

# BigData Spark – Hands-On

SQL, Temporary View  
Hive MetaStore, ExternalCatalog, Table  
Partitions

Arnaud Nauwynck

Nov 2022

# Objectives of Hands-On



Reminders:

Local Install, spark-shell

Spark File IO

1/ CreateTempView, use Dataset from SQL

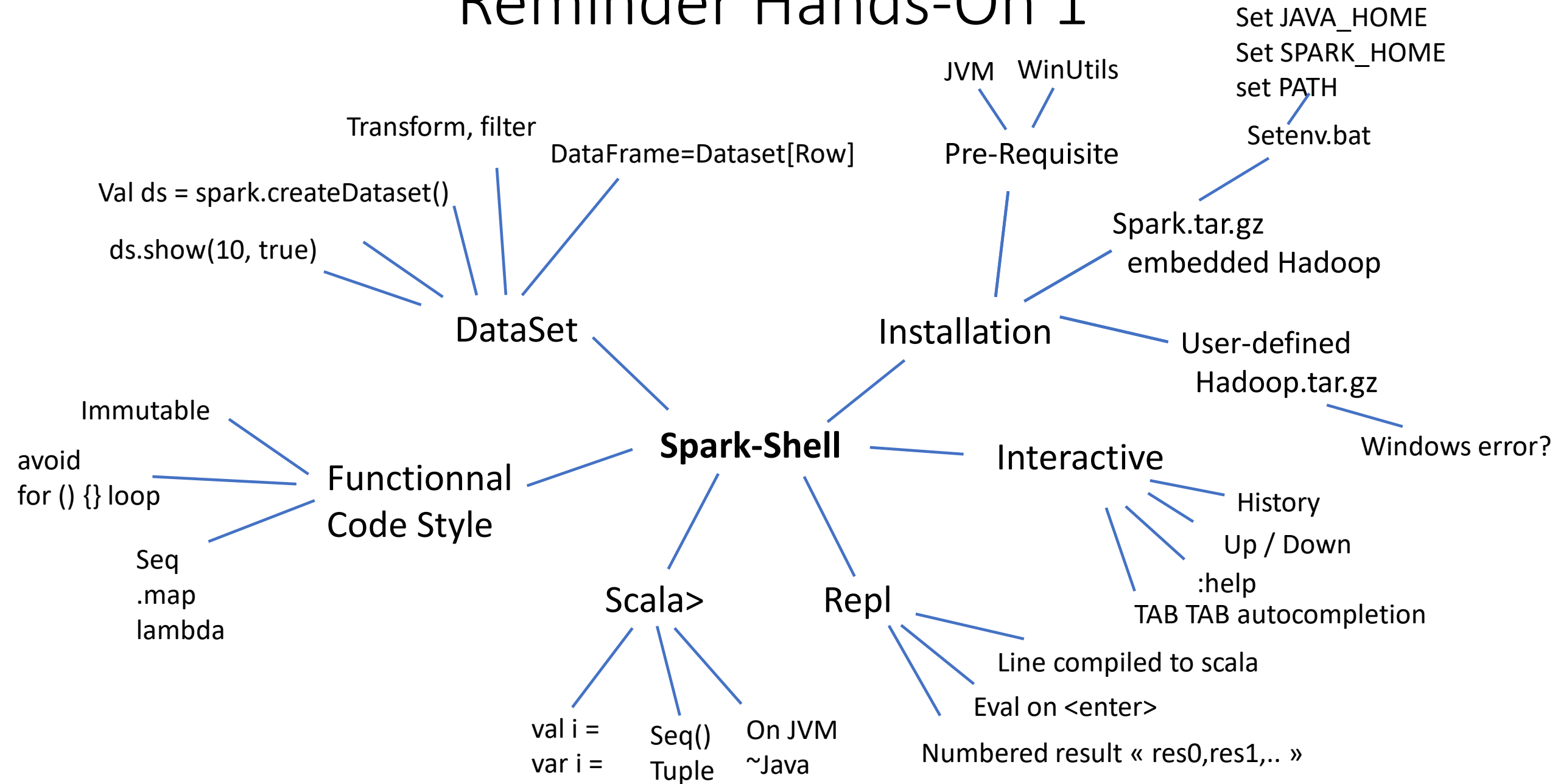
2/ Configure HiveMetastore DB

3/ connect Spark External Catalog to MetaStore

