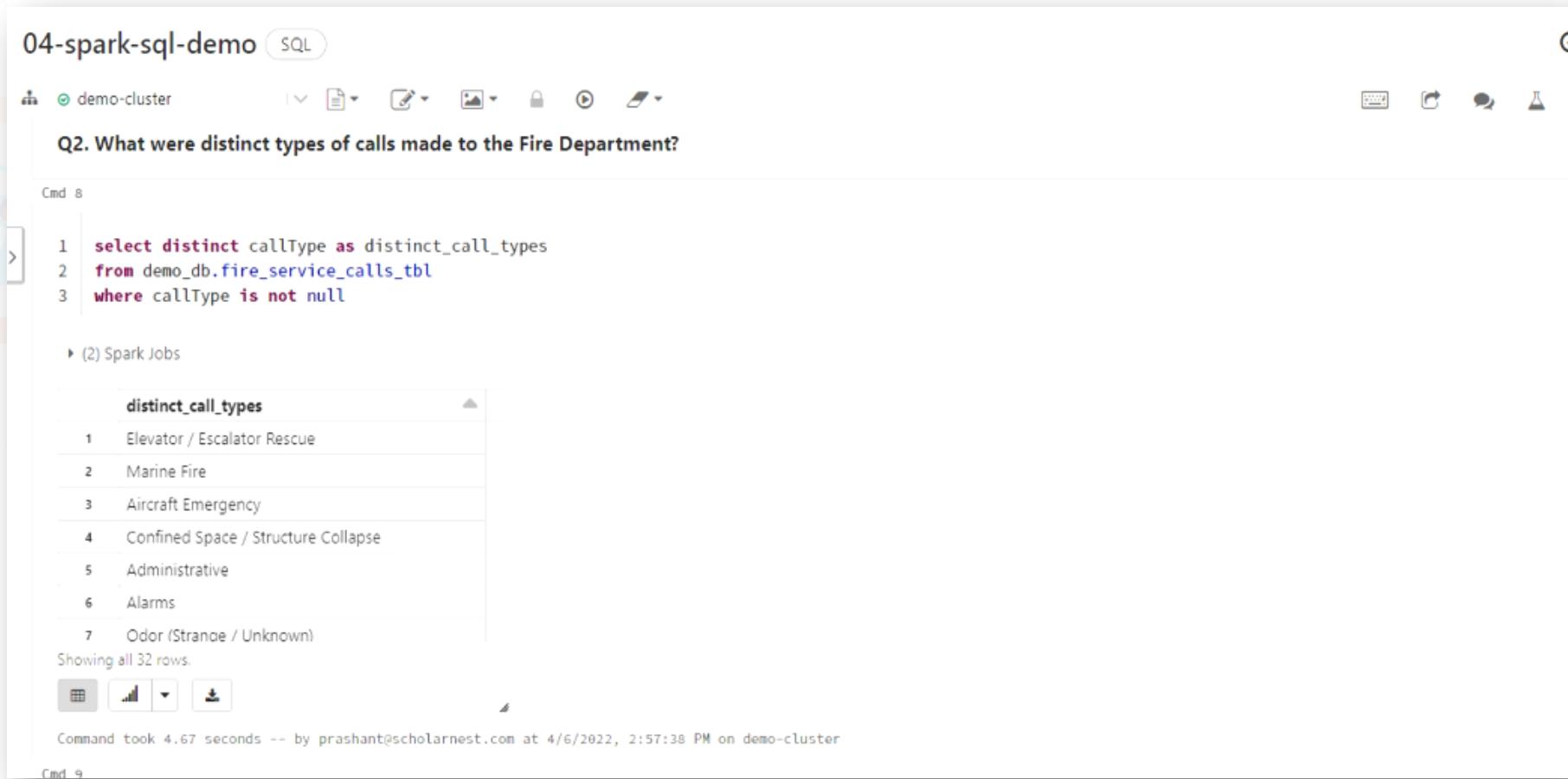
Command took 6.65 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:53:34 PM on demo-cluster

Let us look at the second question.

So the question is almost the same as the first one, but this time, we need a list of distinct call types.

Earlier, we wanted to count distinct call types, but now we want the list.

We have the code for the same shown below. I removed the count function from the select expression we used in the first question, and we can see the expected results.



The screenshot shows a Jupyter Notebook cell titled "Q2. What were distinct types of calls made to the Fire Department?". The cell contains the following SQL code:

```
1 select distinct callType as distinct_call_types
2 from demo_db.fire_service_calls_tbl
3 where callType is not null
```

Below the code, it says "(2) Spark Jobs". A table titled "distinct_call_types" is displayed, showing the following data:

distinct_call_types
1 Elevator / Escalator Rescue
2 Marine Fire
3 Aircraft Emergency
4 Confined Space / Structure Collapse
5 Administrative
6 Alarms
7 Odor (Strange / Unknown)

At the bottom of the cell, it says "Showing all 32 rows." and "Command took 4.67 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:57:38 PM on demo-cluster".

Let us look at the third question.

We want to list the call numbers where the delay is greater than 5 minutes.

We have the code and desired results shown below.

04-spark-sql-demo SQL

demo-cluster

Command took 4.67 seconds -- by prashant@scholarnest.com at 4/6/2022, 2:57:38 PM on demo-cluster

Cmd 9

Q3. Find out all response for delayed times greater than 5 mins?

Cmd 10

```
1 select callNumber, Delay
2 from demo_db.fire_service_calls_tbl
3 where Delay > 5
```

▶ (1) Spark Jobs

	callNumber	Delay
1	111060009	5.4
2	111060015	5.5333333
3	111060023	5.25
4	111060028	5.91666665
5	111060028	5.016667
6	111060065	16.75
7	111060076	32.833332

Truncated results showing first 1000 rows.
Click to re-execute with maximum result limits.

grid icon, chart icon, download icon, refresh icon

Let us look at the fourth question.

We have to find the most common call types. So, we need to take a count on the call type and sort the result in descending order, and we will get the most common calls at the top. I have written the code for the same and you can see the most common types in sorted order.

04-spark-sql-demo SQL

demo-cluster Cmd 11

Q4. What were the most common call types?

Cmd 12

```
1 select callType, count(*) as count
2 from demo_db.fire_service_calls_tbl
3 where callType is not null
4 group by callType
5 order by count desc
```

(2) Spark Jobs

callType	count
Medical Incident	2843475
Structure Fire	578998
Alarms	483518
Traffic Collision	175507
Citizen Assist / Service Call	65360
Other	56961
Outside Fire	51603

Showing all 32 rows.

Command took 5.93 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:03:57 PM on demo-cluster

Let us look at the fifth question.

Here we want to find the most common call types, but we also want to see the zip code for the most common call types. I can do that by adding a zip code column in my group by clause.

04-spark-sql-demo SQL

demo-cluster Cmd 13

Q5. What zip codes accounted for most common calls?

Cmd 14

```
1 select callType, zipCode, count(*) as count
2 from demo_db.fire_service_calls_tbl
3 where callType is not null
4 group by callType, zipCode
5 order by count desc
```

(2) Spark Jobs

callType	zipCode	count
1 Medical Incident	94102	401457
2 Medical Incident	94103	370215
3 Medical Incident	94110	249279
4 Medical Incident	94109	238087
5 Medical Incident	94124	147564
6 Medical Incident	94112	139565
7 Medical Incident	94115	120087

Showing all 714 rows.

Command took 5.64 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:09:33 PM on demo-cluster

Let us look at the sixth question.

This question wants us to filter the records for two zip codes - 94102 and 94103. Then we want to see the names of unique neighbourhoods in these zip codes.

04-spark-sql-demo SQL

demo-cluster ▼ File Edit Insert Image Lock Run Help

Q6. What San Francisco neighborhoods are in the zip codes 94102 and 94103?

Cmd 16

```
1 select zipCode, neighborhood
2 from demo_db.fire_service_calls_tbl
3 where zipCode == 94102 or zipCode == 94103
```

▶ (1) Spark Jobs

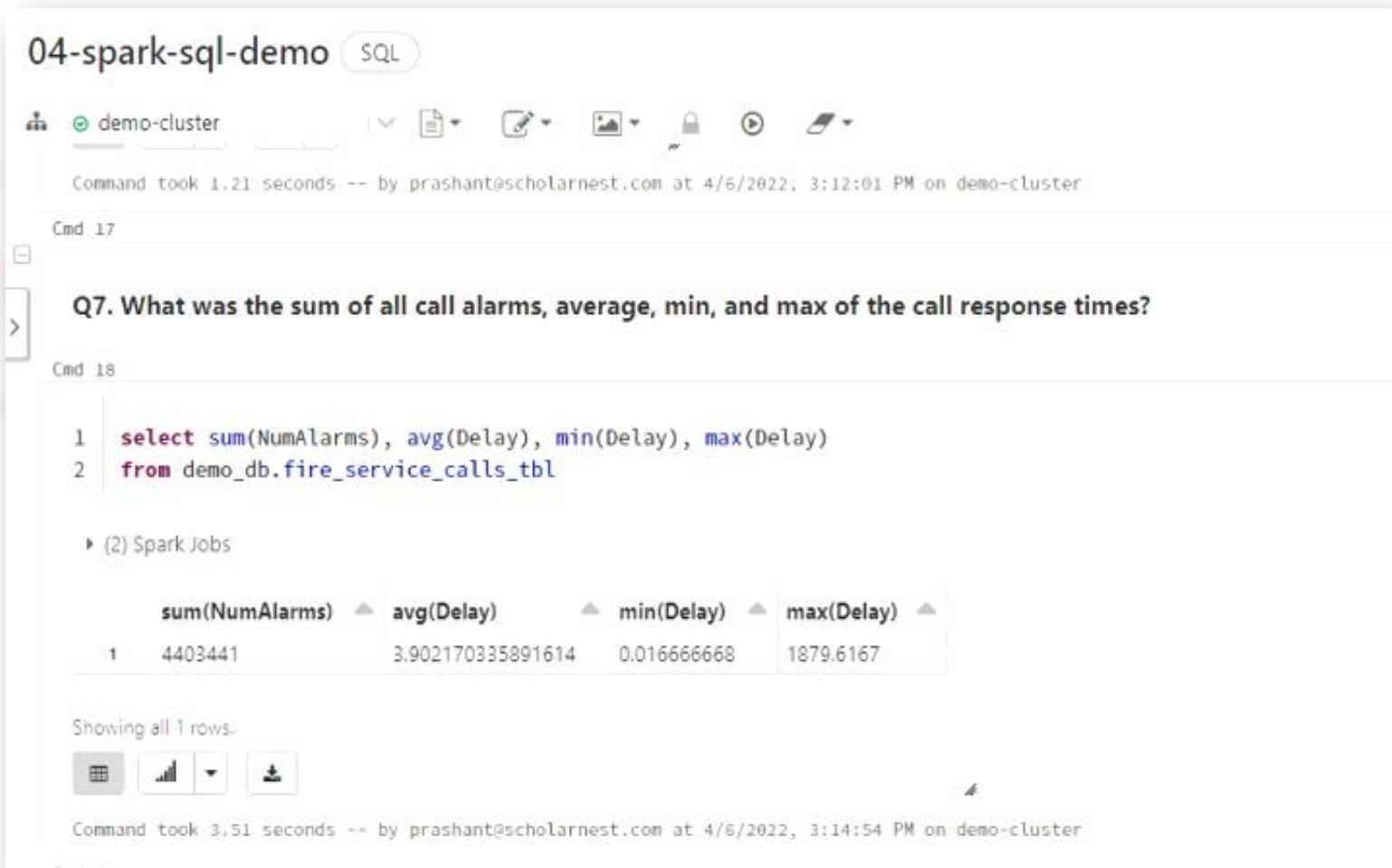
zipCode	neighborhood
1	South of Market
2	South of Market
3	Tenderloin
4	Tenderloin
5	Tenderloin
6	Tenderloin
7	Tenderloin

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 1.21 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:12:01 PM on demo-cluster

Let us look at the seventh question.

The first part of the question wants us to find the sum of *NumAlarms*. The second part of the question wants us to find avg, min, and max response time delay. And we want to find it for all the calls. So no filter and no grouping.



