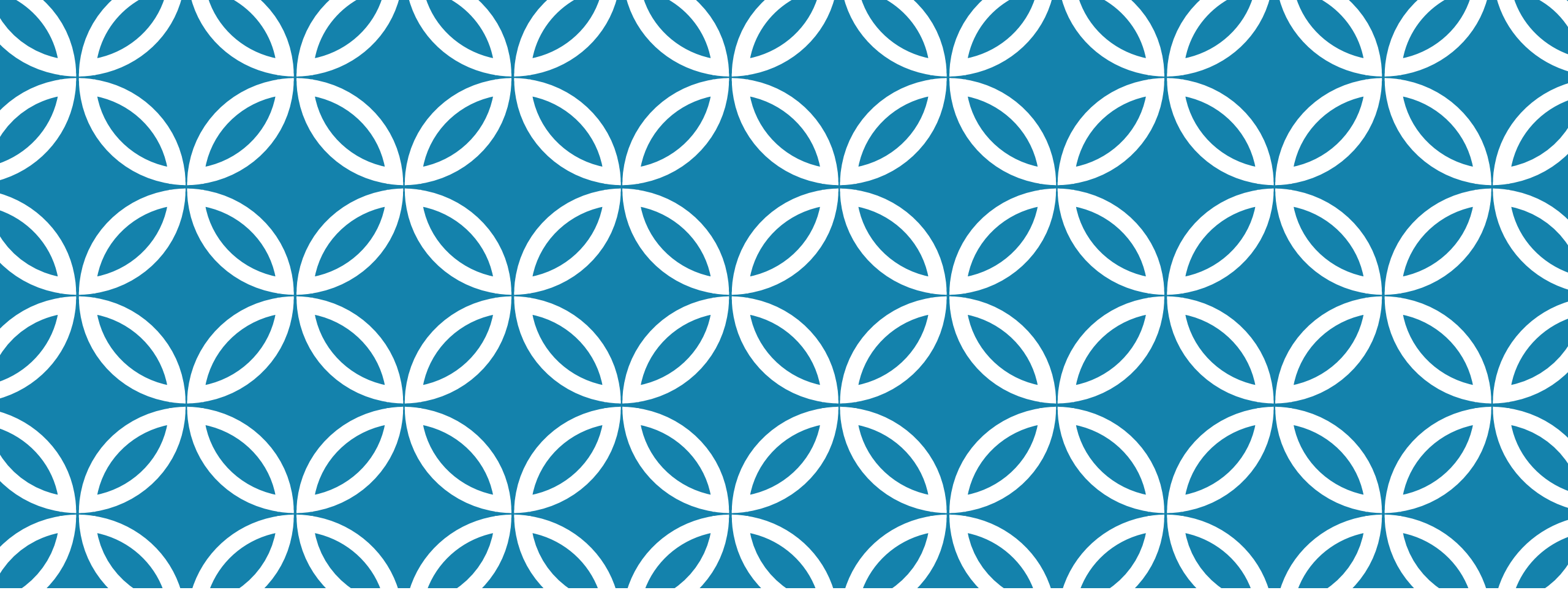

CHAPITRE 3 SPARK SQL ET DATAFRAME

Introduction a Apache Spark

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

1) Utilisation de Spark SQL dans une Application Spark

Spark SQL est un module de Spark permettant d'exécuter des requêtes SQL sur des données stockées sous forme de **DataFrame** ou de **tables SQL.**

Il permet de travailler sur des **données structurées** avec un langage familier (**SQL**) tout en bénéficiant des performances de Spark.

On peut utiliser Spark SQL directement dans une **application Spark** à travers :

- L'**API DataFrame** en Scala, Java, Python et R.
- Le **Spark SQL Shell** pour exécuter des requêtes interactives.
- Les **interfaces JDBC/ODBC** pour se connecter à des outils BI (ex : Tableau).

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

2) Tables et Vues SQL

a) Tables gérées vs tables non gérées

- **Tables gérées (Managed Tables) :**
  - Spark gère les métadonnées et les fichiers des données.
  - Si la table est supprimée, les fichiers sous-jacents sont aussi supprimés.
  - Stockage par défaut dans le répertoire `spark-warehouse`.
- **Tables non gérées (External Tables) :**
  - Spark gère uniquement les métadonnées, mais les fichiers sont stockés ailleurs (HDFS, S3, etc.).
  - La suppression de la table ne supprime pas les données physiques.

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

b) Création des bases de données et des tables

Une **base de données** est un espace de noms pour organiser les tables.

Commandes principales :

```
CREATE DATABASE ma_base;
USE ma_base;
CREATE TABLE ma_table (id INT, nom STRING) USING PARQUET;
```

c) Création des vues

Une **vue** est une table virtuelle basée sur une requête SQL.

