About:

Driven and detail-oriented Data Engineer with 2 years of experience in designing, building, and optimizing scalable data pipelines and architectures. Adept at transforming complex data requirements into efficient ETL processes, ensuring high data quality, and supporting data-driven decision-making. Seeking to leverage my technical skills and experience in a dynamic environment to enhance data infrastructure and contribute to impactful business outcomes.

Experience:

Experience in building data pipelines using AWS services such as EC2, ECS, Glue, Lambda and Databricks.

- Developed ETL workflows using PySpark in Glue / Databricks to transform, validate, and load large amounts of data from various sources to the AWS data lake.

- Analyzed the SQL scripts and designed it by using PySpark SQL for faster performance.

- Expertise in designing and implementing scalable data architectures in AWS, including data modelling and database design using technologies like Redshift, RDS and Snowflake.

- Develop programs in PySpark and Python as part of data cleaning and processing.

- Optimize and fine-tune data models, SQL queries, and transformations for performance and scalability.

- Experience with Amazon SQS for scalable decoupling of components in distributed systems, enabling asynchronous operations and improved fault tolerance in data pipelines.

- Experience using Amazon SNS for pub/sub messaging to coordinate and notify components in data pipelines,

- Develop Glue Jobs to read CSV data and write to Parquet format.

- Enabling event-driven execution and orchestration of data projects.

- Working with JIRA for bug tracking and communicating for better results.

-Possess good problem-solving & interpersonal skills, having capability to work alone

1. AWS CodeBuild and CodeDeploy
2. Key points of PySpark and Spark architecture
3. Default partition size in Spark
4. How to read a file using DataFrame Reader API when the headers are not given?
5. How to create a Spark table on top of a DataFrame?
6. A query to join two tables using a LEFT OUTER JOIN
7. What is the use of WITH in CTE (Common Table Expressions)?
8. Difference between list and tuple
9. How to iterate and get values in a dictionary?
10. How to add a value at the end of a list?
11. How to add values from the beginning of a list?
12. Limitations of AWS Lambda
13. Uses of CodePipeline
14. Glue catalog and Glue crawler
15. CloudFormation Template (CFT) in AWS
16. Tell me about yourself, your projects, and your roles and responsibilities
17. What is workflow management in Redshift?
18. Redshift Spectrum vs External Table
19. QuickSight vs Athena in visualization
20. Redshift Spectrum vs Athena in data retrieval from S3
21. How to store information in an S3 bucket from a PySpark job?
22. How to read data from a Hive table and load it into an S3 bucket?
23. How to access a Hive table in a Glue job?
24. Overview of PySpark in terms of data transformation

25. What is Databricks?
26. Main components of Databricks
27. How does Apache Spark work in Databricks?
28. Catalyst optimizer in Spark
29. How do you manage Spark applications in a Databricks cluster?
30. How to create a notebook in Databricks?
31. Key features of a Databricks notebook
32. How do you perform ETL operations in Databricks?
33. How do you use Databricks in managing data pipelines?
34. Databricks runtime
35. Security in AWS
36. How to optimize PySpark jobs?
37. How to handle null values in DataFrames?
38. Schema evaluation in Databricks
39. Advantages of Databricks compared to traditional data processing engines
40. How do you handle data transformations from an external system (RDBMS)?
41. Types of joins in Spark
42. How to handle out-of-memory errors in Spark?
43. Why do we get analysis exception errors in Spark?
44. Key configurations in designing data pipelines
45. How to handle data quality issues in data pipelines?
46. Fault tolerance in Spark
47. How can transformation be done using AWS Glue?
48. AWS Data Pipeline's role in orchestration of data workflows
49. How do you handle data duplication?
50. Does Glue support data versioning in ETL jobs?
51. Write a PySpark code to generate 80 distinct values and assign branch names
52. Write a Python code to reverse a string without using slicing
53. How to write SQL queries for performance tuning?

About:

As a data engineer, I design and execute data pipelines using Python, AWS, SQL, Spark, Power BI, Azure, GCP, Snowflake, Apache Spark, Apache Airflow and other tools and technologies. I have successfully delivered data solutions for projects in the various domains, extracting, processing, and structuring unstructured data from multiple sources, applying business logic and NLP techniques, and creating data visualizations to interpret data insights.

I graduated with a Bachelor of Technology degree in Mechanical Engineering from Amrita School of Engineering Bangalore in 2021, where I was also the president of the Astrophysics Club and have won the third prize at the International Space Development Conference in 2016. I am an enthusiastic developer with a passion for learning new skills and technologies, and a motivation to contribute to the development and success of esteemed organizations.

Skill Set :- Python, SQL, AWS, Azure, Snowflake, Data Structures and Algorithms, OOP, Excel, Spark, Git, Jenkins, Microsoft Power Bi, Tableau, Apache Airflow, Hadoop, Hive, Data Warehousing, Data Modelling, GCP, Scala, MySQL, PostgreSQL, ETL, Big Data, Data Marts, NoSQL, Big Query, Data Bricks, Delta Lake, Data Lake, Glue Studio.

