ASSIGNMENT SOLUTION

The following Common Boilerplate code to create a Spark Session has to be executed before running the queries.

```
from pyspark.sql import SparkSession

import getpass

username = getpass.getuser()

spark= SparkSession. \

builder. \

config('spark.ui.port','0'). \

config("spark.sql.warehouse.dir", f"/user/{username}/warehouse"). \

enableHiveSupport(). \

master('yarn'). \

getOrCreate()
```

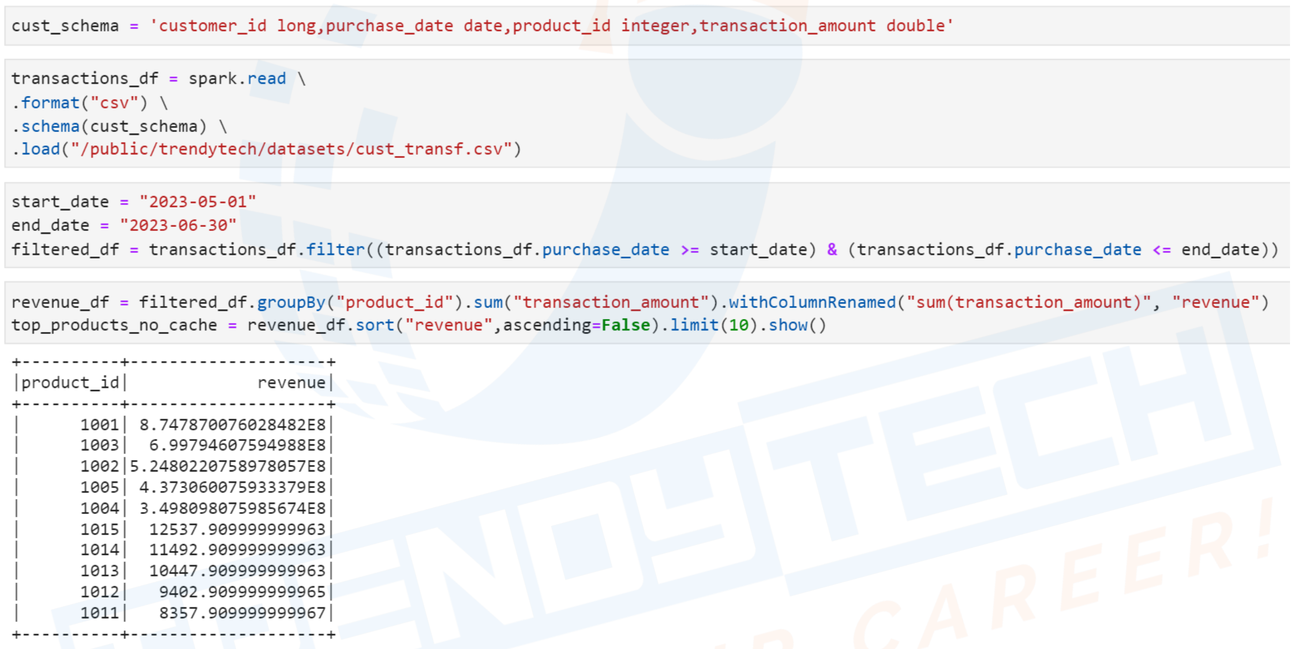
Note : Use Pyspark2 for executing the below queries.

Question 1

A.1

```
cust_schema = 'customer_id long,purchase_date date,product_id
integer,transaction_amount double'


transactions_df = spark.read \

.format("csv") \

.schema(cust_schema) \

.load("/public/trendytech/datasets/cust_transf.csv")


start_date = "2023-05-01"
```

```
end_date = "2023-06-30"

filtered_df = transactions_df.filter((transactions_df.purchase_date >=
start_date) & (transactions_df.purchase_date <= end_date))


revenue_df =
filtered_df.groupBy("product_id").sum("transaction_amount").withColumnRena
med("sum(transaction_amount)", "revenue")

top_products_no_cache =
revenue_df.sort("revenue",ascending=False).limit(10).show()
```