Deux types :

- **Vue temporaire** : Valable pour la session en cours.
- **Vue persistante** : Stockée et réutilisable entre sessions.

```
CREATE OR REPLACE TEMP VIEW vue_temp AS SELECT * FROM ma_table;
```

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

d) Affichage des métadonnées

- Permet d'obtenir des informations sur les bases, tables et vues :

```
SHOW DATABASES;
SHOW TABLES;
DESCRIBE TABLE ma_table;
```

e) Tables SQL et mise en cache

- **Mise en cache** : Améliore la performance en gardant les données en mémoire.

```
CACHE TABLE ma_table;
UNCACHE TABLE ma_table;
```

f) Lire les tables dans les DataFrame

- Convertir une table SQL en DataFrame :

```
df = spark.table("ma_table")
df.show()
```

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

3) Sources de Données pour les DataFrame et les Tables SQL

a) DataFrameReader

- Utilisé pour **charger des données** depuis différentes sources.

```
df = spark.read.format("json").load("data.json")
```

b) DataFrameWriter

- Utilisé pour **enregistrer des DataFrames** dans divers formats.

```
df.write.format("parquet").save("output/")
```

c) Parquet

- Format **colonne** optimisé pour **la compression et la rapidité.**

d) JSON

- Format **semi-structuré** adapté aux **API et logs.**

A) INTRODUCTION AUX SOURCES DE DONNÉES INTÉGRÉES

e) Avro

- Format **compact** utilisé dans **Apache Kafka et Hadoop.**

f) ORC

- Similaire à **Parquet,** mais optimisé pour **Hive.**

g) Images

- Spark peut traiter les images pour **l'analyse et l'IA.**

h) Binary File

- Supporte le **stockage et l'analyse de fichiers binaires.**

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

1) Spark SQL et Apache Hive

a) Fonctions définies par l'utilisateur (UDFs)

Permet d'ajouter des **fonctions SQL personnalisées** en Python ou Scala.

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def majuscule(nom):
    return nom.upper()

majuscule_udf = udf(majuscule, StringType())
df.withColumn("nom_majuscules", majuscule_udf(df["nom"])).show()
```

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

2) Interagir avec Spark SQL, Beeline et Tableau

a) Utilisation de Spark SQL Shell

Interface en ligne de commande pour exécuter des requêtes SQL Spark.

b) Travailler avec Beeline

Client JDBC pour interroger **Apache Hive.**

c) Travailler avec Tableau

Connecter Tableau à Spark via **JDBC/ODBC** pour **visualiser les données.**

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

3) Sources de Données Externes

a) JDBC et SQL Databases

- Spark permet de lire et écrire dans des **bases relationnelles** via JDBC.

```
df = spark.read.format("jdbc").option("url", "jdbc:mysql://host:port/db").load()
```

b) PostgreSQL

- Base SQL Open Source très utilisée.

c) MySQL

- Base SQL populaire compatible avec Spark JDBC.

d) Azure Cosmos DB

- Base **NoSQL** multi-modèle sur le cloud Microsoft.

e) MS SQL Server

- Intégration avec **Microsoft SQL Server** via JDBC.

f) Autres sources (MongoDB, Cassandra, Snowflake)

- Spark permet d'interagir avec des **bases NoSQL** via des connecteurs spécialisés.

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

4) Fonction d'Ordre Supérieur dans DataFrame et Spark SQL

a) Option 1 : Explode et Collect

- `explode()` transforme une colonne **array** en plusieurs lignes.

```
df.select(explode(df.liste)).show()
```

- `collect_list()` regroupe plusieurs valeurs en un **tableau**.

b) Option 2 : Fonctions définies par l'utilisateur (UDFs)

- Voir la section précédente sur les **UDFs**.

c) Fonction intégrale pour le type de données complexes

- Spark permet de manipuler **des structures imbriquées** (array, struct, map).

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

a) Union

- Fusionner deux DataFrames avec le **même schéma.**

```
df_union = df1.union(df2)
```

b) Joins

- Effectuer des jointures entre tables ou DataFrames.

```
df_join = df1.join(df2, "id", "inner")
```

B) INTERROGER LES SOURCES DE DONNÉES EXTERNES

c) Windowing

- Appliquer des **fonctions de fenêtre** pour des analyses avancées.

```
from pyspark.sql.window import Window

from pyspark.sql.functions import row_number

 windowSpec = Window.partitionBy("groupe").orderBy("valeur")
 df.withColumn("rang", row_number().over(windowSpec)).show()
```

d) Modification

- Transformer ou mettre à jour les données dans Spark.

```
df = df.withColumn("nouvelle_colonne", df["ancienne_colonne"] * 2)
```