

Image Data Sets with Databricks Notebooks, Databricks SQL and Power BI

One of the core capabilities of Databricks is to work with Structured, Unstructured & Semi-Structured Datasets. Recently there was an ask as how to work with Image Datasets and access it via Notebooks, Databricks SQL and Power BI in the light of Unity Catalog.

Here by in this blog post, I will walk you through as what is needed interns of access in Catalog, Schema, External Location and then save it as External Table, Access it via Notebooks using Pyspark and Databricks SQL and finally in Power BI.

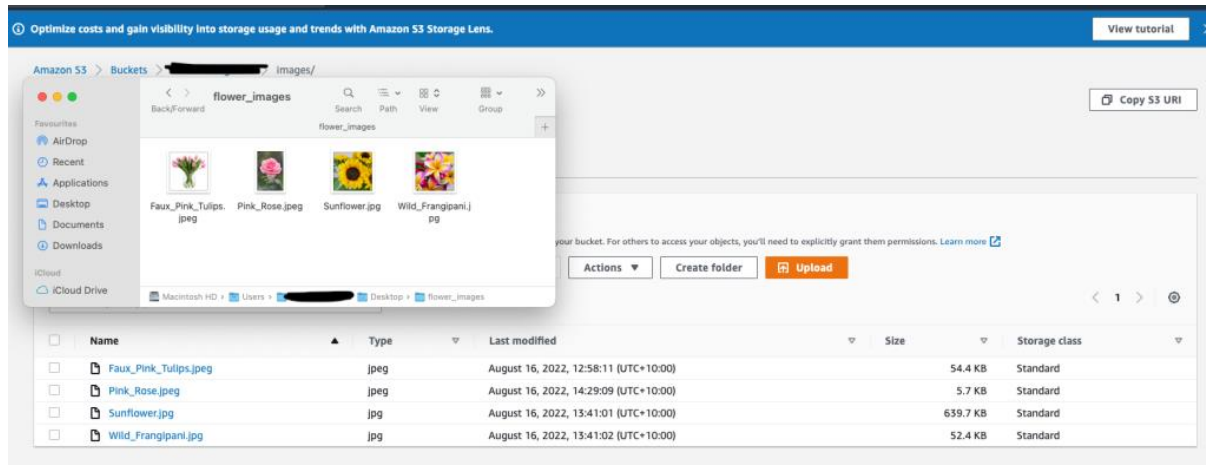
Access In Unity Catalog

Unity Catalog is a fine-grained governance solution for Data & AI in Lakehouse. Please follow Databricks Unity Catalog documentation to more about this topic ([AWS](#) / [Azure](#)).

Catalog	CREATE USAGE
SCHEMA / Database	CREATE USAGE
EXTERNAL LOCATION	CREATE TABLE READ FILES WRITE FILES

Setup

Image Files– I have sample image files in s3 bucket and this bucket is added as External Location ([AWS](#) / [Azure](#)) in Unity Catalog. Bucket name is like *s3:/<Bucket Name>/images*



Databricks Cluster – Currently when Unity Catalog is enabled for a workspace, while provisioning a cluster we have to specify Cluster Access Mode and currently only 2 choice we have which are:

- Single User – support workload using all programming languages and support Unity Catalogs
- Shared – Support Python & SQL programming languages and support Unity Catalogs but has some limitations ([AWS](#) / [Azure](#))

I have used **SINGLE USER** as it does support Python UDF's which are currently not supported by Shared Cluster Access Mode.

How to Read Image Files

In order to read Image Files in Databricks we can either use format of type as “**image**” which normally work for JPEG, PNG and other known formats. However there are few [limitations with image format](#) which can be avoided by using “**binary file**” format.

Binary File format, reads file as binary and converts each file into a single record that contains the raw contents and metadata of the file. If a file has file name with extension in its name, then its image preview is automatically enabled. Alternatively, we can force image preview functionality by using “**mimeType**” option with a string value

"image/*" to annotate the binary column. Images are decoded based on their format information in the binary content. Unsupported files appear as a broken image icon.

