SQL + PYSPARK ASSIGNMENT

Use your SQL database to perform the following:

1. Create the following tables:
    - `orders(order_id, cust_id, amount, order_date, region)`
    - `customers(cust_id, name, gender, age)`
    - `products(product_id, category, price)`
2. Insert at least **5 records** in each table.
3. Write a SQL query to get the **total order amount** per region.
4. Write a SQL query to get the **average customer age** per region (join `orders` and `customers`).
5. Get a list of customers who have placed **more than one order**.
6. Get the **top customer (by amount)** from each region using SQL.
7. Export all 3 tables (`orders`, `customers`, `products`) to CSV files.

Use the exported CSVs to complete the following in PySpark:

8. Read the `orders.csv`, `customers.csv`, and `products.csv` using `.read.csv()` with header and schema inference.
9. Join `orders` with `customers` on `cust_id`. What type of join did you use?
10. Add a new column called `high_value`:

- Value = `"Yes"` if `amount > 5000`, else `"No"`.

11. Perform `groupBy()` on `region` and `gender`:

- Count total orders
- Sum total revenue

12. Using `row_number()`, find the **top customer by amount** in each region.
13. Write the final DataFrame to Parquet, **partitioned by `region`**.
14. Use `.explain()` to print the **execution plan** for both the aggregation and window operations.

Functional Practice

15. Use `select()` to extract only `cust_id`, `name`, `region`, and `high_value`.
16. Filter the data to show only rows where `high_value = 'Yes'`.
17. Sort the final DataFrame by `amount` in descending order.
18. Drop duplicate `order_id` entries using `dropDuplicates()`.
19. Rename the column `amount` to `order_amount` using `withColumnRenamed()`.
20. Use `when()` and `withColumn()` to tag customers as:

- `"Youth"` if age < 30
- `"Adult"` if age between 30–59
- `"Senior"` if age ≥ 60

21. Define a **UDF** to classify customers by loyalty:

- `"Loyal"` if the customer appears more than once in the dataset
- `"New"` otherwise

Conceptual Questions

22. What is the difference between **transformations** and **actions** in PySpark?

23. What's the benefit of **lazy evaluation** in Spark?

24. List and briefly explain at least **4 types of joins** in PySpark.

25. How does a **window function** differ from a groupBy?

26. When should you use a **UDF**, and when should you avoid it?

27. What is the purpose of `.partitionBy()` while writing files?

28. Why is **Parquet** format preferred over CSV in Spark pipelines?

29. What does `.explain()` output show, and why is it useful?

30. How is **PySpark** different from regular Python when handling large data?