# PySpark DataFrame Case Study with SQL

## Scenario:

You have a dataset of e-commerce transactions, and you want to analyze customer purchase patterns, total spending, product preferences, and various other insights using PySpark DataFrame operations and SQL queries. The dataset contains the following columns:  
- transaction\_id: Unique ID for each transaction  
- customer\_id: Unique ID for each customer  
- product\_id: Unique ID for each product  
- product\_name: Name of the product  
- category: Category of the product  
- price: Price of the product  
- quantity: Quantity purchased  
This case study will demonstrate the use of common PySpark DataFrame transformations and actions, as well as SQL queries.

## Sample Data:

1, 101, 5001, 'Laptop', 'Electronics', 1000.0, 1  
2, 102, 5002, 'Headphones', 'Electronics', 50.0, 2  
3, 101, 5003, 'Book', 'Books', 20.0, 3  
4, 103, 5004, 'Laptop', 'Electronics', 1000.0, 1  
5, 102, 5005, 'Chair', 'Furniture', 150.0, 1

## Step 1: Loading the Data into a DataFrame

from pyspark.sql import SparkSession  
  
# Initialize SparkSession  
spark = SparkSession.builder.appName("E-Commerce Analysis").getOrCreate()  
  
# Sample data  
data = [  
 (1, 101, 5001, 'Laptop', 'Electronics', 1000.0, 1),  
 (2, 102, 5002, 'Headphones', 'Electronics', 50.0, 2),  
 (3, 101, 5003, 'Book', 'Books', 20.0, 3),  
 (4, 103, 5004, 'Laptop', 'Electronics', 1000.0, 1),  
 (5, 102, 5005, 'Chair', 'Furniture', 150.0, 1)  
]  
  
columns = ["transaction\_id", "customer\_id", "product\_id", "product\_name", "category", "price", "quantity"]  
  
# Create DataFrame  
df = spark.createDataFrame(data, columns)  
df.show()

## Step 2: Using Filter Transformation

# Filter transactions where quantity is greater than 1  
df\_filtered = df.filter(df.quantity > 1)  
df\_filtered.show()

## Step 3: Handling Null Values

# Filling null values in price column with the average price  
average\_price = df.selectExpr("avg(price)").collect()[0][0]  
df\_filled = df.na.fill({"price": average\_price})  
df\_filled.show()

## Step 4: Dropping Duplicates

# Drop duplicate rows based on customer\_id and product\_id  
df\_no\_duplicates = df.dropDuplicates(["customer\_id", "product\_id"])  
df\_no\_duplicates.show()

## Step 5: Selecting Specific Columns

# Select specific columns  
df\_selected = df.select("customer\_id", "product\_name", "price")  
df\_selected.show()

## Step 6: Grouping and Aggregating Data

# Calculate the total spending per customer  
df\_grouped = df.groupBy("customer\_id").agg({"price": "sum"})  
df\_grouped.show()

## Step 7: Joining DataFrames

# Assume we have another DataFrame with customer details  
customer\_data = [  
 (101, "John Doe", "john@example.com"),  
 (102, "Jane Smith", "jane@example.com"),  
 (103, "Alice Johnson", "alice@example.com")  
]  
customer\_columns = ["customer\_id", "customer\_name", "email"]  
df\_customers = spark.createDataFrame(customer\_data, customer\_columns)  
  
# Join on customer\_id  
df\_joined = df.join(df\_customers, on="customer\_id", how="inner")  
df\_joined.show()

## Step 8: Union of Two DataFrames

# Create another DataFrame with similar schema  
new\_data = [  
 (6, 104, 5006, 'Table', 'Furniture', 200.0, 1)  
]  
df\_new = spark.createDataFrame(new\_data, columns)  
  
# Union the DataFrames  
df\_union = df.union(df\_new)  
df\_union.show()

## Step 9: Creating Temporary Views and Using SQL

# Create a temporary view  
df.createOrReplaceTempView("transactions")  
  
# Run SQL query  
sql\_result = spark.sql("SELECT customer\_id, SUM(price \* quantity) as total\_spent FROM transactions GROUP BY customer\_id")  
sql\_result.show()

## Summary of Transformations and Actions Used:

- `filter`: Filter rows based on a condition.  
- `na.fill`: Fill null values.  
- `dropDuplicates`: Remove duplicate rows based on specific columns.  
- `select`: Select specific columns.  
- `groupBy` and `agg`: Group data and perform aggregation.  
- `join`: Join two DataFrames on a key.  
- `union`: Combine two DataFrames with the same schema.  
- `createOrReplaceTempView` and `spark.sql`: Create a temporary view and execute SQL queries on the DataFrame.