Lorsque vous travaillez avec PySpark dans Databricks, il est essentiel de connaître certaines commandes de base pour manipuler les données et exécuter des opérations sur les DataFrames. Voici une liste des commandes les plus importantes et couramment utilisées :

1. Initialisation du contexte Spark

- **`SparkSession`** : Point d'entrée pour travailler avec Spark. Databricks fournit généralement un SparkSession déjà initialisé nommé `spark`.

```
python
Copier le code
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("NomApp").getOrCreate()
```

2. Chargement des données

- **`read`** : Charger des données dans un DataFrame.

```
python
Copier le code
df = spark.read.csv("chemin/vers/fichier.csv", header=True,
inferSchema=True)
df = spark.read.json("chemin/vers/fichier.json")
df = spark.read.parquet("chemin/vers/fichier.parquet")
```

3. Affichage des données

- **`show`** : Afficher les premières lignes d'un DataFrame.

```
python
Copier le code
df.show()
df.show(10, truncate=False)  # Afficher 10 lignes sans tronquer les
chaînes de caractères
```

- **`printSchema`** : Afficher le schéma du DataFrame.

```
python
Copier le code
df.printSchema()
```

- **`head`** et **`take`** : Récupérer les premières lignes.

```
python
Copier le code
df.head(5)
df.take(10)
```

4. Opérations sur les colonnes

- **`select`** : Sélectionner des colonnes spécifiques.

```
python
Copier le code
```

```
df.select("colonne1", "colonne2").show()
```

- **`withColumn`** : Ajouter ou modifier une colonne.

```
python
Copier le code
from pyspark.sql.functions import col
df = df.withColumn("nouvelle_colonne", col("colonne_existante") * 2)
```

- **`drop`** : Supprimer une colonne.

```
python
Copier le code
df = df.drop("colonne_a_supprimer")
```

5. Filtrage des données

- **`filter`** **ou** **`where`** : Filtrer les lignes en fonction d'une condition.

```
python
Copier le code
df_filtered = df.filter(col("age") > 30)
df_filtered = df.where("age > 30")
```

6. Groupement et agrégation

- **`groupBy`** : Grouper les données par une ou plusieurs colonnes.

```
python
Copier le code
df_grouped = df.groupBy("colonne").count()
df_grouped = df.groupBy("colonne").agg({"age": "avg", "salary":
"sum"})
```

- **`agg`** : Appliquer des fonctions d'agrégation.

```
python
Copier le code
from pyspark.sql.functions import avg, sum
df_agg = df.groupBy("colonne").agg(avg("age").alias("moyenne_age"),
sum("salaire").alias("total_salaire"))
```

7. Jointures

- **`join`** : Joindre deux DataFrames.

```
python
Copier le code
df_joined = df1.join(df2, df1["id"] == df2["id"], "inner")  # Types :
inner, left, right, full
```

8. Enregistrement des données

- **`write`** : Enregistrer un DataFrame dans un format de fichier.

```
python
Copier le code
df.write.csv("chemin/vers/sortie.csv", header=True)
df.write.json("chemin/vers/sortie.json")
df.write.parquet("chemin/vers/sortie.parquet")
```

9. Opérations SQL

- **`createOrReplaceTempView`** : Créer une vue temporaire pour exécuter des requêtes SQL.

```
python
Copier le code
df.createOrReplaceTempView("vue_temporaire")
result = spark.sql("SELECT * FROM vue_temporaire WHERE age > 30")
result.show()
```

10. Gestion des partitions

- **`repartition`** et **`coalesce`** : Gérer le nombre de partitions des données.

```
python
Copier le code
df_repartitioned = df.repartition(10)  # Augmenter le nombre de
partitions
df_coalesced = df.coalesce(5)  # Réduire le nombre de partitions
```

11. Cache et persistance

- **`cache`** et **`persist`** : Mettre en cache un DataFrame pour améliorer les performances lors de plusieurs utilisations.

```
python
Copier le code
df.cache()  # Cache en mémoire
df.persist(StorageLevel.MEMORY_AND_DISK)  # Cache en mémoire et
disque
df.unpersist()  # Libérer le cache
```

12. Analyse et diagnostics

- **`explain`** : Afficher le plan d'exécution du DataFrame.

```
python
Copier le code
df.explain(True)
```

- **`count`** : Compter le nombre de lignes dans un DataFrame.

```
python
Copier le code
df.count()
```

Ces commandes vous permettront de manipuler efficacement vos données et de tirer parti des fonctionnalités de PySpark dans un environnement Databricks.