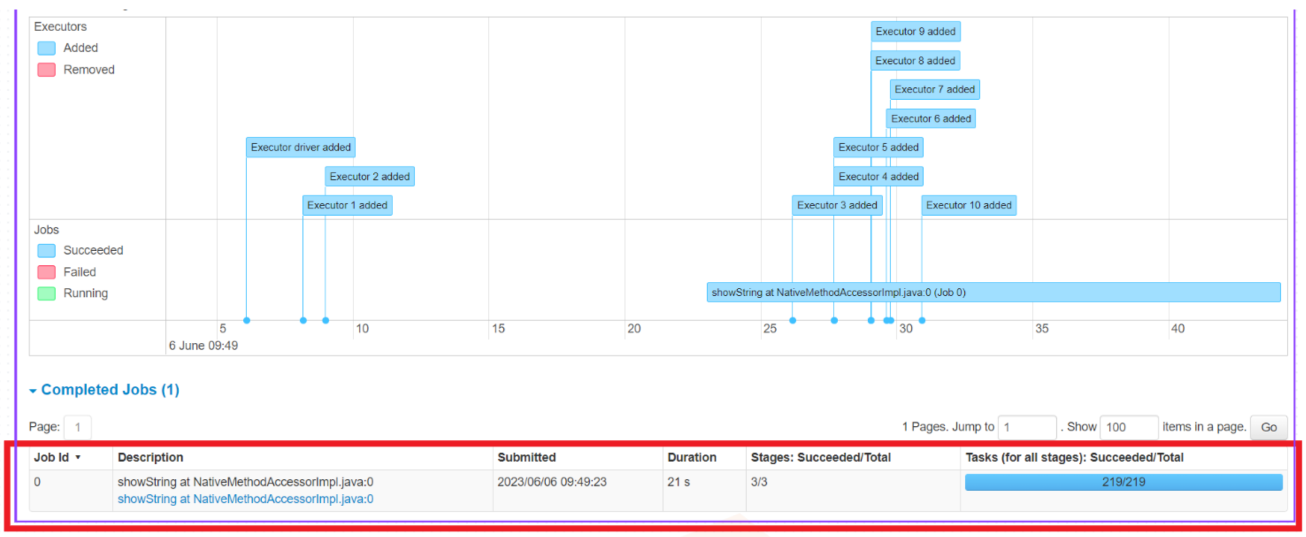
```
cust_schema = 'customer_id long,purchase_date date,product_id integer,transaction_amount double'
```

```
transactions_df = spark.read \
.format("csv") \
.schema(cust_schema) \
.load("/public/trendytech/datasets/cust_transf.csv")
```

```
start_date = "2023-05-01"
end_date = "2023-06-30"
filtered_df = transactions_df.filter((transactions_df.purchase_date >= start_date) & (transactions_df.purchase_date <= end_date))
```

```
revenue_df = filtered_df.groupBy("product_id").sum("transaction_amount").withColumnRenamed("sum(transaction_amount)", "revenue")
top_products_no_cache = revenue_df.sort("revenue",ascending=False).limit(10).show()
```

```
+----------+--------------------+
|product_id|             revenue|
+----------+--------------------+
|      1001| 8.747870076028482E8|
|      1003|  6.99794607594988E8|
|      1002|5.2480220758978057E8|
|      1005| 4.373060075933379E8|
|      1004| 3.498098075985674E8|
|      1015|  12537.909999999963|
|      1014|  11492.909999999963|
|      1013|  10447.909999999963|
|      1012|   9402.909999999965|
|      1011|   8357.909999999967|
+----------+--------------------+
```



▾ Completed Jobs (1)

Page: 1 · 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 09:49:23	21 s	3/3	219/219

#with caching

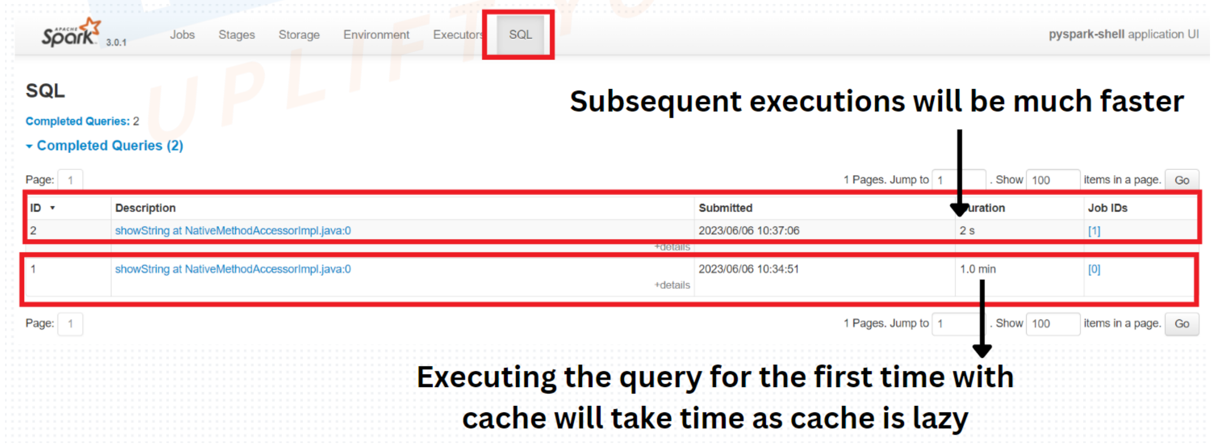
start_date = "2023-05-01"

end_date = "2023-06-30"

cached_filtered_df = transactions_df.filter((transactions_df.purchase_date >= start_date) & (transactions_df.purchase_date <= end_date)).cache()

revenue_df_with_cache = cached_filtered_df.groupBy("product_id").sum("transaction_amount").withColumnRenamed("sum(transaction_amount)", "revenue")

top_products_with_cache = revenue_df_with_cache.orderBy("revenue", ascending=False).limit(10).show()



SQL

Completed Queries: 2

▾ Completed Queries (2)

