Azure Databricks - session 12

==========

dbutils.fs.mount(

source = 'wasbs://inputdatasets@ttstorageaccount100.blob.core.windows.net',

mount_point = '/mnt/retaildb',

extra_configs={'fs.azure.account.key.ttstorageaccount100.blob.core.windows.net':'SuTUyxyYr/ooc0gF

Abqwq3cRt5ApE3sAbCMBSLL8trA/jBt0BEiPRYliwgCCPDSBNY4zvC1eHLBj+AStdIk73A==''}

)

3 ways to access the storage account

==========

1. access key/account key - all access to the storage account

2. SAS key - Shared access signature - (we can give access at container also)

3. Service principal - fine grained control on folder level

storage account -> multiple containers -> inside each container there can be multiple

directories/folders

dbutils.fs.mount(

source = 'wasbs://retaildb@ttstorageaccount102.blob.core.windows.net',

mount_point = '/mnt/retaildb',

extra_configs={'fs.azure.account.key.ttstorageaccount102.blob.core.windows.net':'WLRLRZvhDf0FtiEr

75hG5CWQYZXzhhs573DxUGBJPvSxWutOxo+WG0mVQxBp+QI+bVkhYBSaDh5X+ASt+wLEMw==''}

)

df1 = spark.read.csv('/mnt/retaildb/raw/customers.csv',header = True)

display(df1)

Secret Scope

==========

1. azure key vault backed secret scope - recommended

any other azure service can also reuse the key

create an azure key vault

secrets

databricksdemokey

#secrets/createScope

secret scope -> store many secret keys

databricks-demo-scope

in databricks we created a secret scope

the keys are stored in azure key vault

databricks secrets list-scopes

Scope Backend KeyVault URL

---------- ---------- ----------

databricks-demo-scope AZURE_KEYVAULT
https://ttdatabricksdemokv.vault.azure.net/

dbutils.fs.mount(

source = 'wasbs://retaildb@ttstorageaccount102.blob.core.windows.net',

mount_point = '/mnt/retaildb1',

extra_configs={'fs.azure.account.key.ttstorageaccount102.blob.core.windows.net':

dbutils.secrets.get('databricks-demo-scope','databricksnewkey')}

)

df1 = spark.read.csv('/mnt/retaildb1/raw/customers.csv',header = True)

display(df1)

dbutils.fs.mount(

source = 'wasbs://retaildb@ttstorageaccount102.blob.core.windows.net',

mount_point = '/mnt/retaildb1',

extra_configs={'fs.azure.account.key.ttstorageaccount102.blob.core.windows.net':

dbutils.secrets.get('databricks-demo-scope','databricksnewkey')}

)

databricks secrets list --scope databricks-demo-scope

Key name Last updated

---------- ----------

databricksdemokey 1660770175000

databricksnewkey 1660771040000

2. databricks backed secret scope - encrytped databricks database

Azure Databricks - session 13

==========

Databricks backed secret scope

CLI, or API, but we cannot do using UI

databricks secrets create-scope --scope dempscopedatabricksbacked --initial-manage-principal users

databricks secrets list-scopes

Scope Backend KeyVault URL

---------- ---------- ----------

databricks-demo-scope AZURE_KEYVAULT https://ttdatabricksdemokv.vault.azure.net/

dempscopedatabricksbacked DATABRICKS N/A

databricks secrets put --scope dempscopedatabricksbacked --key storageKey

databricks secrets list --scope dempscopedatabricksbacked

Key name Last updated

---------- ----------

storageKey 1660772322857

dbutils.fs.mount(

source = 'wasbs://retaildb@ttstorageaccount102.blob.core.windows.net',

mount_point = '/mnt/retaildb2',

extra_configs={'fs.azure.account.key.ttstorageaccount102.blob.core.windows.net':

dbutils.secrets.get('dempscopedatabricksbacked','storageKey')}

)

df1 = spark.read.csv('/mnt/retaildb2/raw/customers.csv',header = True)

display(df1)

spark.conf.set("fs.azure.account.auth.type.ttstorageaccount102.dfs.core.windows.net","SAS")

