# BigData Spark – Hands-On

## SQL, Temporary View
## Hive MetaStore, ExternalCatalog, Table Partitions

Arnaud Nauwynck

Nov 2023

# Objectives of Hands-On

Reminders:
Local Install, spark-shell
Spark File IO

1/ CreateTempView,  use Dataset from SQL

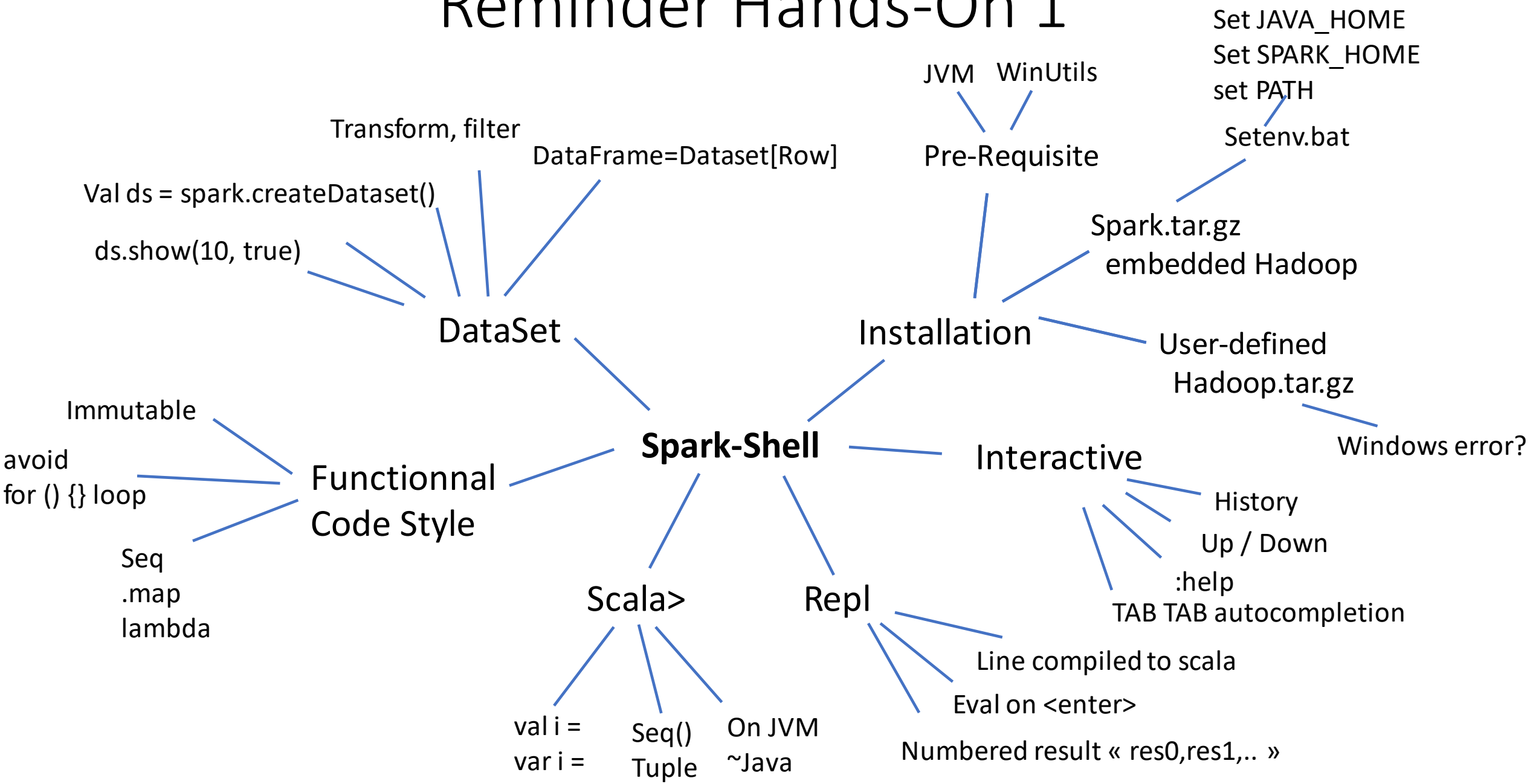
2/ Configure HiveMetastore DB

3/ connect Spark External Catalog  to MetaStore

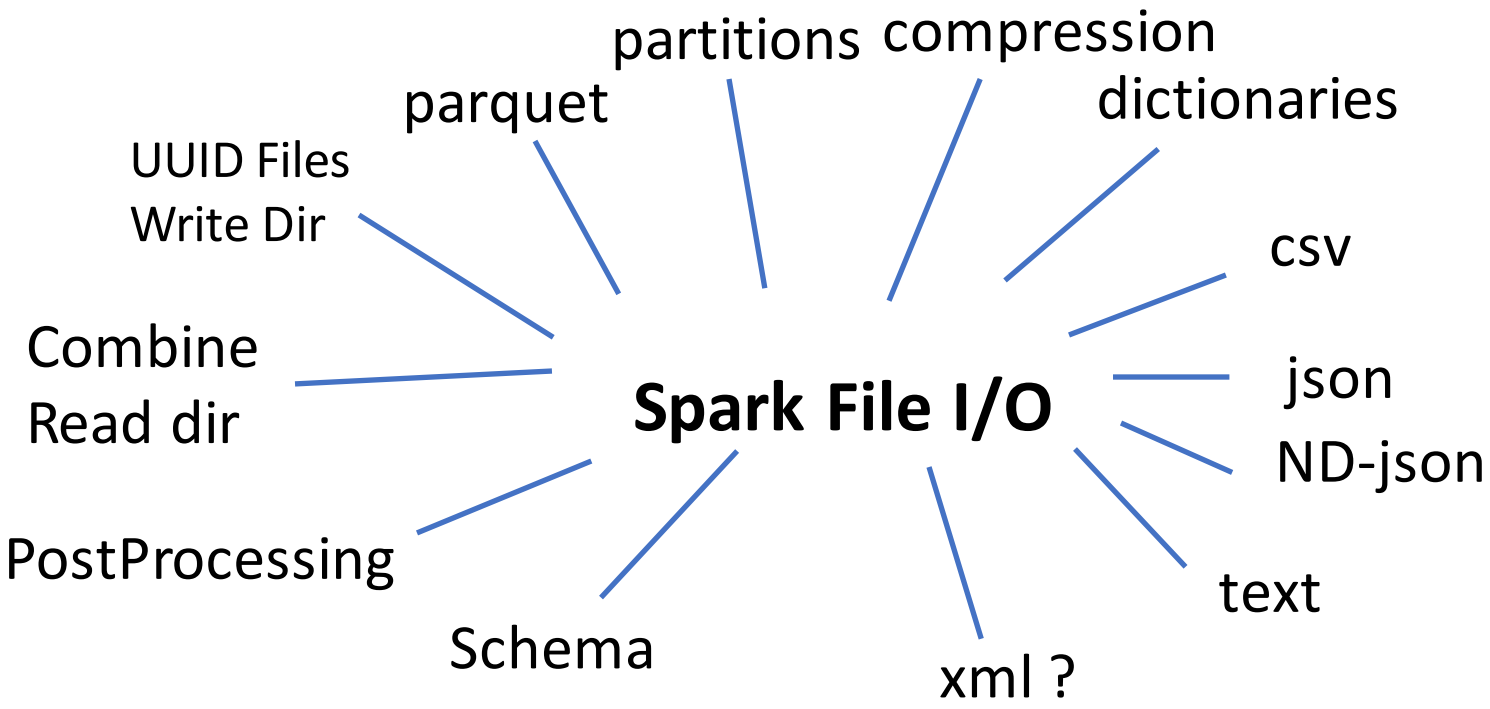
4/ Create Table

5/ (Directory) Partitioned Table

# Reminder Hands-On 1



# Reminder Hands-On 2



# Objectives of Hands-On 3

1/ CreateTempView,  use Dataset from SQL

2/ Configure HiveMetastore DB

3/ connect Spark External Catalog  to MetaStore

4/ Create Table

5/ (Directory) Partitioned Table

# Exercise 1: check spark conf spark.sql.catalogImplementation=in-memory … not « hive »

a/ execute line

scala> **spark.sharedState.sparkContext.getConf.get("spark.sql.catalogImplementation")**

Res1: String = in-memory

b/ expecting for now to have «**in-memory**»

.. not « **hive** » (not configured / running yet !)

c/ If not correct, relaunch spark-shell using

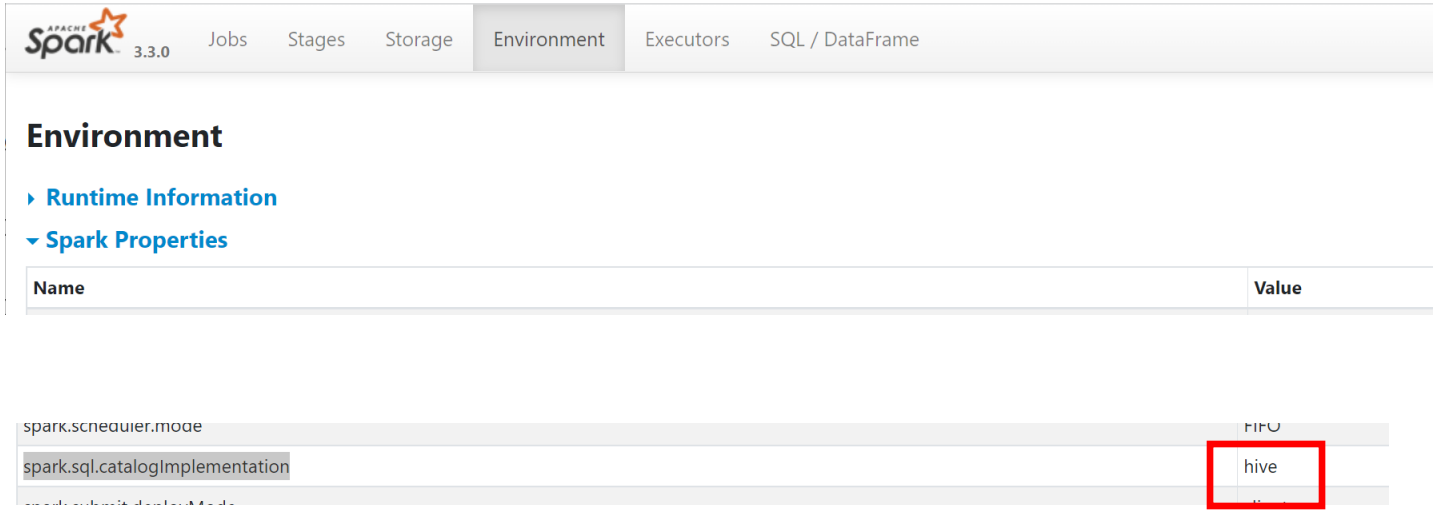
**spark-shell --conf spark.sql.catalogImplementation=in-memory**

# Checking from Spark-UI

a/ Open Spark-UI http://localhost:4040 … more on this later

b/ browse to Environment > Spark Properties >

c/ ensure you have « in-memory »




WRONG for now..

# Re-Checking Second Time …

println(spark.sharedState.externalCatalog.unwrapped)

```
scala> println(spark.sharedState.externalCatalog.unwrapped)
org.apache.spark.sql.catalyst.catalog.InMemoryCatalog@50f6eb17
```

Else, even this will fail …

```
Caused by: org.apache.hadoop.hive.metastore.api.MetaException: Could not connect to meta store using any of the URIs provided. Most recent failure: org.apach