Nielsen Data Engineer Interview Questions with Solutions

1. How to set up an EMR cluster to install a JAR file?

Steps:

- Upload the JAR file to an S3 bucket.

- Create a shell script to download the JAR from S3 to the instance and upload the script to S3.

- Configure a Bootstrap Action in EMR using the S3 path of the shell script. This ensures the script runs during cluster initialization to download the JAR.

2. How to configure Spark on EMR to read data from an S3 bucket encrypted with AWS KMS?

- Grant the EMR instance profile permissions for both S3 objects and the KMS key.

3. How to restrict S3 bucket access to a specific EC2 instance while following the principle of least privilege?

- Attach an Elastic IP to the EC2 instance and set an S3 bucket policy that limits access to the specific IP.

4. Purpose of AWS KMS (Key Management Service) in data engineering?

- AWS KMS provides key management, encryption, and access control across AWS services, ensuring data security and compliance.

5. How to make an EMR cluster resilient for large data processing?

- Approach:

- Use high-performance instance types (e.g., m5.xlarge).

- Enable auto-scaling to dynamically adjust resources based on workload.

- Store data in S3 for scalability and decoupling storage from compute.

- Optimize configurations for Spark and Hadoop, such as increasing partitions for parallel processing and tuning memory settings.

6. What are EMR Instance Fleets, and how do they help?

- They allow multiple instance types and purchasing options within a cluster, enhancing resilience and cost management.

7. Difference Between OLAP and OLTP with Examples:

- OLTP (Online Transaction Processing):

- Use Case: Designed for managing transactional data (e.g., banking systems, online shopping).

- Example: A retail transaction system where every sale, return, or inquiry is processed immediately.

- OLAP (Online Analytical Processing):

- Use Case: Designed for data analysis and complex queries (e.g., business reporting, data mining).

- Example: A sales data analysis system that aggregates data across multiple dimensions (time, region, product).

8. Difference Between Data Lake and Data Warehouse with Examples:

- Data Lake:

- Stores raw, unstructured data (e.g., S3 for logs).

- Data Warehouse:

- Stores processed, structured data for reporting (e.g., Redshift for sales data).

9. Example of Delta Lake (Data Lake + Warehouse):

- Combines data lake and warehouse features, providing ACID transactions and scalable metadata (e.g., Databricks for real-time and historical data processing).

EY Data Engineer Interview Questions with Solutions

1. What is the difference between wide and narrow transformations in Spark?

- Narrow Transformations: Only require data from one partition (e.g., 'map', 'filter'). Data movement is limited.

- Wide Transformations: Require data shuffling across partitions (e.g., 'groupByKey', 'reduceByKey'), which involves a full shuffle across the network.

2. How would you perform an anti join in Spark?

- Use the 'left_anti' join to find rows in the left DataFrame that don't have a match in the right DataFrame:

```
df1.join(df2, on='column', how='left_anti')
```

3. Can you explain what a semi join is in Spark?

- A semi join returns all rows from the left DataFrame that have a match in the right DataFrame without duplicating columns from the right:

```
df1.join(df2, on='column', how='left_semi')
```

4. What are the types of anti joins?

- The main type in Spark is the 'left_anti' join, which excludes rows that have matching keys in the right DataFrame.

5. What is the difference between cache and checkpoint in Spark?

- Cache: Stores data in memory for quick access. Used for iterative algorithms where data is reused.

- Checkpoint: Saves data to a reliable storage (e.g., HDFS) and truncates the lineage graph, used for fault tolerance in long lineage operations.

6. What is the purpose of a DAG in Spark?

- A Directed Acyclic Graph (DAG) represents a sequence of computations and stages in Spark, helping to optimize the execution plan by determining the best order of operations.

7. How does speculative execution work in Spark?

- Speculative execution reduces stragglers by launching duplicate tasks on different nodes. The fastest task completes first, and the rest are canceled.

8. Where is data stored when caching in Spark?

- Data is stored in memory by default. If memory is insufficient, it spills over to disk.

9. What are the main components of Spark architecture?

- Driver: Coordinates tasks on the cluster.

- Executors: Run tasks assigned by the driver.

- Cluster Manager: Manages resources (e.g., YARN, Mesos, Standalone).

10. Once you cache a DataFrame, can you uncache it? How?

- Yes, using 'df.unpersist()', which removes the DataFrame from memory/disk.

11. What is the significance of removing cached data in Spark?

- Freeing up memory for other processes, preventing memory bottlenecks.

12. What happens if you forget to uncache data in Spark? How does it affect performance?

- Leaving data cached unnecessarily can lead to memory overflow, causing disk spills and degraded performance.