spark.conf.set("fs.azure.sas.token.provider.type.ttstorageaccount102.dfs.core.windows.net","org.apa

che.hadoop.fs.azurebfs.sas.FixedSASTokenProvider")

spark.conf.set("fs.azure.sas.fixed.token.ttstorageaccount102.dfs.core.windows.net","sv=2021-06-08&

ss=bfqt&srt=sco&sp=rwdlacupyx&se=2022-08-18T05:48:16Z&st=2022-08-17T21:48:16Z&spr=https&

sig=warKiMqKWEqHF5H%2BQtAUnMZarilRkpHISOOtGwqzrls%3D")

Azure Databricks - session 14

==========

3 ways to access the storage account

==========

1. access key/account key - all access to the storage account

2. SAS key - Shared access signature - (we can give access at container also)

3. Service principal - fine grained control on folder level

SAS Key

========

sv=2021-06-08&ss=bfqt&srt=sco&sp=rwdlacupyx&se=2022-08-18T06:01:06Z&st=2022-08-17T22:01:

06Z&spr=https&sig=yoe%2FgjGkmmhwDJkqbpQHJaDmfEtDjGulKPgMzkBRUuQ%3D

spark.conf.set("fs.azure.account.auth.type.ttstorageaccount102.dfs.core.windows.net","SAS")

spark.conf.set("fs.azure.sas.token.provider.type.ttstorageaccount102.dfs.core.windows.net","org.apa

che.hadoop.fs.azurebfs.sas.FixedSASTokenProvider")

spark.conf.set("fs.azure.sas.fixed.token.ttstorageaccount102.dfs.core.windows.net","sv=2021-06-08&

ss=bfqt&srt=sco&sp=rwdlacupyx&se=2022-08-18T06:01:06Z&st=2022-08-17T22:01:06Z&spr=https&

sig=yoe%2FgjGkmmhwDJkqbpQHJaDmfEtDjGulKPgMzkBRUuQ%3D")

df =

spark.read.csv("abfs://retaildb@ttstorageaccount102.dfs.core.windows.net/raw/orders.csv",header

= True)

display(df)

Service Principal

==================

azure active directory -> app registrations

databricks-demo-spnew

Application (client) ID

9f178d90-1bb3-43dc-b60f-51552e64e418

Object ID

b258c88f-091a-43f2-9314-9d2cd4c9f873

Directory (tenant) ID

a216c39d-a41f-4225-8bca-11c3f51bb637

go to certificates and secrets

pQV8Q~VXv9CLGePn.ER3M51tMb-lNsygcG-fVbrC

service principal is created but we need to give access to our storage account

click on access control IAM

spark.conf.set("fs.azure.account.auth.type.ttstorageaccount102.dfs.core.windows.net","OAuth")

spark.conf.set("fs.azure.account.oauth.provider.type.ttstorageaccount102.dfs.core.windows.net","or

g.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set("fs.azure.account.oauth2.client.id.ttstorageaccount102.dfs.core.windows.net","9f178

d90-1bb3-43dc-b60f-51552e64e418")

spark.conf.set("fs.azure.account.oauth2.client.secret.ttstorageaccount102.dfs.core.windows.net","p

QV8Q~VXv9CLGePn.ER3M51tMb-lNsygcG-fVbrC")

spark.conf.set("fs.azure.account.oauth2.client.endpoint.ttstorageaccount102.dfs.core.windows.net",

"https://login.microsoftonline.com/a216c39d-a41f-4225-8bca-11c3f51bb637/oauth2/token")

df =

spark.read.csv("abfs://retaildb@ttstorageaccount102.dfs.core.windows.net/raw/orders.csv",header

= True)

display(df)

Delta Lake Session 1

==========

Its a file format..

1. what is a data lake?

A data lake is a storage repository that holds a vast amount of raw data in its native format.

Data lake stores all the data in the form of files.

can hold any kind of data.

examples are: Amazon s3, ADLS gen2, GCS etc..

