Data Engineer – Proposal Interview TD

01

02

03

Data Pipeline Design

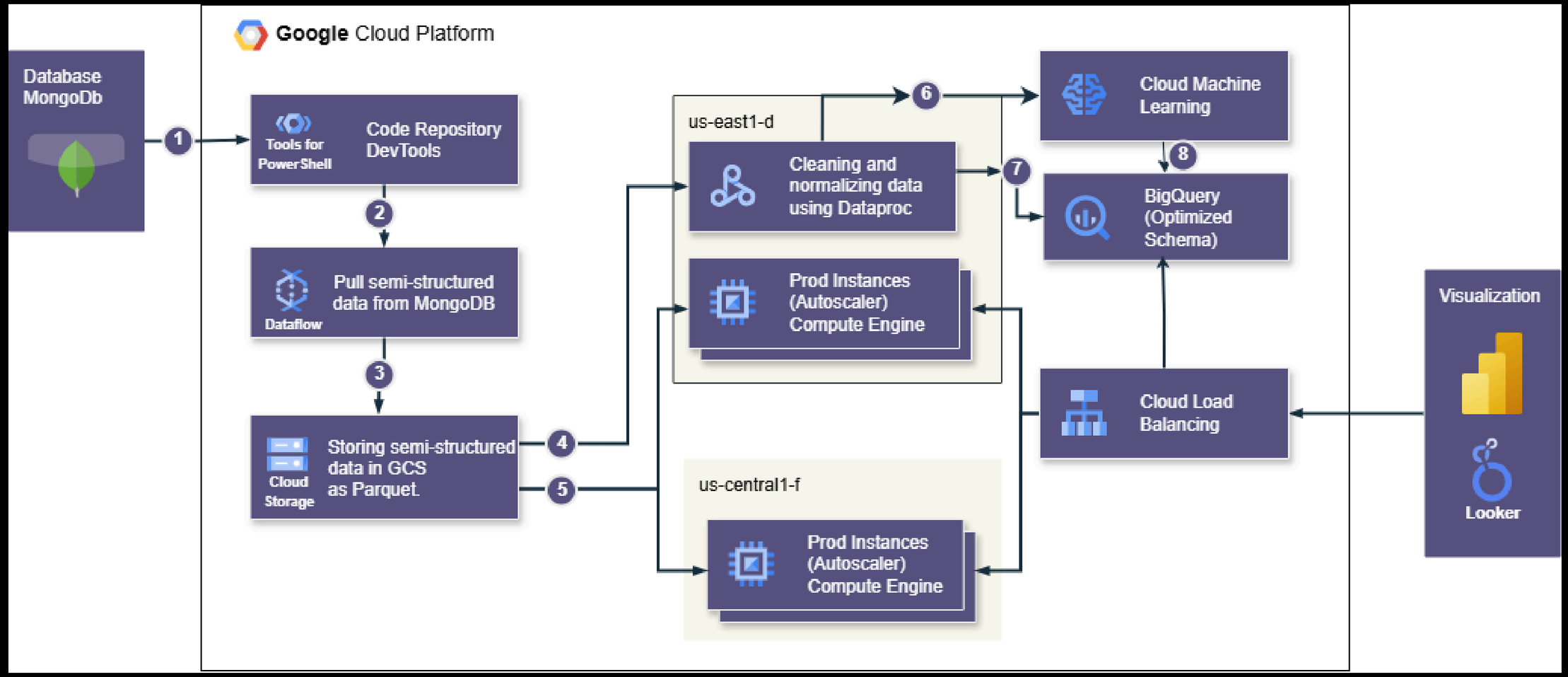
We would like to know your thought process and how you understand & handle the high level thinking around data.

01

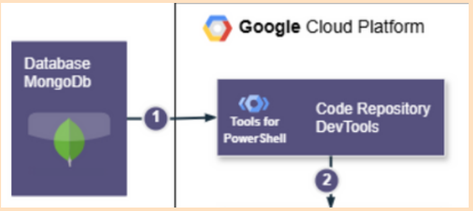
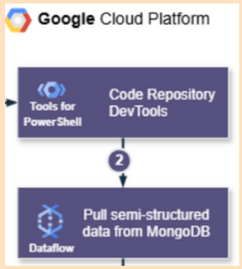
02