The screenshot shows the Databricks SQL interface with the following details:

- Project:** 04-spark-sql-demo
- Cluster:** demo-cluster
- Command:** Command took 1.21 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:12:01 PM on demo-cluster
- Query:** Q7. What was the sum of all call alarms, average, min, and max of the call response times?
- Code:**

```
1 select sum(NumAlarms), avg(Delay), min(Delay), max(Delay)
2 from demo_db.fire_service_calls_tbl
```
- Execution:** Cmd 18, (2) Spark Jobs
- Result:**

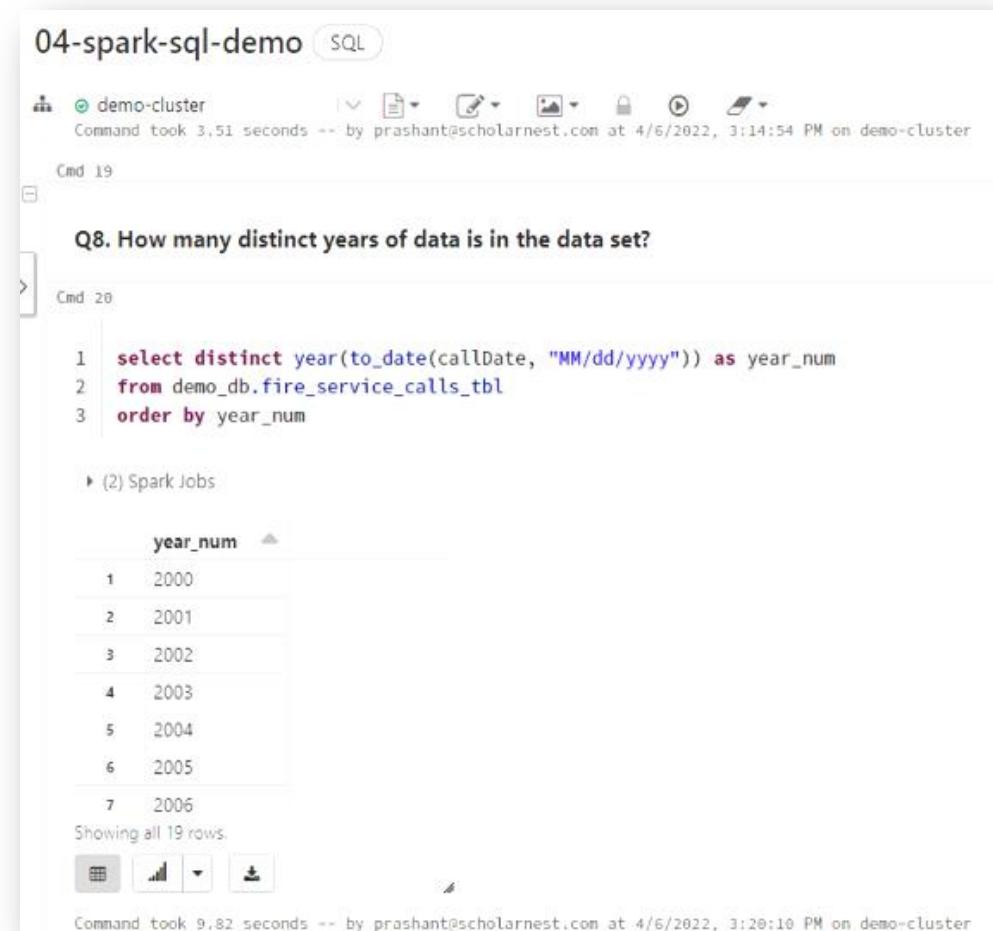
	sum(NumAlarms)	avg(Delay)	min(Delay)	max(Delay)
1	4403441	3.902170335891614	0.016666668	1879.6167

Showing all 1 rows.
- Timing:** Command took 3.51 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:14:54 PM on demo-cluster

Let us look at the eighth question.

They want us to find the number of distinct years in the data set. We have a Call Date field, and we can take out the year from that field. However, the Call Date is a string field.

But we can convert it to a date field and take out the year. Once we have the year number, we can apply a distinct clause to find unique years.



The screenshot shows a Jupyter Notebook cell titled "04-spark-sql-demo" with the "SQL" tab selected. The cell contains the following SQL query:

```
1 select distinct year(to_date(callDate, "MM/dd/yyyy")) as year_num
2 from demo_db.fire_service_calls_tbl
3 order by year_num
```

Below the query, under "(2) Spark Jobs", is a table with the following data:

year_num
1 2000
2 2001
3 2002
4 2003
5 2004
6 2005
7 2006

Text at the bottom of the cell indicates: "Showing all 19 rows."

At the very bottom of the cell, the command took 9.82 seconds.

Let us look at the ninth question.

They want us to filter the table for the year 2018. We can convert the CallDate into a date filed, take out the year and filter it for 2018. Then we can take out the week number from the CallDate and count it by week number. Once counted, you can sort it in descending order to find the most calls.

The screenshot shows a Jupyter Notebook cell titled "04-spark-sql-demo" with the "SQL" tab selected. The cell contains the following SQL query:

```
1 select weekofyear(to_date(callDate, "MM/dd/yyyy")) week_year, count(*) as count
2 from demo_db.fire_service_calls_tbl
3 where year(to_date(callDate, "MM/dd/yyyy")) == 2018
4 group by week_year
5 order by count desc
```

Below the query, there is a section titled "(2) Spark Jobs" which displays a table of results:

week_year	count
1	6401
25	6163
13	6103
22	6060
44	6048
27	6042
16	6009

At the bottom of the cell, it says "Showing all 45 rows." and includes standard Jupyter Notebook navigation icons for cell selection, execution, and file operations. The footer of the notebook cell indicates: "Command took 8.23 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:24:15 PM on demo-cluster".

Let us look at the tenth question.

We just need to filter records for the year 2018. Then look for the neighbourhoods and delay columns. And finally, sort it by delay in descending order.

04-spark-sql-demo SQL

demo-cluster

Cmd 23

Q10. What neighborhoods in San Francisco had the worst response time in 2018?

Cmd 24

```
1 select neighborhood, delay
2 from demo_db.fire_service_calls_tbl
3 where year(to_date(callDate, "MM/dd/yyyy")) = 2018
4 order by delay desc
```

(1) Spark Jobs

neighborhood	delay
1 West of Twin Peaks	754.0833
2 Mission	745.93335
3 Chinatown	734.86664
4 Bayview Hunters Point	715.76666
5 Bayview Hunters Point	714.73334
6 Bayview Hunters Point	713.05
7 Bayview Hunters Point	700

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 7.89 seconds -- by prashant@scholarnest.com at 4/6/2022, 3:26:09 PM on demo-cluster



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

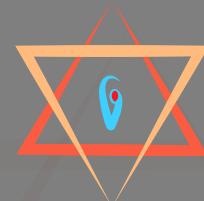
Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks



Dataframe Methods

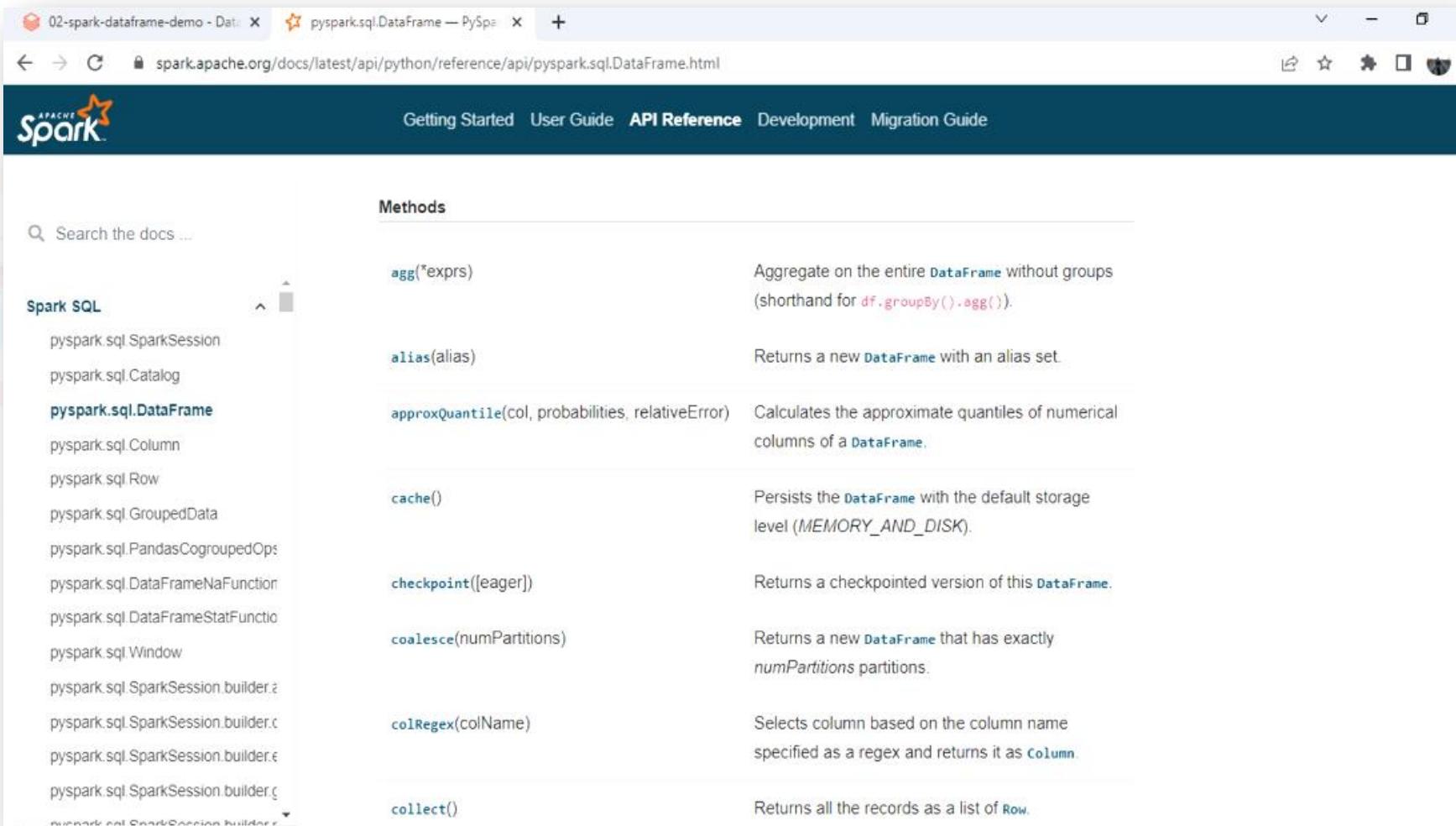
Go to your Databricks workspace and open spark-dataframe-demo notebook. (Reference: 02-spark-dataframe-demo.ipynb)

```
02-spark-dataframe-demo - DataFrames +  
← → ⌂ https://community.cloud.databricks.com/?o=946780288864704#notebook/2787702214819532/command/2909488897709251  
02-spark-dataframe-demo Python  
demo-cluster Cmd 1  
1 fire_df = spark.read \  
2     .format("csv") \  
3     .option("header", "true") \  
4     .option("inferSchema", "true") \  
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")  
Cmd 2  
▶ (2) Spark Jobs  
Command took 1.24 minutes -- by prashant@scholarnest.com at 4/6/2022, 2:34:53 PM on demo-cluster  
1 fire_df.show(10)  
Cmd 2  
▶ (1) Spark Jobs  
+---+---+---+---+---+---+---+---+---+---+  
|Call Number|Unit ID|Incident Number| CallType| Call Date|Watch Date|Call Final Disposition| Available DtTm| Address|City|Zipcode of  
Incident|Battalion|Station Area| Box|OrigPriority|Priority|Final Priority|ALS Unit|Call Type Group|NumAlarms| UnitType|Unit sequence in call dispatch|Fire  
Prevention District|Supervisor District| Neighborhood| Location| RowID| Delay|  
+---+---+---+---+---+---+---+---+---+---+  
| 20110014| M29| 2003234|Medical Incident|01/11/2002|01/10/2002| Other|01/11/2002 01:58:...| 10TH ST/MARKET ST| SF|  
94103| B02| 36|2338| 1| 1| 2| true| null| 1| MEDIC| 1|  
2| 6| Tenderloin|(37.7765408927183...|020110014-M29| 5.233333333333333|  
| 20110015| M08| 2002222|Medical Incident|01/11/2002|01/10/2002| Other|01/11/2002 02:10:...| 1-200 Block of 5TH ST| SF|
```

Now that we have created a Dataframe, we can apply Dataframe methods to process the data.

You can visit

<https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html> to get the list of all the Dataframe methods.



The screenshot shows a web browser window with the Apache Spark API Reference page for DataFrame methods. The URL in the address bar is <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html>. The page has a dark blue header with the Apache logo and navigation links for Getting Started, User Guide, API Reference (which is highlighted), Development, and Migration Guide. A sidebar on the left lists various Spark SQL classes and methods under the heading "Spark SQL". The main content area is titled "Methods" and lists several methods with their descriptions:

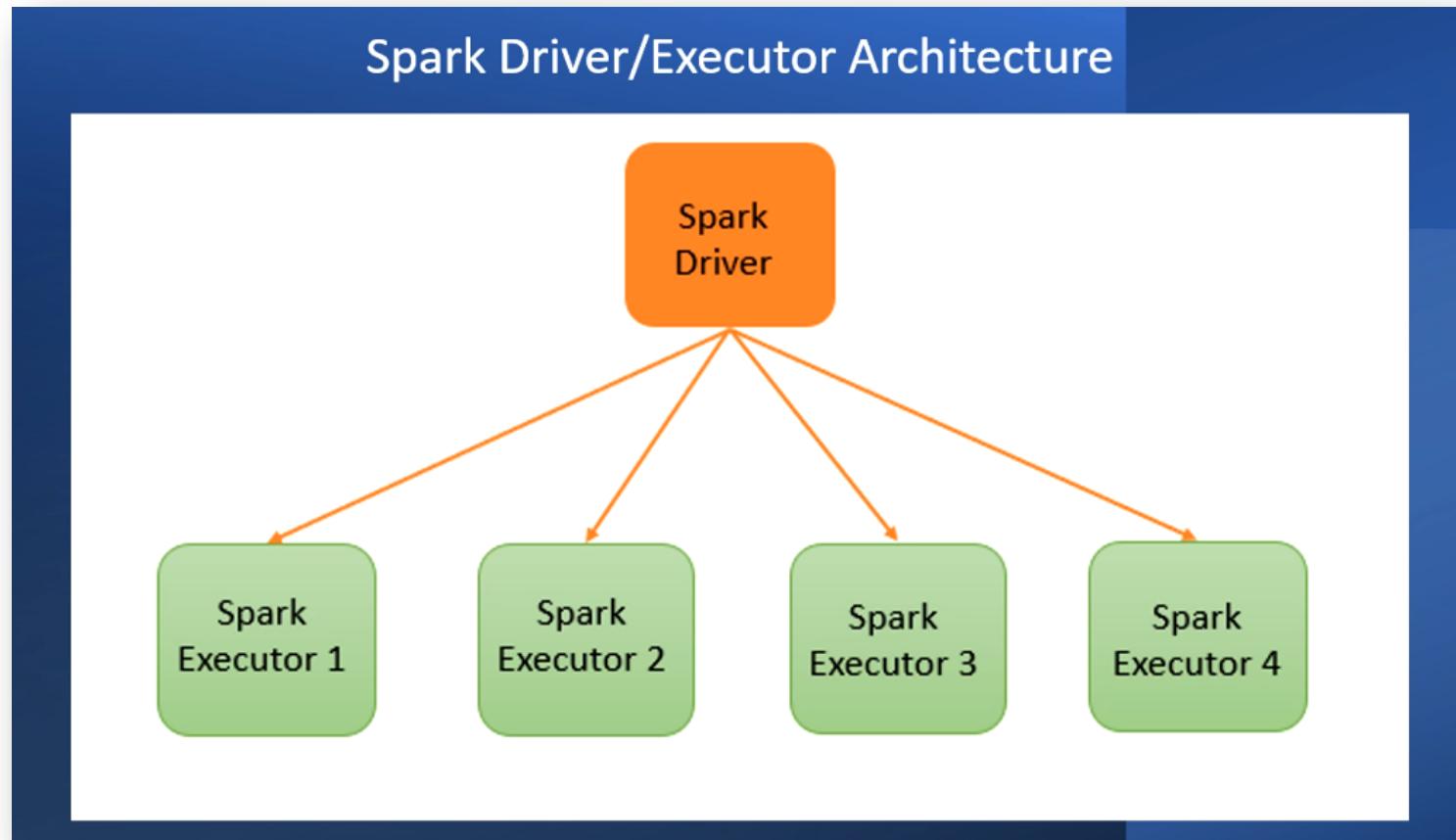
Method	Description
<code>agg(*exprs)</code>	Aggregate on the entire <code>DataFrame</code> without groups (shorthand for <code>df.groupBy().agg()</code>).
<code>alias(alias)</code>	Returns a new <code>DataFrame</code> with an alias set.
<code>approxQuantile(col, probabilities, relativeError)</code>	Calculates the approximate quantiles of numerical columns of a <code>DataFrame</code> .
<code>cache()</code>	Persists the <code>DataFrame</code> with the default storage level (<code>MEMORY_AND_DISK</code>).
<code>checkpoint([eager])</code>	Returns a checkpointed version of this <code>DataFrame</code> .
<code>coalesce(numPartitions)</code>	Returns a new <code>DataFrame</code> that has exactly <code>numPartitions</code> partitions.
<code>colRegex(colName)</code>	Selects column based on the column name specified as a regex and returns it as <code>Column</code> .
<code>collect()</code>	Returns all the records as a list of <code>Row</code> .

Dataframe offers you a long list of methods. However, we can classify them into three logical groups:

1. Actions
2. Transformations
3. Functions/Methods

Actions: Actions trigger a Spark Job and return to the Spark driver.
Spark is a distributed processing system. So every Spark program runs as one driver and one or more executors
And Actions are Dataframe operations that performs two things:

1. Kick off a Spark Job
2. Return to the Driver



Transformations: Spark Dataframe transformation produces a newly transformed Dataframe.

Transformations are different than actions. They do not kick off a Spark Job, and they do not return to the driver.

They simply create a new Dataframe and return it.

Functions and Methods : These are DataFrame methods or functions which are not categorized into Actions or Transformations.

So everything other than the Actions and Transformations is a method or function.

Actions

1. collect
2. count
3. describe
4. first
5. foreach
6. foreachPartition
7. head
8. show
9. summary
10. tail
11. take
12. toLocalIterator

Transformations

- | | | |
|--------------------|--------------------------|-----------------------|
| 1. agg | 16. intersect | 31. subtract |
| 2. alias | 17. intersectAll | 32. transform |
| 3. coalesce | 18. join | 33. union |
| 4. colRegex | 19. limit | 34. unionAll |
| 5. crossJoin | 20. orderBy | 35. unionByName |
| 6. crosstab | 21. randomSplit | 36. where |
| 7. cube | 22. repartition | 37. withColumn |
| 8. distinct | 23. repartitionByRange | 38. withColumnRenamed |
| 9. drop | 24. rollup | |
| 10. dropDuplicates | 25. sample | |
| 11. dropDuplicates | 26. sampleBy | |
| 12. dropna | 27. select | |
| 13. exceptAll | 28. selectExpr | |
| 14. filter | 29. sort | |
| 15. groupby | 30. sortWithinPartitions | |

Here is a list of Dataframe Functions and Methods.

We have 20 functions and methods. These are utility functions or methods.

Functions/Methods

1. approxQuantile
2. cache
3. checkpoint
4. createGlobalTempView
5. createOrReplaceGlobalTempView
6. createOrReplaceTempView
7. createTempView
8. explain
9. hint
10. inputFiles
11. isLocal
12. localCheckpoint
13. persist
14. printSchema
15. registerTempTable
16. toDF
17. toJSON
18. unpersist
19. writeTo
20. withWatermark

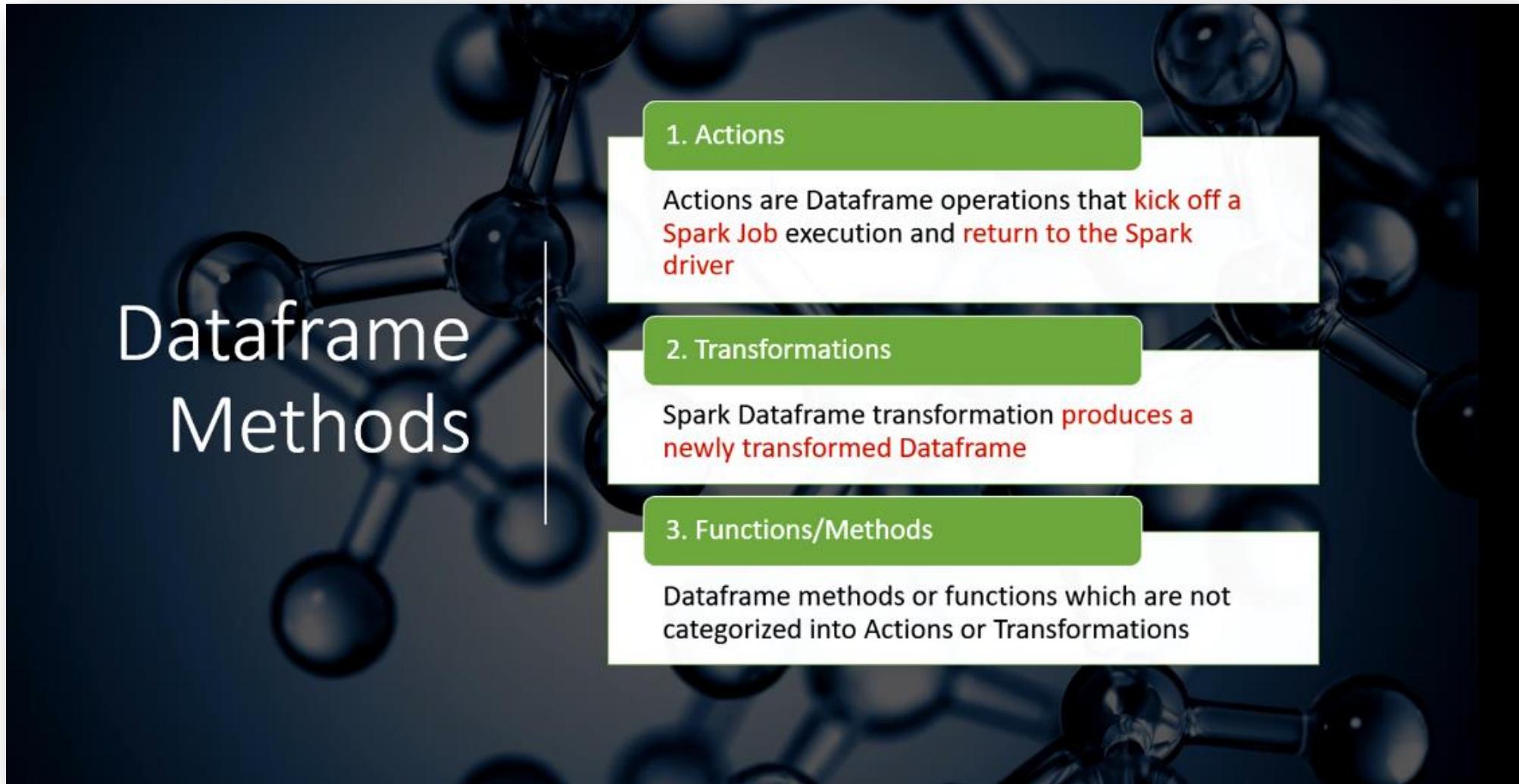
There are ten more functions that I excluded from the list that I have highlighted in the image below. I have excluded these because those functions are only available in Python and not in other languages, and it is not frequently used.

Functions/Methods

- 1. approxQuantile
- 2. cache
- 3. checkpoint
- 4. createGlobalTempView
- 5. createOrReplaceGlobalTempView
- 6. createOrReplaceTempView
- 7. createTempView
- 8. explain
- 9. hint
- 10. inputFiles
- 11. isLocal
- 12. localCheckpoint
- 13. persist
- 14. printSchema
- 15. registerTempTable
- 16. toDF
- 17. toJSON
- 18. unpersist
- 19. writeTo
- 20. withWatermark

- 1. corr
- 2. cov
- 3. freqItems
- 4. mapInPandas
- 5. replace
- 6. sameSemantics
- 7. semanticHash
- 8. to_koalas
- 9. to_pandas_on_spark
- 10. toPandas

Finally, here is the definition of all the three types of Dataframe Methods.





Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey

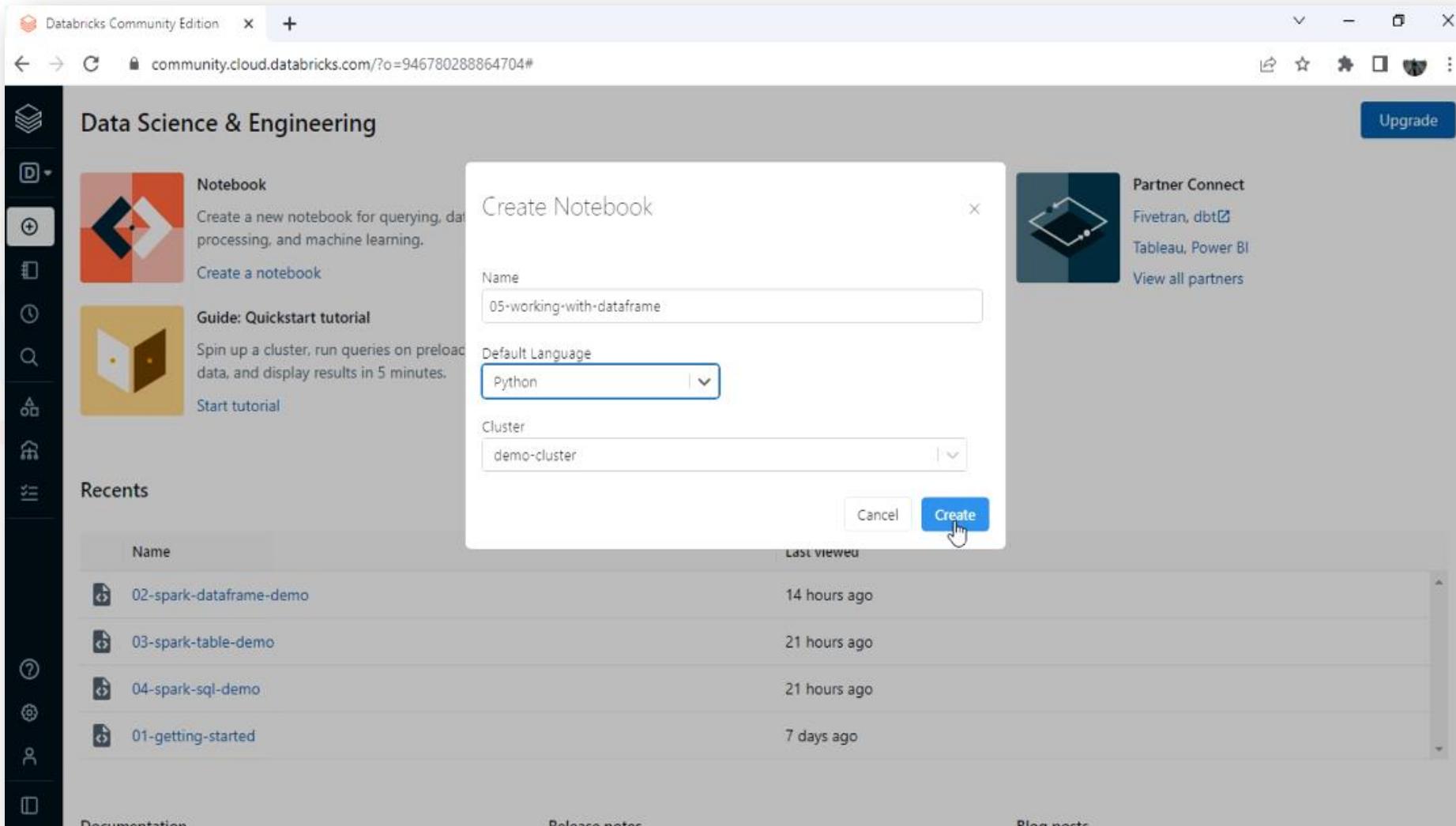


Absolute Beginner to Specialization in Apache Spark and Azure Databricks

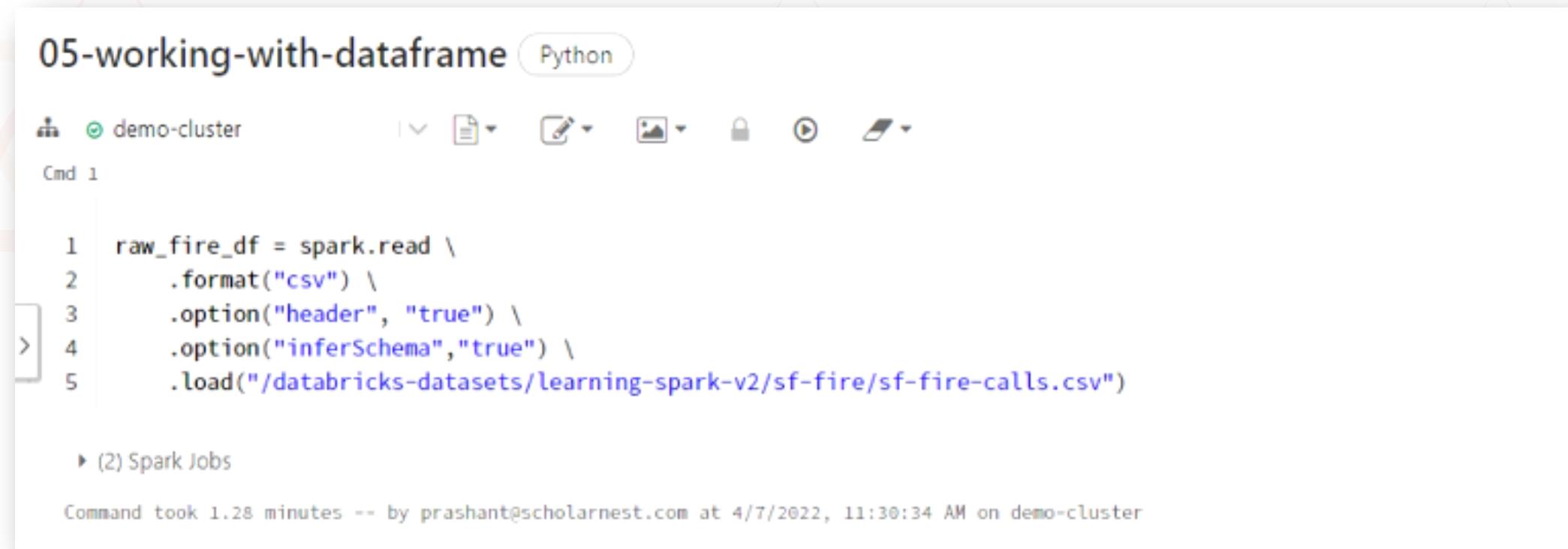


Applying Dataframe Transformations

Go to your Databricks workspace and create a new notebook as given below. (**Reference: 05-working-with-dataframe.ipynb**)



Here we the older code which we used earlier to create a data frame for the fire call response data set.



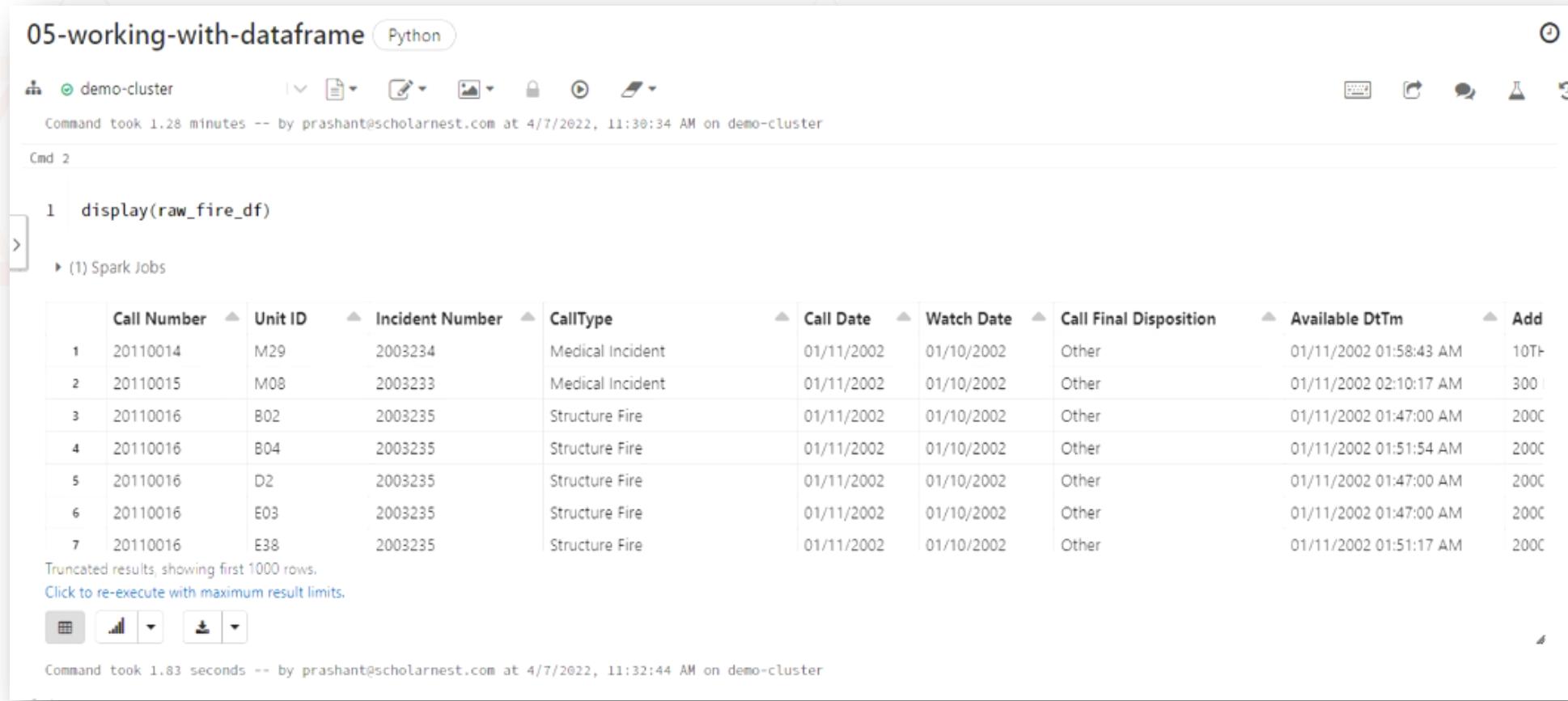
The screenshot shows a Databricks notebook interface. The title bar says "05-working-with-dataframe" and "Python". The code editor contains the following Python code:

```
1 raw_fire_df = spark.read \
2     .format("csv") \
3     .option("header", "true") \
4     .option("inferSchema","true") \
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

Below the code, it says "(2) Spark Jobs". At the bottom, there is a timestamp: "Command took 1.28 minutes -- by prashant@scholarnest.com at 4/7/2022, 11:30:34 AM on demo-cluster".

If we display the older `raw_fire_df` Dataframe and see the results as given below. We can see there are two problems in this Dataframe:

1. Column Names are not standardized
2. Date fields are of string type



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" running on a "demo-cluster". The cell contains the command `display(raw_fire_df)`. Below the command, it says "(1) Spark Jobs". The resulting DataFrame is displayed as a table with the following columns and data:

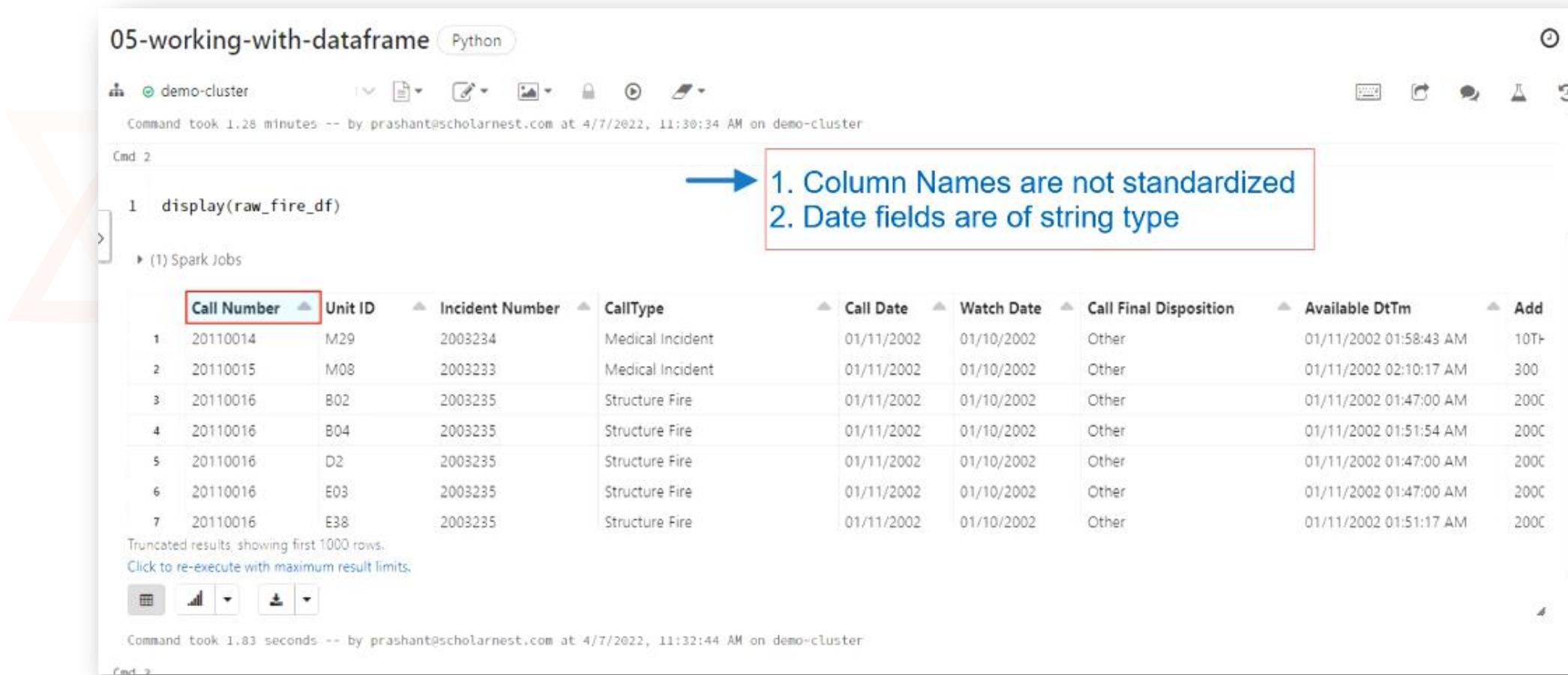
	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	200C

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 1.83 seconds -- by prashant@scholarnest.com at 4/7/2022, 11:32:44 AM on demo-cluster

Column Names are not standardized:

You can see a blank space between call and number. That's not standard. Spark Dataframe column names are case insensitive. However, space in a column name is not a common thing. Let me fix these column names and rename them to remove spaces.



05-working-with-dataframe Python

demo-cluster

Command took 1.28 minutes -- by prashant@scholarnest.com at 4/7/2022, 11:30:34 AM on demo-cluster

Cmd 2