Bien sûr ! Voici d'autres commandes PySpark importantes qui vous permettront d'approfondir la manipulation et l'analyse des données dans Databricks.

13. Opérations avancées sur les colonnes

- **`alias`** : Donner un alias à une colonne ou à un DataFrame.

```
python
Copier le code
df.select(col("nom").alias("nom_utilisateur")).show()
```

- **`cast`** : Changer le type de données d'une colonne.

```
python
Copier le code
df = df.withColumn("age", col("age").cast("int"))
```

- **`substr`** : Extraire une sous-chaîne d'une colonne de type chaîne de caractères.

```
python
Copier le code
df.select(col("nom").substr(1, 3).alias("initiales")).show()
```

14. Fonctions sur les chaînes de caractères

- **`lower`** et **`upper`** : Convertir une chaîne en minuscules ou majuscules.

```
python
Copier le code
from pyspark.sql.functions import lower, upper
df = df.withColumn("nom_minuscule", lower(col("nom")))
df = df.withColumn("nom_majuscule", upper(col("nom")))
```

- **`concat`** : Concaténer plusieurs colonnes.

```
python
Copier le code
from pyspark.sql.functions import concat, lit
df = df.withColumn("nom_complet", concat(col("prenom"), lit(" "),
col("nom")))
```

- **`trim`** : Supprimer les espaces en début et fin de chaîne.

```
python
Copier le code
from pyspark.sql.functions import trim
df = df.withColumn("nom", trim(col("nom")))
```

15. Fonctions sur les dates et les heures

- **`current_date`** et **`current_timestamp`** : Obtenir la date ou l'heure actuelle.

```
python
Copier le code
from pyspark.sql.functions import current_date, current_timestamp
df = df.withColumn("date_du_jour", current_date())
df = df.withColumn("heure_actuelle", current_timestamp())
```

- **`date_format`** : Formater une colonne de type date.

```
python
Copier le code
from pyspark.sql.functions import date_format
df = df.withColumn("date_formatee", date_format(col("date"),
"dd/MM/yyyy"))
```

- **`datediff`** et **`add_months`** : Calculer la différence entre des dates ou ajouter des mois.

```
python
Copier le code
from pyspark.sql.functions import datediff, add_months
df = df.withColumn("jours_ecoules", datediff(current_date(),
col("date_naissance")))
df = df.withColumn("date_plus_3_mois", add_months(col("date"), 3))
```

16. Fonctions de fenêtre (Window Functions)

- **`window`** et **`row_number`** : Appliquer des calculs sur des fenêtres de données.

```
python
Copier le code
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec = Window.partitionBy("colonne").orderBy("date")
df = df.withColumn("rang", row_number().over(windowSpec))
```

- **`lag`** et **`lead`** : Accéder aux valeurs précédentes ou suivantes dans une partition.

```
python
Copier le code
from pyspark.sql.functions import lag, lead
df = df.withColumn("valeur_precedente", lag("colonne",
1).over(windowSpec))
df = df.withColumn("valeur_suivante", lead("colonne",
1).over(windowSpec))
```

17. Fonctions de pivot et unpivot

- **`pivot`** : Transformer les lignes en colonnes.

```
python
Copier le code
df_pivot = df.groupBy("categorie").pivot("type").sum("valeur")
df_pivot.show()
```

- **`melt`** **(unpivot)** : Transformer les colonnes en lignes (utilisation via `selectExpr`).

```
python
```

```
Copier le code
df_unpivot = df.selectExpr("id", "stack(3, 'col1', col1, 'col2',
col2, 'col3', col3) as (variable, valeur)")
df_unpivot.show()
```