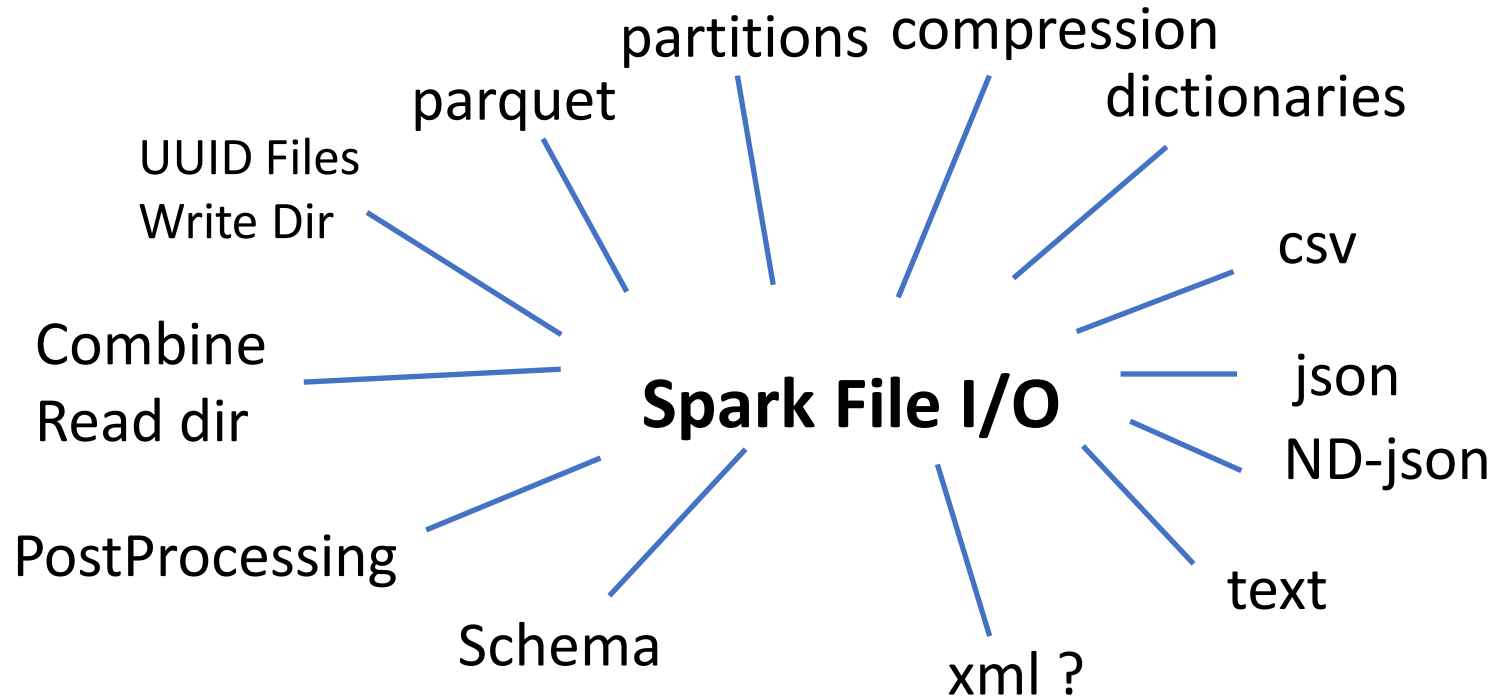
4/ Create Table

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

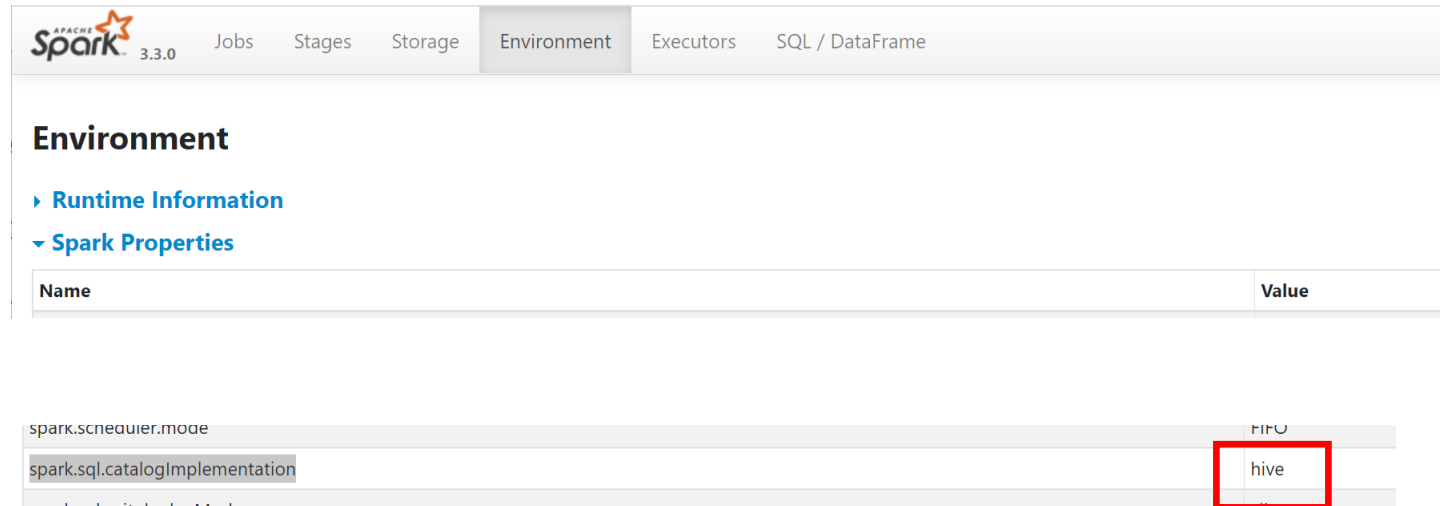
```
spark-shell --conf spark.sql.catalogImplementation=in-memory
```

# 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 »



APACHE Spark 3.3.0		Jobs	Stages	Storage	Environment	Executors	SQL / DataFrame
<b>Environment</b>							
▶ Runtime Information							
▼ Spark Properties							
Name						Value	
spark.scheduler.mode						FIFO	
spark.sql.catalogImplementation						hive	
spark.sql.catalog.spark_catalog						hive	

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.apache.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)
```

# Ensure Memory JVM Argument -Xmx3g

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



Command Line:

```
C:\apps\jdk\jdk-8\bin\java -cp "C:\apps\hadoop\spark-3.2.0-hadoop-3.3\bin\..\conf\;C:\apps\hadoop\spark-3.3.0-hadoop-3.3\jars\*" "-Dscala.usejavacp=true" -Xmx3g -XX:+IgnoreUnrecognizedVMOptions "--add-opens=java.base/java.lang=ALL-UNNAMED" "--add-opens=java.base/java.lang.invoke=ALL-UNNAMED" "--add-opens=java.base/java.lang.reflect=ALL-UNNAMED" "--add-opens=java.base/java.io=ALL-UNNAMED" "--add-opens=java.base/java.net=ALL-UNNAMED" "--add-opens=java.base/java.nio=ALL-UNNAMED" "--add-opens=java.base/java.util=ALL-UNNAMED" "--add-opens=java.base/java.util.concurrent=ALL-UNNAMED" "--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED" "--add-opens=java.base/sun.nio.ch=ALL-UNNAMED" "--add-opens=java.base/sun.nio.cs=ALL-UNNAMED" "--add-opens=java.base/sun.security.action=ALL-UNNAMED" "--add-opens=java.base/sun.util.calendar=ALL-UNNAMED" "--add-opens=java.security.jgss/sun.security.krb5=ALL-UNNAMED" org.apache.spark.deploy.SparkSubmit --conf "spark.driver.memory=3g" --conf "spark.sql.catalogImplementation=in-memory" --class org.apache.spark.repl.Main --name "Spark shell" spark-shell
```

Path:

C:\apps\jdk\jdk-8\bin\java.exe

# 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 disappeared !

d/ recreate it

# 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



# Install Hive Standalone Metastore (Not Hive)








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

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

## Index of /hive

<u>Name</u>	<u>Last modified</u>
 <a href="#">Parent Directory</a>	
 <a href="#">hive-1.2.2/</a>	2022-06-17 12:34
 <a href="#">hive-2.3.9/</a>	2022-06-17 12:34
 <a href="#">hive-3.1.2/</a>	2022-06-17 12:34
 <a href="#">hive-3.1.3/</a>	2022-06-17 12:34
 <a href="#">hive-4.0.0-alpha-1/</a>	2022-06-17 12:34
 <a href="#">hive-standalone-metastore-3.0.0/</a>	2022-06-17 12:34
 <a href="#">hive-storage-2.7.3/</a>	2022-06-17 12:34
 <a href="#">hive-storage-2.8.1/</a>	2022-06-17 12:34
 <a href="#">stable-2/</a>	2022-06-17 12:34
 <a href="#">KEYS</a>	2022-03-23 18:19










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

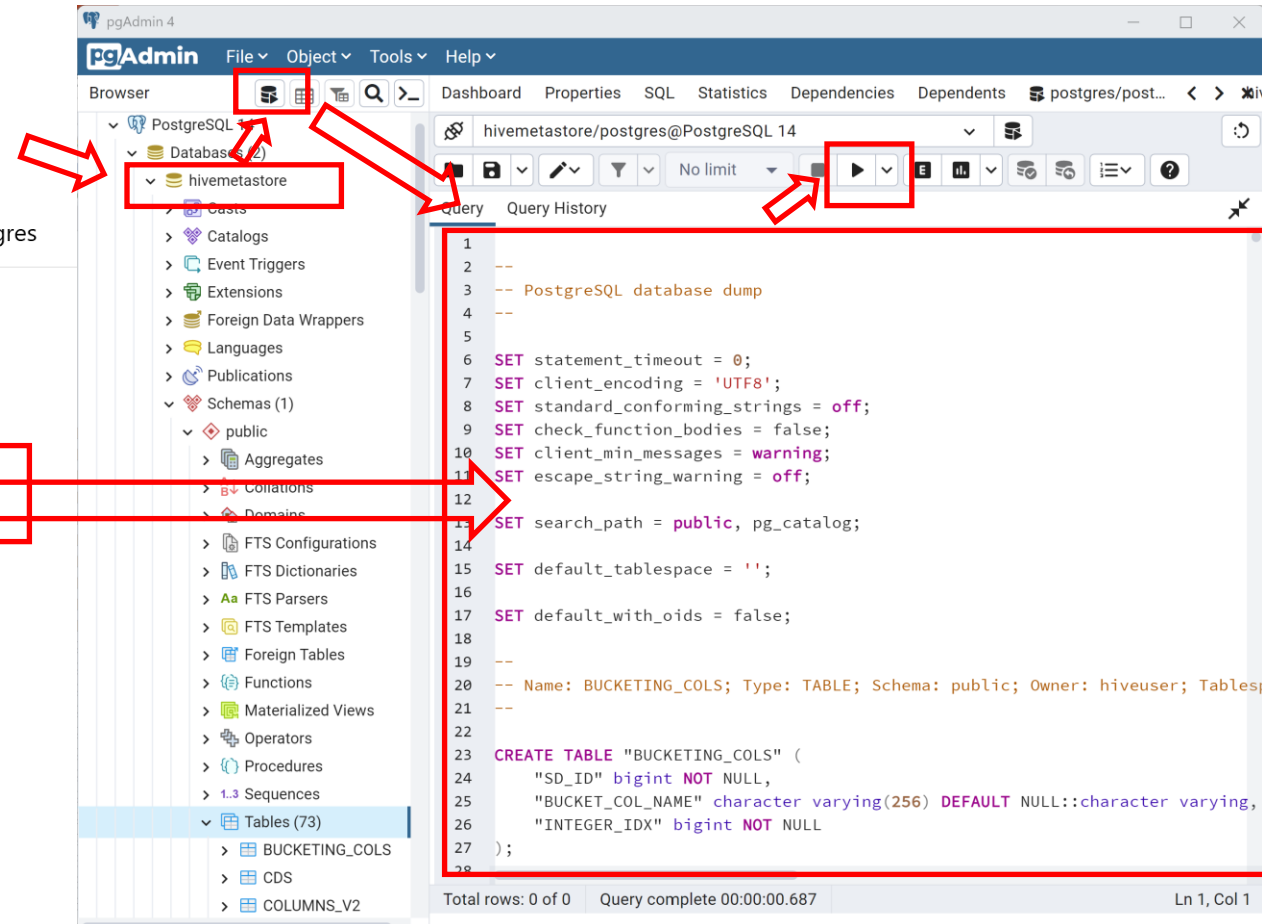