13. Which operator in PySpark can you use to check if two DataFrames are the same? What is a fast and specific function for this?

- Use 'df1.subtract(df2).isEmpty()' to quickly check if two DataFrames are identical.

Why Snowflake is a Game-Changer for Data Engineering Interviews ❄️

Prepping for a Data Engineering interview? Snowflake often comes up as a hot topic, and for good reason! Its advanced features and cloud-native architecture set it apart from traditional data warehouses.

1. **ELT Over ETL: Simplifying Data Pipelines**
Snowflake promotes the ELT approach—load raw data into Snowflake first, then transform using its powerful SQL engine. This method:
- **Reduces Complexity**: No need for complex pre-processing outside the warehouse.
- **Scales Easily**: Elastic compute resources handle transformations without performance issues.

Interview Tip: Be ready to explain why ELT is more efficient than ETL, especially in cloud-native environments.

2. **Advanced Features That Set Snowflake Apart**
Snowflake isn't just another data warehouse; it's packed with features that make it a top choice:
- **Time Travel**: Access historical data effortlessly, perfect for undoing mistakes or comparing changes over time.
- **Zero-Copy Cloning**: Instantly duplicate databases, schemas, or tables without extra storage costs—ideal for testing.
- **Data Sharing**: Share data across accounts or externally without making copies.
- **Automatic Scaling**: Multi-cluster architecture scales compute resources automatically based on workload.

Interview Tip: Highlight Snowflake's ability to handle both structured and semi-structured data (JSON, Avro, Parquet) natively—often a strong differentiator.

3. **Snowflake vs. Other Data Warehouses**
Snowflake's unique architecture separates compute and storage, allowing independent scaling:
- **Consistent Performance**: Handles high concurrency without slowing down.
- **Cost Efficiency**: Pay only for what you use, with separate billing for storage and compute.

Compared to Redshift or BigQuery, Snowflake's ease of use and performance make it the go-to choice for modern data pipelines.

Interview Tip: Be prepared to discuss scenarios where features like Time Travel or Zero-Copy Cloning can solve business problems or streamline workflows.

4. **Real-World Application: Why Snowflake?**

Snowflake simplifies data management with stored procedures, views, and robust data modeling. It’s not just about loading data—it's about turning data into actionable insights.

Evoke Technologies Data Engineer Interview Questions with Solutions

1. Can you explain a recent project you worked on in detail?

I built a data pipeline using AWS and PySpark to process large customer datasets for behavioral analysis. PySpark was used for data cleaning and transformation, with results stored in AWS S3. AWS Lambda automated job execution, reducing processing time significantly.

2. How would you extract a list of emails from a SAS dataset using Python?

Use the 'saspy' library to connect to SAS, load the data into a Pandas DataFrame, and use regex to extract email addresses.

3. How can you run a Python script on AWS Lambda that involves heavy data processing?

Package the script and dependencies into a ZIP file, upload to AWS Lambda, set the correct handler function, and ensure Lambda has permissions to access required data.

4. What’s the difference between Pandas DataFrame and PySpark DataFrame?

- Pandas: Best for small datasets; easy to use but not suited for large-scale data.
- PySpark: Built for big data, works well with large datasets across distributed systems.

5. How do you handle errors in a DAG in Apache Airflow?

Use task retries, failure notifications, and set rules for next steps if a task fails, such as sending alerts or pausing the pipeline.

6. How would you find departments that have no employees?

```
SELECT Dep.depid, Dep.depname
FROM Dep
LEFT JOIN Emp ON Dep.depid = Emp.depid
WHERE Emp.empid IS NULL;
```

7. How do you optimize a PySpark job for performance?

Use DataFrames, partition data correctly, cache reused data, and set an optimal number of partitions. Avoid unnecessary user-defined functions and use built-in Spark functions.

8. Explain how you would handle schema evolution in Parquet files stored in AWS S3.

Use AWS Glue to manage schema changes, and ensure Spark jobs can handle new or missing fields to read updated data formats.

9. How would you set up a data pipeline in AWS Glue?

Create a Glue job using PySpark or Python, define data sources (like S3, RDS), transform data using Spark SQL, and load it into targets like S3 or a data warehouse. AWS Glue crawlers detect schemas and catalog data automatically.

Virtusa Data Engineer Interview Questions with Solutions

(Preparing for AWS Data Engineer role at Virtusa? Here are key questions from my friend's interview experience!)

1. What is the role of Apache Kafka in Data Engineering?
Apache Kafka is a distributed streaming platform for real-time data pipelines and applications. It enables low-latency messaging between data producers and consumers.

2. How to secure data in AWS S3?
Secure S3 data with encryption (SSE-S3, SSE-KMS), proper bucket policies, ACLs, IAM roles, and VPC endpoints for private access.

3. What is ETL, and how does it differ from ELT?
ETL (Extract, Transform, Load) transforms data before loading into the target. ELT loads raw data first, then transforms it, suitable for modern data lakes.