03

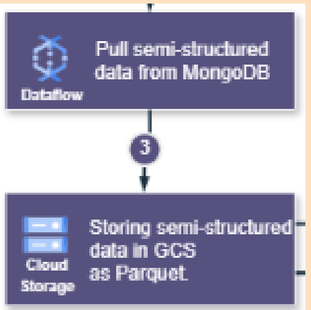
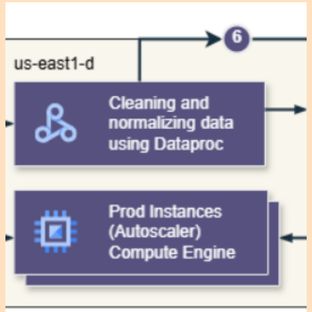
Cloud Architecture



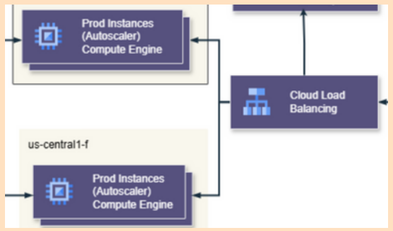
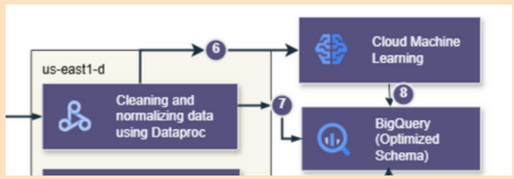
Process & Solution Table

Feature	Process	GCP Service
1.Extract	• PowerShell or automated scripts fetch semi-structured,denormalized data from MongoDB. • Code is stored and versioned in a code repository. • The script imports the following libraries `#Assumtion codeing` `import apache_beam as beam` `from apache_beam.options.pipeline_options import PipelineOptions` `import pymongo` `import json` `import pandas as pd` `import pyarrow.parquet as pq` `from google.cloud import storage, bigquery` `from pyspark.sql import SparkSession` `from pyspark.sql.functions import col` `# MongoDB Connection` `MONGO_URI = "mongodb://your_mongo_uri_path"` `MONGO_DB = "your_database"` `MONGO_COLLECTION = "your_collection"`	PowerShell, DevTools 
2.Ingest Data	• Extract semi-structured JSON data from MongoDB. • Batch or streaming ingestion into Google Cloud. • Handles schema mapping, deduplication, and basic transformations. `# GCP Bucket & BigQuery Config` `GCS_BUCKET = "your-bucket-name_path"` `GCS_PARQUET_PATH = f"gs://{GCS_BUCKET}/data.parquet"` `BIGQUERY_DATASET = "your_dataset"` `BIGQUERY_TABLE = "your_table"` `# Step 1: Extract - Read data from MongoDB` `def read_from_mongo():` `client = pymongo.MongoClient(MONGO_URI)` `db = client[MONGO_DB]` `collection = db[MONGO_COLLECTION]` `return list(collection.find({}))`	Cloud Dataflow (Apache Beam) 

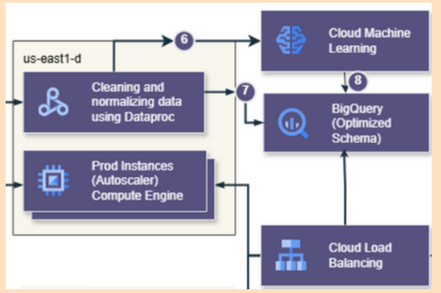
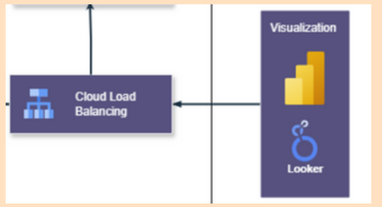
Process & Solution Table

Feature	Process	GCP Service
3.Store Raw Data	• The extracted data is stored in GCS as Parquet format (optimized for analytics). • This allows efficient querying & downstream processing. ``` # Step 2: Transform - Convert to DataFrame & Store as Parquet def transform_and_store_parquet(data): df = pd.DataFrame(data) table = pa.Table.from_pandas(df) pq.write_table(table, "/tmp/data.parquet") # Upload to GCS storage_client = storage.Client() bucket = storage_client.bucket(GCS_BUCKET) blob = bucket.blob("data.parquet") blob.upload_from_filename("/tmp/data.parquet") ```	Cloud Storage (Parquet) 
4.Transform Data	• Cleans & structures raw data (handling duplicates, missing values, normalization). • Runs data transformations (flattening nested JSON, applying business logic). • Prepares data for BigQuery. ``` # Step 3: Load - Process Data using Dataproc & Load to BigQuery def clean_data_with_spark(): spark = SparkSession.builder.appName("Data Cleaning").getOrCreate() df = spark.read.parquet(GCS_PARQUET_PATH) # Cleaning Steps cleaned_df = df.select(col("field1").alias("new_field1"), col("field2").cast("integer"), col("timestamp").cast("timestamp")) # Save cleaned data back to GCS cleaned_df.write.mode("overwrite").parquet(f"gs://{GCS_BUCKET}/cleaned_data.parquet") ```	Cloud Dataproc (Spark) 