e.thrift.transport.TTransportException: java.net.ConnectException: Connection refused: connect
        at org.apache.thrift.transport.TSocket.open(TSocket.java:226)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.open(HiveMetaStoreClient.java:478)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.<init>(HiveMetaStoreClient.java:245)
```

# Exercise 2: reload address DataSet ensure no OutOfMemoryError

a/ Load the Dataset of addresses from previous hands-on.

scala> **val allAddressDs = spark.read.parquet**("C:/data/OpenData-gouv.fr/bal/adresses-parquet")

b/ check the data… using « **.count** » and « **.show**(2, false) »

c/ if encountering OutOfMemory Error .. Then increase memory (cf next)

```
scala> allAddressDs.count
22/09/29 16:11:08 WARN BlockManager: Block rdd_8_1 could not be removed as it was not found on disk or in memory
22/09/29 16:11:08 WARN BlockManager: Block rdd_8_5 could not be removed as it was not found on disk or in memory
22/09/29 16:11:08 WARN BlockManager: Block rdd_8_4 could not be removed as it was not found on disk or in memory
22/09/29 16:11:08 ERROR Executor: Exception in task 5.0 in stage 2.0 (TID 7)
java.lang.OutOfMemoryError: Java heap space
22/09/29 16:11:08 ERROR Executor: Exception in task 1.0 in stage 2.0 (TID 3)
java.lang.OutOfMemoryError: Java heap space
        at java.nio.HeapByteBuffer.<init>(HeapByteBuffer.java:57)
        at java.nio.ByteBuffer.allocate(ByteBuffer.java:335)
        at org.apache.parquet.bytes.HeapByteBufferAllocator.allocate(HeapByteBufferAllocator.java:32)
        at org.apache.parquet.hadoop.ParquetFileReader$ConsecutivePartList.readAll(ParquetFileReader.java:1696)
```