```
1 display(raw_fire_df)
```

→ 1. Column Names are not standardized
2. Date fields are of string type

(1) Spark Jobs

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	200C

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

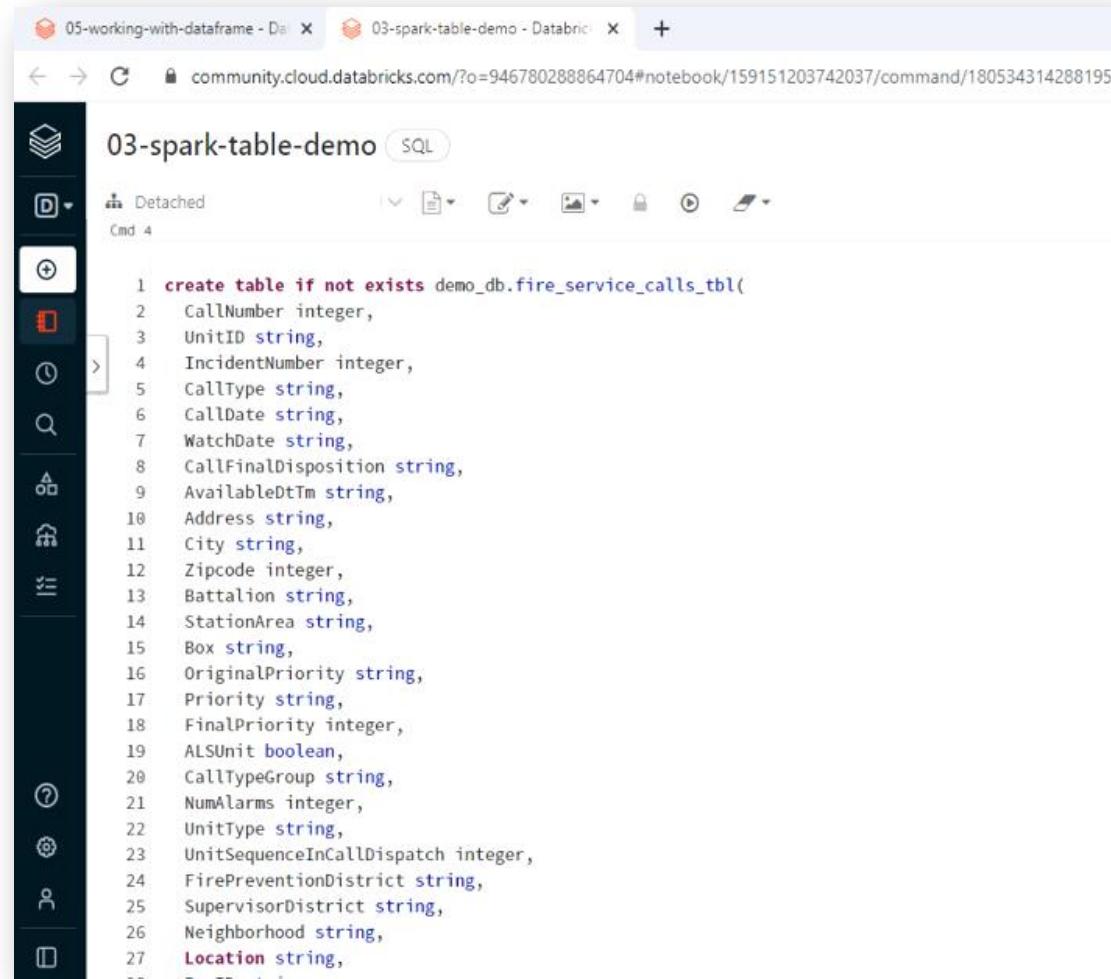
Command took 1.83 seconds -- by prashant@scholarnest.com at 4/7/2022, 11:32:44 AM on demo-cluster

Cmd 2

Column Names are not standardized:

We fixed this problem while creating the table. (**Reference: 03-spark-table-demo.sql**)

You can see in the image below that we created column names without spaces when we created the table. Then we loaded data into this table. So we already fixed the column names while creating a table.



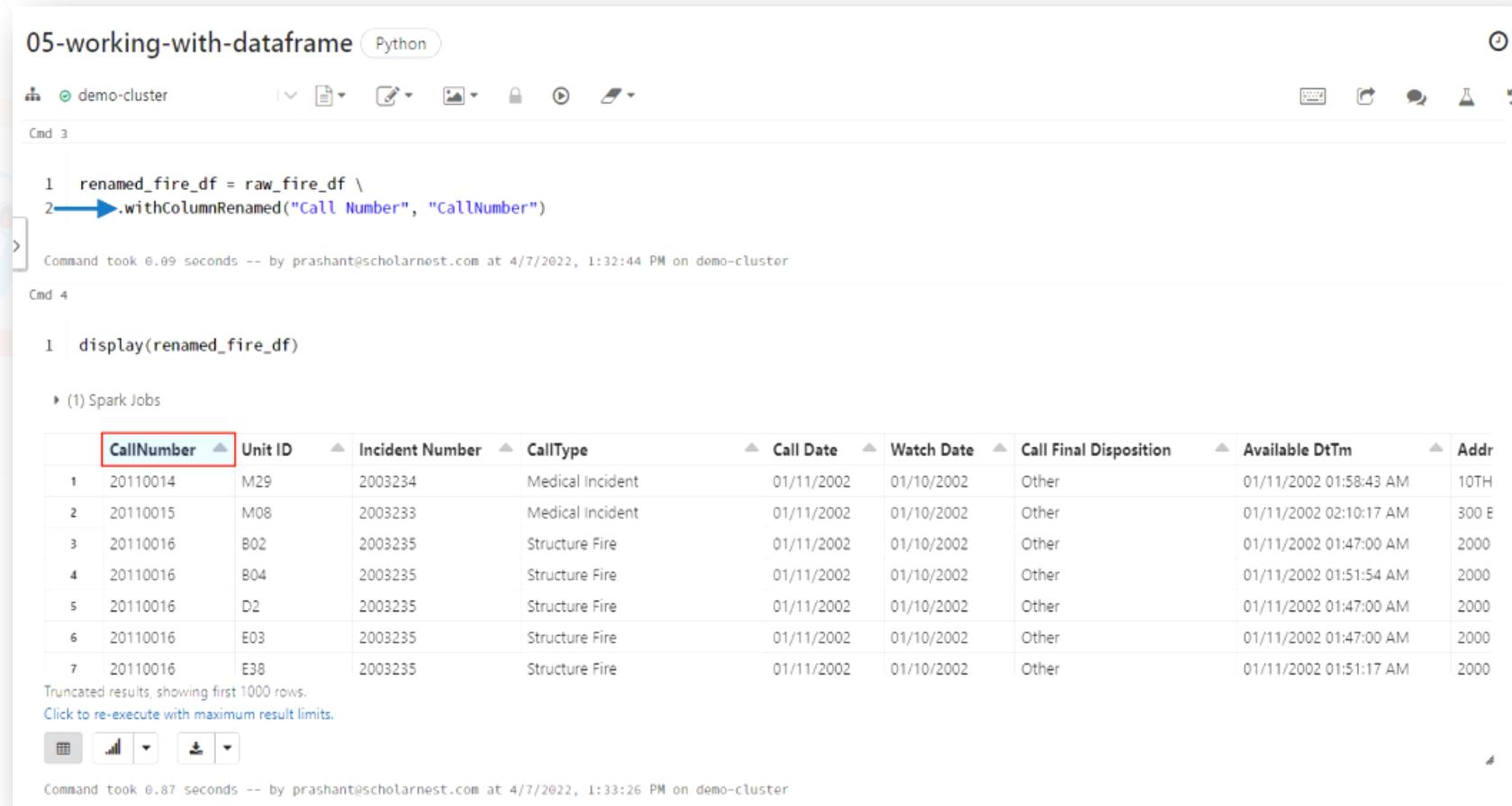
The screenshot shows the Databricks SQL interface. The left sidebar has a dark theme with icons for file operations, a search bar, and a notebook icon. The main area is titled "03-spark-table-demo" and shows a SQL command being run. The command is as follows:

```
1 create table if not exists demo_db.fire_service_calls_tbl(
2   CallNumber integer,
3   UnitID string,
4   IncidentNumber integer,
5   CallType string,
6   CallDate string,
7   WatchDate string,
8   CallFinalDisposition string,
9   AvailableDtM string,
10  Address string,
11  City string,
12  Zipcode integer,
13  Battalion string,
14  StationArea string,
15  Box string,
16  OriginalPriority string,
17  Priority string,
18  FinalPriority integer,
19  ALSUnit boolean,
20  CallTypeGroup string,
21  NumAlarms integer,
22  UnitType string,
23  UnitSequenceInCallDispatch integer,
24  FirePreventionDistrict string,
25  SupervisorDistrict string,
26  Neighborhood string,
27  Location string,
28  RowID integer)
```

Column Names are not standardized:

Here is how we can fix the column names while working on Dataframes. (**Reference: 05-working-with-dataframe.ipynb**)

Spark Dataframe offers you withColumnRenamed() transformation to rename a column. You can see in the screenshot below that withColumnRenamed() takes two arguments: Old column name and New column name. And we can see the desired results when we are displaying the Dataframe below.



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" in Python mode. The cell contains the following code:

```
1 renamed_fire_df = raw_fire_df \
2 .withColumnRenamed("Call Number", "CallNumber")
```

The second line of code, which renames the column, is highlighted with a blue arrow pointing to the ".withColumnRenamed" method. Below the code, a message indicates the command took 0.09 seconds. The cell then displays the result of the `display` command:

```
1 display(renamed_fire_df)
```

结果显示了一个名为 "(1) Spark Jobs" 的表格，列名包括 CallNumber, Unit ID, Incident Number, CallType, Call Date, Watch Date, Call Final Disposition, Available DtM, 和 Addr。CallNumber 列被高亮显示并选中。表格数据如下：

	CallNumber	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtM	Addr
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300 E
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	2000
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	2000
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	2000
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	2000
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	2000

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 6.87 seconds -- by prashant@scholarnest.com at 4/7/2022, 1:33:26 PM on demo-cluster

Column Names are not standardized:

Similarly, we can rename all the problem columns as shown in the code below. If we run this code and display the results, our first problem is resolved and our column names are standardized.



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" in Python mode. The cell contains the following code, which renames 16 columns in a DataFrame:

```
1 renamed_fire_df = raw_fire_df \
2     .withColumnRenamed("Call Number", "CallNumber") \
3     .withColumnRenamed("Unit ID", "UnitID") \
4     .withColumnRenamed("Incident Number", "IncidentNumber") \
5     .withColumnRenamed("Call Date", "CallDate") \
6     .withColumnRenamed("Watch Date", "WatchDate") \
7     .withColumnRenamed("Call Final Disposition", "CallFinalDisposition") \
8     .withColumnRenamed("Available DtTm", "AvailableDtTm") \
9     .withColumnRenamed("Zipcode of Incident", "Zipcode") \
10    .withColumnRenamed("Station Area", "StationArea") \
11    .withColumnRenamed("Final Priority", "FinalPriority") \
12    .withColumnRenamed("ALS Unit", "ALSUnit") \
13    .withColumnRenamed("Call Type Group", "CallTypeGroup") \
14    .withColumnRenamed("Unit sequence in call dispatch", "UnitSequenceInCallDispatch") \
15    .withColumnRenamed("Fire Prevention District", "FirePreventionDistrict") \
16    .withColumnRenamed("Supervisor District", "SupervisorDistrict")
```

The code is highlighted with a red box around the renaming logic. Below the code, the command took 0.28 seconds to execute.

Spark Dataframe programming is all about doing two things:

1. Read your raw data into a Dataframe.
2. Transform your raw Dataframe and create a new transformed Dataframe.

We can also narrow down 3 key observations from what we have learned till now:

1. You can create a chain of Spark transformation methods one after the other.
2. Spark transformation returns a new Dataframe after transforming the old Dataframe.
3. Spark Dataframe is immutable. We cannot change or modify an existing Dataframe. Instead, we transform the existing Dataframe and create a new Dataframe.

Now let us move ahead to the second problem.

You can see call date, watch date, and AvailableDtTm fields in the data set. The call date and watch date represent a date. And the AvailableDtTm represents a timestamp. However, these fields are of the type string, which is not desirable.

05-working-with-dataframe Python

demo-cluster

Command took 0.28 seconds -- by prashant@scholarnest.com at 4/7/2022, 1:37:17 PM on demo-cluster

Cmd 4

```
1 display(raw_fire_df)
```

▶ (1) Spark Jobs

1. Column Names are not standardized
2. Date fields are of string type

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:17 AM	200C

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 0.56 seconds -- by prashant@scholarnest.com at 4/7/2022, 1:41:01 PM on demo-cluster

Cmd 5

Date fields are of string type:

You can see the data type information using the `printSchema()` utility method. We have used this method on `remaned_fire_df` as we have transformed the older Dataframe into this new Dataframe by removing the problem column names. You can see that all the three fields are of the string type. Let me fix this problem.

05-working-with-dataframe Python

demo-cluster UMA 6

```
1 renamed_fire_df.printSchema()

root
|-- CallNumber: integer (nullable = true)
|-- UnitID: string (nullable = true)
|-- IncidentNumber: integer (nullable = true)
|-- CallType: string (nullable = true)
|-- CallDate: string (nullable = true)  
|-- WatchDate: string (nullable = true)  
|-- CallFinalDisposition: string (nullable = true)
|-- AvailableDtTm: string (nullable = true)   ←
|-- Address: string (nullable = true)
|-- City: string (nullable = true)
|-- Zipcode: integer (nullable = true)
|-- Battalion: string (nullable = true)
|-- StationArea: string (nullable = true)
|-- Box: string (nullable = true)
|-- OrigPriority: string (nullable = true)
|-- Priority: string (nullable = true)
|-- FinalPriority: integer (nullable = true)
|-- ALSUnit: boolean (nullable = true)
|-- CallTypeGroup: string (nullable = true)
|-- NumAlarms: integer (nullable = true)
```

Command took 0.04 seconds -- by prashant@scholarnest.com at 4/7/2022, 1:44:08 PM on demo-cluster

Date fields are of string type:

We had this problem while working on spark-SQL-demo notebook. (**Reference: 04-spark-sql-demo.sql**)

You can see the solution of question number 8, that we converted the Call Date to a date filed. Then we applied the year() function. So, if your date and timestamp fields are stored in a string column, you may have to convert it to date on almost every use of the column.

The screenshot shows a Jupyter Notebook interface with the title '04-spark-sql-demo'. The notebook has a detached tab bar. The first cell, 'Cmd 19', contains the question 'Q8. How many distinct years of data is in the data set?'. The second cell, 'Cmd 20', contains the following SQL code:

```
1 select distinct year(to_date(callDate, "MM/dd/yyyy")) as year_num
2 from demo_db.fire_service_calls_tbl
3 order by year_num
```

An arrow points to the 'year' function in the code. Below the code, under '(2) Spark Jobs', is a table with the following data:

year_num
1 2000
2 2001
3 2002
4 2003
5 2004
6 2005
7 2006

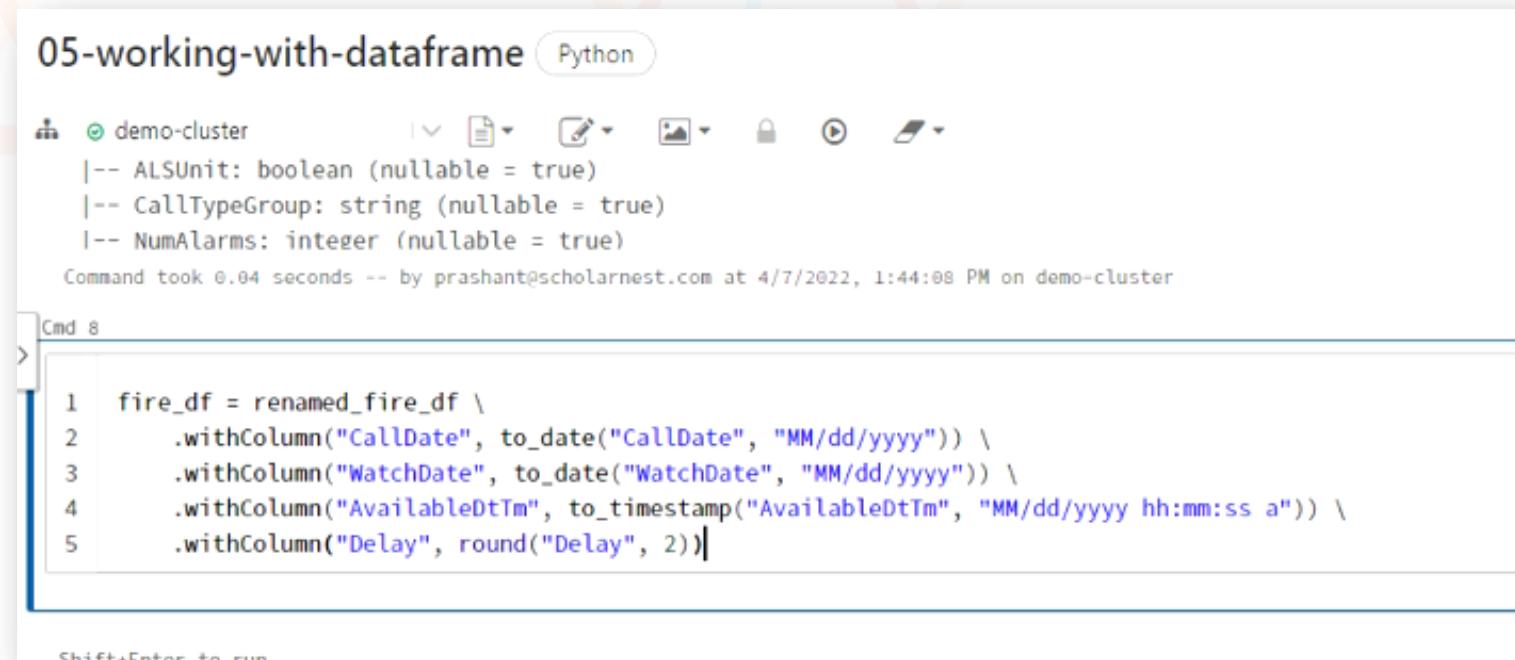
At the bottom of the table, it says 'Showing all 19 rows.' The footer of the notebook cell displays the command took 9.82 seconds and was run by prashant@scholarnest.com at 4/6/2022, 3:20:10 PM on demo-cluster.

Date fields are of string type:

You can rectify this issue by transforming the `renamed_fire_df` to a new Dataframe `fire_df` by applying the `.withColumn()` transformation. The `withColumn` method takes two arguments:

1. The first column is the one you want to transform. I want to transform the Call Date column, so I give the column name here.
2. The second argument is the logic to transform the given column. We want to convert the Call Date to a Date. So I am using the `to_date()` function. And the Call Date column in the `renamed_fire_df` is represented using MM/dd/yyyy string. Similarly, we have the corresponding logic for AvailableDtTm.

Make sure you import the `to_date()` and `to_timestamp()` function from the `pyspark.sql.functions` package. We are also transforming the Delay column to round the Delay column for two digits.



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" in Python mode. The cell displays the schema of a DataFrame named `fire_df`, which includes columns `ALSUnit` (boolean), `CallTypeGroup` (string), and `NumAlarms` (integer). Below the schema, a command history entry shows the execution of the code. The code itself is as follows:

```
1 fire_df = renamed_fire_df \
2     .withColumn("CallDate", to_date("CallDate", "MM/dd/yyyy")) \
3     .withColumn("WatchDate", to_date("WatchDate", "MM/dd/yyyy")) \
4     .withColumn("AvailableDtTm", to_timestamp("AvailableDtTm", "MM/dd/yyyy hh:mm:ss a")) \
5     .withColumn("Delay", round("Delay", 2))
```

A tooltip at the bottom of the cell says "Shift+Enter to run".

We can print the schema of the new fire_df Dataframe and see the data types. And we see the desired results in the output, the date and timestamp transformation is successful.

05-working-with-dataframe Python

demo-cluster

```
1 fire_df.printSchema()

root
|-- CallNumber: integer (nullable = true)
|-- UnitID: string (nullable = true)
|-- IncidentNumber: integer (nullable = true)
|-- CallType: string (nullable = true)
|-- CallDate: date (nullable = true)
|-- WatchDate: date (nullable = true)
|-- CallFinalDisposition: string (nullable = true)
|-- AvailableDtTm: timestamp (nullable = true) ←
|-- Address: string (nullable = true)
|-- City: string (nullable = true)
|-- Zipcode: integer (nullable = true)
|-- Battalion: string (nullable = true)
|-- StationArea: string (nullable = true)
|-- Box: string (nullable = true)
|-- OrigPriority: string (nullable = true)
|-- Priority: string (nullable = true)
|-- FinalPriority: integer (nullable = true)
|-- ALSUnit: boolean (nullable = true)
|-- CallTypeGroup: string (nullable = true)
|-- NumAlarms: integer (nullable = true)
```

Command took 0.03 seconds -- by prashant@scholarnest.com at 4/7/2022, 3:18:42 PM on demo-cluster

We learned about three new methods in this lecture.

Column Transformation Methods:

1. `withColumnRenamed`
2. `withColumn`

Utility Method:

1. `printSchema()`



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Working with Spark Dataframe – Part I

Go to your Databricks workspace and open the notebook we created in the previous lecture. (**Reference: 05-working-with-dataframe.ipynb**)

05-working-with-dataframe - Da x +

community.cloud.databricks.com/?o=946780288864704#notebook/384359114294295/command/384359114294308

05-working-with-dataframe Python

demo-cluster

Cmd 1

```
1 from pyspark.sql.functions import *
```

Command took 0.04 seconds -- by prashant@scholarnest.com at 4/7/2022, 3:15:00 PM on demo-cluster

Cmd 2

```
1 raw_fire_df = spark.read \
2     .format("csv") \
3     .option("header", "true") \
4     .option("inferSchema","true") \
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

▶ (2) Spark Jobs

Command took 1.28 minutes -- by prashant@scholarnest.com at 4/7/2022, 11:30:34 AM on demo-cluster

Cmd 3

```
1 display(raw_fire_df)
```

▶ (1) Spark Jobs

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtM	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B01	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C

We have transformed the created a Dataframe fire_df which is ready for analysis. You can see the fire_df Dataframe here in the screenshot below.

05-working-with-dataframe Python

```
demo-cluster
4     .withColumn("AvailableDtTm", to_timestamp("AvailableDtTm", "MM/dd/yyyy hh:mm:ss a")) \
5     .withColumn("Delay", round("Delay", 2))
```

Command took 0.15 seconds -- by prashant@scholarnest.com at 4/7/2022, 3:16:56 PM on demo-cluster

Cmd 8

```
1 display(fire_df)
```

▶ (1) Spark Jobs

	CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm
1	20110014	M29	2003234	Medical Incident	2002-01-11	2002-01-10	Other	2002-01-11T01:58:43.000+0000
2	20110015	M08	2003233	Medical Incident	2002-01-11	2002-01-10	Other	2002-01-11T02:10:17.000+0000
3	20110016	B02	2003235	Structure Fire	2002-01-11	2002-01-10	Other	2002-01-11T01:47:00.000+0000
4	20110016	B04	2003235	Structure Fire	2002-01-11	2002-01-10	Other	2002-01-11T01:51:54.000+0000
5	20110016	D2	2003235	Structure Fire	2002-01-11	2002-01-10	Other	2002-01-11T01:47:00.000+0000
6	20110016	E03	2003235	Structure Fire	2002-01-11	2002-01-10	Other	2002-01-11T01:47:00.000+0000
7	20110016	E38	2003235	Structure Fire	2002-01-11	2002-01-10	Other	2002-01-11T01:51:17.000+0000

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 1.98 seconds -- by prashant@scholarnest.com at 4/7/2022, 3:17:24 PM on demo-cluster

We can document all the 10 micro project question in this notebook along with it's solution SQL expression which we have already seen in the earlier lectures.

The screenshot shows a Jupyter Notebook interface with the title "05-working-with-dataframe" and a Python tab selected. The notebook contains the following content:

- Q1. How many distinct types of calls were made to the Fire Department?**

```
select count(distinct CallType) as distinct_call_type_count
from fire_service_calls_tbl
where CallType is not null
```
- Cmd 11**
1
- Q2. What were distinct types of calls made to the Fire Department?**

```
select distinct CallType as distinct_call_types
from fire_service_calls_tbl
where CallType is not null
```
- Cmd 13**
1
- Q3. Find out all response for delayed times greater than 5 mins?**

Before I start analysis, let me cache the Dataframe. You can use the cache() utility method to cache the Dataframe into memory. Because I want to run ten analysis transformations on the same Dataframe.

The screenshot shows a Jupyter Notebook interface with a single code cell. The cell title is "05-working-with-dataframe" and the language is "Python". The cell content is:

```
1 fire_df.cache()
```

The output of the cell is:

```
Out[12]: DataFrame[CallNumber: int, UnitID: string, IncidentNumber: int, CallType: string, CallDate: date, WatchDate: date, CallFinalDisposition: string, AvailableDtTm: timestamp, Address: string, City: string, Zipcode: int, Battalion: string, StationArea: string, Box: string, OrigPriority: string, Priority: string, FinalPriority: int, ALSUnit: boolean, CallTypeGroup: string, NumAlarms: int, UnitType: string, UnitSequenceInCallDispatch: int, FirePreventionDistrict: string, SupervisorDistrict: string, Neighborhood: string, Location: string, RowID: string, Delay: double]
```

Below the output, a note states: "Command took 0.15 seconds -- by prashant@scholarnest.com at 4/7/2022, 9:13:44 PM on demo-cluster".

We already know the logic behind the 10 question as we implemented the same using SQL. But SQL runs on a table. And we want to learn how to run those same 10 logic on a Dataframe.

We have two approaches:

1. SQL Approach
2. Dataframe Transformation Approach

Further, the SQL approach is a two-step process:

1. Convert your Dataframe to a temporary view
2. Run your SQL on the view

Let me proceed with the SQL approach, the first thing is to convert my Dataframe to a temporary view. So I am using the same SQL that I used on the table. But I changed the table name to the view name. Now, the second step is to run SQL queries on the view. The second step will give you a Dataframe.

The screenshot shows a Jupyter Notebook interface with a single cell titled "05-working-with-dataframe". The cell contains the following code:

```
Q1. How many distinct types of calls were made to the Fire Department?  
select count(distinct CallType) as distinct_call_type_count  
from fire_service_calls_tbl  
where CallType is not null  
  
Cmd 12  
1 fire_df.createOrReplaceTempView("fire_service_calls_view")  
2 q1_sql_df = spark.sql("""  
3     select count(distinct CallType) as distinct_call_type_count  
4     from fire_service_calls_view  
5     where CallType is not null  
6     """)  
7 display(q1_sql_df)
```

Below the code, there is a section titled "(3) Spark Jobs" which displays a table:

distinct_call_type_count
1 32

At the bottom of the cell, it says "Showing all 1 rows."

At the very bottom of the notebook, a status bar indicates: "Command took 2.46 minutes -- by prashant@scholarnest.com at 4/7/2022, 9:27:23 PM on demo-cluster".

Now let us see the Dataframe Transformation approach. Here we have the logic for this approach given below. We can build the logic easily just by looking at SQL solutions.

The screenshot shows a Jupyter Notebook interface with the title "05-working-with-dataframe" and a Python tab selected. The notebook has three cells:

- Cmd 11:** A question "Q1. How many distinct types of calls were made to the Fire Department?" followed by an SQL query:

```
select count(distinct CallType) as distinct_call_type_count
from fire_service_calls_tbl
where CallType is not null
```
- Cmd 12:** Python code to create a temporary view and run the same SQL query:

```
1 fire_df.createOrReplaceTempView("fire_service_calls_view")
2 q1_sql_df = spark.sql("""
3     select count(distinct CallType) as distinct_call_type_count
4     from fire_service_calls_view
5     where CallType is not null
6     """)
7 display(q1_sql_df)
```
- Cmd 13:** A "Show result" button.

A red box highlights the steps in the Cmd 12 code, which correspond to the numbered steps in the list below:

1. Filter the records and take only those where CallType is not null
2. Select the CallType column
3. Take only distinct call types
4. Show the count

Here is the code for the Dataframe Transformation approach. We got the same desired output with this approach as well.