4. How would you handle a large-scale data migration in AWS?
Use AWS DMS for migrations and AWS Snowball or Snowmobile for large datasets. AWS Step Functions can automate and orchestrate the migration steps.

5. How to perform incremental data loading in Spark?
Use timestamps or incremental columns to fetch new data. Structured streaming with watermarking helps manage state and discard old data.

6. Explain Data Partitioning in Apache Spark
Partitioning divides data into chunks for parallel processing, optimizing tasks like joins and aggregations by grouping related data.

7. Difference between DataFrame and Dataset in PySpark
DataFrames are like SQL tables, while Datasets offer type safety and use encoders, balancing performance and ease of use.

8. Explain about PySpark SQL and its use cases
PySpark SQL processes structured data, executes SQL queries, reads data sources, and integrates with BI tools in data processing pipelines.

9. How to handle schema evolution in AWS Glue?
AWS Glue manages schema evolution via the Data Catalog, automatically updating schemas with Glue crawlers.

10. Common file formats in Big Data and why?
Parquet, ORC, Avro, and JSON. Parquet and ORC are efficient for analytical queries, Avro is great for write-heavy, row-based storage.

11. How to handle NULL values in SQL
Use 'COALESCE()', 'IFNULL()', 'NVL()', or 'CASE WHEN' to substitute NULLs with default values in queries.

12. SQL query to find the second highest salary

```
SELECT MAX(salary) AS Second_Highest_Salary
FROM employee
WHERE salary < (SELECT MAX(salary) FROM employee);
```

13. Significance of Data Lakes in modern data architectures
Data lakes store all types of data at any scale, support multiple processing frameworks, and provide a unified repository for analytics and ML.

Nielsen Data Engineer Interview Questions with Solutions

(Prepping for an AWS Data Engineer role? Here are key questions from my personal interview experiences that you might encounter!)

1. Infrastructure as Code (IaC) in AWS
IaC lets you manage infrastructure using code. AWS services like CloudFormation and Terraform automate provisioning, making deployments consistent.

2. How to create and deploy AWS Lambda function using CLI? What is Runtime of Lambda?
Use AWS CLI commands like 'aws lambda create-function' to deploy functions. Lambda runtime specifies the environment (e.g., Python, Node.js) where the code runs.

3. What is Data Smoothing?
Data Smoothing removes noise from data, making trends clearer. Common methods include moving averages and exponential smoothing.

4. Explain the difference between 'is' and '==' in Python
'is' checks if two variables point to the same object in memory, while '==' checks if their values are equal.

5. How to delete duplicate elements from a list?

```
my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(set(my_list))
print(unique_list)
```

7. Explain Spark Context and Streaming Context
Spark Context is the entry point for any Spark app, connecting to the cluster. Streaming Context is used in Spark Streaming for real-time data.

8. Explain About RDDs
RDDs (Resilient Distributed Datasets) are the core data structure in Spark, providing fault tolerance, parallel processing, and transformations.

9. Difference between Map and Reduce
Map applies a function to each dataset element, creating a new dataset. Reduce aggregates by combining elements into a single output.

10. Different types of Joins in PySpark
- Left Semi Join: Returns rows from the left DataFrame with a match in the right.
- Left Anti Join: Returns rows from the left DataFrame without a match in the right.
- Broadcast Join: Joins a small DataFrame with a larger one by broadcasting the smaller DataFrame to all nodes.
- Sort-Merge Join: Used for large datasets with sorted keys, efficient for big data joins.

11. How are 'startswith' and 'endswith' methods used in Python?

```
text = "Hello, World!"
print(text.startswith("Hello")) # True
print(text.endswith("World!"))  # True
```

12. DAG Scheduler in PySpark
The DAG Scheduler in PySpark breaks down operators into stages of tasks, optimizing execution and managing dependencies.

13. Write a Python program that counts the number of times each character appears in a string

```
from collections import Counter
def count_characters(s):
s = s.replace(" ", "") # Remove spaces
return Counter(s)
print(count_characters("Hello World!"))
```

Exponentia.ai Data Engineer Interview Questions with Solutions

1. What's the difference between a data warehouse and a data lake?
- A data warehouse is for structured data, optimized for SQL-based reporting with predefined schemas, ideal for historical data analysis. A data lake stores structured, semi-structured, and unstructured data, making it flexible for big data, advanced analytics, and machine learning by storing raw data in its native format.

2. How do you handle semi-structured data like JSON or XML?
- Tools like Apache Spark or pandas in Python can handle JSON and XML by parsing, reading, and manipulating the data into DataFrames, facilitating seamless data processing and analysis in pipelines.

3. What's the difference between ETL and ELT?
- ETL (Extract, Transform, Load) transforms data before loading it into the target system, suitable for traditional on-premises data warehouses. ELT (Extract, Load, Transform) loads raw data into the target first, using the system's processing power for transformations, which is efficient for large-scale, cloud-based environments.

