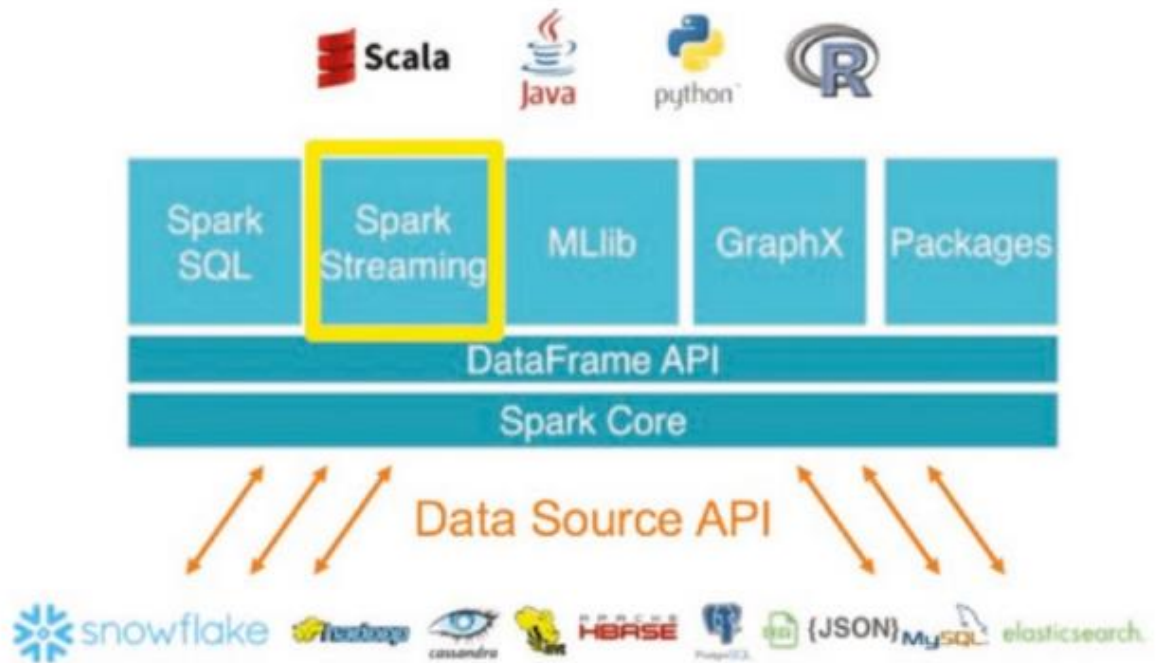


## **Chapitre 3 :**

## **Streaming structure**

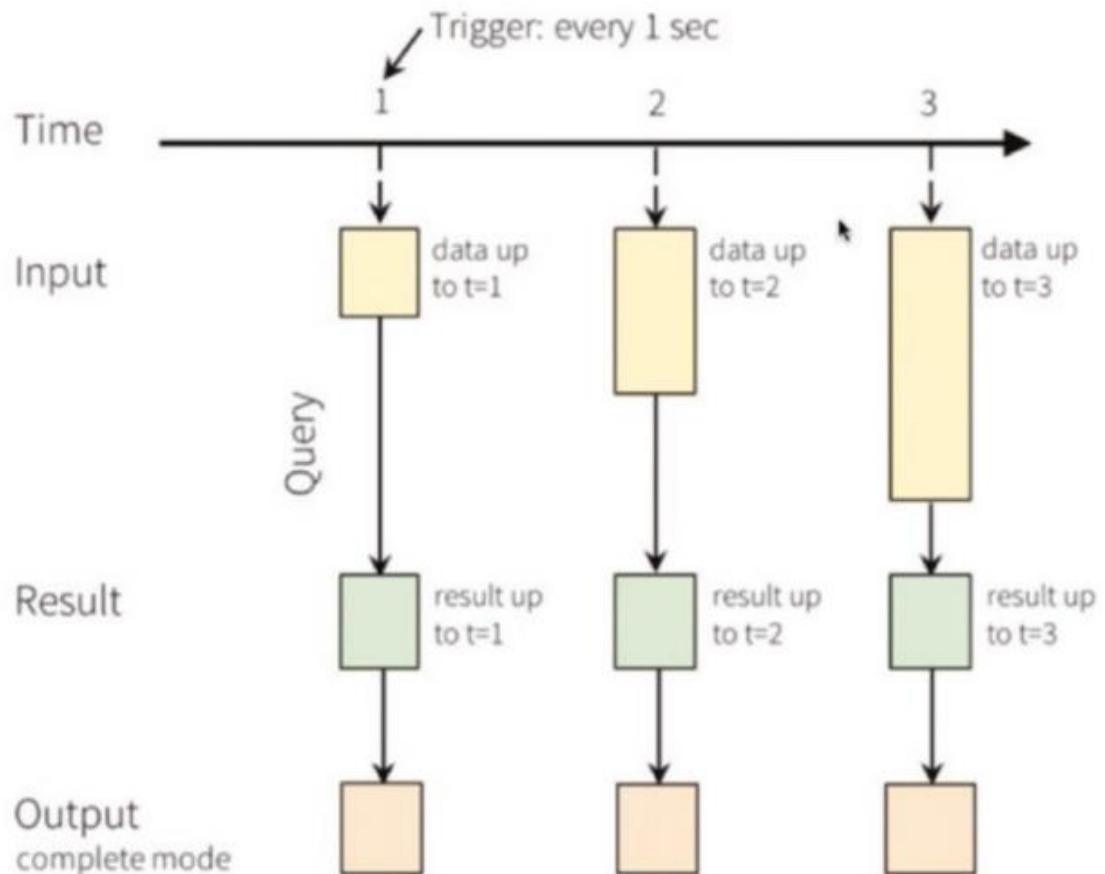
### **A. Notes de cours**

- La différence entre traitement de données par lots et par flux est que :
  - le lot fait référence à un groupe d'enregistrements rassemblés sur une période de temps et utilisés ultérieurement pour le traitement et l'analyse. Étant donné que ces enregistrements sont collectés sur une période de temps, en termes de taille, les données de lot sont généralement plus volumineuses que les données de flux (dans certains cas, cependant, les données de flux peuvent être plus volumineuses que les données de lot) et sont souvent utilisées pour effectuer des post-mortem à diverses fins d'analyse.
  - Le traitement de flux fait référence au traitement des enregistrements en temps réel ou quasi réel. On n'attend pas la fin de la journée pour ensuite traiter ou analyser les données. Au lieu de cela, les enregistrements de l'ensemble de données sont traités un par un dès qu'ils deviennent disponibles ou sur la base d'une période de fenêtre.