18. Gestion des valeurs manquantes

- **`fillna`** : Remplacer les valeurs nulles par une valeur spécifique.

```
python
Copier le code
df = df.fillna({"age": 0, "nom": "inconnu"})
```

- **`dropna`** : Supprimer les lignes contenant des valeurs nulles.

```
python
Copier le code
df = df.dropna()
```

- **`replace`** : Remplacer des valeurs spécifiques dans une colonne.

```
python
Copier le code
df = df.replace(["ancien_valeur"], ["nouvelle_valeur"], "colonne")
```

19. Opérations sur les ensembles

- **`union`** et **`unionByName`** : Fusionner deux DataFrames.

```
python
Copier le code
df_union = df1.union(df2)  # Par position
df_union = df1.unionByName(df2)  # Par nom de colonne
```

- **`intersect`** et **`except`** : Intersecter ou exclure des lignes entre deux DataFrames.

```
python
Copier le code
df_intersection = df1.intersect(df2)
df_diff = df1.except(df2)
```

20. Échantillonnage et gestion aléatoire

- **`sample`** : Prendre un échantillon aléatoire des données.

```
python
Copier le code
df_sample = df.sample(withReplacement=False, fraction=0.1)
```

- **`randomSplit`** : Diviser un DataFrame en plusieurs sous-ensembles de manière aléatoire.

```
python
Copier le code
```

```
train_df, test_df = df.randomSplit([0.8, 0.2])
```

21. Récupération des statistiques

- **`describe`** : Obtenir des statistiques descriptives sur les colonnes numériques.

```
python
Copier le code
df.describe().show()
```

- **`summary`** : Obtenir des statistiques plus complètes, y compris la médiane, les percentiles, etc.

```
python
Copier le code
df.summary().show()
```

- **`corr`** et **`cov`** : Calculer la corrélation ou la covariance entre deux colonnes.

```
python
Copier le code
correlation = df.stat.corr("colonne1", "colonne2")
covariance = df.stat.cov("colonne1", "colonne2")
```

22. Fonctions de cartographie

- **`explode`** : Éclater un tableau ou une structure imbriquée en lignes individuelles.

```
python
Copier le code
from pyspark.sql.functions import explode
df = df.withColumn("element", explode(col("colonne_tableau")))
```

- **`map`** : Appliquer une fonction Python à chaque élément d'une colonne via `rdd`.

```
python
Copier le code
df = df.rdd.map(lambda row: (row.col1, row.col2 * 2)).toDF(["col1",
"col2_multiplié"])
```

23. Conversion et manipulation des types

- **`to_date`** et **`to_timestamp`** : Convertir une chaîne en date ou en timestamp.

```
python
Copier le code
from pyspark.sql.functions import to_date, to_timestamp
df = df.withColumn("date_col", to_date(col("date_string"), "yyyy-MM-
dd"))
df = df.withColumn("timestamp_col",
to_timestamp(col("timestamp_string"), "yyyy-MM-dd HH:mm:ss"))
```

24. Manipulation des données structurées

- **`selectExpr`** : Utiliser des expressions SQL dans un `select`.

```
python
Copier le code
df = df.selectExpr("col1 + col2 as somme", "col3 as alias_col3")
```

- **`distinct`** et **`dropDuplicates`** : Obtenir des lignes uniques dans un DataFrame.

```
python
Copier le code
df_distinct = df.distinct()
df_no_duplicates = df.dropDuplicates(["col1", "col2"])
```

Ces commandes avancées vous permettront de mieux exploiter les capacités de PySpark dans Databricks, en manipulant les données de manière plus complexe et en appliquant des analyses plus sophistiquées.

SQL

Databricks permet d'exécuter des requêtes SQL directement dans des notebooks, ce qui peut être très puissant pour manipuler et analyser des données de manière structurée. Voici un aperçu des fonctions SQL les plus couramment utilisées dans Databricks.