Reminder

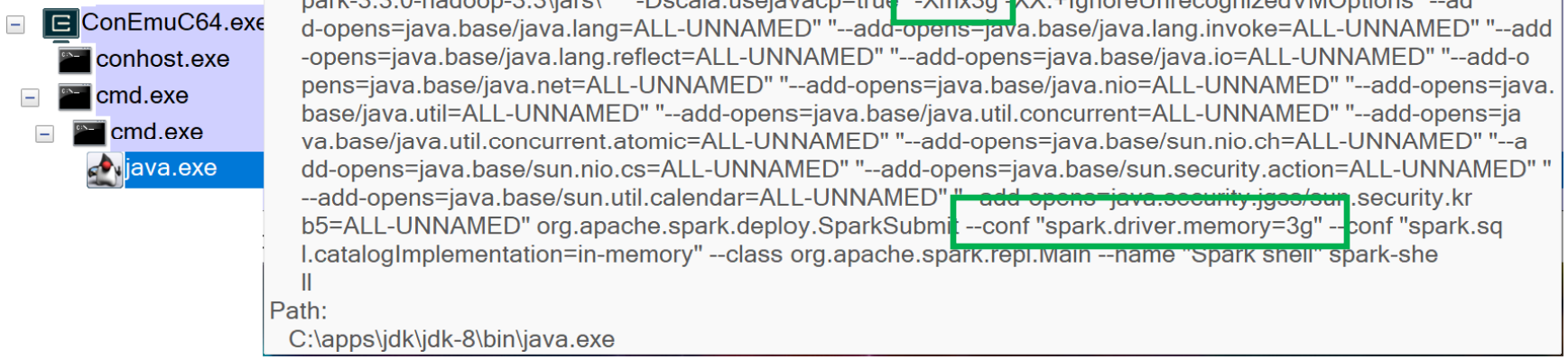
# Dataset… read CSV / write Parquet / load

```
val csvDs = spark.read.format("csv").option("delimiter",";").option("header",true)
.schema("""
uid_adresse string,  cle_interop string NOT NULL, commune_insee string NOT NULL, commune_nom string,
commune_deleguee_insee integer,  commune_deleguee_nom string,
voie_nom string, lieudit_complement_nom string, numero integer NOT NULL,
suffixe string, position string,  x double, y double, long double, lat double,  cad_parcelles string, source string,
date_der_maj string, certification_commune integer NOT NULL
""")
.load("C:/data/OpenData-gouv.fr/bal")

csvDs.printSchema
csvDs.show(1, false)

val csvDsNoXY = csvDs.drop("x", "y").sortWithinPartitions("commune_insee", "voie_nom", "numero")
csvDsNoXY.write.format("parquet").save("C:/data/OpenData-gouv.fr/bal-parquet")

val ds = spark.read.format("parquet").load("C:/data/OpenData-gouv.fr/bal-parquet")
ds.cache()
```

Details

# Ensure Memory JVM Argument –Xmx3g

**spark-shell --conf spark.sql.catalogImplementation=in-memory --driver-memory=3g**



# Exercise 3:  createTempView … then SQL

Execute following

```
allAdressDs.createTempView("tmp_address")

spark.sql("SELECT count(*) FROM tmp_address").show

spark.sql("SELECT * FROM tmp_address").show(2, false)

spark.sql("SELECT * FROM tmp_address WHERE commune_nom='Paris'").show(2, false)
```

# Exercise 4: list Tables

```
spark.sql("SHOW TABLES").show

spark.catalog.listTables.show(false)
```

# Exercise 5: Describe Table / printSchema

Execute

a/ spark.sql("select * from tmp_address")**.printSchema**

b/ spark.sql("**describe table** tmp_address").show(false)

c/ spark.sql("**show create table** tmp_address").show(false) // …. WILL FAIL … no Sql DDL for it
// error: not a « table » (but describe ok!)

# Exercise 6: stop + restart .. temporary

a/ stop spark-shell

b/ restart spark-shell

c/ list tables …
Check temporary tables have disapeared !

Optional

# Optional Exercise 7: createGlobalTemporaryView

https://spark.apache.org/docs/latest/sql-getting-started.html#global-temporary-view

a/ similar to Exercise 3,
« **createGlobalTemporaryView** » from your dataset
instead of a « createTempView »

b/ forget about finding it from default spark catalog ( … confusing )
list explicitly from:
spark.catalog.listTables(**« global_temp »**).show(false)

c/ execute SQL query on « **global_temp.**address »
select count(*) from global_temp.address

d/ explain difference between global temp and temp

# Objectives of Hands-On

1/ CreateTempView,  use Dataset from SQL

2/ Configure HiveMetastore DB

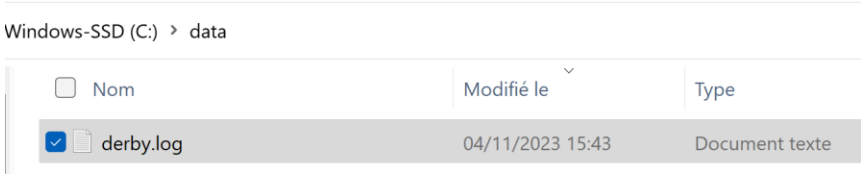
3/ connect Spark External Catalog  to MetaStore