Subsequent executions will be much faster

Page: 1 · 1 Pages. Jump to 1 . Show 100 items in a page. Go

ID ▾	Description	Submitted	Duration	Job IDs
2	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/06 10:37:06	2 s	[1]
1	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/06 10:34:51	1.0 min	[0]

Page: 1 · 1 Pages. Jump to 1 . Show 100 items in a page. Go

Executing the query for the first time with cache will take time as cache is lazy

A.2

```
customer_transactions =
filtered_df.groupBy("customer_id").sum("transaction_amount").withColumnRe
named("sum(transaction_amount)", "cust_amount")


customer_transactions.show()


top_customers = customer_transactions.sort("cust_amount",
ascending=False)


top_10_customers = top_customers.limit(10).show()
```

A.3

```
spark.sql("create database tt_cust_transaction")
```

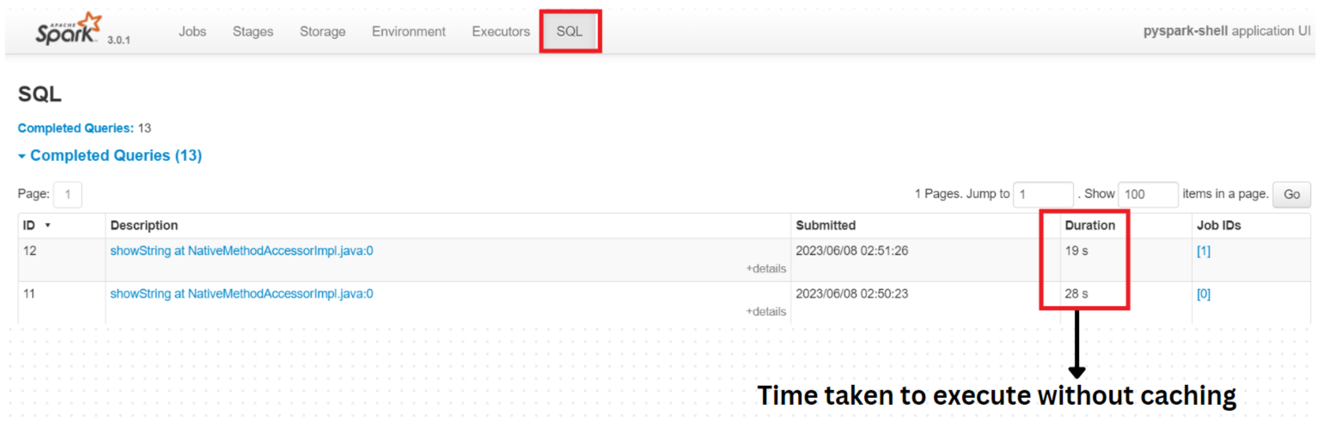
#before caching

```
spark.sql("create table
tt_cust_transaction.customer_transactions_ext(customer_id
long,purchase_date date,product_id integer,transaction_amount double)
USING csv location '/public/trendytech/datasets/cust_transf.csv'")


spark.sql("SELECT product_id, SUM(transaction_amount) AS revenue FROM
tt_cust_transaction.customer_transactions_ext WHERE purchase_date >=
'2023-05-01' AND purchase_date <= '2023-06-30' GROUP BY product_id
ORDER BY revenue DESC LIMIT 10").show()
```

```
+----------+--------------------+
|product_id|             revenue|
+----------+--------------------+
|      1001| 8.747870076028483E8|
|      1003| 6.997946075949881E8|
|      1002|5.2480220758978045E8|
|      1005| 4.373060075933379E8|
|      1004| 3.498098075985674E8|
|      1015|  12537.909999999963|
|      1014|  11492.909999999963|
|      1013|  10447.909999999963|
|      1012|   9402.909999999965|
|      1011|   8357.909999999967|
+----------+--------------------+
```

```
spark.sql("SELECT customer_id, SUM(transaction_amount) AS cust_amount
FROM tt_cust_transaction.customer_transactions_ext WHERE
purchase_date >= '2023-05-01' AND purchase_date <= '2023-06-30' GROUP
BY customer_id ORDER BY cust_amount DESC LIMIT 10").show()
```

```
+-----------+--------------------+
|customer_id|         cust_amount|
+-----------+--------------------+
|       1001| 3.180884580005336E8|
|       1004| 3.101342580008687E8|
|       1005|2.6240905800151232E8|
|       1003|2.1468385800145328E8|
|       1002|2.0672965800144082E8|
|       1011|1.9086143271084768E8|
|       1006|1.9085620771084768E8|
|       1015|1.6700301271081635E8|
|       1010|1.6699778771081635E8|
|       1014|     1.510935677079E8|
+-----------+--------------------+
```



Spark 3.0.1 — Jobs | Stages | Storage | Environment | Executors | SQL — pyspark-shell application UI

SQL

Completed Queries: 13

Completed Queries (13)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
12	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 02:51:26	19 s	[1]
11	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 02:50:23	28 s	[0]