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">hive-standalone-metastore-3.0.0-bin.tar.gz</a>	2018-06-07 17:56	25M	
 <a href="#">hive-standalone-metastore-3.0.0-bin.tar.gz.asc</a>	2018-06-07 17:56	832	
 <a href="#">hive-standalone-metastore-3.0.0-bin.tar.gz.sha256</a>	2018-06-07 17:56	109	
 <a href="#">hive-standalone-metastore-3.0.0-src.tar.gz</a>	2018-06-07 17:56	2.3M	
 <a href="#">hive-standalone-metastore-3.0.0-src.tar.gz.asc</a>	2018-06-07 17:56	832	
 <a href="#">hive-standalone-metastore-3.0.0-src.tar.gz.sha256</a>	2018-06-07 17:56	109	

# 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>

SSD (C:) > apps > hadoop > hive-metastore > hive-metastore-3.0.0 > scripts > metastore > upgrade > postgres

<input type="checkbox"/> Nom	Modifié le	Type	Taille
 create-user.postgres.sql	15/05/2018 23:42	Fichier SQL	1 Ko
 hive-schema-1.2.0.postgres.sql	15/05/2018 23:42	Fichier SQL	43 Ko
<input checked="" type="checkbox"/>  hive-schema-3.0.0.postgres.sql	15/05/2018 23:42	Fichier SQL	52 Ko
 upgrade.order.postgres	15/05/2018 23:42	Fichier POSTGRES	1 Ko
 upgrade-1.2.0-to-2.0.0.postgres.sql	15/05/2018 23:42	Fichier SQL	3 Ko
 upgrade-2.0.0-to-2.1.0.postgres.sql	15/05/2018 23:42	Fichier SQL	2 Ko
 upgrade-2.1.0-to-2.2.0.postgres.sql	15/05/2018 23:42	Fichier SQL	3 Ko
 upgrade-2.2.0-to-2.3.0.postgres.sql	15/05/2018 23:42	Fichier SQL	1 Ko
 upgrade-2.3.0-to-3.0.0.postgres.sql	15/05/2018 23:42	Fichier SQL	13 Ko



pgAdmin 4

Browser: PostgreSQL

- Databases (2)
- hivemetastore
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences
      - Tables (73)
        - BUCKETING\_COLS
        - CDS
        - COLUMNS\_V2

Query: hivemetastore/postgres@PostgreSQL 14

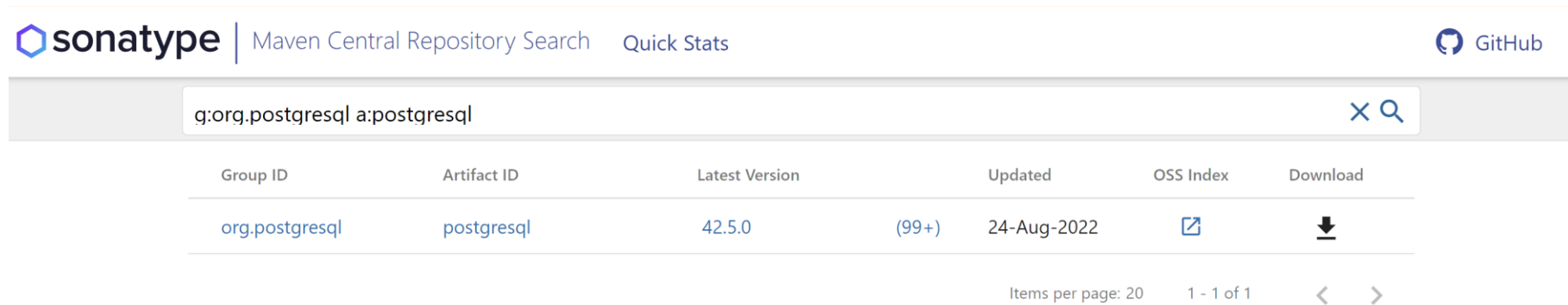
```
1 --
2 --
3 -- PostgreSQL database dump
4 --
5
6 SET statement_timeout = 0;
7 SET client_encoding = 'UTF8';
8 SET standard_conforming_strings = off;
9 SET check_function_bodies = false;
10 SET client_min_messages = warning;
11 SET escape_string_warning = off;
12
13 SET search_path = public, pg_catalog;
14
15 SET default_tablespace = '';
16
17 SET default_with_oids = false;
18
19 --
20 -- Name: BUCKETING_COLS; Type: TABLE; Schema: public; Owner: hiveuser; Tablespace:
21 --
22
23 CREATE TABLE "BUCKETING_COLS" (