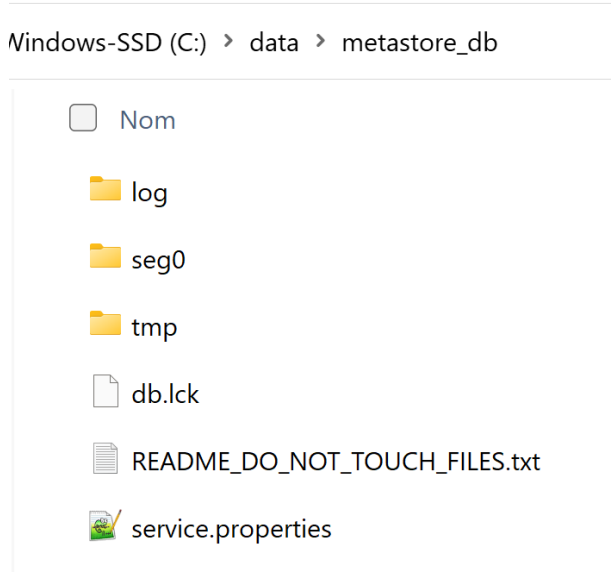
4/ Create Table

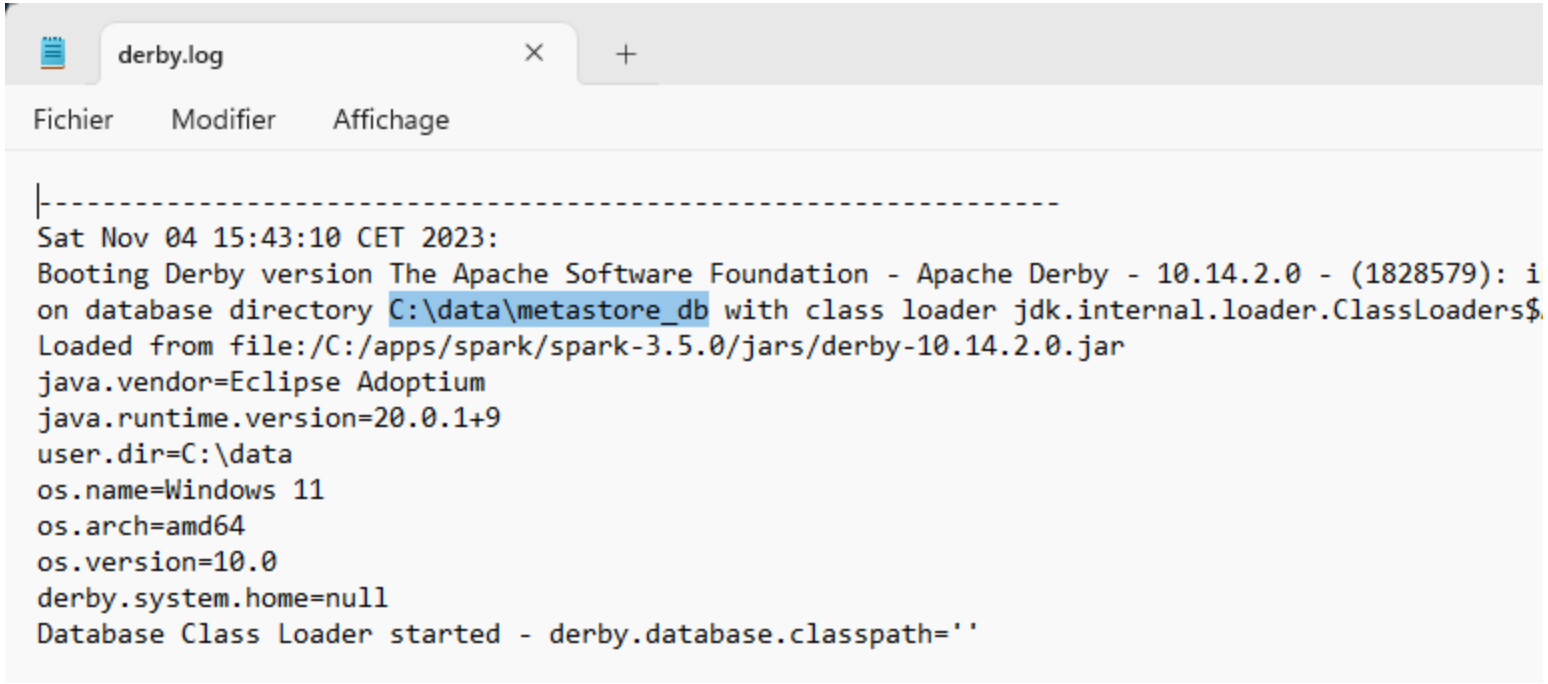
5/ (Directory) Partitioned Table

Optional

# Default hive… using Derby metastore_db
# cf next slides if explicitly using Postgresql DB



Optional

# Install Hive Standalone Metastore

https://downloads.apache.org/hive/hive-standalone-metastore-3.0.0/

https://downloads.apache.org/hive/

**Index of /hive**

| Name | Last modified |
|---|---|
| Parent Directory | |
| hive-1.2.2/ | 2022-06-17 12:34 |
| hive-2.3.9/ | 2022-06-17 12:34 |
| hive-3.1.2/ | 2022-06-17 12:34 |
| hive-3.1.3/ | 2022-06-17 12:34 |
| hive-4.0.0-alpha-1/ | 2022-06-17 12:34 |
| hive-standalone-metastore-3.0.0/ | 2022-06-17 12:34 |
| hive-storage-2.7.3/ | 2022-06-17 12:34 |
| hive-storage-2.8.1/ | 2022-06-17 12:34 |
| stable-2/ | 2022-06-17 12:34 |
| KEYS | 2022-03-23 18:19 |

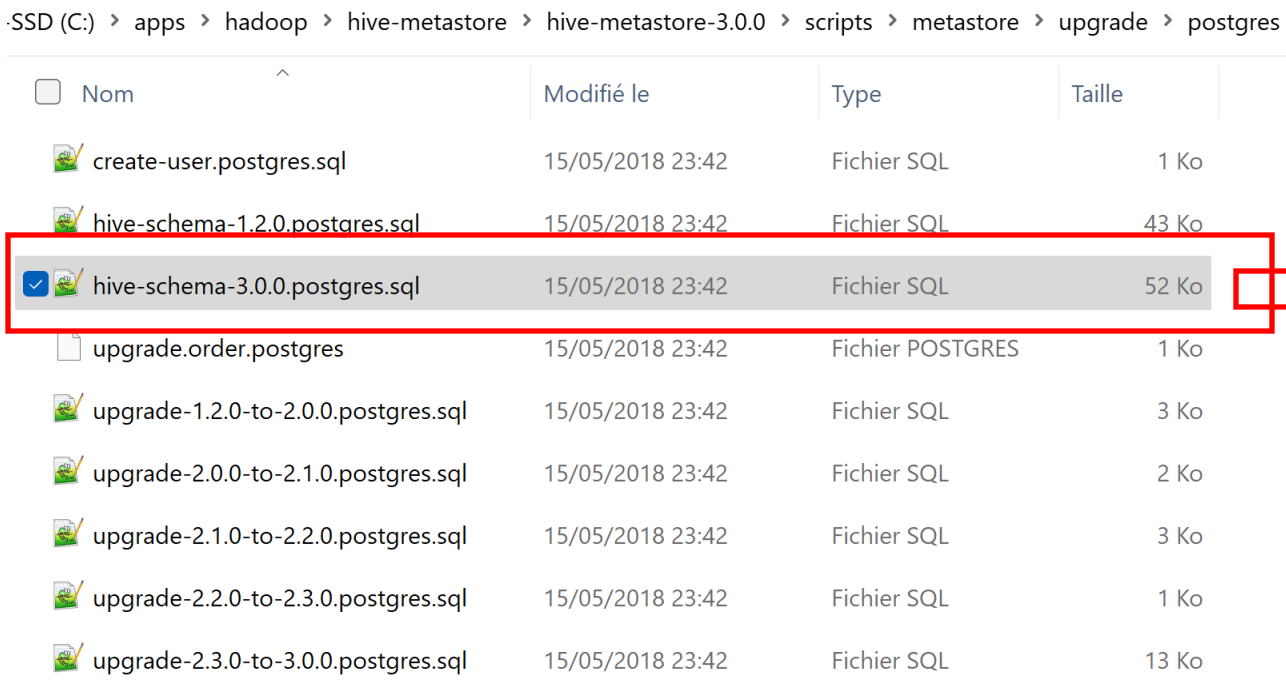
**Index of /hive/hive-standalone-metastore-3.0.0**