2. advantages of a data lake

cost effective

scalable

any kind of data (structured, semi structured, unstructured)

3. challenges of a data lake

ACID guarantees

when we perform transactions we required ACID guarantees.

Any database will provide these ACID guarantees.

HDFC Bank account

25000 Rs in your account

your friend initially has 50k

min balance of 5000

ACID

Atomicity - all or none

10k will be deducted from your account

your friends account will be added with 10k

Consistency -

10k will be deducted from your account

your friends account will be added with 10k

lets say you are transferring 25k to your friend

0 Rs

75k Rs

Isolation -

before 10 pm

10k will be deducted from your account - 10 pm

your friends account will be added with 10k - 10.02 pm

Reader - 10.01 (your balance and your friend)

10 < 10.01 < 10.02

your balance is 15k

your friends balance is 50k

Durability - changes should be permanent and system failure should not hamper the results..

ACID properties in a database

Delta Lake Session 2

==========

Data lake do not provide ACID guarantees

when you run a spark job then multiple tasks underneath and each generate a part file..

1. A job is failing while appending the data

output folder - 5 part files.

you are running a job which will also create 5 part files.

2 files are generated...

after that the job fails...

7 files

the reader will read 7 files

Atomicity, Consistency

2. A job is failing while overwriting the data

overwrite is a 2 step process..

1) deleting what is already there

2) write the new files

5 part files are already present in the output folder

and after that we ran the job

2 part files are generated and then the job fails

Atomicty, Consistency, Durability

3. Simultaneous reads/writes

5 part available

and then you invoked the spark job

2 part are written

reader is trying to read while part file 3 is getting processed.

totally 7 files

here isolation has gone for a toss

4. Append the data with a different Schema

you have 5 part files... and each of these files have 5 columns..

Now we are running a job which will write 5 more part files.. but all these new files have 7

columns..

totally 10 files are there

5 files have 5 columns each

5 files have 7 columns each

Data Reliablity issues

no data validation

we are corrupting our data in failure cases

NO DML operations

Data Quality issues

Difficult to maintain historical versions

=> Delta Lake comes for our rescue

Delta Lake Session 3

==========

Some improvement on top of data lakes that solves the above challenges.

Delta lake is an open source storage layer

it is like a small utility installed on your spark cluster.

Parquet format

- it is a column based file format

- very well used with Apache spark

- it embeds the metadata

Delta = Parquet + Transaction logs

df for orders data

df.write.format("parquet")

df.write.format("delta")

5 tasks are running

5 partfiles will be created

output folder - 5 partfiles

_delta_logs

00000.json

operation 1 - write

==========

part-000.parquet

part-001.parquet

part-002.parquet

part-003.parquet

part-004.parquet

_delta_log

00000.json

Operation 2 - append

==========

part-005.parquet

part-006.parquet

_delta_log

00001.json

Operation 3 - append

==========

part-007.parquet

_delta_log

00002.json

for each write operation

- All part files is written first

- A transaction log file is added to _delta_log folder (Json format)

for each read operation

- transaction log files are read first

- part files are read based on the above log files

1. A job failing while appending

part-000.parquet

part-001.parquet

part-002.parquet

part-003.parquet

part-004.parquet

_delta_log

00000.json -

add part-000.parquet

add part-001.parquet

..

..

..

5 more part files

part-005.parquet

part-006.parquet

--> job fails

If a reader reads they will see only 5 part files...

Atomicity, Consistency

2. A job failing while overwriting

overwrite is a 2 step process..

1) deleting what is already there

2) write the new files

5 part files are already present in the output folder

and after that we ran the job

2 part files are generated and then the job fails

Atomicty, Consistency, Durability

with Delta Lake the previous files are not deleted

it will start writing the new files

if all the 5 files are written properly...