Process & Solution Table

Feature	Process	GCP Service
5.Compute Resources	• Compute instances dynamically scale up/down based on the processing workload. • Supports parallel execution of transformation jobs. `# Run the Pipeline` `if __name__ == "__main__":` `  data = read_from_mongo()` `  transform_and_store_parquet(data)` `  clean_data_with_spark()` `  load_to_bigquery()`	Compute Engine (Autoscaler) 
6.Machine Learning (Optional)	• Prepares data for AI/ML model training. • Generates predictions or recommendations for analytics.	Cloud Machine Learning 

Process & Solution Table

Feature	Process	GCP Service
7. Load to BigQuery	• Writes structured, partitioned, clustered tables for fast querying. • Business-friendly schema for SQL-based users. • Enables advanced analytics & reporting. `def load_to_bigquery():` `  client = bigquery.Client()` `  dataset_ref = client.dataset(BIGQUERY_DATASET)` `  table_ref = dataset_ref.table(BIGQUERY_TABLE)` `  job_config = bigquery.LoadJobConfig(source_format=bigquery.SourceFormat.PARQUET)` `  uri = f"gs://{GCS_BUCKET}/cleaned_data.parquet"` `  load_job = client.load_table_from_uri(uri, table_ref, job_config=job_config)` `  load_job.result()`	BigQuery 
8.Visualization	• Connects BigQuery data to Looker dashboards. • Uses Cloud Load Balancing for performance optimization. • Enables data-driven decision-making.	Looker, Cloud Load Balancer 

Text Sanitizer

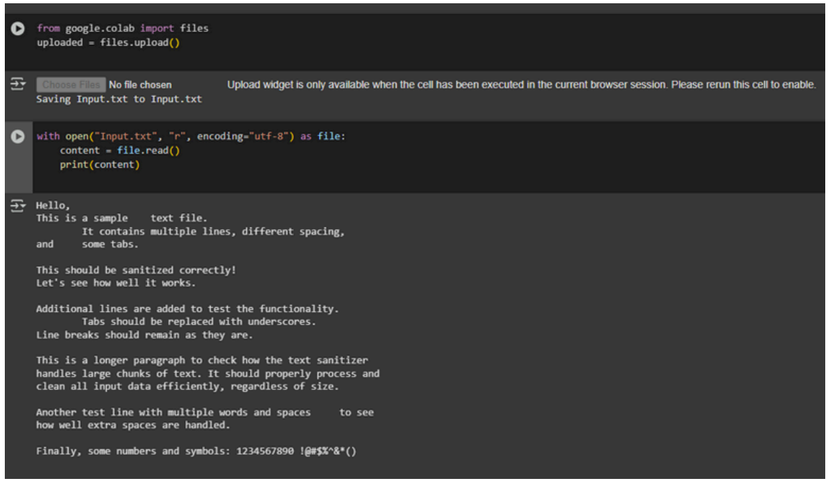
We would like to know how you design the application and how you implement the code based on the given requirements.

01

02

03

Process & Solution Table

Feature	Process	Solution
Upload Input.txt	Step 1: Import necessary libraries Step 2: Read file contents • Open the uploaded file in read mode ("r"), ensuring encoding is set as "utf-8". • Read and print the content to verify the input.	
Sanitize text	Step 3: Sanitize text • Initialization • Create an instance of the TextSanitizer class. Step 4: Read Text (read_file) • Open the provided file and read its content. • Store the text content into an internal variable.	

```
from google.colab import files
uploaded = files.upload()
```