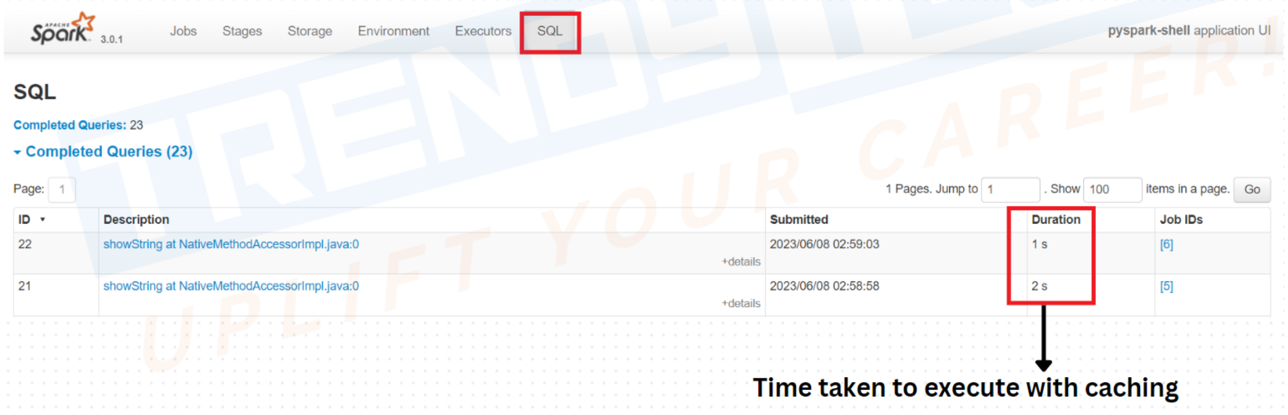
Time taken to execute without caching

#after caching

spark.sql("cache table tt_cust_transaction.customer_transactions_ext")

spark.sql("SELECT product_id, SUM(transaction_amount) AS revenue FROM tt_cust_transaction.customer_transactions_ext WHERE purchase_date >= '2023-05-01' AND purchase_date <= '2023-06-30' GROUP BY product_id ORDER BY revenue DESC LIMIT 10").show()

spark.sql("SELECT customer_id, SUM(transaction_amount) AS cust_amount FROM tt_cust_transaction.customer_transactions_ext WHERE purchase_date >= '2023-05-01' AND purchase_date <= '2023-06-30' GROUP BY customer_id ORDER BY cust_amount DESC LIMIT 10").show()



Spark 3.0.1 — Jobs | Stages | Storage | Environment | Executors | SQL — pyspark-shell application UI

SQL

Completed Queries: 23

Completed Queries (23)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
22	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 02:59:03	1 s	[6]
21	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 02:58:58	2 s	[5]

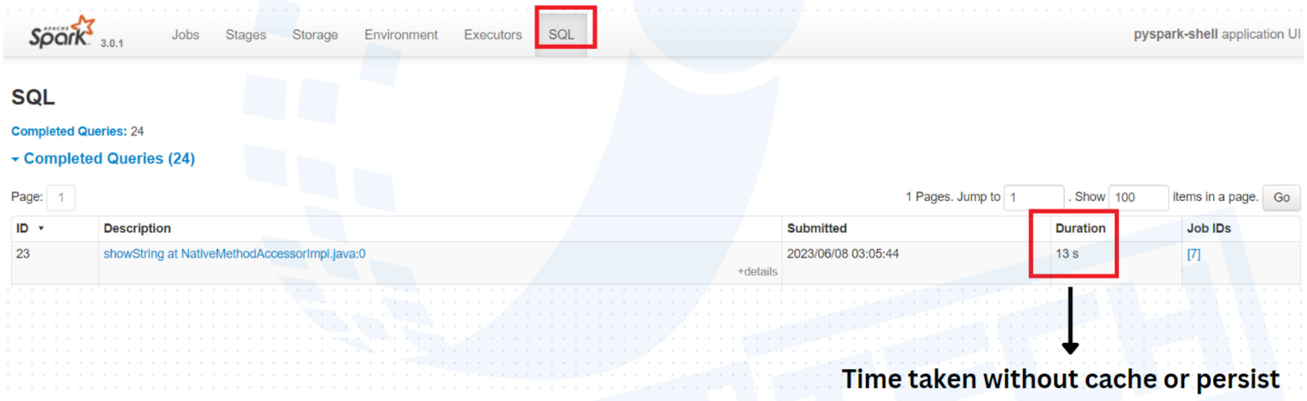
Time taken to execute with caching

A.4

```
from pyspark.sql.functions import year, month

from pyspark.sql.functions import countDistinct


cust_schema = 'customer_id long,purchase_date date,product_id
integer,transaction_amount double'


transactions_df = spark.read \

.format("csv") \

.schema(cust_schema) \

.load("/public/trendytech/datasets/cust_transf.csv")


new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))


customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs"))


regular_customers = customer_month_counts.filter("distinct_months = 1") \

    .groupBy("customer_id").count() \

    .orderBy("count", ascending=False).limit(10).show()
```

```
regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```