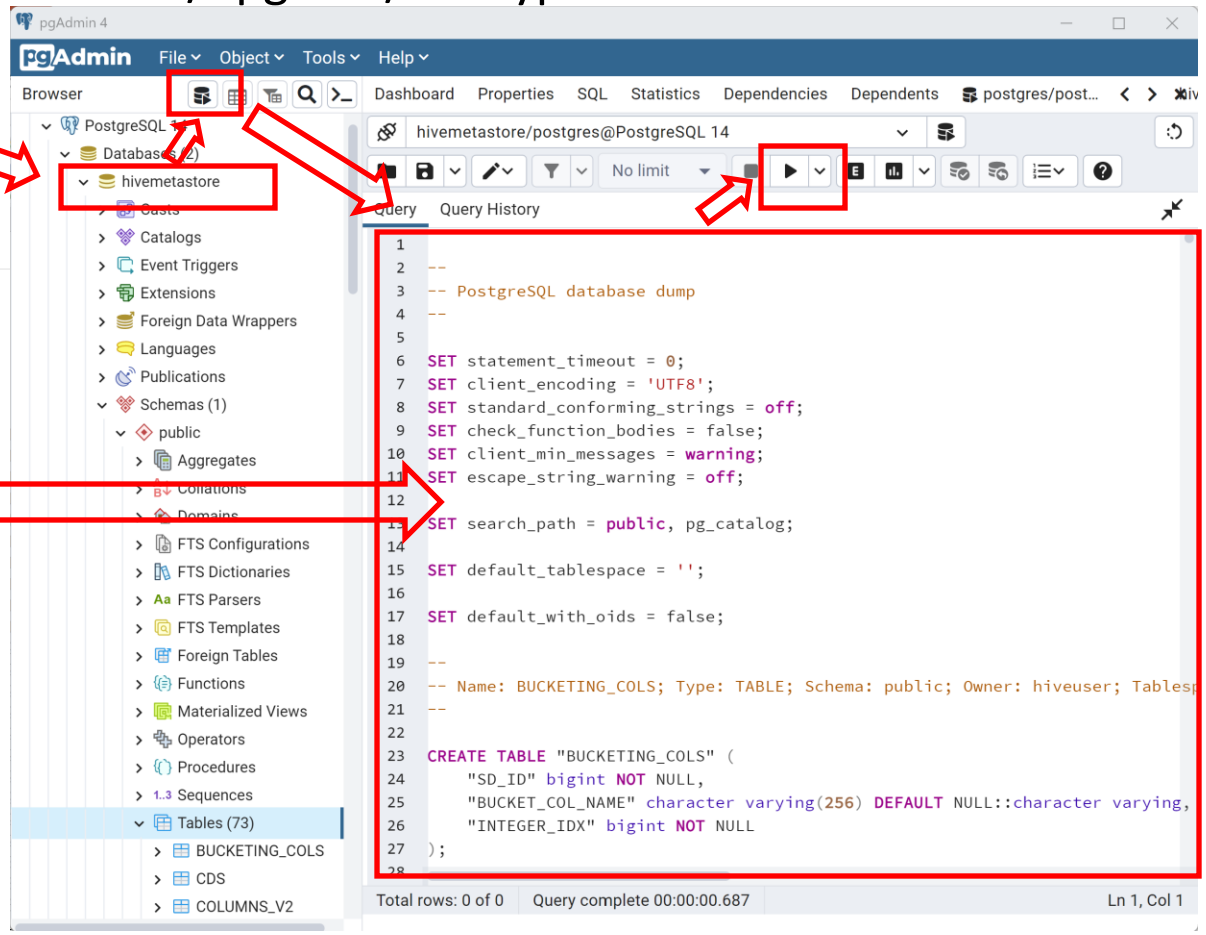
| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| hive-standalone-metastore-3.0.0-bin.tar.gz | 2018-06-07 17:56 | 25M | |
| hive-standalone-metastore-3.0.0-bin.tar.gz.asc | 2018-06-07 17:56 | 832 | |
| hive-standalone-metastore-3.0.0-bin.tar.gz.sha256 | 2018-06-07 17:56 | 109 | |
| hive-standalone-metastore-3.0.0-src.tar.gz | 2018-06-07 17:56 | 2.3M | |
| hive-standalone-metastore-3.0.0-src.tar.gz.asc | 2018-06-07 17:56 | 832 | |
| hive-standalone-metastore-3.0.0-src.tar.gz.sha256 | 2018-06-07 17:56 | 109 | |

Optional

# Example using Postgres : Create Hive Schema

1/ Open PgAdmin
2/ Create a « Database »
3/ execute SQL script « hive-schema-<dbType>.sql »
copied from <downloadedHiveDir>/scripts/metastore/upgrade/<dbType>