1. Fonctions d'agrégation

- **`COUNT`** : Compter le nombre de lignes.

```
sql
Copier le code
SELECT COUNT(*) FROM table;
```

- **`SUM`** : Calculer la somme d'une colonne.

```
sql
Copier le code
SELECT SUM(salaire) FROM employes;
```

- **`AVG`** : Calculer la moyenne d'une colonne.

```
sql
Copier le code
SELECT AVG(salaire) FROM employes;
```

- **`MAX`** et **`MIN`** : Trouver la valeur maximale ou minimale d'une colonne.

```
sql
Copier le code
SELECT MAX(salaire) FROM employes;
SELECT MIN(salaire) FROM employes;
```

- **`GROUP BY`** : Grouper les données et appliquer une fonction d'agrégation.

```
sql
```

```
Copier le code
SELECT departement, AVG(salaire) FROM employes GROUP BY departement;
```

2. Fonctions sur les chaînes de caractères

- **`CONCAT`** : Concaténer plusieurs chaînes de caractères.

```
sql
Copier le code
SELECT CONCAT(prenom, ' ', nom) AS nom_complet FROM employes;
```

- **`SUBSTRING`** : Extraire une sous-chaîne d'une chaîne.

```
sql
Copier le code
SELECT SUBSTRING(nom, 1, 3) AS initiales FROM employes;
```

- **`LOWER`** et **`UPPER`** : Convertir une chaîne en minuscules ou majuscules.

```
sql
Copier le code
SELECT LOWER(nom) AS nom_minuscule, UPPER(nom) AS nom_majuscule FROM
employes;
```

- **`TRIM`** : Supprimer les espaces en début et fin de chaîne.

```
sql
Copier le code
SELECT TRIM(nom) FROM employes;
```

3. Fonctions de date et d'heure

- **`CURRENT_DATE`** et **`CURRENT_TIMESTAMP`** : Obtenir la date ou l'heure actuelle.

```
sql
Copier le code
SELECT CURRENT_DATE() AS date_du_jour;
SELECT CURRENT_TIMESTAMP() AS heure_actuelle;
```

- **`DATEDIFF`** : Calculer la différence en jours entre deux dates.

```
sql
Copier le code
SELECT DATEDIFF(CURRENT_DATE(), date_naissance) AS age_en_jours FROM
employes;
```

- **`DATE_ADD`** et **`DATE_SUB`** : Ajouter ou soustraire des jours à une date.

```
sql
Copier le code
SELECT DATE_ADD(CURRENT_DATE(), 10) AS date_future;
SELECT DATE_SUB(CURRENT_DATE(), 10) AS date_passee;
```

- **`YEAR`**, **`MONTH`**, **`DAY`** : Extraire l'année, le mois ou le jour d'une date.

```
sql
Copier le code
SELECT YEAR(date_embauche) AS annee_embauche FROM employes;
```

4. Fonctions conditionnelles

- **`CASE`** : Créer des conditions dans une requête SQL.

```
sql
Copier le code
SELECT nom,
       CASE
           WHEN salaire > 50000 THEN 'Elevé'
           ELSE 'Bas'
       END AS niveau_salaire
FROM employes;
```

- **`COALESCE`** : Retourner la première valeur non nulle parmi plusieurs colonnes.

```
sql
Copier le code
SELECT COALESCE(prenom, 'Nom inconnu') AS prenom FROM employes;
```

5. Opérations sur les tables

- **`CREATE TABLE`** : Créer une nouvelle table.

```
sql
Copier le code
CREATE TABLE nouvelle_table (
    id INT,
    nom STRING,
    salaire DOUBLE
);
```

- **`INSERT INTO`** : Insérer des données dans une table existante.

```
sql
Copier le code
INSERT INTO nouvelle_table VALUES (1, 'John Doe', 50000);
```

- **`UPDATE`** : Mettre à jour les données dans une table.

```
sql
Copier le code
UPDATE employes SET salaire = salaire * 1.1 WHERE departement =
'Ventes';
```

- **`DELETE`** : Supprimer des lignes d'une table.

```
sql
Copier le code
DELETE FROM employes WHERE salaire < 30000;
```