_delta_log

00000.json

==========

add part-000.parquet

add part-001.parquet

add part-002.parquet

add part-003.parquet

add part-004.parquet

00001.json

==========

add part-005.parquet

add part-006.parquet

add part-007.parquet

add part-008.parquet

add part-009.parquet

remove part-000.parquet

remove part-001.parquet

remove part-002.parquet

remove part-003.parquet

remove part-004.parquet

add part-005.parquet

add part-006.parquet

add part-007.parquet

add part-008.parquet

add part-009.parquet

3) Simultaneous reads/writes

Simultaneous reads/writes

5 part available

and then you invoked the spark job

2 part are written

reader is trying to read while part file 3 is getting processed.

totally 7 files

here isolation has gone for a toss

00000.json

==========

add part-000.parquet

add part-001.parquet

add part-002.parquet

add part-003.parquet

add part-004.parquet

00001.json

==========

when a reader is reading they will still see just 00000.json

Updates and Deletes

==========

how a delta lake provides updates and deletes (DML operations) without compromising on ACID

guarantees.

Updates

========

Data files

==========

part-000.parquet

part-001.parquet

part-002.parquet

part-003.parquet

part-004.parquet

00000.json - transaction log files

==========

add part-000.parquet

add part-001.parquet

add part-002.parquet

add part-003.parquet

add part-004.parquet

employee id, employee name, salary

101, Kapil, 10000

this particular record is in part-001.parquet

part-001.parquet

================

101, Kapil, 10000

102, Sachin, 20000

103, Ramesh, 25000

Part-005.parquet

==========

101, Kapil, 15000

102, Sachin, 20000

103, Ramesh, 25000

00000.json - transaction log files

==========

add part-000.parquet

add part-001.parquet

add part-002.parquet

add part-003.parquet

add part-004.parquet

00001.json - transaction log files

==========

add Part-005.parquet

remove part-001.parquet

part-000.parquet

part-002.parquet

part-003.parquet

part-004.parquet

part-005.parquet

Delete

========

Data files

==========

part-000.parquet

part-001.parquet

part-002.parquet

part-003.parquet

part-004.parquet

00000.json - transaction log files

==========

add part-000.parquet

add part-001.parquet

add part-002.parquet

add part-003.parquet

add part-004.parquet

part-001.parquet

==========

101, Kapil, 10000

102, Sachin, 20000

103, Ramesh, 25000

part-005.parquet

==========

102, Sachin, 20000

103, Ramesh, 25000

00001.json - transaction log files

==========

add part-005.parquet

remove part-001.parquet

when a reader is reading

part-000.parquet

part-002.parquet

part-003.parquet

part-004.parquet

part-005.parquet

issues with schema change can be taken care..

version history using time travel

apply data quality checks

performance improvement...

delta lake provides database like features on top of your data lake.

Delta Lake session 4

==========

Setup

azure portal -

we created a databricks workspace with the name tt-azure-databricks-ws

we created a storage account with the name ttstorageaccount100

we created a container with the name inputdatasets

we uploaded customers.csv and orders.csv to the inputdatasets container

we have created a minimum configuration cluster in databricks. (Standard_F4)

dbutils.fs.mount(

source = 'wasbs://inputdatasets@ttstorageaccount100.blob.core.windows.net',

mount_point = '/mnt/retaildb',

extra_configs={'fs.azure.account.key.ttstorageaccount100.blob.core.windows.net':'LwbzpkkuVw1Yu4

cV96g5an1MTvF4c7T+SgKgtjI8B4JHRjmIiA1HX794W0kFsNfC9XfymQS3KLEH+ASTt28LTw==''}

)

%fs ls /mnt/retaildb

df = spark.read.csv("/mnt/retaildb/orders.csv", header = True)

display(df)

the orders data I want to write in parquet format at some location

df.write.mode("overwrite").partitionBy("order_status").format("parquet").save("/mnt/retaildb/parq

uet/orders.parquet")

df.write.mode("overwrite").partitionBy("order_status").format("delta").save("/mnt/retaildb/delta/or

ders.delta")

Delta Lake session 5

==========

%sql

create database if not exists retaildb

%sql

create table retaildb.ordersparquet using parquet location "/mnt/retaildb/parquet/orders.parquet"

%sql

drop table retaildb.ordersparquet

%sql

create table retaildb.ordersparquet using parquet location

"/mnt/retaildb/parquet/orders.parquet/*"