  - Les entreprises veulent utiliser les données les plus récentes ou les plus récentes pour générer des informations utiles qui peuvent aider à la prise de décision. Le traitement par lots ne peut pas offrir d'analyse à la volée, car il ne fonctionne pas en temps réel, tandis que le traitement des données en continu peut aider plus efficacement dans des cas tels que la détection de fraude.
- Spark Streaming l'un des composants du framework Spark présente cette architecture



- Structured Streaming : est la dernière version du composant de streaming dans Spark et offre la même API pour les travaux de traitement de données par lots et par flux.

Son processus est la suivante :



Les trois domaines principaux dans lesquels nous pouvons diviser ce cadre de diffusion de streaming structuré :

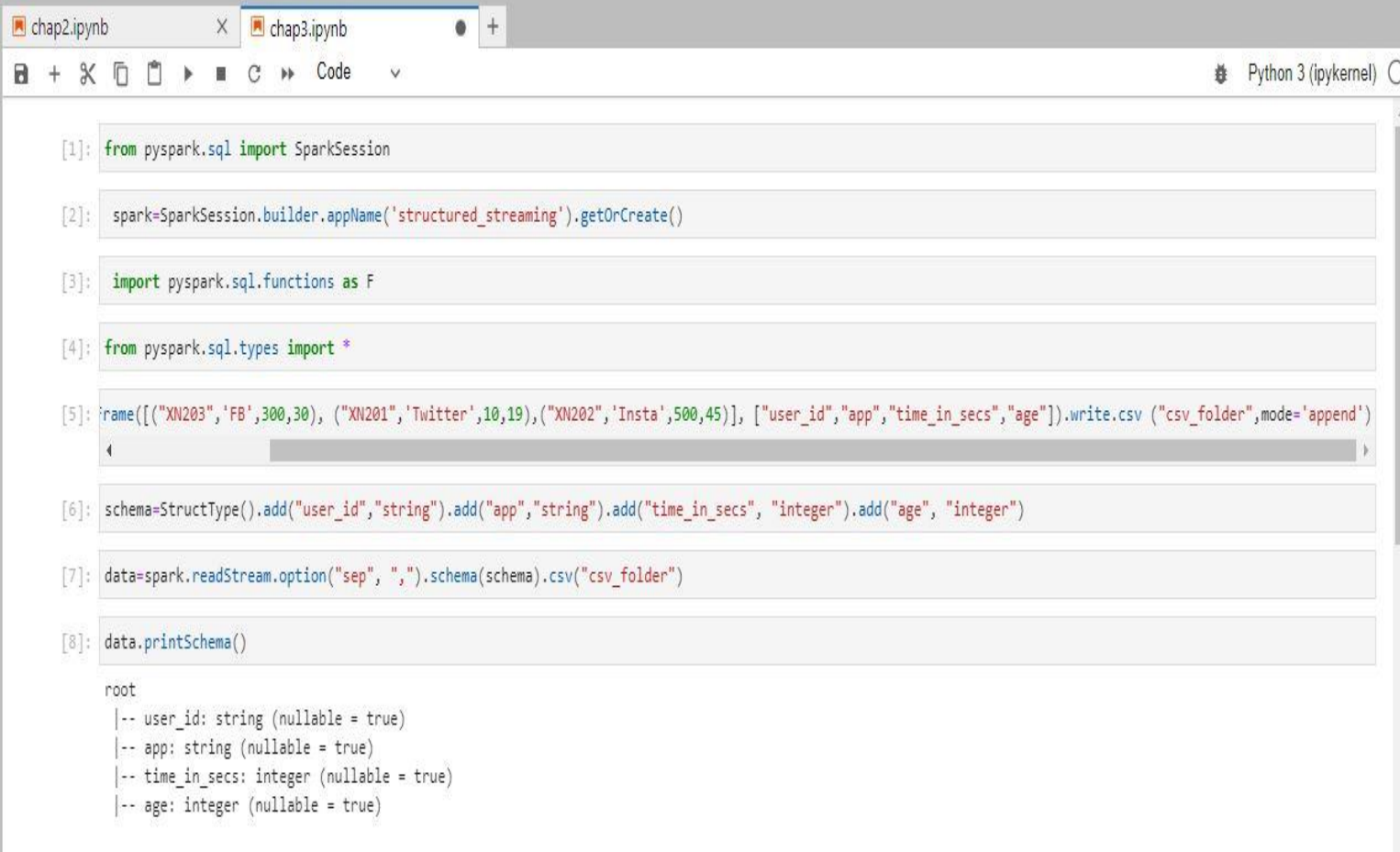
- Data input
- Data processing (real time or near real time)
- Final output