05-working-with-dataframe Python

demo-cluster Show result

Cmd 13

```
1 q1_df = fire_df.where("CallType is not null") \
2     .select("CallType") \
3     .distinct()
4 print(q1_df.count())
```

▶ (3) Spark Jobs

32

Command took 2.00 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:34:24 AM on demo-cluster

We can also write the code for the Dataframe approach in the manner shown below. Both the approach will give you the same answer, but it is recommended to follow the first approach shown in the image.

05-working-with-dataframe Python

demo-cluster Show result

Cmd 13

```
1 q1_df = fire_df.where("CallType is not null") \
    .select("CallType") \
    .distinct()
4 print(q1_df.count())
```

▶ (3) Spark Jobs

32

Command took 2.00 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:34:24 AM on demo-cluster

Cmd 14

```
1 q1_df1 = fire_df.where("CallType is not null")
2 q1_df2 = q1_df1.select("CallType")
3 q1_df3 = q1_df2.distinct()
4 print(q1_df3.count())
```

▶ (3) Spark Jobs

32

Command took 2.79 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:37:20 AM on demo-cluster

We can also write the code for the Dataframe approach in the manner shown below. Both the approach will give you the same answer, but it is recommended to follow the first approach shown in the image.

05-working-with-dataframe Python

demo-cluster Show result

Cmd 13

```
1 q1_df = fire_df.where("CallType is not null") \
    .select("CallType") \
    .distinct()
4 print(q1_df.count())
```

▶ (3) Spark Jobs

32

Command took 2.00 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:34:24 AM on demo-cluster

Cmd 14

```
1 q1_df1 = fire_df.where("CallType is not null")
2 q1_df2 = q1_df1.select("CallType")
3 q1_df3 = q1_df2.distinct()
4 print(q1_df3.count())
```

▶ (3) Spark Jobs

32

Command took 2.79 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:37:20 AM on demo-cluster



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com



ScholarNest

Spark Azure Databricks

Databricks Spark Certification and beyond

Instructor: Prashant Kumar Pandey



Absolute Beginner to Specialization in Apache Spark and Azure Databricks





Working with Spark Dataframe – Part II

Go to your Databricks workspace and open the notebook we worked with in the previous lecture. (**Reference: 05-working-with-dataframe.ipynb**)

05-working-with-dataframe - Da x +

community.cloud.databricks.com/?o=946780288864704#notebook/384359114294295/command/384359114294308

05-working-with-dataframe Python

Cmd 1

```
1 from pyspark.sql.functions import *
```

Command took 0.04 seconds -- by prashant@scholarnest.com at 4/7/2022, 3:15:00 PM on demo-cluster

Cmd 2

```
1 raw_fire_df = spark.read \
2     .format("csv") \
3     .option("header", "true") \
4     .option("inferSchema","true") \
5     .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

▶ (2) Spark Jobs

Command took 1.28 minutes -- by prashant@scholarnest.com at 4/7/2022, 11:30:34 AM on demo-cluster

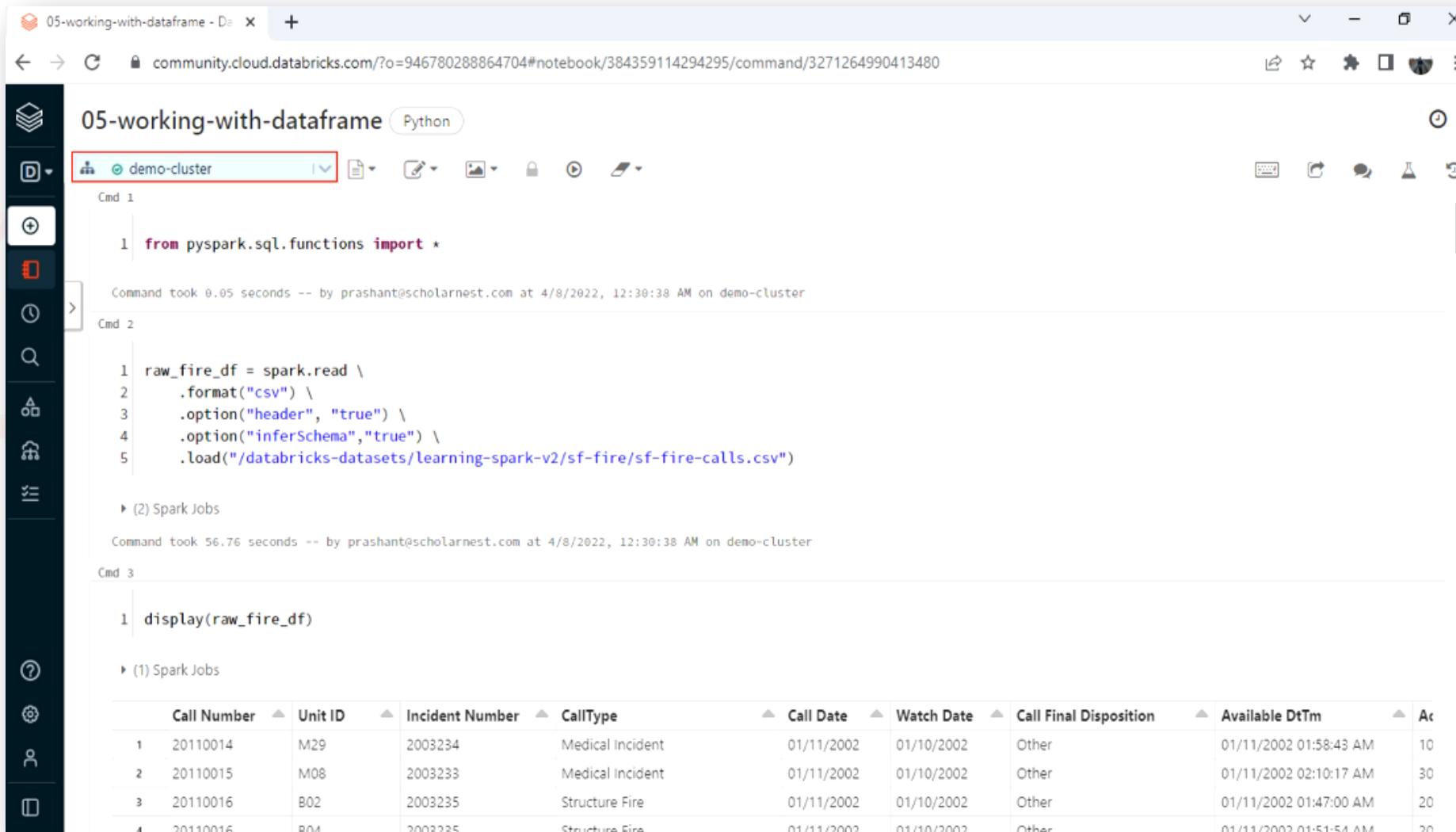
Cmd 3

```
1 display(raw_fire_df)
```

▶ (1) Spark Jobs

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtMm	Add
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10TH
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	300
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	200C
4	20110016	B01	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	200C

Make sure you have attached a running cluster to your notebook, and run the entire notebook once before you start working on this notebook further.



05-working-with-dataframe - D + New

community.cloud.databricks.com/?o=946780288864704#notebook/384359114294295/command/3271264990413480

05-working-with-dataframe Python

demo-cluster

Cmd 1

```
1 from pyspark.sql.functions import *
```

Command took 0.05 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:30:38 AM on demo-cluster

Cmd 2

```
1 raw_fire_df = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema","true") \
    .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

▶ (2) Spark Jobs

Command took 56.76 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:30:38 AM on demo-cluster

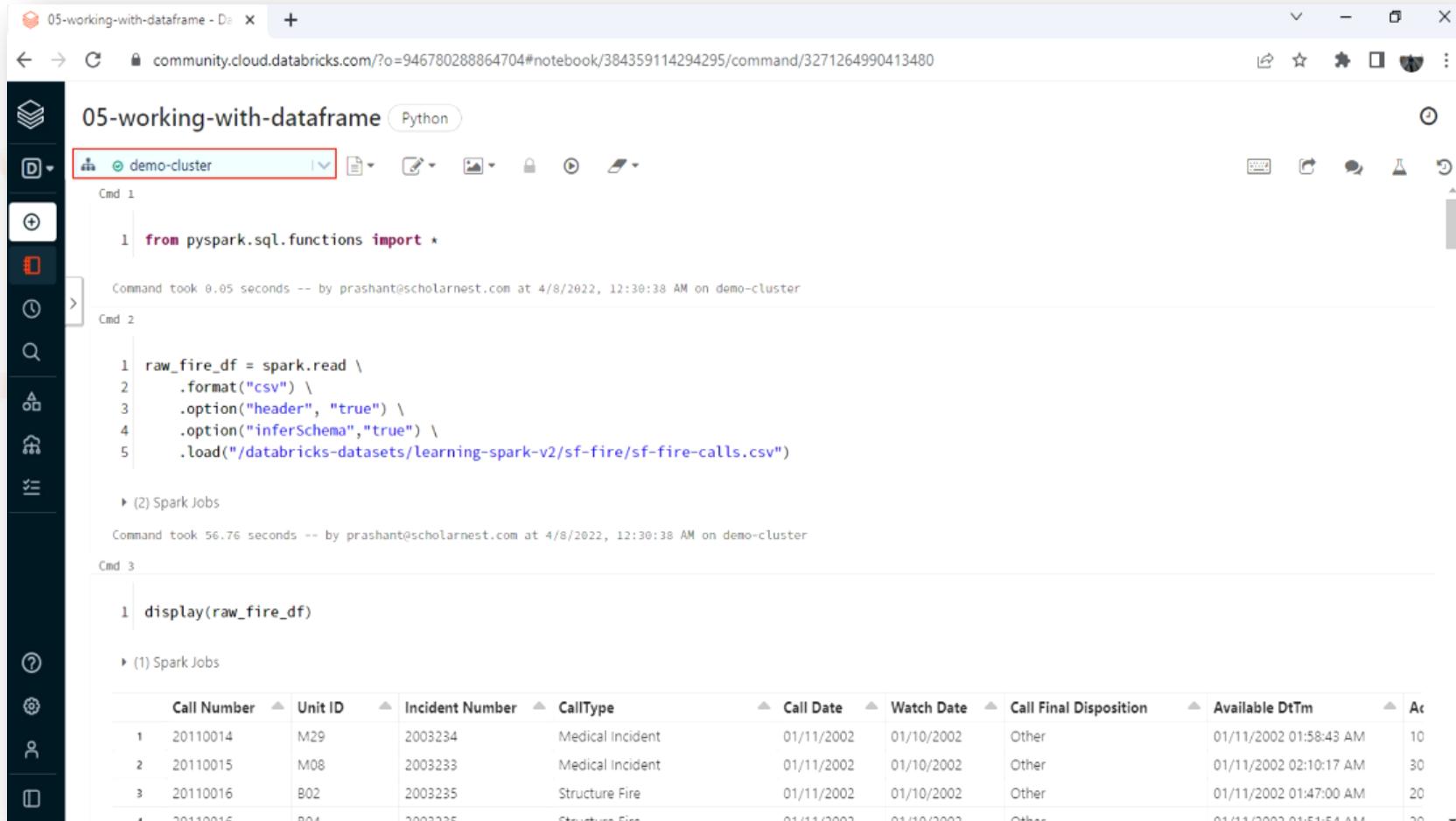
Cmd 3

```
1 display(raw_fire_df)
```

▶ (1) Spark Jobs

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtM	Ac
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	30
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	20
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	20

Make sure you have attached a running cluster to your notebook, and run the entire notebook once before you start working on this notebook further. Whenever you stop working on the Databricks cloud and want to return, you must start a new cluster, attach it to your notebook and recreate all the objects you want to use.



05-working-with-dataframe - Databricks Notebook

community.cloud.databricks.com/?o=946780288864704#notebook/384359114294295/command/3271264990413480

05-working-with-dataframe Python

demo-cluster

Cmd 1

```
1 from pyspark.sql.functions import *
```

Command took 0.05 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:30:38 AM on demo-cluster

Cmd 2

```
1 raw_fire_df = spark.read \
2   .format("csv") \
3   .option("header", "true") \
4   .option("inferSchema","true") \
5   .load("/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv")
```

▶ (2) Spark Jobs

Command took 56.76 seconds -- by prashant@scholarnest.com at 4/8/2022, 12:30:38 AM on demo-cluster

Cmd 3

```
1 display(raw_fire_df)
```

▶ (1) Spark Jobs

	Call Number	Unit ID	Incident Number	CallType	Call Date	Watch Date	Call Final Disposition	Available DtTm	Ac
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58:43 AM	10
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10:17 AM	30
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47:00 AM	20
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:54 AM	20

We already solved the first question in the previous lecture. Now, let us look at the second question. We have the logic framed for this question in the image below.

The screenshot shows a Jupyter Notebook interface with the title "05-working-with-dataframe" and a Python tab selected. The notebook has a cluster named "demo-cluster". A command history shows "Command took 1.93 seconds -- by prashant@scholarnest.com at 4/8/2022, 1:20:37 AM on demo-cluster".

Cmd 15 contains the question:

Q2. What were distinct types of calls made to the Fire Department?

select distinct CallType as distinct_call_types
from fire_service_calls_tbl
where CallType is not null

Cmd 16 contains the result of the query:

1 |

1. Filter the records and take only those where CallType is not null --> `where()`
2. Select the CallType column as distinct_call_type --> `select()`
3. Take only distinct call types --> `distinct()`
4. Show the result --> `show()` or `display()`

Here is the code for the logic we framed in the second question.

I am using fire_df and applying where transformation. The filter condition takes only those records where CallType is not null. Then I am chaining the select transformation which takes a column name. However, I wanted to give an expression. So, I must use the expr() function to evaluate my expression. Then I am chaining the distinct() transformation. And last step is to take action and show the results.