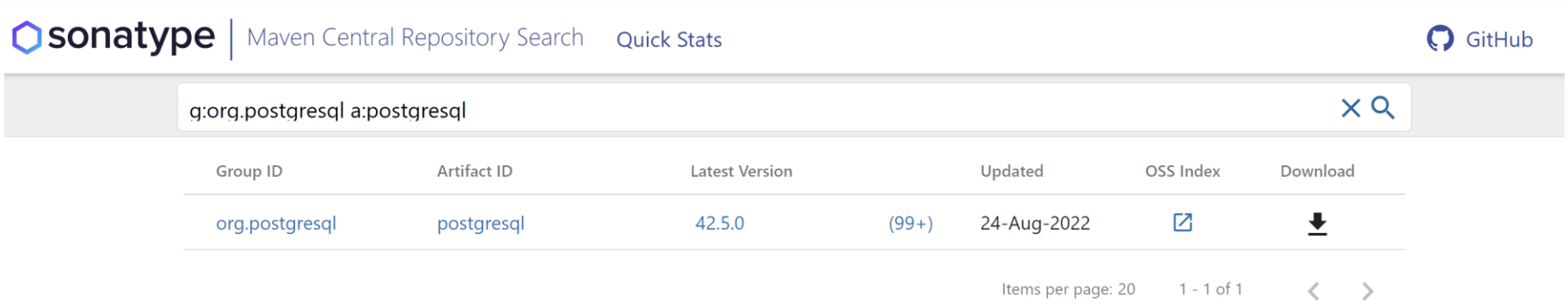



Optional

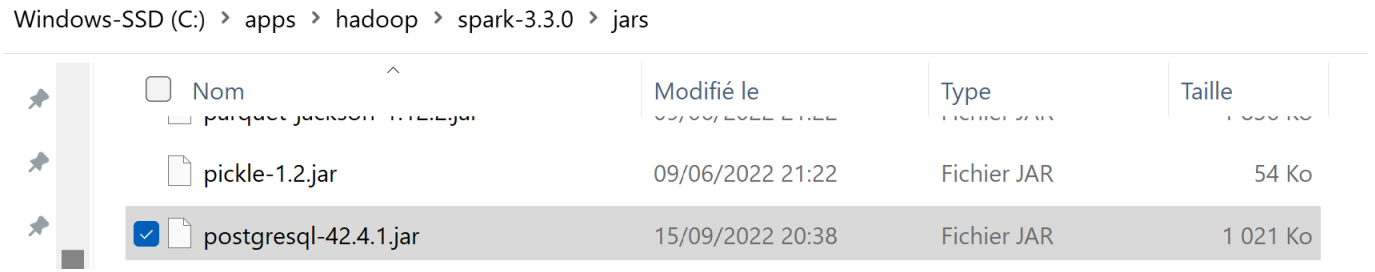
# Copy Jar .. Example using Postgresql

Step 1/ find g:org.postgresql a:postgresql

https://search.maven.org/search?q=g:org.postgresql%20a:postgresql




Step 2/ copy jar to SPARK_HOME/jars



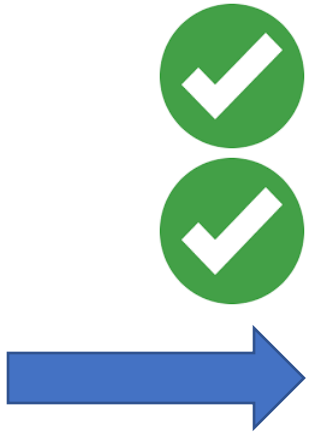
Optional

# Configure hive metastore in HADOOP_CONF_DIR (or HADOOP_HOME) /conf/hive-site.xml

hive-site.xml

```
22  <property>
23    <name>hive.metastore.db.type</name>
24    <value>postgres</value>
25  </property>
26  <property>
27    <name>javax.jdo.option.ConnectionDriverName</name>
28    <value>org.postgresql.Driver</value>
29  </property>
30  <property>
31    <name>javax.jdo.option.ConnectionURL</name>
32    <value>jdbc:postgresql://localhost/hivemetastore</value>
33  </property>
34  <property>
35    <name>javax.jdo.option.ConnectionUserName</name>
36    <value>hivemetastore-user</value>
37  </property>
38  <property>
39    <name>javax.jdo.option.ConnectionPassword</name>
40    <!-- <value> ... </value>  -->
41  </property>
42
```

# Objectives of Hands-On



1/ CreateTempView,  use Dataset from SQL

2/ Configure HiveMetastore DB

3/ connect Spark External Catalog  to MetaStore

4/ Create Database, Table

5/ (Directory) Partitioned Table

# Restart spark-shell + Check externalCatalog

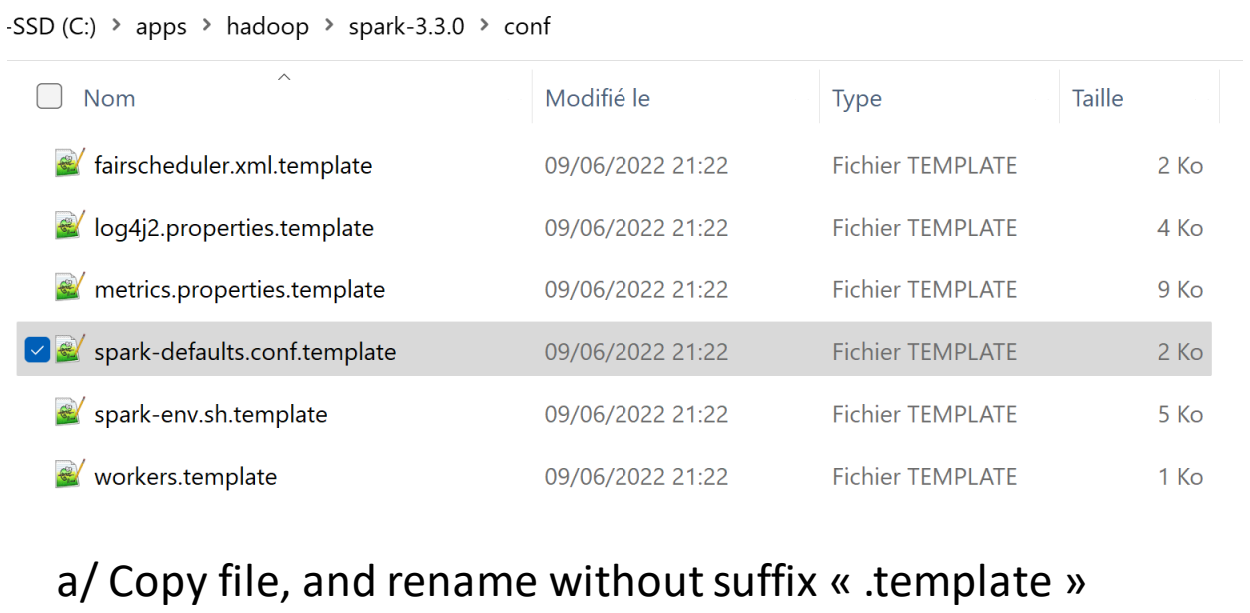
relaunch spark-shell
**spark-shell --driver-memory 5g --conf spark.sql.catalogImplementation=hive**

Check in sparkContext
**spark.sharedState.sparkContext.getConf.get("spark.sql.catalogImplementation")**
**res1: String = "hive"**

**println(spark.sharedState.externalCatalog.unwrapped)**

Check in Spark UI

# Optional Exercise 8: View/Change spark-defaults.conf




a/ Copy file, and rename without suffix « .template »
Edit to add line

**spark.sql.catalogImplementation hive**

Check conf can still be overloadded by « --conf key=value »
check conf at runtime:
spark.sharedState.sparkContext.getConf.get(« key »)
Or in Spark UI

# Objectives of Hands-On

1/ CreateTempView,  use Dataset from SQL

2/ Configure HiveMetastore DB

3/ connect Spark External Catalog  to MetaStore

4/ Create Database, Table

5/ (Directory) Partitioned Table

# List Databases

```
scala> spark.catalog.listDatabases().show(false)
+-------+--------------------+--------------------------------------+
|name   |description         |locationUri                           |
+-------+--------------------+--------------------------------------+
|default|Default Hive database|file:/C:/apps/hadoop/spark-warehouse|
+-------+--------------------+--------------------------------------+
```

# Exercise 9: Create Database, Create if not exists

Execute:

spark.sql("**CREATE DATABASE** db1").show(false)

spark.sql("**CREATE DATABASE IF NOT EXISTS** db1").show(false)

Then, search created database

10 mn pause

# Exercise 10: saveAsTable

Try some variations:

```
// val ds = spark.read.format("parquet").load("C:/data/OpenData-gouv.fr/bal-parquet")

ds.write.saveAsTable("addr")

ds.write.saveAsTable("db1.addr")

ds.write.format("parquet").saveAsTable("db1.addr")
```

# Exercise 10 : CTAS
# CREATE TABLE <db>.<table> AS SELECT

CREATE TABLE … => STUPID default format « text »

```
scala> spark.sql("CREATE TABLE db1.address_copy AS SELECT * FROM tmp_address LIMIT 100")
22/10/01 15:20:38 WARN ResolveSessionCatalog: A Hive serde table will be created as there is no table provider specified. You can set spark.sql.legacy.create
HiveTableByDefault to false so that native data source table will be created instead.
22/10/01 15:20:38 WARN HiveMetaStore: Location: file:/C:/apps/hadoop/spark-warehouse/db1.db/address_copy specified for non-external table:address_copy
res33: org.apache.spark.sql.DataFrame = []
```

CREATE TABLE … STORED AS PARQUET => OK

```
scala> spark.sql("CREATE TABLE db1.address_copy STORED AS PARQUET AS SELECT * FROM tmp_address LIMIT 100")
22/10/01 15:28:12 WARN HiveMetaStore: Location: file:/C:/apps/hadoop/spark-warehouse/db1.db/address_copy specified for non-external table:address_copy
res40: org.apache.spark.sql.DataFrame = []
```

Anyway… it is only to do next « show create table » ..
CTAS => can NOT create EXTERNAL tables / can NOT create PARTITIONED tables !!!