```
+-----------+-----+
|customer_id|count|
+-----------+-----+
|       1009|    2|
|       1012|    2|
|       1001|    2|
|       1011|    2|
|       1007|    2|
|       1005|    2|
|       1010|    2|
|       1002|    2|
|       1006|    2|
|       1013|    2|
+-----------+-----+
```



SQL

Completed Queries: 24

▾ Completed Queries (24)

Page: 1   1 Pages. Jump to 1 . Show 100 items in a page. Go

ID ▾	Description	Submitted	Duration	Job IDs
23	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 03:05:44	13 s	[7]

Time taken without cache or persist

#using persist

customer_month_counts = new_df.groupBy("customer_id", "purchase_year", "purchase_month").agg(countDistinct("purchase_month").alias("distinct_months")).persist()

regular_customers = customer_month_counts.filter("distinct_months = 1") \

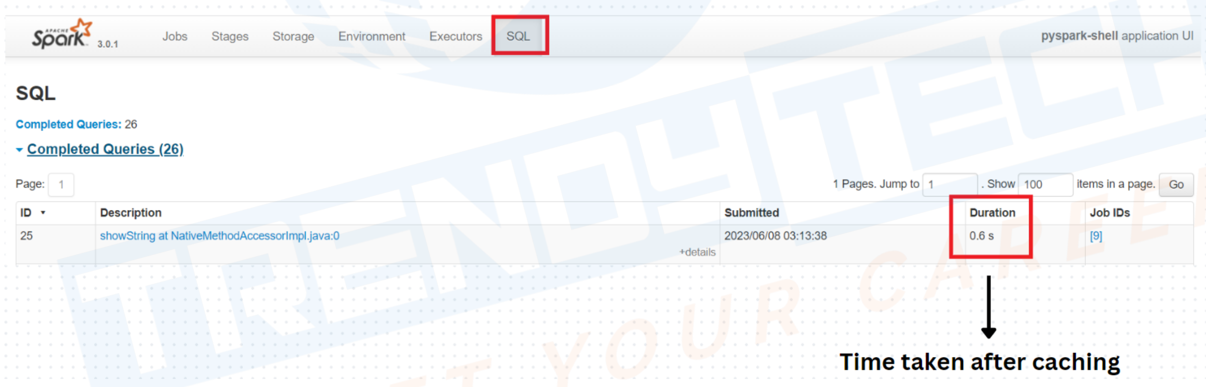
.groupBy("customer_id").count() \

.orderBy("count", ascending=False).limit(10).show()

A.5

#using cache

```
from pyspark.sql.functions import year, month

new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))

from pyspark.sql.functions import countDistinct


customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).cache()


regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```



SQL

Completed Queries: 26

Completed Queries (26)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
25	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 03:13:38	0.6 s	[9]

Time taken after caching

#using persist

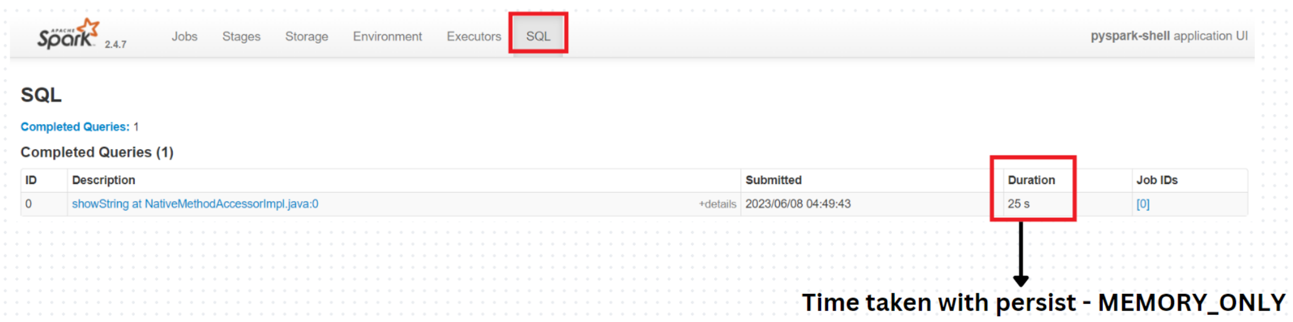
```
from pyspark.sql.functions import year, month

from pyspark.sql.functions import countDistinct

from pyspark.storagelevel import StorageLevel

new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))
```

```
customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.MEMORY_AND_DISK_SER)


regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```

A.6

#MEMORY ONLY