In below screenshot, we are using `format = image` and as all files are of extension JPEG hence it provides an image preview.

Cmd 4

```
1 sample_img_dir = "s3://[REDACTED]/images"
2
3 image_df = spark.read.format("image").load(sample_img_dir)
4
5 display(image_df)
```





▸ (2) Spark Jobs

▼ image_df: pyspark.sql.dataframe.DataFrame

▼ image: struct

- origin: string
- height: integer
- width: integer
- nChannels: integer
- mode: integer
- data: binary

Table Data Profile

	image
1	
2	
3	
	

Showing all 4 rows.

☒ Show image preview ⓘ

In below screenshot, when we use **format = binaryFile** and **option = mimeType**, ***"image/*"***, it not only provides a preview of the image but also some other properties as well.

Cmd 6

```

1 df = spark.read.format("binaryFile").option("mimeType","image/*").load("s3://[REDACTED]/images")
2 display(df) # image thumbnails are rendered in the "content" column

```





▸ (2) Spark Jobs

df: pyspark.sql.dataframe.DataFrame

path: string
modificationTime: timestamp
length: long
content: binary

Table

Data Profile

	path	modificationTime	length	content
1	s3://[REDACTED]/images/Sunflower.jpg	2022-08-16T03:41:01.000+0000	655056	
2	s3://[REDACTED]/images/Faux_Pink_Tulips.jpeg	2022-08-16T02:58:11.000+0000	55744	
3	s3://[REDACTED]/images/Wild_Frangipani.jpg	2022-08-16T03:41:02.000+0000	53610	
4	s3://[REDACTED]/images/Pink_Rose.jpeg	2022-08-16T02:58:12.000+0000	5871	

How to save data pertaining to images and image file as a DELTA Table

Above we just displayed image datasets but haven't saved it yet. Let's see how to save data and image as DELTA table.

In below screenshot, we created a dummy dataset with 3 columns i.e.. Id, Flower_Name & Location.


Please note: We just specified the location of s3 bucket where files are residing and a display of dataframe does not render it as image yet.

```

1 from pyspark.sql.types import StructType, StructField, StringType, IntegerType
2
3 schema = StructType([
4     StructField("id", IntegerType(), True),
5     StructField("Flower_Name", StringType(), True),
6     StructField("Location", StringType(), True)
7 ])
8
9 data = [(1, "Faux Pink Tulip", "s3://[REDACTED]/images/Faux_Pink_Tulips.jpeg"),
10         (2, "Pink Rose", "s3://[REDACTED]/images/Pink_Rose.jpeg"),
11         (3, "Sunflower", "s3://[REDACTED]/images/Sunflower.jpg"),
12         (4, "Wild Frangipani", "s3://[REDACTED]/images/Wild_Frangipani.jpg")
13     ]
14
15 df = spark.createDataFrame(data=data, schema=schema)
16 df.printSchema()
17 display(df)

```

▸ (2) Spark Jobs

▸  df: pyspark.sql.dataframe.DataFrame = [id: integer, Flower_Name: string ... 1 more field]

root

```

|-- id: integer (nullable = true)
|-- Flower_Name: string (nullable = true)
|-- Location: string (nullable = true)

