* **Unified Platform**: Databricks combines data engineering, data science, and machine learning into a single platform, built on Apache Spark.
* **Collaborative Environment**: It offers a collaborative workspace with features like notebooks, libraries, and dashboards, allowing team members to share code and computing resources.
* **Scalability and Integration**: Databricks provides scalable clusters and seamless integration with cloud storage (AWS, Azure, Google Cloud) and other data tools.
* **Key Features**: Includes Delta Lake for reliable data lakes, Unity Catalog for security and governance, and Databricks Runtime for optimized performance.
* **Prerequisites**: You need a cloud account (Azure, Google Cloud, or AWS) or you can use the Databricks Community Edition for free.
* **Creating an Account**: The video walks through the process of creating a Databricks account, including filling in personal details and selecting a cloud provider.
* **Workspace Setup**: It demonstrates setting up a Databricks workspace on Azure, including creating a resource group, selecting a region, and choosing a pricing tier.

External id: 4258eb63-c2cd-4ab9-93cb-920b0b6be49d

* **Unified Interface**: DBFS provides a unified interface for accessing and managing data in the Databricks workspace, similar to a local file system.
* **Cloud-Based Storage**: DBFS is built on top of cloud storage (Azure Data Lake Storage, AWS, or Google Cloud Storage), making it highly scalable and distributed.
* **Data Persistence**: Data stored in DBFS is persistent, meaning it remains available even if clusters are started or stopped.
* **Integration and Use Cases**: DBFS integrates seamlessly with Databricks notebooks, clusters, and workflows, and is used for data loading, writing, backup, and sharing files between team members.

files = dbutils.fs.ls("dbfs:/sample.parquet")

for file in files:

    dbutils.fs.rm(file.path, True)  # Recursively delete each file/folder

dbutils.fs.rm("dbfs:/sample.parquet", True)  # True ensures recursive deletion

%fs

ls /FileStore/tables

understand the dataframe

#dbfs:/FileStore/tables/customer.csv

#df=spark.read.csv("/FileStore/tables/customer.csv")

df=spark.read.csv("/FileStore/tables/customer.csv", header=True)

#df.show()

display(df)

df.printSchema()

#dbfs:/FileStore/tables/customer.csv

#df=spark.read.csv("/FileStore/tables/customer.csv")

df=spark.read.csv("/FileStore/tables/customer.csv", header=True, inferSchema=True)

#df.show()

display(df)

from pyspark.sql.types import \*

# Define the schema

schema = StructType([

    StructField("customer\_id", IntegerType(), True),

    StructField("name", StringType(), True),

    StructField("email", StringType(), True),

    StructField("country", StringType(), True),

    StructField("customer\_type", StringType(), True),

    StructField("registration\_date", DateType(), True),

    StructField("age", IntegerType(), True),

    StructField("gender", StringType(), True),

    StructField("total\_purchases", IntegerType(), True),

    StructField("ingestion\_timestamp", TimestampType(), True)

])

df1 = spark.read.csv("/FileStore/tables/customer.csv", header=True, schema=schema)

df1.printSchema()

#dbfs:/FileStore/tables/product.json

df = spark.read.json("/FileStore/tables/product.json")

display(df)

df.printSchema()

from pyspark.sql.types import StructType, StructField, StringType, BooleanType, DoubleType, LongType

# Define the schema

schema = StructType([

    StructField("brand", StringType(), True),

    StructField("category", StringType(), True),

    StructField("ingestion\_timestamp", StringType(), True),  # Using StringType for ingestion\_timestamp

    StructField("is\_active", BooleanType(), True),

    StructField("name", BooleanType(), True),

    StructField("price", DoubleType(), True),

    StructField("product\_id", LongType(), True),

    StructField("rating", DoubleType(), True),

    StructField("stock\_quantity", LongType(), True)

])

df1 = spark.read.json("/FileStore/tables/product.json", schema=schema)

#/dbfs/FileStore/tables/transactions\_snappy.parquet

df = spark.read.parquet("/FileStore/tables/transactions\_snappy.parquet")

display(df)

from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType, TimestampType

# Define the schema

schema = StructType([

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

    StructField("customer\_id", IntegerType(), True),

    StructField("product\_id", IntegerType(), True),

    StructField("quantity", IntegerType(), True),

    StructField("total\_amount", DoubleType(), True),

    StructField("transaction\_date", StringType(), True),  # Using StringType for the date column

    StructField("payment\_method", StringType(), True),

    StructField("store\_type", StringType(), True),

    StructField("ingestion\_timestamp", TimestampType(), True)

])

df1 = spark.read.schema(schema).parquet("/FileStore/tables/transactions\_snappy.parquet")

**Filter the records**

**Different way to do the same thing**

df1=df.filter(df["customer\_type"]=='VIP')

display(df1)

from pyspark.sql.functions import col, column

df1 = df.filter(df["customer\_type"]=='VIP')

df2 = df.filter(col("customer\_type")=='Regular')

df3 = df.filter(column("customer\_type")=='Premimum')

df4 = df.filter(df.customer\_type =='VIP')

df5 = df.filter((df.customer\_type =='VIP') &(df.country =='USA'))

df6 = df.where((df.customer\_type =='VIP') &(df.country =='USA'))

df7 = df.where((df.customer\_type =='VIP') | (df.country =='USA'))

display(df7)