# Exercise 11 : SHOW CREATE TABLE

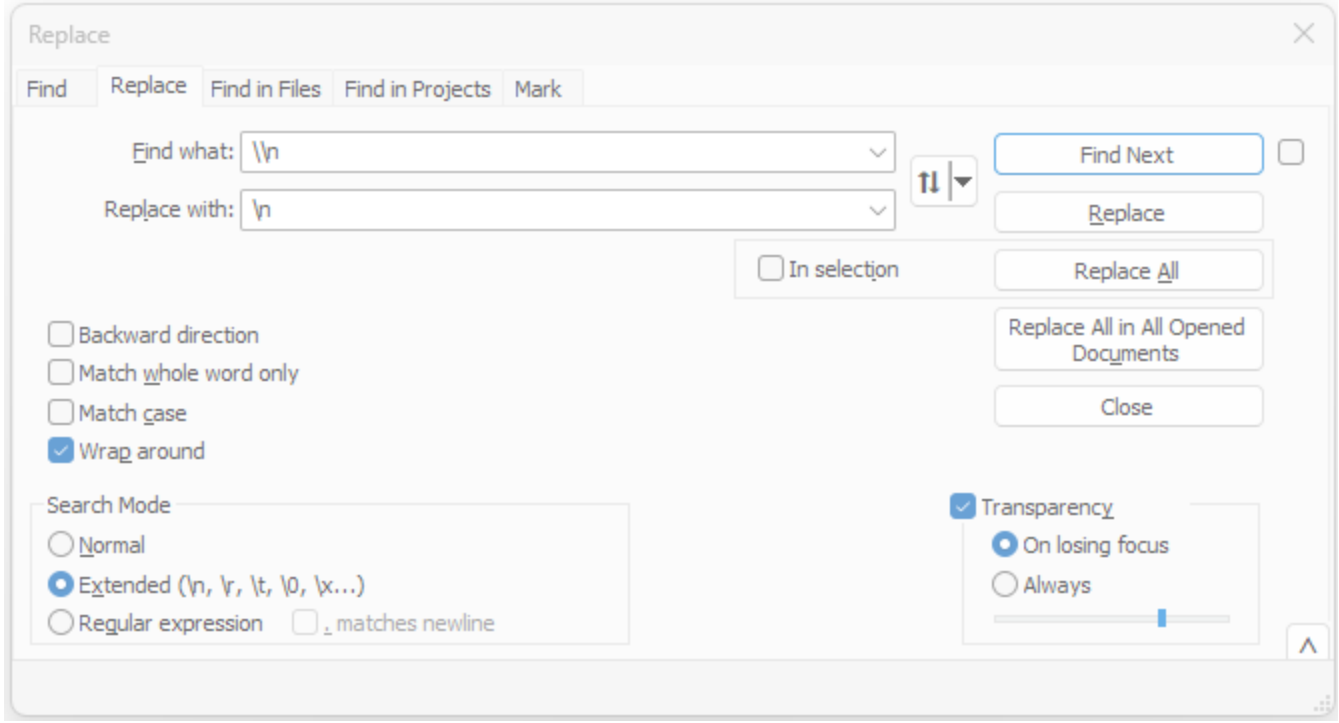
a/ Execute SQL

```
spark.sql("SHOW TABLES in db1").show(false)
spark.sql("DESCRIBE TABLE db1.addr").show(false)
spark.sql("SHOW CREATE TABLE db1.addr").show(false)
```

b/ check you can RE-Execute the result SQL DDL script
(ensure modify table name)

c/ compare difference with « dataset.printSchema » or Sql « DESCRIBE TABLE »
(ASCII table vs Sql)

# Hint Exercise 11: Replacing \\n by \n



# SHOW CREATE TABLE

```
CREATE TABLE spark_catalog.db1.addr (
uid_adresse STRING,
cle_interop STRING,  commune_insee STRING,
commune_nom STRING,
commune_deleguee_insee INT,
commune_deleguee_nom STRING,
voie_nom STRING,
lieudit_complement_nom STRING,
numero INT,  suffixe STRING,  position STRING,  x DOUBLE,  y DOUBLE,
long DOUBLE,  lat DOUBLE,  cad_parcelles STRING,  source STRING,
date_der_maj STRING,  certification_commune INT
)
USING parquet
```

# Exercise 12: Create « EXTERNAL » TABLE instead of default «MANAGED » Table

a/ Execute Sql « DROP TABLE db1.address »

b/ Verify that it has deleted all Directories and Files !

c/ recreate your table, but using « CREATE EXTERNAL TABLE .. LOCATION .. »

d/ fill your table using « INSERT OVERWRITE … SELECT .. »)

e/ Execute « DROP TABLE » on the external table
Verify that Dir/Files are NOT affected !

f/ RE-Create EXTERNAL TABLE … adding « LOCATION 'c:/data/your-dir' » (cf next slide)
Verify it still has data « select count(*) from .. »

# CREATE EXTERNAL TABLE .. STORED AS .. LOCATION ...

```
spark.sql("""
CREATE EXTERNAL TABLE spark_catalog.db1.addr2 (
uid_adresse STRING,
cle_interop STRING,  commune_insee STRING,
commune_nom STRING,  commune_deleguee_insee INT,  commune_deleguee_nom STRING,
voie_nom STRING,  lieudit_complement_nom STRING,  numero INT,  suffixe STRING,
position STRING,  x DOUBLE,  y DOUBLE,  long DOUBLE,  lat DOUBLE,  cad_parcelles STRING,
source STRING,  date_der_maj STRING,  certification_commune INT
)
STORED AS parquet
LOCATION 'C:/data/OpenData-gouv.fr/bal-table-addr2'
""").show()
```

# Exercise 13: INSERT INTO ..VALUES … result created File(s)

a/ Execute SQL

```
spark.sql("""
INSERT INTO db1.addr2
  (commune_insee,commune_nom,numero,voie_nom)
   VALUES ('75000', 'PARIS', 1, 'rue de la paix')
""").show()
```

check that it has created 1 new File (containing only 1 line)

b/ .. Repeat 10 times … “for(i <- 0 to 10)”,
check it has created 10 additional files

# Exercise 13(next): INSERT INTO SELECT.. … result created File(s)

c/ Execute SQL

```
spark.sql("""
INSERT INTO db1.addr2
   SELECT * FROM db1.addr
""").show()
```

Check how many files are inserted, comparing to the parallelism of the SELECT

```
spark.sql("SELECT * FROM db1.addr").toJavaRDD.getNumPartitions
```

d/ redo with

```
spark.sql("""
INSERT INTO db1.addr2 SELECT /*+REPARTITION(25)*/ * FROM db1.addr
""").show()
```

# Exercise 14: INSERT OVERWRITE ( difference with INSERT INTO )

a/ Question: if you reapeat several time the same « INSERT INTO »,
do you have duplicate rows ?
Can you have unicity with Primary Key constraint ?
can you do sql “UPDATE” ?

b/ Execute SQL, replacing « INTO » by « OVERWRITE »
INSERT **OVERWRITE** db1.address SELECT * FROM tmp_address

c/ what are the files remaining/created after ?

d/ if you repeat several times « INSERT OVERWRITE »,
what happens ?