```

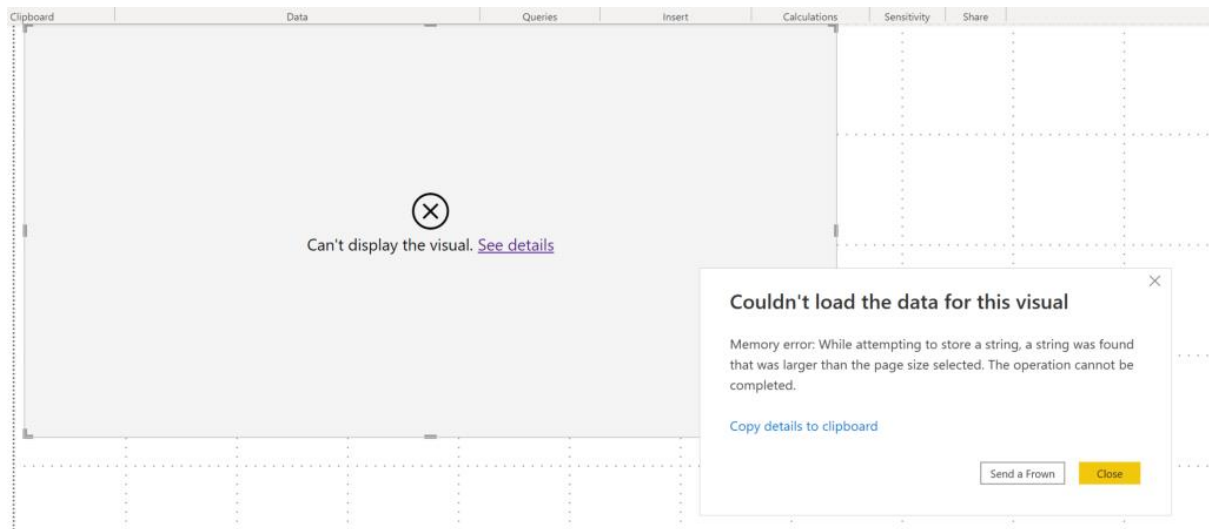
Table Data Profile

	id	Flower_Name	Location
1	1	Faux Pink Tulip	s3://[REDACTED]/images/Faux_Pink_Tulips.jpeg
2	2	Pink Rose	s3://[REDACTED]/images/Pink_Rose.jpeg
3	3	Sunflower	s3://[REDACTED]/images/Sunflower.jpg
4	4	Wild Frangipani	s3://[REDACTED]/images/Wild_Frangipani.jpg

Showing all 4 rows.

Next, we need to read the image files from s3 bucket and create a separate dataframe with image related properties.

Python UDF – We will be using a Python User Defined Function (UDF) in our query so that we can resize the image based on our requirements (This is needed as when we would display this data in Power BI dashboard then we can avoid potential memory errors because of the size of the row/column). Also, it will then convert image to **JPEG Base 64** ([which helps to load files quickly](#)).



In below screenshot, we can change the values in row 11 (marked with a red arrow) to resize image as per our requirements.

User Defined Function (UDF) to resize image and convert to standard JPEG format

This is to make sure lesser data gets moved when rendering this data in Power BI and we don't get any Memory Errors





```

Cmd 10
1 # UDF to resize the image (change value for line 11 to appropriate size based on your requirements)
2 import io
3 from pyspark.sql.functions import concat, base64, lit, col
4 from PIL import Image
5
6 @udf("binary")
7 def resized_image_binary(content):
8
9     buffer = io.BytesIO()
10    img = Image.open(io.BytesIO(content))
11    new_img = img.resize((64, 64)) # Choose your image size
12    new_img.save(buffer, format="JPEG")
13    img_binary = buffer.getvalue()
14    return img_binary
  
```

Next, read the Image Files as binarFiles and extract column: Path, Content and use pass Content column to UDF and convert to Binary format and then use this convert to JPEG base64.

```

Cmd 14
1 flowers_df = spark.read.format("binaryFile").load("s3://databricks-gsethi/images") \
2   .select(
3     col("path"),
4     col("content"),
5     resized_image_binary(col("content")).cast("binary").alias("resized_binary")
6   ) \
7   .withColumn("resized_image_base64",
8     concat(lit('data:image/jpeg;base64, '), base64(col("resized_binary"))))
9
10
11 display(flowers_df)
  
```

Table Data Profile			
	path	content	
1	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Sunflower.jpg		
2	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Flur_Pink_Tulips.jpg		
3	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/White_Frangipani.jpg		
4	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Pink_Rose.jpg		

Now at this stage we have 2 dataframes, 1 with data and 2nd with image related properties. Now we need to combine both datasets so that we can create one single record for both data and image properties.

In below screenshot, we are joining both dataframes and while doing so we are comparing it with the Image File Name from Location field from Dataset and Path filed from image dataframe. Post joining these 2 dataframes, we are saving it as a new dataframe which we then we write as a delta table.