## **B. CAS PRATIQUE (Construction d'une application de streaming structuré)**

- Création de l'objet SparkSession, puis nous créons des données auto-générées qui seront poussées dans un repertoire locale nommé **csv\_folder**. Les données que nous allons générées sont au format csv et contiennent 04 colonnes. Une fois que les données sont

créés, nous allons définir le schéma de fichier afin de le lire en utilisant un traitement de flux.

- Pour valider le schema du dataframe, nous utilisons la fonction **printschema()**.



```
[1]: from pyspark.sql import SparkSession

[2]: spark=SparkSession.builder.appName('structured_streaming').getOrCreate()

[3]: import pyspark.sql.functions as F

[4]: from pyspark.sql.types import *

[5]: rdd=spark.readStream.option("sep", ",").load("csv_folder").write.csv("csv_folder",mode='append')

[6]: schema=StructType().add("user_id","string").add("app","string").add("time_in_secs","integer").add("age","integer")

[7]: data=spark.readStream.option("sep", ",").schema(schema).csv("csv_folder")

[8]: data.printSchema()

root
|-- user_id: string (nullable = true)
|-- app: string (nullable = true)
|-- time_in_secs: integer (nullable = true)
|-- age: integer (nullable = true)
```

- Nous allons maintenant appliquer plusieurs transformations afin d'obtenir différents résultats. La première consiste tout simplement à compter le nombre d'enregistrements de chaque application dans le dataframe. Dans cet exemple, nous écrivons le résultat en mémoire et nous utilisons une simple commande spark sql pour afficher le résultat de la requête que nous avons exécutée sur le dataframe en continu, en le convertissant en un dataframe Pandas.

```
[9]: app_count=data.groupBy('app').count()
[10]: query=(app_count.writeStream.queryName('count_query').outputMode('complete').format('memory').start())
[19]: spark.sql("select * from count_query ").toPandas()
```

	app	count
0	Insta	2
1	FB	2
2	Twitter	2

- Requete pour filtrer uniquement les enregistrements de l'application Facebook(FB) ainsi que le temps moyen passe par chaque utilisateur sur l'application.

```
[13]: fb_data=data.filter(data['app']=='FB')
[14]: fb_avg_time=fb_data.groupBy('user_id').agg(F.avg("time_in_secs"))
[15]: fb_query=(fb_avg_time.writeStream.queryName('fb_query').outputMode('complete').format('memory').start())
[18]: spark.sql("select * from fb_query ").toPandas()
```

	user_id	avg(time_in_secs)
0	XN203	300.0

- Comme il n'y a qu'un seul dataframe dans le dossier local, nous obtenons les resultats d'un utilisateur accedant a FB et le temps passe. Afin d'afficher des resultats plus relatifs, poussons plus de donnees auto-generees vers le dossier.
- Nous pouvons maintenant supposer sans risque que spark structured streaming a lu les nouveaux enregistrements et les a ajoutes dans le dataframe de streaming et par consequent, les nouveaux resultats pour la meme requete seront differents du dernier.

```
[24]: df_2=spark.createDataFrame([("XN203",'FB',100,30),("XN201",'FB',10,19),("XN202",'FB',2000,45)],["user_id","app","time_in_secs","age"]).write.csv("csv_folder",mo
```

```
[27]: spark.sql("select * from fb_query ").toPandas()
```

```
[27]:
```

	user_id	avg(time_in_secs)
0	XN203	209.090909
1	XN201	10.000000
2	XN202	2000.000000