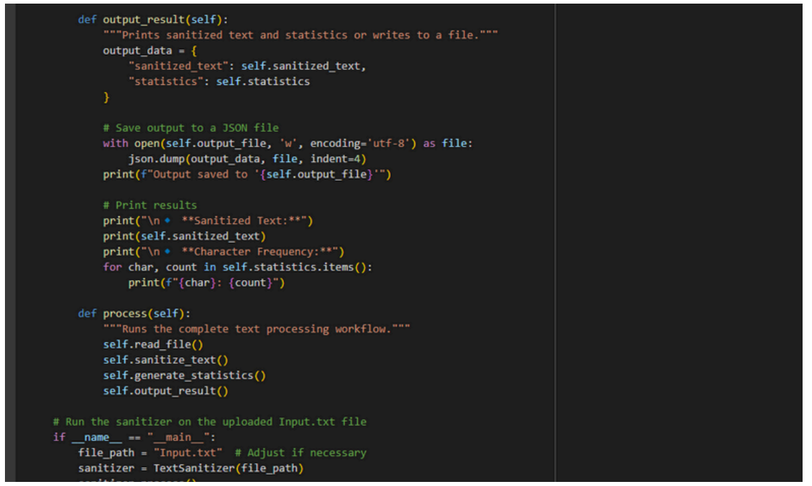
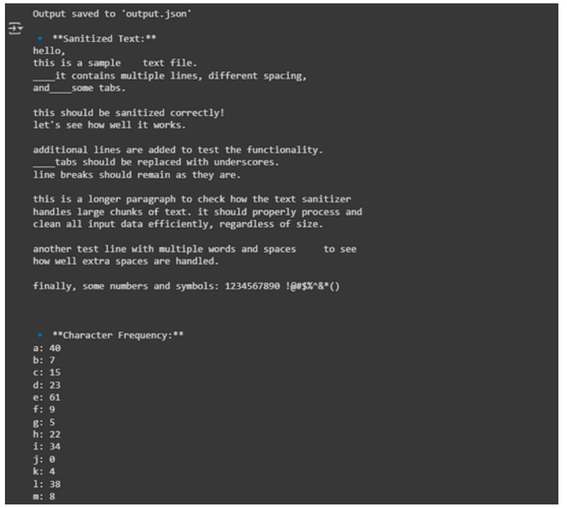
```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Input.txt to Input.txt
```

```
with open("Input.txt", "r", encoding="utf-8") as file:
    content = file.read()
    print(content)
```

```
Hello,
This is a sample    text file.
        It contains multiple lines, different spacing,
and     some tabs.

This should be sanitized correctly!
Let's see how well it works.

Additional lines are added to test the functionality.
        Tabs should be replaced with underscores.
Line breaks should remain as they are.

This is a longer paragraph to check how the text sanitizer
handles large chunks of text. It should properly process and
clean all input data efficiently, regardless of size.

Another test line with multiple words and spaces     to see
how well extra spaces are handled.

Finally, some numbers and symbols: 1234567890 !@#$%^&*()
```

```
import string
import json

class TextSanitizer:
    """Class for reading, sanitizing, and analyzing text."""

    def __init__(self, file_path, output_file="output.json"):
        """
        Initialize the TextSanitizer with a file path.

        :param file_path: Path to the input text file.
        :param output_file: Path to the output JSON file.
        """
        self.file_path = file_path
        self.output_file = output_file
        self.text = ""
        self.sanitized_text = ""
        self.statistics = {}

    def read_file(self):
        """Reads input text from a file."""
        try:
            with open(self.file_path, 'r', encoding='utf-8') as file:
                self.text = file.read()
        except FileNotFoundError:
            print(f"Error: File '{self.file_path}' not found.")
            exit(1)

    def sanitize_text(self):
        """Sanitizes the text by converting it to lowercase and replacing tabs."""
        self.sanitized_text = self.text.lower().replace('\t', '___')

    def generate_statistics(self):
```