4. How do you optimize PySpark performance?
- Optimize PySpark by using DataFrame operations, applying predicate pushdown for early filtering, and minimizing shuffles with partition-aware transformations. Use broadcast joins for small datasets and cache data when reused frequently to enhance performance.

5. What strategies do you use to handle NULL values?
- NULL values can be handled with 'dropna()' to remove rows with nulls, 'fillna()' to replace them with default values, or 'coalesce()' to substitute nulls with the first non-null value.

6. What is the difference between a star schema and a snowflake schema?
- A star schema features a central fact table linked directly to denormalized dimension tables, making queries simpler and faster. A snowflake schema normalizes dimension tables into multiple

related tables, reducing redundancy but requiring more joins, which increases query complexity.

7. What's the difference between a sort key and a distribution key?
- Sort keys optimize query performance by ordering data within a table, improving filtering and range scans. Distribution keys control how data is spread across nodes in a distributed database, affecting how evenly data and queries are balanced, directly impacting performance.

8. How do you remove duplicates in PySpark?
- Use PySpark's 'dropDuplicates()' to eliminate duplicates. For more control, window functions can help manage duplicates:

```
window_spec = Window.partitionBy('column_name').orderBy('another_column')
df = df.withColumn('row_num', row_number().over(window_spec)).filter('row_num == 1')
```

Carelon Global Solutions India Data Engineer Interview Questions with Solutions

(Prepping for an AWS Data Engineer role? Here are some key questions from my personal interview experiences that you might encounter!)

1. What are the key features of Python?
- Python is easy to learn due to its simplicity and readability. It's interpreted, dynamically typed, and supports object-oriented programming. It runs on multiple platforms and has extensive libraries for web development, automation, and data science.

2. Does Spark executor memory get equally divided into executor cores?
- No, Spark executor memory is shared among cores and used dynamically for tasks like caching, shuffling, and execution.

3. How does Spark executor memory get divided?
- It's divided into Execution Memory (shuffles, joins), Storage Memory (caching), User Memory, and Reserved Memory for Spark's internal processes.

4. How to handle out-of-memory issues when joining large datasets in Spark?
- Increase executor memory, use broadcast joins for small datasets, increase parallelism with repartition(), optimize join strategies, persist intermediate data, and handle data skew with techniques like salting.

5. How to add a price_category column in PySpark based on price?

```
df = df.withColumn(
"price_category",
F.when(df["price"] < 50, "low")
.when((df["price"] >= 50) & (df["price"] <= 100), "medium")
.otherwise("high")
)
```

6. Provide an example of a PySpark DataFrame with product name and price.

```
data = [("Product1", 30), ("Product2", 60), ("Product3", 120)]
df = spark.createDataFrame(data, ["product_name", "product_price"])
df = df.withColumn(
"price_category",
F.when(df["product_price"] < 50, "low")
.when((df["product_price"] >= 50) & (df["product_price"] <= 100), "medium")
.otherwise("high")
)
```

7. How to calculate the variation of sales from yesterday to today in SQL?

```
SELECT
product,
date,
sales,
sales - LAG(sales) OVER (PARTITION BY product ORDER BY date) AS sales_variation
FROM
product_sales;
```

8. What are some common use cases for Hadoop?
- Hadoop is commonly used for large-scale data processing, storing and analyzing unstructured data, running distributed applications, and supporting big data analytics with tools like MapReduce, Hive, and Pig.

9. How does partitioning work in Hive, and why is it useful?
- Partitioning in Hive divides large tables into smaller, more manageable pieces, based on column values (like date). This improves query performance by allowing Hive to scan only relevant partitions instead of the entire table.

OLTP vs. OLAP: Important Data Engineer Interview Question You Need to Master!

If you're prepping for a data engineering interview, you've likely heard this: "What's the difference between OLTP and OLAP?" But it's not just about knowing the basics—it's about understanding the continuum between them! Let's dive into this with AWS examples to give you an edge in your next interview.

OLTP (Online Transaction Processing):

- Handles real-time transactions like order placements or user registrations.

- Example: Amazon RDS (e.g., MySQL, PostgreSQL) stores transactional data quickly and efficiently.

OLAP (Online Analytical Processing):

- Optimized for large-scale data analysis, perfect for generating reports and dashboards.

- Example: Amazon Redshift enables fast querying of large datasets, making it ideal for deep dives into sales trends or customer behavior.

The Continuum Explained with AWS:

1. Production Snapshots: Use RDS Snapshots to back up data from your OLTP systems.

2. Master Data: Clean and integrate this data using AWS Glue, then store it in Amazon S3 in a structured format (like Parquet).