- Ajoutons quelques enregistrements supplémentaires au dossier ; on constate qu'il y'a aggregation et le tri de la requete sur le dataframe existant dans le dossier local.

```
[28]: df_3=spark.createDataFrame([("XN203",'FB',500,30), ("XN201",'Insta',30,19),("XN202",'Twitter',100,45)],["user_id","app","time_in_secs","age"]).write.csv("csv_fo
```

```
[31]: spark.sql("select * from fb_query ").toPandas()
```

```
[31]:
```

	user_id	avg(time_in_secs)
0	XN203	223.076923
1	XN201	10.000000
2	XN202	2000.000000

```
[ ]:
```

- Nous regroupons ensuite les enregistrements par application et calculons le temps total passe sur chaque application par ordre décroissant.

```
[32]: app_df=data.groupBy('app').agg(F.sum('time_in_secs').alias('total_time')).orderBy('total_time',ascending=False)
```

```
[33]: app_query=(app_df.writeStream.queryName('app_wise_query').outputMode('complete').format('memory').start())
```

```
[37]: spark.sql("select * from app_wise_query ").toPandas().head(5)
```

```
[37]:
```

	app	total_time
0	FB	14960
1	Insta	3030
2	Twitter	160

- Ajoutons de nouveaux enregistrements et voyons les nouveaux resultats.

```
[43]: df_4=spark.createDataFrame([("XN203", 'FB', 500, 30), ("XN201", 'Insta', 30, 19), ("XN202", 'Twitter', 100, 45)], ["user_id", "app", "time_in_secs", "age"]).write.csv("csv_fol")
```

```
[45]: spark.sql("select * from app_wise_query ").toPandas()
```

```
[45]:
```

	app	total_time
0	FB	17870
1	Insta	3560
2	Twitter	270

- Trouvons maintenant l'age moyen des utilisateurs de chaque application ; pour cela, nous regroupons les donnees par application, puis nous prenons l'age moyen de tous les utilisateurs et nous trions les resultats par ordre decroissant.

```
[46]: age_df=data.groupby('app').agg(F.avg('age').alias('mean_age')).orderBy('mean_age',ascending=False)
```

```
[48]: age_query=(age_df.writeStream.queryName('age_query').outputMode('complete').format('memory').start())
```

```
[49]: df_5=spark.createDataFrame([("XN210", 'FB', 500, 50), ("XN255", 'Insta', 30, 23), ("XN222", 'Twitter', 100, 30)], ["user_id", "app", "time_in_secs", "age"]).write.csv("csv_fol")
```

```
[53]: spark.sql("select * from age_query ").toPandas()
```

```
[53]:
```

	app	mean_age
0	Insta	35.545455
1	FB	30.875000
2	Twitter	28.454545

- Exemple de fusion des donnees entrantes ( dataframe de flux) avec un dataframe statique qui contient le nom complet des applications.

```
[54]: app_df=spark.createDataFrame([('FB','FACEBOOK'),('Insta','INSTAGRAM'),('Twitter','TWITTER')],["app", "full_name"])
```

```
[55]: app_df.show()
```

```
+-----+-----+
|  app |full_name|
+-----+-----+
|   FB |FACEBOOK|
| Insta|INSTAGRAM|
|Twitter| TWITTER|
+-----+-----+
```

- Maintenant que nous disposons d'un dataframe statique, nous pouvons simplement écrire une nouvelle requête pour joindre le dataframe en continu avec lequel nous avons travaillé jusqu'à présent et les fusionner dans une colonne d'application.

```
[56]: app_stream_df=data.join(app_df,'app')
```

```
[57]: join_query=(app_stream_df.writeStream.queryName('join_query').outputMode('append').format('memory').start())
```

```
[60]: spark.sql("select * from join_query ").toPandas().head(10)
```

```
[60]:
```

	app	user_id	time_in_secs	age	full_name
0	FB	XN201	10	19	FACEBOOK
1	FB	XN201	10	19	FACEBOOK
2	FB	XN201	10	19	FACEBOOK
3	FB	XN201	10	19	FACEBOOK
4	FB	XN201	10	19	FACEBOOK
5	FB	XN201	10	19	FACEBOOK
6	FB	XN201	10	19	FACEBOOK
7	FB	XN210	500	50	FACEBOOK
8	FB	XN203	500	30	FACEBOOK
9	FB	XN203	500	30	FACEBOOK