Next, we will join both original data source dataframe and image file dataframe to form a single record to be saved in the delta table





```

Cmd 16

1 import pyspark.sql.functions as F
2
3 final_df = df.join(flowers_df, F.element_at(F.split("Location","\\-"),-1) == F.element_at(F.split("path","\\-"),-1),'left')
4
5

```

A display of table will show data and image metadata nicely stitched for each row and while reading we need not to use any image format either. Data in the table is stored in a format which when displayed rendered as image.

ID	Flower_Name	Location	path	content	related_image_name
1	Flur_Pink_Tulips	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Flur_Pink_Tulips.jpg	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Flur_Pink_Tulips.jpg		efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Flur_Pink_Tulips.jpg
2	Pink_Rose	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Pink_Rose.jpg	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Pink_Rose.jpg		efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Pink_Rose.jpg
3	Sunflower	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Sunflower.jpg	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Sunflower.jpg		efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/Sunflower.jpg
4	White_Frangipani	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/White_Frangipani.jpg	efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/White_Frangipani.jpg		efb3-def-958aa32d48044eaa3b736972a5f9a948e69248f526de493ab67e1873c107b4-29-sg-zoocevmgwmvtdata-images/White_Frangipani.jpg

Query Data in Databricks SQL

Let's query this table in Databrick SQL. And yes, we are able to see image data as well as part of SELECT statement.

New query +

New query ☆





Run (limit: 1000) main_external_loc

```
1 SELECT
2 *
3 FROM
4 flower_images
```

+ Add filter

Table

+ Add visualization

id	Flower_Name	Location	path	content	resized_binary	resized_image_base64
3	Sunflower	s3://[REDACTED]images/Sunflower.jpg	elbs://s3-def:958aa32dd48044eaa63b736972a5f9a94i	ffd8ffe000104a4649460001010000480c	ffd8ffe000104a4649460001010000	
1	Faux Pink Tulip	s3://[REDACTED]images/Faux_Pink_Tulips.jpeg	elbs://s3-def:958aa32dd48044eaa63b736972a5f9a94i	ffd8ffe000104a4649460001010000010c	ffd8ffe000104a4649460001010000	
4	Wild Frangipani	s3://[REDACTED]images/Wild_Frangipani.jpg	elbs://s3-def:958aa32dd48044eaa63b736972a5f9a94i	ffd8ffe000104a4649460001010000480c	ffd8ffe000104a4649460001010000	
2	Pink Rose	s3://[REDACTED]images/Pink_Rose.jpeg	elbs://s3-def:958aa32dd48044eaa63b736972a5f9a94i	ffd8ffe000104a4649460001010000010c	ffd8ffe000104a4649460001010000	

Render Data in Power BI

Let's render this table in Power BI. We need change the **Data Category** to **Image URL** for **resized_image_base64** column and we will be able to see image column rendered properly.

The screenshot shows the Power BI Desktop interface. In the top ribbon, the 'Column tools' tab is active. The 'Data category' dropdown is set to 'Image URL'. A red arrow points from this dropdown to the 'resized_image_base64' column in the 'Fields' pane on the right. Another red arrow points from the 'Image URL' option in the dropdown to the 'resized_image_base64' column in the 'Fields' pane. The 'Fields' pane shows a table with columns: id, Flower_Name, and resized_image_base64. The table contains four rows of data:

id	Flower_Name	resized_image_base64
1	Faux Pink Tulip	[Image]
2	Pink Rose	[Image]
3	Sunflower	[Image]
4	Wild Frangipani	[Image]

The bottom part of the screenshot shows the 'Home' ribbon with the 'Data' group. The 'Data' group contains icons for 'Get data', 'Excel', 'Data Server', 'SQL', 'Enter data', 'Database', 'Recent sources', 'Transform data', 'Refresh', 'New visual', 'Text box', 'More visuals', 'New measure', 'Quick measure', 'Sensitivity', and 'Put data in context'. The 'Data' group is currently selected, and the 'Data' icon is highlighted.