**Add remove column**

from pyspark.sql.functions import when

df1 = df.withColumn("Salary", col("age")\*1000)

df1.printSchema()

display(df1)

df2 = df.withColumn("Seniority", when(df.age>30,'Senior').otherwise('Junior'))

display(df2)

**changing the existing column**

df3 = df2.withColumnRenamed("Seniority","Is\_Senior")

#display(df3)

df4 = df3.drop("Is\_Senior")

df5 = df3.drop("Is\_Senior", "age", "gender")

df5.printSchema()

**Select function**

df1 = df.select("age", "gender", "customer\_type")

df2 = df.select("age", "gender", "customer\_type", col("age")\*1000)

df3 = df.select("age", "gender", "customer\_type", (col("age")\*1000).alias("Salary"))

display(df3)

**Read from the json file**  
df1 = spark.read.**json**("/FileStore/tables/SampleNested.json")

df2 = df1.select("address.city", "address.state", "age", "name", "email")

display(df2)

**UNION and Distinct**

df1 = df.filter("customer\_type=='VIP'")

df2 = df.filter("customer\_type=='Regular'")

df3 = df1.union(df2)

df4 = df1.union(df2).union(df)

df5 = df4.distinct()

df6 = df4.select("customer\_type", "country").distinct()

**Handle NULL**

print(df.count())

df1 = df.na.drop()

print(df1.count())

df2 = df.filter(col("email").isNull())

#display(df2)

df3 = df.na.fill("Unknown")

df4 = df.na.fill({"email":"Unknown", "age":0})

df5 = df.fillna({"email":"Unknown", "age":0})

df6 = df.dropna()

display(df4)

**Order/sort by**

from pyspark.sql.functions import desc

df1 = df.orderBy("age")

df2 = df.orderBy("age", "gender")

df3 = df.orderBy(desc("age"))

df4 = df.orderBy(desc("age"), "gender")

df5 = df.sort(desc("age"))

df6 = df.orderBy(col("email").asc\_nulls\_first())

df7 = df.orderBy(col("email").asc\_nulls\_last())

df8 = df.orderBy(col("email").desc\_nulls\_last())

display(df7)

**Group by and aggregation**

from pyspark.sql.functions import sum,avg,max,min

df1 = df.groupBy("gender").count()

df2 = df.groupBy("gender", "customer\_type").count()

df3 = df.groupBy("gender").sum("age")

df4 = df.groupBy("gender").max("age")

df5 = df.groupBy("gender").min("age")

df6 = df.groupBy("gender").avg("age")

df7 = df.groupBy("gender").agg(sum("age"), max("age"), min("age"), avg("age"))

df8 = df.groupBy("gender").agg(sum("age").alias("Sum"), max("age").alias("Highest"), min("age").alias("Lowest"), avg("age").alias("Average"))

display(df8)

**Manipulating string**

from pyspark.sql.functions import upper,lower,rtrim, ltrim,trim, regexp\_replace,split,  length, concat\_ws

df1 = df.select(upper("country"))

df2 = df.select(lower("country"))

df3 = df.select(ltrim("country").alias("New\_Country"), rtrim("country"), trim("country"))

df4 = df.select(regexp\_replace(col("country"), "Unknown", "Not Sure"))

df5 = df.select("email", split("email", "@"))

df6 = df.select("email", col("email").contains("customer"))

df7 = df.select("email", length("email"))

df7 = df.select("email", length("email"))

df8 = df.select(concat\_ws("@@", "age", "country"))

display(df8)

**Handle date**

from pyspark.sql.functions import \*

df1 = df.select("registration\_date", year("registration\_date"), month("registration\_date"), dayofmonth("registration\_date"))

#display(df1)

df2 = df.select("registration\_date", dayofweek("registration\_date"), weekofyear("registration\_date"), dayofyear("registration\_date"), quarter("registration\_date"))

#display(df2)

df3 = df.select("registration\_date", date\_add("registration\_date", 10), date\_sub("registration\_date", 10))

#display(df3)

df4 = df.select("registration\_date", date\_format("registration\_date", "MMMM dd, yyyy"))

df5 = df.select(current\_date()).limit(1)

display(df5)

**Handel timestamp**

df1= df.select("ingestion\_timestamp", year("ingestion\_timestamp"), month("ingestion\_timestamp"), dayofmonth("ingestion\_timestamp"), hour("ingestion\_timestamp"), minute("ingestion\_timestamp"), second("ingestion\_timestamp"))

#display(df1)

df2 = df.select(current\_timestamp())

df3 = df.select("ingestion\_timestamp", date\_diff(current\_timestamp(), "ingestion\_timestamp"))

display(df3)

#df3.write.mode("overwrite").option("header", "True").csv("/Filestore/tables/Practice\_Output/CSV/")

df3.write.mode("overwrite").json("/Filestore/tables/Practice\_Output/JSON/")

df4.write.mode("overwrite").parquet("/Filestore/tables/Practice\_Output/Parquet/")

**df3.write.mode("append").json("/Filestore/tables/Practice\_Output/JSON/")**

**df.write.mode("overwrite").partitionBy("country").csv("/Filestore/tables/Practice\_Output/JSON\_Parttion/")**