05-working-with-dataframe Python

demo-cluster

Command took 1.93 seconds -- by prashant@scholarnest.com at 4/8/2022, 1:20:37 AM on demo-cluster

Cmd 15

> Q2. What were distinct types of calls made to the Fire Department?

```
select distinct CallType as distinct_call_types
from fire_service_calls_tbl
where CallType is not null
```

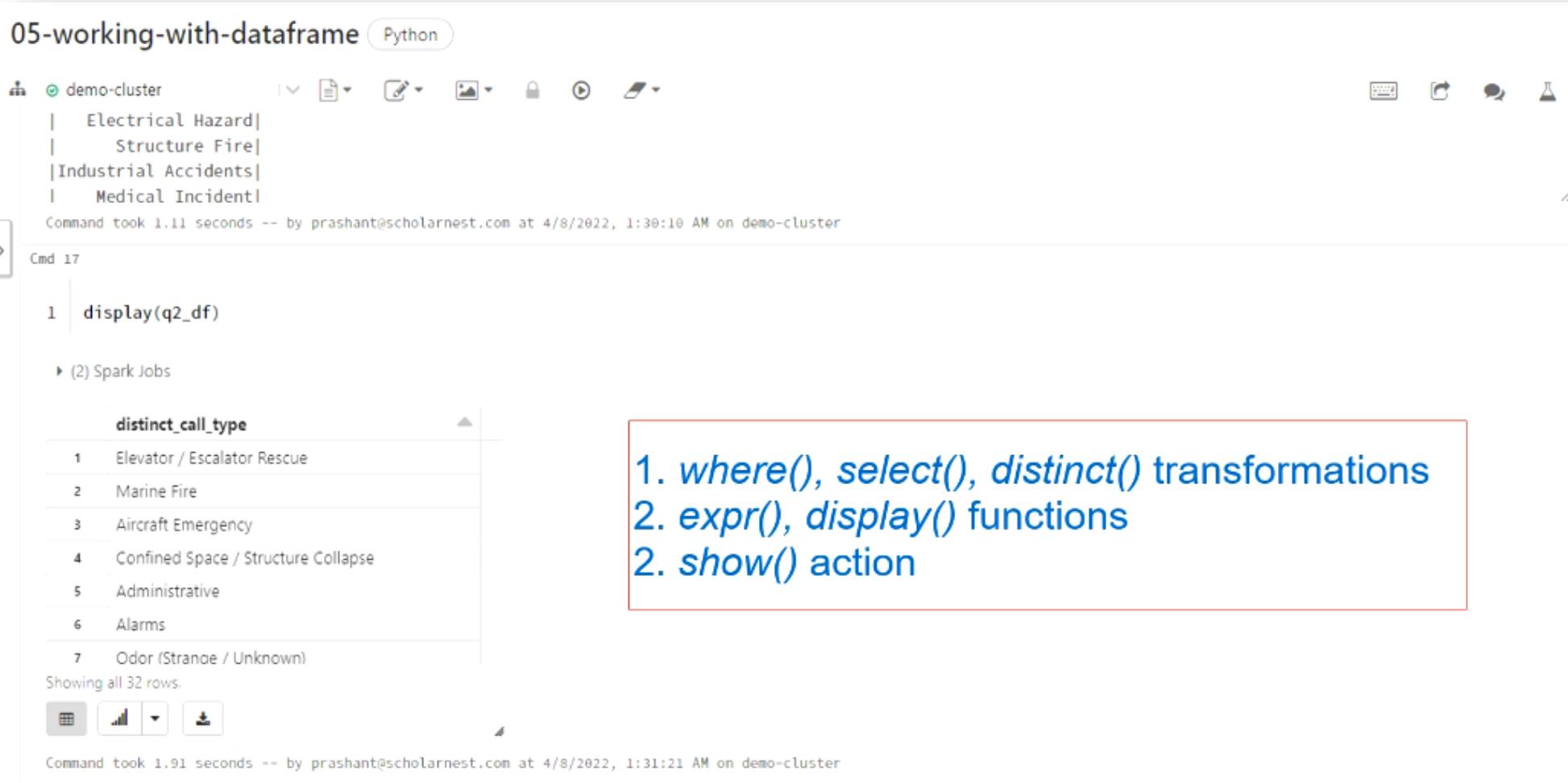
Cmd 16

```
1 q2_df = fire_df.where("CallType is not null") \
2     .select(expr("CallType as distinct_call_type")) \
3     .distinct()
4 q2_df.show()
```

▶ (2) Spark Jobs

distinct_call_type
Elevator / Escala...
Aircraft Emergency
Alarms
Odor (Strange / U...
Citizen Assist / ...
HazMat
oil Spill

You can also use `display()` function which depicts the formatted results. The `display()` function is not part of Apache Spark. It is a utility function added by Databricks. And the `show()` method is a commonly used Spark action which shows a plain text output. So, we learned the following Dataframe methods from the second question.



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" in Python mode. The code cell contains the following:

```
1 display(q2_df)
```

Below the code, the resulting DataFrame is displayed:

distinct_call_type
1 Elevator / Escalator Rescue
2 Marine Fire
3 Aircraft Emergency
4 Confined Space / Structure Collapse
5 Administrative
6 Alarms
7 Odor (Strange / Unknown)

Text at the bottom of the cell indicates: "Showing all 32 rows."

At the bottom of the notebook cell, there are three small icons: a grid, a chart, and a file.

At the very bottom of the screen, there is a footer bar with the text "Command took 1.91 seconds -- by prashant@scholarnest.com at 4/8/2022, 1:31:21 AM on demo-cluster".

1. `where()`, `select()`, `distinct()` transformations
2. `expr()`, `display()` functions
2. `show()` action

Now let us look at the 3rd question.

The question asks for all responses for delayed times greater than 5 mins. We have framed the logic for the same in the image shown below.

05-working-with-dataframe Python

demo-cluster Command took 1.91 seconds -- by prashant@scholarnest.com at 4/8/2022, 1:31:21 AM on demo-cluster

Cmd 18

Q3. Find out all response for delayed times greater than 5 mins?

```
select CallNumber, Delay  
from fire_service_calls_tbl  
where Delay > 5
```

1. Filter the records where Delay is greater than five --> `where("Delay > 5")`
2. Select CallNumber and Delay columns --> `select("CallNumber", "Delay")`

Cmd 19

1 | Python ▶ ▾ - x

Cmd 20

We have implemented the logic into code and we can see the desired output below.

We are doing two new things in this code:

1. We are selecting two columns in the select() method. The select() method takes a comma-separated list of columns.
2. We chained the show() action. Remember that you can chain one and only one Action, and that must be the last method in the chain. So I cannot chain anything else after the show() method.



The screenshot shows a Jupyter Notebook cell titled "05-working-with-dataframe" in Python mode. The cell contains the following Scala code:

```
Q3. Find out all response for delayed times greater than 5 mins?  
select CallNumber, Delay  
from fire_service_calls_tbl  
where Delay > 5
```

Below the code, the cell is labeled "Cmd 19". The code itself is:

```
1 fire_df.where("Delay > 5") \  
2 .select("CallNumber", "Delay") \  
3 .show()
```

After the code, there is a "▶ (1) Spark Jobs" button. Below it, the output is displayed as a table:

CallNumber	Delay
20110014	5.23
20110017	6.93
20110019	6.12
20110039	7.85
20110045	77.33
20110046	5.42
20110055	6.5
20110058	6.85
20110058	6.85
20110061	6.33

Here we have the next question and it's logic along side.

We want to filter records where the call type is not null and select the CallType. But we want to find the most common CallType. So we must count the calls for each CallType. Hence, we will group by the CallType and then count for each group. The last step is to sort the record by the count in descending order to know which ones are the most common.

The screenshot shows a Jupyter Notebook interface with a single cell containing the following code:

```
05-working-with-dataframe Python
demo-cluster
Cmd 20
Q4. What were the most common call types?
select CallType, count(*) as count
from fire_service_calls_tbl
where CallType is not null
group by CallType
order by count desc
```

Below the code, the output of the cell is shown in a table:

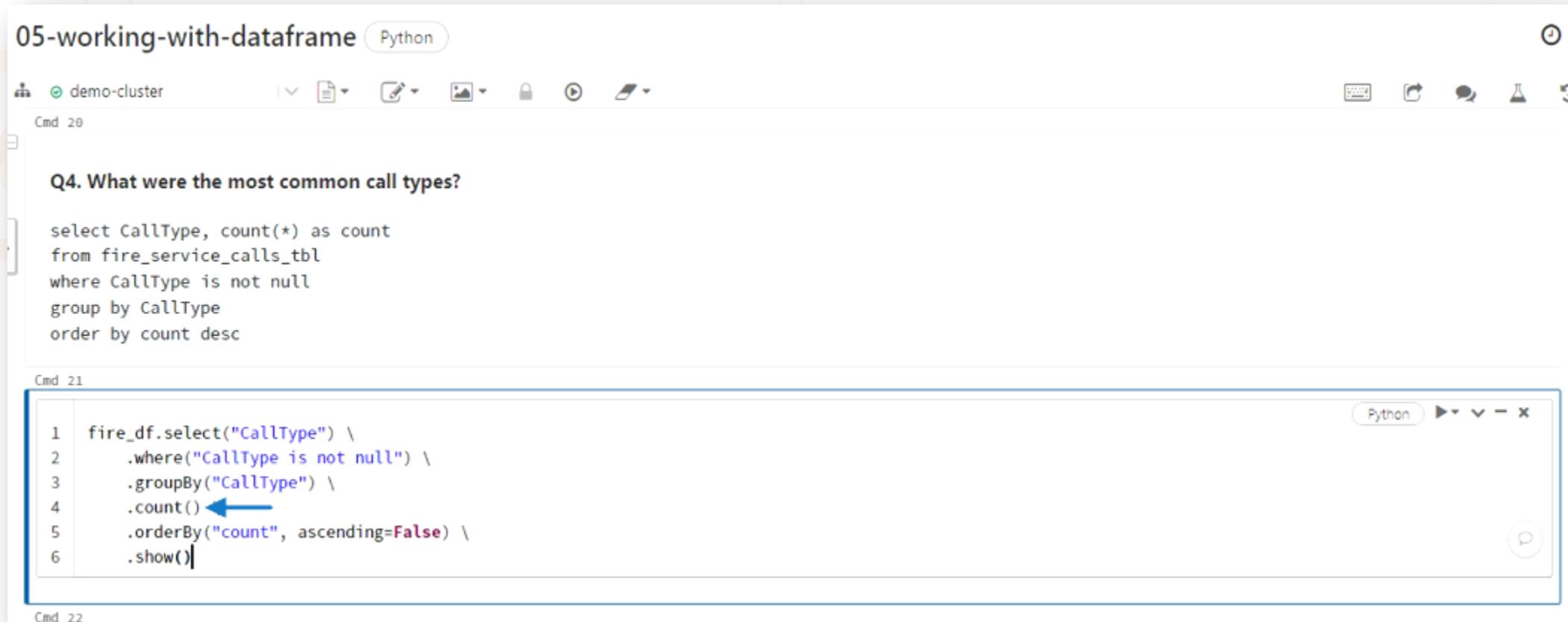
CallType	count
1	

The output table has one row with values '1' and '|'. To the right of the table, a red box contains the following steps:

1. Filter the records where CallType is not null --> `where("CallType is not null")`
2. Select CallType --> `select("CallType")`
3. Group them by CallType --> `groupBy("CallType")`
4. Count the grouped Data Frame --> `count()`
5. Sort it by count in decending order --> `orderBy("count", ascending=False)`
6. Show the results --> `show()`

Here is the code for the fourth question.

But I want to highlight one thing here, I am using count() transformation in this example. But we already learned that the count() is an action.



05-working-with-dataframe Python

demo-cluster Cmd 20

Q4. What were the most common call types?

```
select CallType, count(*) as count
from fire_service_calls_tbl
where CallType is not null
group by CallType
order by count desc
```

Cmd 21

```
1 fire_df.select("CallType") \
2     .where("CallType is not null") \
3     .groupBy("CallType") \
4     .count() ←
5     .orderBy("count", ascending=False) \
6     .show()
```

Python

Cmd 22

So we have two count() methods in Spark:

1. Dataframe.count()
2. GroupedData.count()

If you are using a count before the groupBy(), it will be an action, and you cannot chain any other transformation after the count () Action.

However, if you use a count() method after the groupBy(), it will be a transformation, and you can chain further transformations.

We can see the desired output for the fourth question as well.

The remaining 6 questions are given for practice, you can compare your answer with the solutions in the material provided.

05-working-with-dataframe Python

demo-cluster

Cmd 21

```
1 fire_df.select("CallType") \
2   .where("CallType is not null") \
3   .groupBy("CallType") \
4   .count() \
5   .orderBy("count", ascending=False) \
6   .show()
```

▶ (2) Spark Jobs

CallType	count
Medical Incident	2843475
Structure Fire	578998
Alarms	483518
Traffic Collision	175507
Citizen Assist / ...	65360
Other	56961
Outside Fire	51603
Vehicle Fire	20939
Water Rescue	20037
Gas Leak (Natural...)	17284
Electrical Hazard	12608
Elevator / Escala...	11851
Odor (Strange / U...)	11680
Smoke Investigati...	9796
Fuel Spill	5198



Thank You
ScholarNest Technologies Pvt Ltd.
www.scholarnest.com