```
from pyspark.sql.functions import year, month
from pyspark.sql.functions import countDistinct
from pyspark.storagelevel import StorageLevel


new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))


customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.MEMORY_ONLY)


regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```



SQL

Completed Queries: 1

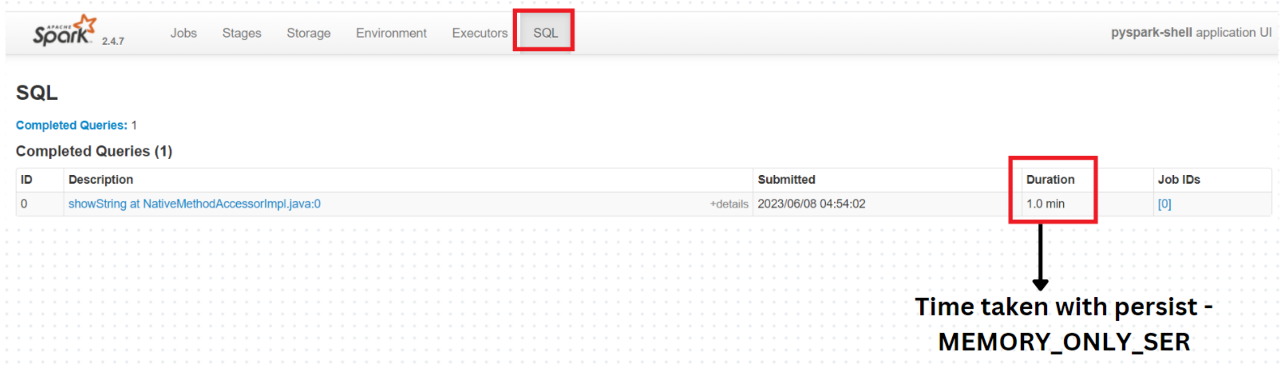
Completed Queries (1)

ID	Description	Submitted	Duration	Job IDs
0	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 04:49:43	25 s	[0]

Time taken with persist - MEMORY_ONLY

# MEMORY_ONLY_SER

```
from pyspark.sql.functions import year, month

from pyspark.sql.functions import countDistinct

from pyspark.storagelevel import StorageLevel


new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))


customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.MEMORY_ONLY_SER)


regular_customers = customer_month_counts.filter("distinct_months = 1") \

    .groupBy("customer_id").count() \

    .orderBy("count", ascending=False).limit(10).show()
```



SQL

Completed Queries: 1

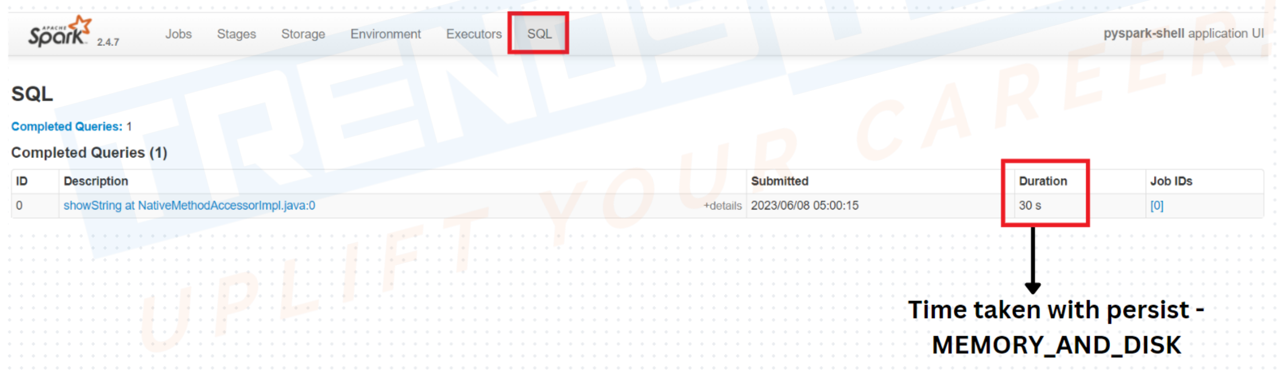
Completed Queries (1)

ID	Description	Submitted	Duration	Job IDs
0	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 04:54:02	1.0 min	[0]

Time taken with persist - MEMORY_ONLY_SER

# MEMORY_AND_DISK

```
from pyspark.sql.functions import year, month

from pyspark.sql.functions import countDistinct

from pyspark.storagelevel import StorageLevel


new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))


customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.MEMORY_AND_DISK)


regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```


Spark 2.4.7
Jobs
Stages
Storage
Environment
Executors
SQL
pyspark-shell application UI
Time taken with persist - MEMORY_AND_DISK

SQL

Completed Queries: 1

Completed Queries (1)

ID	Description	Submitted	Duration	Job IDs
0	showString at NativeMethodAccessorImpl.java:0 +details	2023/06/08 05:00:15	30 s	[0]

# MEMORY_AND_DISK_SER

```
from pyspark.sql.functions import year, month

from pyspark.sql.functions import countDistinct
from pyspark.storagelevel import StorageLevel

new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))

customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.MEMORY_AND_DISK_SER)

regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```

# DISK_ONLY

```
from pyspark.sql.functions import year, month
from pyspark.sql.functions import countDistinct
from pyspark.storagelevel import StorageLevel

new_df = transactions_df.withColumn("purchase_year",
year("purchase_date")).withColumn("purchase_month",
month("purchase_date"))

customer_month_counts = new_df.groupBy("customer_id", "purchase_year",
"purchase_month").agg(countDistinct("purchase_month").alias("distinct_mont
hs")).persist(StorageLevel.DISK_ONLY)

regular_customers = customer_month_counts.filter("distinct_months = 1") \
    .groupBy("customer_id").count() \
    .orderBy("count", ascending=False).limit(10).show()
```