%sql

create table retaildb.ordersdelta using delta location "/mnt/retaildb/delta/orders.delta"

%sql

select * from retaildb.ordersdelta

%sql

describe table extended retaildb.ordersdelta

dbutils.fs.rm("path")

we followed a 2 step process -

1. first we added the files in the storage account

2. we created external table on top of that

step 1:

df.write.mode("overwrite").partitionBy("order_status").format("delta").save("/mnt/retaildb/delta/or

ders.delta")

step 2:

%sql

create table retaildb.ordersdelta using delta location "/mnt/retaildb/delta/orders.delta"

df.write.mode("overwrite").partitionBy("order_status").format("delta").option("path","/mnt/retaildb

/deltalatest/orders.delta").saveAsTable("retaildb.ordersdeltatable")

when we run it multiple times.. we see multiple files getting created... (the previous files are not

deleted)

Delta Lake session 6

==========

The table is created..

Inserting into Delta Table

1. Insert Command (done)

2. Append (done)

3. Copy Command

retaildb.ordersdelta

describe history retaildb.ordersdelta

insert into retaildb.ordersdelta values ('111111111','2013-07-25 00:00:00.0','222222222','CLOSED')

describe history retaildb.ordersdelta

dfnew = spark.read.csv("/mnt/retaildb/ordersappend.csv", header = True)

dfnew.write.mode("append").partitionBy("order_status").format("delta").save("/mnt/retaildb/delta/

orders.delta")

%sql

copy into retaildb.ordersdelta from "/mnt/retaildb/orders1.csv" fileformat = CSV

format_options('header' = 'true')

Delta Lake session 7

==========

schema mismatch

%sql

copy into retaildb.ordersdelta from "/mnt/retaildb/ordersnew.csv" fileformat = CSV

format_options('header' = 'true')

df.write.mode("append").partitionBy("order_status").format("delta").save("/mnt/retaildb/delta/orde

rs.delta")

what if you want schema evolution..

earlier there were rows with 4 columns..

now we want to add rows with 5 columns...

df.write.mode("append").partitionBy("order_status").format("delta").option("mergeSchema","true")

.save("/mnt/retaildb/delta/orders.delta")

how to perform updates and deletes

==========

%sql

select * from retaildb.ordersdelta where order_id = 5

%sql

update retaildb.ordersdelta set order_status = 'CLOSED' where order_id = 5

complete

50 records

remove 50 complete

add complete

49 records

add closed

1 record

describe history retaildb.ordersdelta

50 records in a file1

we are deleting one record from this..

then it will create a file2 with 49 records..

add file2 - 49

delete file1 - 50

Delta Lake session 8

==========

Apply constraints

access history using time travel

NOT NULL

CHECK

```
%sql

alter table retaildb.ordersdelta alter column customer_id SET not NULL

%sql

insert into
retaildb.ordersdelta(order_id,order_date,customer_id,order_status,order_amo
unt)

values('4','2013-07-25 00:00:00.0',null,'CLOSED','40')

%sql

insert into
retaildb.ordersdelta(order_id,order_date,customer_id,order_status,order_amo
unt)

values('4','2013-07-25 00:00:00.0','101','CLOSED',null)

%sql

alter table retaildb.ordersdelta add constraint status_ck check (order_status in

('ON_HOLD','PAYMENT_REVIEW','PROCESSING','CLOSED','SUSPECTED_
FRAUD','COMPLETE','PENDING

','CANCELED','PENDING_PAYMENT'))

%sql

insert into retaildb.ordersdelta values('4','2013-07-25
00:00:00.0','101','CLOSEDNEW','40')
```

Access the history using time travel

==========

```
%sql

select * from retaildb.ordersdelta where order_id = 6

%sql

select * from retaildb.ordersdelta version as of 2 where order_id = 6

%sql

select count(*) from retaildb.ordersdelta version as of 0
```

2022-09-23T14:01:01.000+0000

select * from retaildb.ordersdelta timestamp as of '2022-09-23T14:01:01.000+0000'

the log retention is for 30 days by default.

%sql

restore table retaildb.ordersdelta to version as of 0