24     "SD_ID" bigint NOT NULL,
25     "BUCKET_COL_NAME" character varying(256) DEFAULT NULL::character varying,
26     "INTEGER_IDX" bigint NOT NULL
27 );
28
```

Total rows: 0 of 0 Query complete 00:00:00.687 Ln 1, Col 1

# Copy Jar .. Example using Postgresql

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

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



The screenshot shows the Sonatype Maven Central Repository Search interface. The search bar contains the query 'g:org.postgresql a:postgresql'. The results table shows one entry for 'org.postgresql:postgresql' with version '42.5.0', updated on '24-Aug-2022'. The table has columns: Group ID, Artifact ID, Latest Version, Updated, OSS Index, and Download. The download link is represented by a download icon.

Group ID	Artifact ID	Latest Version	Updated	OSS Index	Download
org.postgresql	postgresql	42.5.0	24-Aug-2022	<a href="#">[Link]</a>	<a href="#">[Download]</a>

Items per page: 20    1 - 1 of 1    < >

Step 2/ copy jar to SPARK\_HOME/jars

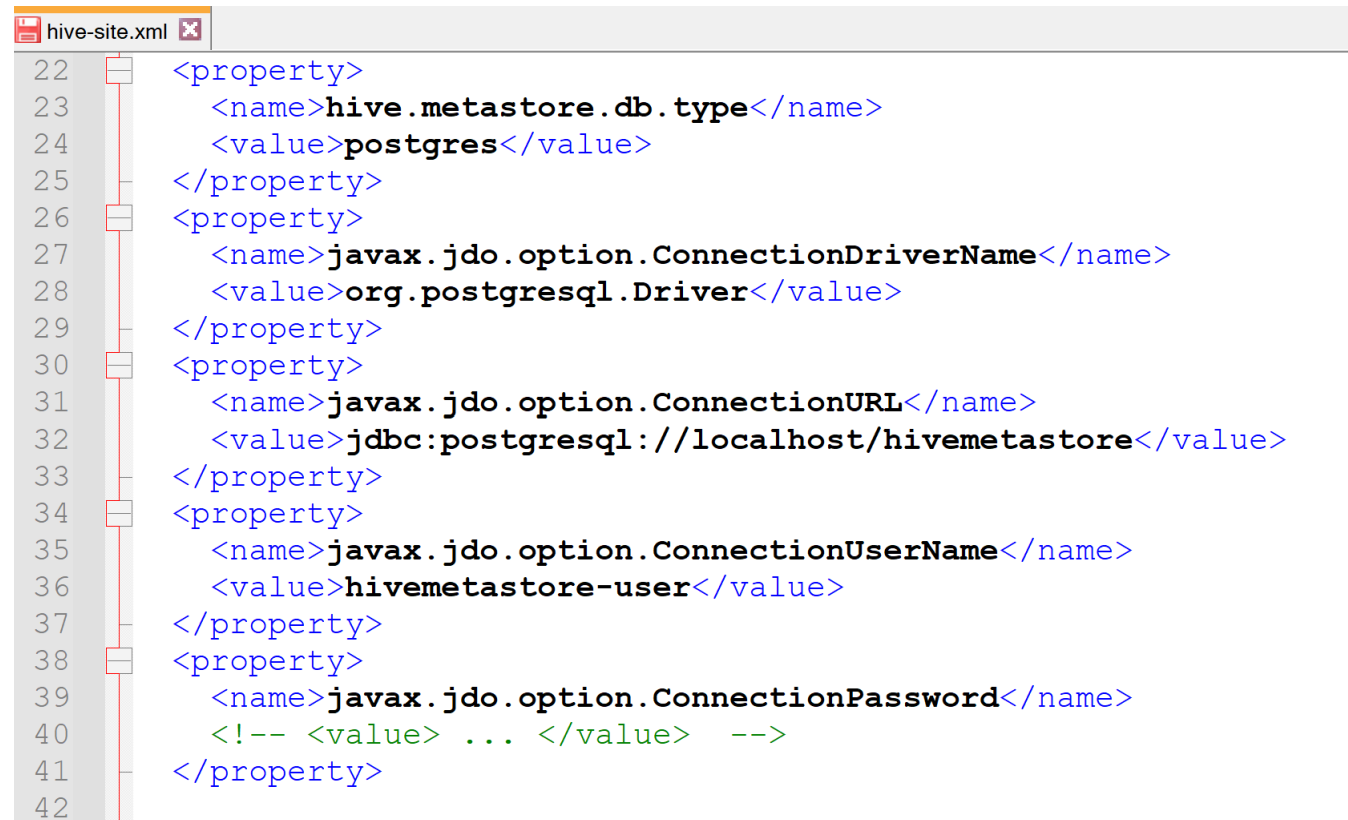
Windows-SSD (C:) > apps > hadoop > spark-3.3.0 > jars



The screenshot shows a Windows File Explorer window displaying the contents of the 'jars' directory. The table lists files with their names, modification dates, types, and sizes. The file 'postgresql-42.4.1.jar' is selected.

Nom	Modifié le	Type	Taille
parquet-jackson-1.12.2.jar	09/06/2022 21:22	Fichier JAR	1 000 Ko
pickle-1.2.jar	09/06/2022 21:22	Fichier JAR	54 Ko
postgresql-42.4.1.jar	15/09/2022 20:38	Fichier JAR	1 021 Ko

# 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

a/ Check env variables

HADOOP\_HOME=..

HADOOP\_CONF\_DIR=.. (default to HADOOP\_HOME if not set )

b/ Check conf/hive-site.xml

c/ relaunch spark-shell

**spark-shell --conf spark.sql.catalogImplementation=hive # not in-memory any more**

d/ Check in sparkContext







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

**res1: String = "hive"**

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

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

-SSD (C:) > apps > hadoop > spark-3.3.0 > conf

<input type="checkbox"/> Nom	Modifié le	Type	Taille