B)

#user defined function

```
def get_customer_history(customer_id):
    customer_history_df = transactions_df.filter(transactions_df.customer_id ==
customer_id).cache()
    return customer_history_df
```

#pass the customer_id you want to get the history

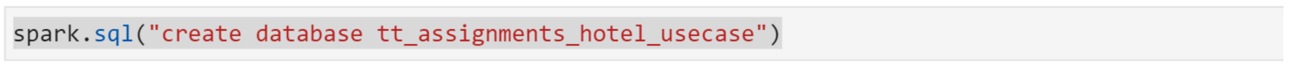
```
customer_id = 1001
customer_history_df = get_customer_history(customer_id)
customer_history_df.show()
```

C)

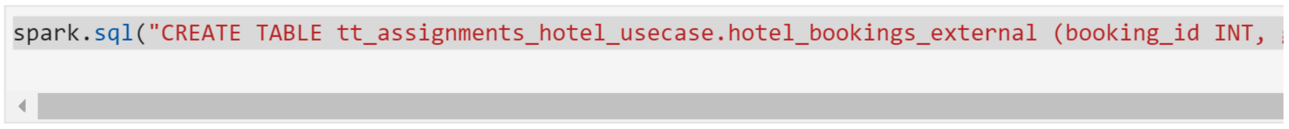
```
cached_filtered_df.unpersist()
spark.sql(“uncache table tt_cust_transaction.customer_transactions_ext”)
```

Question 2

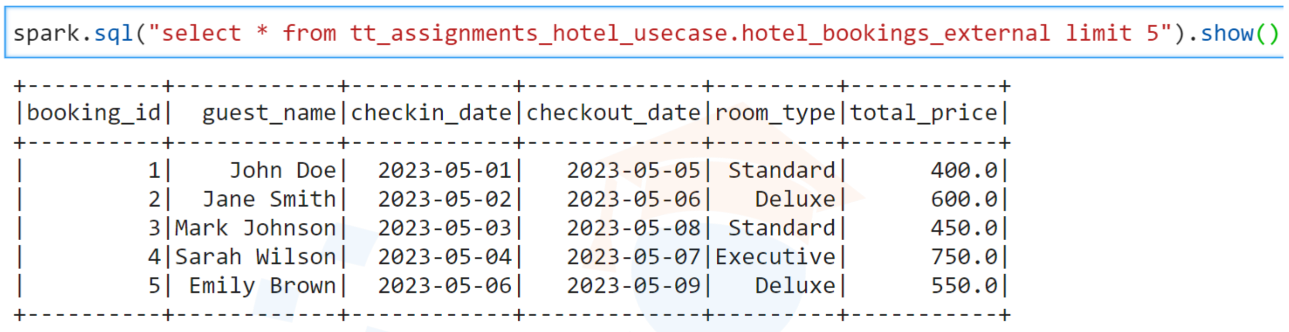
```
spark.sql("create database tt_assignments_hotel_usecase")

spark.sql("CREATE TABLE
tt_assignments_hotel_usecase.hotel_bookings_external (booking_id INT,
guest_name STRING, checkin_date DATE, checkout_date DATE, room_type
STRING, total_price DOUBLE) USING csv location
'/public/trendytech/datasets/hotel_data.csv' ")

spark.sql("select * from
tt_assignments_hotel_usecase.hotel_bookings_external limit 5").show()
```



```
spark.sql("create database tt_assignments_hotel_usecase")
```



```
spark.sql("CREATE TABLE tt_assignments_hotel_usecase.hotel_bookings_external (booking_id INT,
```

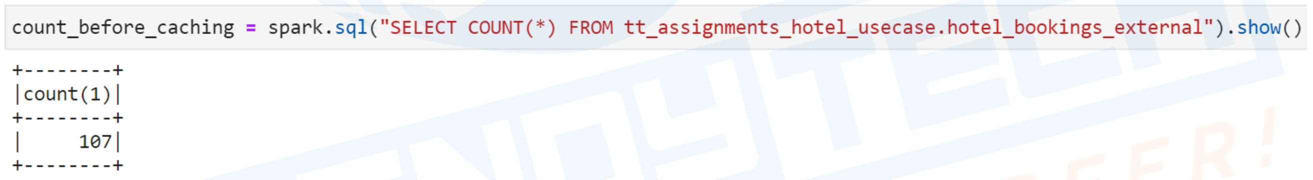


```
spark.sql("select * from tt_assignments_hotel_usecase.hotel_bookings_external limit 5").show()
```

```
+----------+------------+------------+-------------+---------+-----------+
|booking_id|  guest_name|checkin_date|checkout_date|room_type|total_price|
+----------+------------+------------+-------------+---------+-----------+
|         1|    John Doe|  2023-05-01|   2023-05-05| Standard|      400.0|
|         2|  Jane Smith|  2023-05-02|   2023-05-06|   Deluxe|      600.0|
|         3|Mark Johnson|  2023-05-03|   2023-05-08| Standard|      450.0|
|         4|Sarah Wilson|  2023-05-04|   2023-05-07|Executive|      750.0|
|         5| Emily Brown|  2023-05-06|   2023-05-09|   Deluxe|      550.0|
+----------+------------+------------+-------------+---------+-----------+
```

