

Day46 - March 22nd 2024

1. Started my day as usual
2. Solved one leetcode medium problem

3. Started learning PySpark from youtube channel names manish kumar

docs.google.com/document/d/1vtgtCNlkiyHrPocS1pdTSaT2FpQFQBikT_0ljKNIsc/edit

Spark Theory Day3&4

File Edit View Tools Help

Request edit access Share

- Disadvantage: RDDs don't have a schema defining data types. This can lead to runtime errors if transformations operate on unexpected data formats.
- Example: Imagine an RDD with user data, where the age element might accidentally be a string ("thirty") instead of an integer. Processing code expecting an integer would fail during transformations with RDDs.

Spark Session and Spark Context

- If we want to run a spark code in a cluster, then we need to have spark session in the cluster
- Explained : <https://q.co/gemini/share/6b285cd1ed9f>

```
findspark.init()
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.master("local[5]")\
    .appName("testing").getOrCreate()
print(spark)
spark2= spark.newSession()
print(spark2)
sc = spark.sparkContext
print(sc)
```

- Here in this code we have initiated spark session
- We have assigned spark with node "local[5]" and appName of "testing", and .getOrCreate(). If there's existing spark session with this config then it gets that ..or else it creates new

Spark Job, Stages, Tasks

- Potential interview questions

8°C Windy

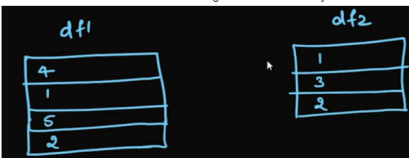
docs.google.com/document/d/1vtgtCNlkiyHrPocS1pdTSaT2FpQFQBikT_0ljKNIsc/edit

Spark Theory Day3&4

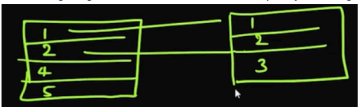
File Edit View Tools Help

Request edit access Share

25. Lets consider we have 2 df., which undergone shuffled already



26. After sorting we get our df's like this..and time complexity of sorting is $n \log n$



27. After sorting it joins the id's of 2 df

Shuffle Sort Merge Join (SSMJ) is the default join strategy in Apache Spark for large datasets (since Spark 2.3). It efficiently joins two DataFrames based on a shared key using a three-step process: shuffle, sort, and merge.

Here's how it works:

- Shuffle:**
 - Both DataFrames are shuffled and repartitioned based on the join key. Spark utilizes a hash function to determine the target partition for each record based on its join key value.
 - Records with the same join key are sent to the same executor node, ensuring they are co-located for efficient joining. This shuffle operation involves data movement across the network.
- Sort:**
 - On each executor node, the repartitioned data within each partition is sorted by the join key. Sorting allows for efficient merging in the next step.
- Merge:**

8°C Windy

4. Please find the doc here : [Spark Theory Day3&4](#)

5. Ended my day by solving a complex SQL question from Ankit's YT

SQLQuery1.sql - KAUSHI\SQLEXPRESS:master (KAUSHI\jamka (52)) - Microsoft SQL Server Management Studio

```

/*Question:
Identify the game_type
1. No Social Interaction(No messages or gifts between gamers in the game)
2. One Sided Interaction(Messages or gifts sent by only one player in the game)
3. Both sided Interaction without custom_typed message
4. Both Sided Interaction with custom_types_message from atleast one player
*/
select * from user_interactions

```

Results Messages

user_id	event	event_date	interaction_type	game_id	event_time
abc	game_start	2024-01-01	NULL	ab0000	10:00:00.0000000
def	game_start	2024-01-01	NULL	ab0000	10:00:00.0000000
def	send_emoji	2024-01-01	emoji1	ab0000	10:03:20.0000000
def	send_message	2024-01-01	preloaded_quick	ab0000	10:03:49.0000000
abc	send_gift	2024-01-01	gift1	ab0000	10:04:40.0000000
abc	game_end	2024-01-01	NULL	ab0000	10:10:00.0000000
def	game_end	2024-01-01	NULL	ab0000	10:10:00.0000000
abc	game_start	2024-01-01	NULL	ab9999	10:00:00.0000000
def	game_start	2024-01-01	NULL	ab9999	10:00:00.0000000
abc	send_message	2024-01-01	custom_typed	ab9999	10:02:43.0000000
abc	send_gift	2024-01-01	gift1	ab9999	10:04:40.0000000
abc	game_end	2024-01-01	NULL	ab9999	10:10:00.0000000
abc	game_end	2024-01-01	NULL	ab9999	10:10:00.0000000
abc	game_start	2024-01-01	NULL	ab1111	10:00:00.0000000
def	game_start	2024-01-01	NULL	ab1111	10:00:00.0000000
abc	game_end	2024-01-01	NULL	ab1111	10:10:00.0000000
def	game_end	2024-01-01	NULL	ab1111	10:10:00.0000000
abc	game_start	2024-01-01	NULL	ab1234	10:00:00.0000000
def	game_start	2024-01-01	NULL	ab1234	10:00:00.0000000
abc	send_message	2024-01-01	custom_typed	ab1234	10:02:43.0000000
def	send_emoji	2024-01-01	emoji1	ab1234	10:03:20.0000000
def	send_message	2024-01-01	preloaded_quick	ab1234	10:03:49.0000000
abc	send_gift	2024-01-01	gift1	ab1234	10:04:40.0000000
abc	game_end	2024-01-01	NULL	ab1234	10:10:00.0000000
def	game_end	2024-01-01	NULL	ab1234	10:10:00.0000000

Query executed successfully.

SQLQuery1.sql - KAUSHI\SQLEXPRESS:master (KAUSHI\jamka (52)) - Microsoft SQL Server Management Studio

```

select game_id,case when count(interaction_type) = 0 then 'No Social Interaction'
when count(distinct case when interaction_type is not null then user_id end)=1 then 'One Sided Interaction'
when count(distinct case when interaction_type is not null then user_id end)=2 and
count(distinct case when interaction_type = 'custom_typed' then user_id end) = 0
then 'Both sided interaction without custom_typed messages'
when count(distinct case when interaction_type is not null then user_id end)=2 and
count(distinct case when interaction_type = 'custom_typed' then user_id end) >= 1
then 'Both sided interaction with custom_types_messages from atleast one player'
end as game_type
from user_interactions
group by game_id

```

Results Messages

game_id	game_type
ab0000	Both sided interaction without custom_typed mess...
ab1111	No Social Interaction
ab1234	Both sided interaction with custom_types_message...
ab9999	One Sided Interaction

Query executed successfully.