 fairscheduler.xml.template	09/06/2022 21:22	Fichier TEMPLATE	2 Ko
 log4j2.properties.template	09/06/2022 21:22	Fichier TEMPLATE	4 Ko
 metrics.properties.template	09/06/2022 21:22	Fichier TEMPLATE	9 Ko
<input checked="" type="checkbox"/>  spark-defaults.conf.template	09/06/2022 21:22	Fichier TEMPLATE	2 Ko
 spark-env.sh.template	09/06/2022 21:22	Fichier TEMPLATE	5 Ko
 workers.template	09/06/2022 21:22	Fichier TEMPLATE	1 Ko



a/ Copy file, and rename without suffix « .template »  
Edit to change default « key value »

b/ Check conf can still be overloadded by « --conf key=value »

c/ check conf at runtime  
`spark.sharedState.sparkContext.getConf.get(« key »)`

d/ check 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:

```
dataset.saveAsTable(« tableName »)
```

```
dataset.saveAsTable(« dbName.tableName »)
```

```
dataset.format(« parquet »).saveAsTable(« dbName.tableName »)
```

# 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.createHiveTableByDefault 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 « SHOW CREATE TABLE db1.address »
- 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)

# 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

Verify it still has data « select count(\*) from .. »

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

a/ Execute SQL

**INSERT INTO** db1.address (col1, col2... colN) **VALUES** (... ..)

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

c/ .. Repeat 100 times ... check it has created 100 files!!

c/ **INSERT INTO** db1.address **SELECT** /\* +REPARTITION(25) \*/ \* FROM tmp\_address

d/ check it has created N new File(s), where N is the parallelism of the SELECT



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

a/ Question: if you repeat several times the same « INSERT INTO », do you have duplicate rows ?

Can you have unicity with Primary Key constraint ?

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

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

# 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”

**EXPLAIN** select count(\*) FROM db1.address\_by\_dept WHERE dept=92

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 it 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
- ...