3. OLAP Cubes: Load master data into Redshift for quick, multi-dimensional analysis (e.g., sales by time and region).

4. Metrics: Visualize insights using Amazon QuickSight, turning your data into actionable dashboards.

Why This Matters in Interviews :

Understanding the OLTP-OLAP continuum helps you explain how to streamline data flows from real-time transactions to strategic insights. It shows you can design systems that cater to both operational needs and analytical queries—skills highly valued in data engineering roles.

Carelon Global Solutions India AWS Data Engineer Interview Questions with Solutions

(Prepping for an AWS Data Engineer role? Here are some key questions from my personal interview experiences that you might encounter!)

1. There are two SQL tables, one with values 1, 2, 3, 4, 5, 6 and the other with 4, 5, 6, 7. Write an SQL query to get the desired non-repeated values from both.

```
(SELECT number FROM table1 UNION SELECT number FROM table2)
EXCEPT
(SELECT number FROM table1 INTERSECT SELECT number FROM table2);
```

This will return 1, 2, 3, 7.

2. You want to execute PySpark code from a Python script on an EC2 instance. How would you run it efficiently?
-> Ensure PySpark is installed:
 pip install pyspark
-> Upload your script to EC2 using SCP:
 scp -i /path/to/your-key.pem file.py ec2-user@ec2-instance- address:/home/ec2-user/
-> Run the script with Spark:
 spark-submit /home/ec2-user/file.py

3. You need to copy files from your local machine to an S3 bucket. What is the most efficient way to do this using AWS CLI?
Use the AWS CLI command:
- aws s3 cp /path/to/local/file s3://your-bucket-name/path/to/destination/

4. If you create and then overwrite a file in S3, how many files are there, considering versioning?
- Without Versioning: There will be only 1 file since each overwrite replaces the previous one.
- With Versioning Enabled: The number of files (versions) equals the number of times the file has been overwritten plus the original. For example, creating and overwriting once results in 2 versions.

5. Compare Snowflake and Spark. Which is generally faster and in what scenarios when running the same query?
 - Snowflake is generally faster for SQL queries due to its optimized architecture and automatic scaling. Spark may require more tuning and is usually slower for straightforward SQL queries due to in-memory computation overhead.

6. When running the same queries, is it more cost-effective to use Snowflake or EC2 with Spark, and why?
 - Snowflake is typically more cost-effective for SQL queries with its pay-as-you-go model and managed scaling. EC2 with Spark can be costlier due to manual setup, tuning, and potentially underutilized resources.

 Spark questions can be tricky, especially when it comes to resource allocation.

I've faced this type of scenario-based questions many times in the interviews I've personally attended.

Question :
You need to process a 1 TB dataset on a Spark cluster with 20 nodes, each with 128 GB of memory and 32 cores. What's the best way to allocate resources?

Step-by-Step Solution:

1. **Understand Your Cluster**:
- Each node has 128 GB memory and 32 cores.
- Total: 20 nodes available.

2. **Allocate Memory Smartly**:
- Keep 10% of memory for Spark's internal processes.
- Usable Memory per Node: ~115 GB (128 x 0.9)

3. **Distribute Executors Efficiently**:
- Use 3 executors per node to balance memory and processing power.
- Memory per Executor: ~38 GB (115 GB ÷ 3).
- Cores per Executor: 5 (to avoid overhead).

4. **Total Executors in the Cluster**:
- Total No of Executors: 60 (3 executors x 20 nodes).

5. **Optimize Spark Configuration**:
- spark.executor.memory: 38g
- spark.executor.cores: 5
- spark.executor.instances: 60
- spark.driver.memory: 64g (adjust as needed)
- spark.driver.cores: 8

6. **Enable Dynamic Allocation** (**if workload varies**):
- Allows Spark to automatically adjust resources.
- spark.dynamicAllocation.enabled = true
- spark.dynamicAllocation.minExecutors = 30
- spark.dynamicAllocation.maxExecutors = 60

7. **Adjust Shuffle Partitions**:
- Tuning the number of shuffle partitions improves performance.
- spark.sql.shuffle.partitions: 300 (or based on job needs).

1. PySpark Code Challenge:

Write a PySpark code to perform these steps:

- Check for duplicates and drop them if found in primary keys (1-10).

- Filter out rows with null or empty primary keys.

- Replace nulls in non-primary keys with 0.

- Reject rows where all values in columns 11 to 20 are null (non-primary keys).

```
# Check for duplicates in primary keys (pk1 to pk10 assumed)
df = df.dropDuplicates(['pk1', 'pk2', ..., 'pk10'])

# Filter out rows with null or empty primary keys
primary_keys = ['pk1', 'pk2', ..., 'pk10']
df = df.filter(F.reduce(lambda x, y: x & y, [(~F.isnull(df[col]) & (df[col] != "")) for col in primary_keys]))

# Replace nulls in non-primary keys with 0
non_primary_keys = ['col11', 'col12', ..., 'col20']
df = df.fillna(0, subset=non_primary_keys)

# Reject rows where all values in columns 11 to 20 are null (Non-primary keys)
df = df.filter(~F.reduce(lambda x, y: x & y, [F.isnull(df[col]) for col in non_primary_keys]))
```

2. Generate unique non-repeating pairs like 'A-B' from a given table using SQL:

Table

col
A
B
C
D

```
SELECT DISTINCT a.col || '-' || b.col AS combinations
FROM (SELECT col FROM table) a
JOIN (SELECT col FROM table) b ON a.col < b.col;
```

3. How do you return a parameter from a child notebook to a parent notebook?

- Use 'dbutils.notebook.exit()' in the child notebook and capture the value with 'dbutils.notebook.run()' in the parent notebook.