A)

count_before_caching = spark.sql("SELECT COUNT(*) FROM tt_assignments_hotel_usecase.hotel_bookings_external").show()

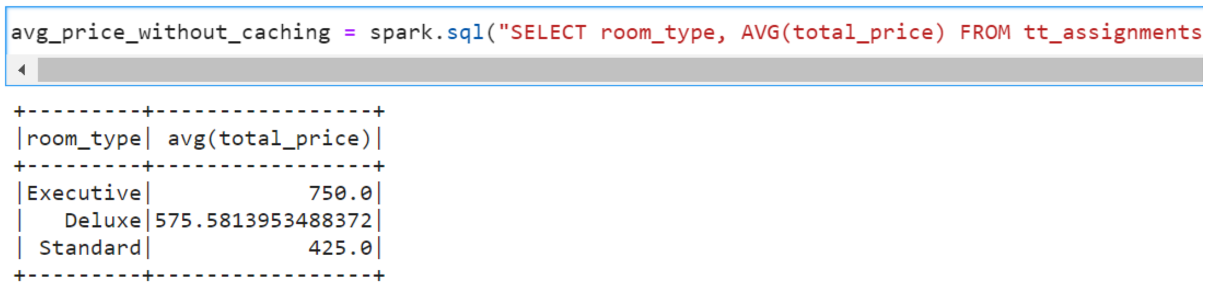


```
count_before_caching = spark.sql("SELECT COUNT(*) FROM tt_assignments_hotel_usecase.hotel_bookings_external").show()
```

```
+--------+
|count(1)|
+--------+
|     107|
+--------+
```

B)

avg_price_without_caching = spark.sql("SELECT room_type, AVG(total_price) FROM tt_assignments_hotel_usecase.hotel_bookings_external GROUP BY room_type limit 100").show()



```
avg_price_without_caching = spark.sql("SELECT room_type, AVG(total_price) FROM tt_assignments
```

```
+---------+-----------------+
|room_type| avg(total_price)|
+---------+-----------------+
|Executive|            750.0|
|   Deluxe|575.5813953488372|
| Standard|            425.0|
+---------+-----------------+
```

#with caching

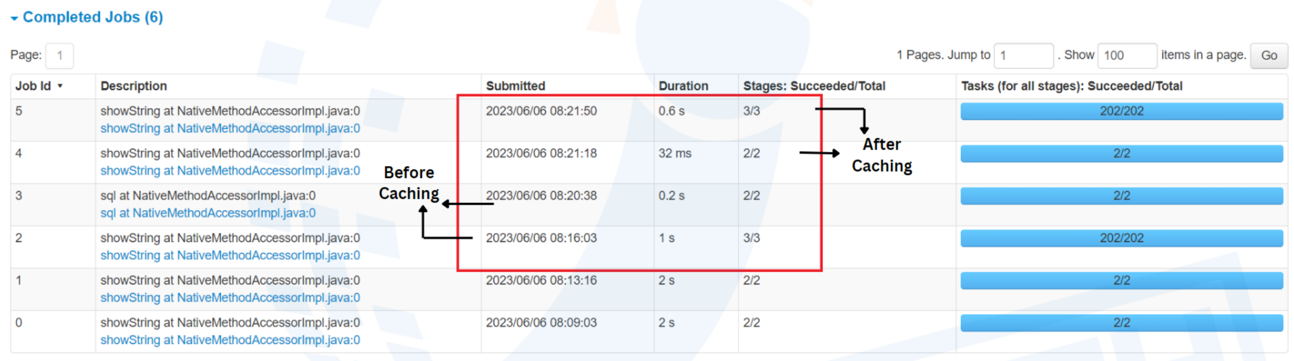
```
spark.sql("cache table tt_hotel.hotel_bookings_external")
```

A)

```
count_after_caching = spark.sql("SELECT COUNT(*) FROM tt_hotel.hotel_bookings_external").show()
```

B)

```
avg_price_with_caching = spark.sql("SELECT room_type, AVG(total_price) FROM tt_hotel.hotel_bookings_external GROUP BY room_type limit 100").show()
```



Completed Jobs (6)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 08:21:50	0.6 s	3/3	202/202
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 08:21:18	32 ms	2/2	2/2
3	sql at NativeMethodAccessorImpl.java:0 sql at NativeMethodAccessorImpl.java:0	2023/06/06 08:20:38	0.2 s	2/2	2/2
2	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 08:16:03	1 s	3/3	202/202
1	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 08:13:16	2 s	2/2	2/2
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/06/06 08:09:03	2 s	2/2	2/2

Job	Without caching	With caching
1	1s	0.6s
2	0.2s	32ms

Note: You can see a large difference when dealing with really big data. Here since the data is small, the comparisons might be very less and might be varying.

C)

```
spark.sql("uncache table tt_hotel.hotel_bookings_external")
```