Process & Solution Table

Feature	Process	Solution
Sanitize text	Step 5: Sanitize Text • Replace tabs (\t) with underscores (_). • Convert text to lowercase (implied by counting only lowercase characters later). • Retain line breaks as they are. Step 6: Analyze Text • Calculate the frequency of each alphabetic character (a to z). • Store these counts in a dictionary. Step 7: Output Results • Save sanitized text and character frequency statistics into a JSON-formatted output file (output.json by default). • Print sanitized text to the console. • Print frequency of each alphabet character.	 

```
    def output_result(self):
        """Prints sanitized text and statistics or writes to a file."""
        output_data = {
            "sanitized_text": self.sanitized_text,
            "statistics": self.statistics
        }

        # Save output to a JSON file
        with open(self.output_file, 'w', encoding='utf-8') as file:
            json.dump(output_data, file, indent=4)
        print(f"Output saved to '{self.output_file}'")

        # Print results
        print("\n🔹 **Sanitized Text:**")
        print(self.sanitized_text)
        print("\n🔹 **Character Frequency:**")
        for char, count in self.statistics.items():
            print(f"{char}: {count}")

    def process(self):
        """Runs the complete text processing workflow."""
        self.read_file()
        self.sanitize_text()
        self.generate_statistics()
        self.output_result()

# Run the sanitizer on the uploaded Input.txt file
if __name__ == "__main__":
    file_path = "Input.txt"  # Adjust if necessary
    sanitizer = TextSanitizer(file_path)
```

```
Output saved to 'output.json'

🔹 **Sanitized Text:**
hello,
this is a sample    text file.
____it contains multiple lines, different spacing,
and____some tabs.

this should be sanitized correctly!
let's see how well it works.

additional lines are added to test the functionality.
____tabs should be replaced with underscores.
line breaks should remain as they are.

this is a longer paragraph to check how the text sanitizer
handles large chunks of text. it should properly process and
clean all input data efficiently, regardless of size.

another test line with multiple words and spaces     to see
how well extra spaces are handled.

finally, some numbers and symbols: 1234567890 !@#$%^&*()



🔹 **Character Frequency:**
a: 40
b: 7
c: 15
d: 23
e: 61
f: 9
g: 5
h: 22
i: 34
j: 0
k: 4
l: 38
m: 8
```

SQL

We would like to know your thought process and how you implement SQL based on the given requirements.

01

02

03

Process & Solution Table

Feature	Process	Solution
SQL **Step 1: Aggregate Sales Data**	• Retrieve data from three tables: ◦ Sales_Transaction – transaction records with details like quantities and retail prices. ◦ Product – provides information on product IDs, product names, and class IDs. ◦ Product_class – includes details about product class IDs and their names. • Join these tables using the product IDs and transaction IDs to connect transactions with product details, creating a comprehensive dataset containing each sale's product and class information.	e Blame 37 lines (37 loc) · 985 Bytes 1 WITH SalesData AS (2 SELECT 3 p.product_id, 4 p.product_name, 5 pc.product_class_id, 6 pc.product_class_name, 7 SUM(s.quantity) AS total_quantity, 8 SUM(s.quantity * s.retail_price) AS sales_value 9 FROM Sales_Transaction s 10 JOIN Product p ON s.product_id = p.product_id 11 JOIN Product_Class pc ON p.product_class_id = pc.product_class_id 12 GROUP BY p.product_id, p.product_name, pc.product_class_id, pc.product_class_name **Sales Transaction** transaction_id | product_id | quantity 1 | 1 | 5 1 | 2 | 7 2 | 3 | 1 3 | 2 | 3 .. | .. | ..

Process & Solution Table

Feature	Process	Solution
SQL **Step 2: Rank Products within Each Product Class**	• Calculate key metrics for each product: ◦ Total Quantity Sold (SUM(s.quantity)): the total units sold for each product. ◦ Total Sales Value (SUM(s.quantity * s.retail_price)): the total revenue generated by each product. • Group these aggregations by product details (product_id, product_name) and class details (product_class_id, product_class_name), producing a summarized view of sales performance per product.	15 SELECT 16 product_class_id, 17 product_class_name, 18 product_id, 19 product_name, 20 total_quantity, 21 sales_value, 22 RANK() OVER (23 PARTITION BY product_class_id 24 ORDER BY sales_value DESC, total_quantity ASC 25) AS rank 26 FROM SalesData 27)

Product

product_id	product_name	retail_price	product_class_id
1	aa	10	1
2	bb	20	1
3	cc	30	2
..	..	..	..

Product Class

product_class_id	product_class_name
1	Class A
2	Class B
3	Class C
..	..

Process & Solution Table

Feature	Process	Solution
SQL **Step 3: Filter Top 2 Ranked Products**	• Apply a ranking algorithm (RANK()) partitioned by product_class_id. • Within each product class, products are sorted first by sales value (sales_value DESC) to prioritize products generating the highest revenue. • If two products have the same sales value, the secondary criterion of total quantity (total_quantity ASC) is used to break ties, ranking products with fewer quantities sold higher (a typical business decision might be prioritizing high-value sales even if lower in quantity).	

```
28    SELECT
29        product_class_name,
30        product_id,
31        product_name,
32        total_quantity,
33        sales_value,
34        rank
35    FROM RankedSales
36    WHERE rank <= 2
37    ORDER BY product_class_id, rank;
```

Expected Output

product_class_name	rank	product_name	sal
Class A	1	aa	1234
Class A	2	bb	9999
Class B	1	cc	2500
Class B	2	dd	2500
..	..	..	..

Thank You

FOR YOUR ATTENTION

01

02

03