4. Parameterizing Values at Runtime in Databricks:

- Use Databricks widgets like 'dbutils.widgets.text()' or 'dbutils.widgets.dropdown()' to input parameters during runtime.

5. Delta Tables vs. Traditional RDBMS:

- Delta tables support ACID transactions, schema evolution, and versioning, making them robust for big data. Traditional RDBMS are row-based, have strict schema requirements, and are less suited for large, constantly updating datasets.

6. What do you mean by Databricks runtime, and where can we see it?

- Databricks runtime includes Apache Spark and other components optimized for running on Databricks. You can select and view runtimes when creating or configuring clusters in the Databricks UI.

1. **Core SQL Commands**:
   - Get the basics right: ‘SELECT’, ‘FROM’, ‘WHERE’, ‘JOIN’, ‘GROUP BY’, ‘ORDER BY’.
   - Master operators: ‘=’, ‘!=’, ‘<’, ‘>’, ‘<=’, ‘>=’, ‘IN’, ‘BETWEEN’, ‘LIKE’.
   - Use ‘HAVING’ to filter grouped data!

   2. **Aggregate Functions for Data Analysis**:
   - Summarize data efficiently: ‘COUNT()’, ‘SUM()’, ‘AVG()’, ‘MIN()’, ‘MAX()’.
   - Combine with ‘GROUP BY’ for insightful summaries.

   3. **Mastering Joins**:
   - Combine data from multiple tables: ‘INNER JOIN’, ‘LEFT JOIN’, ‘RIGHT JOIN’, ‘FULL OUTER JOIN’, ‘CROSS JOIN’.
   - Interview Tip: Be ready to explain the differences and use cases for each join type.

   4. **String Manipulation**:
   - Clean and transform text data: ‘CONCAT()’, ‘UPPER()’, ‘LOWER()’, ‘REPLACE()’, ‘TRIM()’.
   - Essential for prepping data for analysis.

   5. **Date Functions**:
   - Handle dates effortlessly: ‘DAY()’, ‘MONTH()’, ‘YEAR()’, ‘DATEDIFF()’, ‘DATE_ADD()’.
   - Key for time-series analysis and data transformations.

6. **Data Cleaning & Transformation**:
- Tidy up your data: ‘CAST()’, ‘COALESCE()’, ‘CASE WHEN’, ‘IFNULL()’.
- Essential for preparing data for ETL processes.

7. **Window Functions**:
- Perform advanced analytics: ‘ROW_NUMBER()’, ‘RANK()’, ‘DENSE_RANK()’, ‘LEAD()’, ‘LAG()’.
- Interview Tip: Be prepared to discuss how these functions can simplify complex analytics.

8. **Advanced SQL Techniques**:
- Enhance your queries: ‘CTEs’ (Common Table Expressions), ‘Subqueries’, ‘UNION’, ‘INTERSECT’, ‘EXCEPT’.
- Learn to optimize and troubleshoot complex queries, especially with large datasets.

9. **SQL Optimization Tips**:
- Understand indexing, query plans, and execution times to improve performance on large-scale data systems.

Accenture Data Engineer Interview Questions with Solutions

(Planning to land a Data Engineer role? Start prepping with these key questions!)

These questions are from my personal experience during interviews.

1. Explain your projects in detail.
 - Focus on your role, technologies used (e.g., Python, SQL, AWS), and the outcomes. Highlight specific problems you solved and the impact of your work.

2. Write a Python function to rotate an array by K.

```
def rotate_array(arr, k):
k %= len(arr)
return arr[-k:] + arr[:-k]

print(rotate_array([5, 4, 3, 2, 1], 2))
```

3. Python function to find the length of the longest substring without repeating characters.

```
def longest_unique_substring(s):
char_map = {}
left = max_length = 0
for right in range(len(s)):
if s[right] in char_map:
```

```
left = max(left, char_map[s[right]] + 1)
char_map[s[right]] = right
max_length = max(max_length, right - left + 1)
return max_length

print(longest_unique_substring("abcabcbb"))
```

4. How do you handle discrepancies and null values in numerical columns in DataFrames?
 - Use methods like 'fillna()' to fill missing values, 'dropna()' to remove rows/columns with nulls, 'interpolate()' for estimating missing values, and 'replace()' for correcting discrepancies.