- **`DROP TABLE`** : Supprimer une table.

```
sql
Copier le code
DROP TABLE IF EXISTS ancienne_table;
```

6. Jointures

- **`INNER JOIN`** : Joindre des tables sur une clé commune.

```
sql
Copier le code
SELECT e.nom, d.nom_departement
FROM employes e
INNER JOIN departements d ON e.departement_id = d.id;
```

- **`LEFT JOIN`** : Obtenir toutes les lignes de la table de gauche et les correspondances de la table de droite.

```
sql
Copier le code
SELECT e.nom, d.nom_departement
FROM employes e
LEFT JOIN departements d ON e.departement_id = d.id;
```

- **`RIGHT JOIN`** : Obtenir toutes les lignes de la table de droite et les correspondances de la table de gauche.

```
sql
Copier le code
SELECT e.nom, d.nom_departement
FROM employes e
RIGHT JOIN departements d ON e.departement_id = d.id;
```

- **`FULL OUTER JOIN`** : Combiner toutes les lignes des deux tables.

```
sql
Copier le code
SELECT e.nom, d.nom_departement
FROM employes e
FULL OUTER JOIN departements d ON e.departement_id = d.id;
```

7. Fonctions analytiques

- **`ROW_NUMBER`** : Attribuer un numéro de ligne à chaque enregistrement dans une partition.

```
sql
Copier le code
SELECT nom, salaire,
       ROW_NUMBER() OVER (PARTITION BY departement ORDER BY salaire
DESC) AS rang
FROM employes;
```

- **`RANK`** et **`DENSE_RANK`** : Attribuer un rang à chaque enregistrement dans une partition.

```
sql
```

```
Copier le code
SELECT nom, salaire,
       RANK() OVER (PARTITION BY departement ORDER BY salaire DESC)
AS rang
FROM employes;
```

- **`LAG` et `LEAD`** : Accéder aux valeurs de la ligne précédente ou suivante dans une partition.

```
sql
Copier le code
SELECT nom, salaire,
       LAG(salaire, 1) OVER (ORDER BY salaire) AS salaire_precedent
FROM employes;
```

8. Manipulation de données structurées

- **`PIVOT`** : Convertir les lignes en colonnes.

```
sql
Copier le code
SELECT * FROM (
    SELECT departement, sexe, salaire
    FROM employes
) PIVOT (
    AVG(salaire) FOR sexe IN ('Homme' AS salaire_homme, 'Femme' AS
salaire_femme)
);
```

- **`UNPIVOT`** : Convertir les colonnes en lignes.

```
sql
Copier le code
SELECT * FROM (
    SELECT id, salaire_homme, salaire_femme
    FROM employes_pivot
) UNPIVOT (
    salaire FOR sexe IN (salaire_homme AS 'Homme', salaire_femme AS
'Femme')
);
```

9. Fonctions de fenêtre

- **`WINDOW`** : Appliquer des fonctions sur un ensemble de lignes dans une fenêtre.

```
sql
Copier le code
SELECT nom, salaire,
       AVG(salaire) OVER (PARTITION BY departement ORDER BY salaire
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS moyenne_salaire
FROM employes;
```

10. Fonctions de manipulation des types de données

- **`CAST`** : Convertir une colonne d'un type de données à un autre.

```
sql
Copier le code
SELECT nom, CAST(salaire AS STRING) AS salaire_texte FROM employes;
```

- **`PARSE_DATE`** et **`PARSE_TIMESTAMP`** : Analyser une chaîne et la convertir en date ou en timestamp.

```
sql
Copier le code
SELECT PARSE_DATE('yyyy-MM-dd', '2024-08-05') AS date;
SELECT PARSE_TIMESTAMP('yyyy-MM-dd HH:mm:ss', '2024-08-05 12:34:56')
AS timestamp;
```

Ces fonctions SQL vous permettent de réaliser une large gamme d'opérations analytiques et de manipulation de données dans Databricks, en exploitant toute la puissance de SQL. Databricks supporte également des extensions spécifiques à Spark SQL, ce qui enrichit davantage les capacités de traitement de données.