# Objectives of Hands-On

1/ CreateTempView,  use Dataset from SQL

2/ Configure HiveMetastore DB

3/ connect Spark External Catalog  to MetaStore

4/ Create Database, Table

5/ (Directory) Partitioned Table

# Exercise 15: enrich Dataset / table address by computing column « dept » from column « commune_insee »

We want to partition by department : only 99 values … instead of by 50 000 cities

Use column « commune_insee »: it is a String of 5 chars,
padding with « 0 » on left,
and ignore the 3 chars on right (city code within department)

Create a column to extract the numeric value of dept
Recreate your table (or dataset) with this extra column

HINT: use « .withColumn(« dept », regexp_replace( col(srcCol), pattern, replacement) ) »

Example:
« 75100 » => « 75 » => 75
« 01200 » => « 01 » => « 1 » => 1

# Exercise 16 : Create PARTITIONED Table

a/ do « show create table »  on UN-partitioned table

b/ edit SQL + execute to change to a new table «  address_by_dept »,
partitioned by column « dept Int »

```
CREATE EXTERNAL TABLE db1.address_by_dept (
    … <all columns except dept column> … )
PARTITIONED BY ( dept Int)
STORED AS PARQUET
LOCATION ‘file:///c/data/db1/address_by_dept’
```

c/ execute

```
allAdressDept.write.format("parquet").saveAsTable("db1.tmp_addr_with_dept")
```

or..

```
INSERT OVERWRITE db1.address_by_dept  SELECT * FROM db1. tmp_addr_with_dept
```

# Exercise 17: Dirs for Partitioned Table

a/ explore directory / files for partitioned table

b/ check the partitioned column does not exist in the sub-files content, only in directory names

HINT: reload an individual parquet file using « spark.read.parquet(« … »)

10 mn pause

# Exercise 18: Query a Partitioned Table … optim « Partition Pruning »

a/ do a SQL query using a « WHERE dept=92 »

b/ do a SQL query using a « WHERE dept in (75, 78, 91, 92) »

c/ do a SQL query without partition column condition

d/ Can you measure that

Query a/ is ~4x faster than Query b/ (1 part << 4 parts)

Query a/ is ~99x faster than Query c/ (1 part << 99 parts)

e/ check that Spark prunes the unnecessary partitions dirs from scanning

# Exercise 19: SQL Explain plan: SQL « EXPLAIN select … from .. Where ..»

a/ Execute SQL prefixed by “EXPLAIN”

```
spark.sql("""
EXPLAIN select count(*) FROM db1.address_by_dept WHERE dept=92
""").foreach(println(_))
```

b/ Check that Spark has read ONLY files within directory « …/dept=92/*.parquet »

# Exercise 20: Query by condition, NOT by partition column

a/ Execute SQL with any condition but not on column “dept”
SELECT .. WHERE … somethingElse …

b/ Execute SQL:
EXPLAIN SELECT …

b/ Check that Spark has read ALL directories/files
… Check that is is slower than the unpartitioned table

# Exercise 21: INSERT OVERWRITE... partitioned

a/ Execute SQL
INSERT OVERWRITE address_by_dept
SELECT .. FROM .. WHERE … commune='Nanterre'

b/ check that ALL files from directory /dept=92/*
have been deleted… and rewritten with partial new results

c/ .. check that all cities != 'Nanterre' from dept=92 have been deleted
(not re-created).
an "INSERT OVERWRITE" can INSERT+UPDATE+DELETE rows !

d/ check that ALL other directories dept != 92 are unmodified

# Exercise 22: SAME insert overwrite … on 2 differently partitioned tables

Execute twice the same SQL, on 2 different tables, containing the same data
a/ compare select count(*) on both tables

b/ execute SQL on first (partitioned) table:
INSERT OVERWRITE address_by_dept
SELECT .. FROM .. WHERE … dept=92

c/ execute Same SQL on second (un-partitioned) table
INSERT OVERWRITE address
SELECT .. FROM .. WHERE … dept=92

d/ compare counts, Explain

# Exercise 23: INSERT OVERWRITE… for fully « UPDATING » partition(s)

a/ Execute SQL:
INSERT OVERWRITE address_by_dept
SELECT * FROM address  WHERE … dept=92

b/ check that counts are same:
  select count(*) from address where dept=92
  select count(*) from address_by_dept where dept=92

c/ Explain that you can replay safely an « UPDATE » batch for any dept
  .. NO duplicate, NO delete
  Update fully the expected result  but no side effect elsewhere

# Exercise 24: ACID Update / Delete in Spark ?

Questions

a/ Does Spark support SQL Update or Delete per rows?

b/ Can you « insert overwrite » data in Spark ? With which granularity ?

c/ can you emulate applicative updates or deletes on rows
by append-only events ?

d/ Do you know « DeltaLake » or « Iceberg » ? (google it)

# Exercise 25: Question on Partition.. What for ?

Questions

a/ Are partitions improving performances ?

b/ is it true that same SQL INSERT OVERWRITE
can have different results on tables with different partitionning ?

c/ Why/How should you use partitions ?

d/ Is is advisable to have deep sub-sub-sub partitioning ?

# Optionnal Exercise 26 – DeltaLake / Iceberg …

Have a look to DeltaLake

https://docs.delta.io/latest/quick-start.html

```
spark-shell --packages io.delta:delta-core_2.12:2.2.0 \
 --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension  \
 --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
```

CREATE TABLE … USING DELTA …  (instead of STORED AS PARQUET … )

# Exercise 27 : MindMap

Draw a MindMap to summarize
what you did and learn from this Hands-On session

Your MindMap should
start with word « Spark Catalog - Metastore» in the middle
Then draw star edges to other word chapters and sub-chapters

# Questions ?

# Take-Away

# What You learned ?

# Next Steps

More Lessons

More Hands-On

Spark concepts:
- Spark UI, DAG, Optimisation, Predicate-Push-Down
- Spark Clustering
- Java binding, UDF, map
- Spark Streaming
- …