5. Name some transformations performed using pandas.
 - Common transformations include:
 - 'groupby()' for aggregating data.
 - 'pivot()' / 'pivot_table()' for reshaping data into summary tables.
 - 'melt()' to convert wide data into a long format.
 - 'apply()' to apply custom functions.
 - `merge()` / `join()` to combine DataFrames.

6. What are wide and long DataFrames, and how does pivot help in it?
 - Wide Format: More columns, used for different variables in columns.
 - Long Format: Fewer columns, more rows with all observations in a single column.
 - Pivoting: Helps transform long format data into a wide format, making it more structured and readable.

7. Explain the basics of data structures and algorithms.
 - Understanding arrays, lists, stacks, queues, and dictionaries, and how algorithms like sorting and searching utilize these structures is crucial.

8. Which data structure is used under the hood of a dictionary in Python?
 - Python dictionaries use a hash table for storing key-value pairs, enabling efficient lookups, insertions, and deletions.

9. Importance of Data Engineering in the current world.
 - Data Engineering is essential for building scalable data pipelines that enable data-driven decisions, crucial for analytics, machine learning, and business intelligence.

Dun & Bradstreet Data Engineer Interview Questions with Solutions

1. Difference Between LIKE and = in SQL
 - '=' is used for exact comparisons between values, typically leveraging indexes for quick lookups.

- 'LIKE' is used for pattern matching with wildcards (`%` and `_`), which allows flexible searches but can lead to performance hits due to potential full table scans.

2. Why is = Faster than LIKE in SQL?
- '=' utilizes indexes effectively because it searches for exact matches, resulting in faster query execution.
- 'LIKE', especially with leading wildcards, often bypasses indexes and performs sequential scans, making it significantly slower for large datasets.

3. Indexes in SQL
- Indexes are database structures that enhance query performance by reducing the amount of data the database needs to process. They work like a roadmap, pointing directly to where data resides, but they do increase overhead on data modification operations (INSERT, UPDATE, DELETE) due to the need for index maintenance.

4. Provide an example of multithreading in Python

```
import threading

def print_numbers():
for i in range(5):
print(i)

thread = threading.Thread(target=print_numbers)
thread.start()
thread.join()
```

5. What is Agile Methodology?
- Agile is a project management and software development approach that prioritizes iterative progress, frequent feedback, and cross-functional team collaboration. It breaks projects into smaller, manageable units called sprints, allowing teams to respond swiftly to changes and deliver value continuously.

6. How to Implement SCD Type 1 in a Stored Procedure?
- SCD Type 1 overwrites existing data with updated data without retaining history. Here's an example of a stored procedure implementing SCD Type 1:

```
CREATE PROCEDURE UpdateCustomerData AS
BEGIN
 -- Overwrite existing records with new data
 TRUNCATE TABLE target_table;

 INSERT INTO target_table (CustomerID, CustomerName, Address)
 SELECT CustomerID, CustomerName, Address
 FROM source_table;
END;
```

7. What are wildcards in SQL?
 - Wildcards are special characters used with the 'LIKE' operator to filter data based on patterns:
 - '%' represents any sequence of characters, making it useful for broad searches.
 - '_' represents a single character, allowing for precise pattern matching with variability.

1. How does partitioning improve performance in PySpark?
- Partitioning distributes data across multiple nodes, enabling parallel processing. This reduces shuffle operations and speeds up task execution.

2. What is a Broadcast Join in PySpark, and when should you use it?
- A Broadcast Join sends a small DataFrame to all worker nodes, making joins faster by avoiding shuffling large datasets. Use it when one DataFrame is significantly smaller than the other.

3. How do you find duplicates in a list using Python?

```
 from collections import Counter
 duplicates = [item for item, count in Counter(my_list).items() if count > 1]
```

4. What is the purpose of the getOrCreate() method in Spark?
- 'getOrCreate()' checks if there’s an existing SparkSession; if not, it creates a new one. This avoids creating multiple sessions within the same application.

5. Write a Python script to print the following pattern:

```
 for i in range(1, 5):
 print('*' * i)
```

6. How can you determine the cluster size and configure memory settings in PySpark?
- Use 'spark.conf' or the Spark UI to monitor memory usage. Set configurations using '.config("spark.executor.memory", "4g")' and other related settings in the Spark session builder.

7. How do you decide the number of nodes and workers for a PySpark job?
- Base it on the data size, job complexity, and resource availability. Rule of thumb: more nodes for bigger data and more workers for parallelism.

8. How do coalesce and repartition handle data differently in PySpark?

- 'coalesce()' reduces partitions without shuffle, suitable for reducing partitions; 'repartition()' shuffles data to increase or decrease partitions, providing a balanced distribution.

9. How do you resolve OOM(Out of Memory) error?
 - Error: Out of memory exceptions during shuffle.
 - Resolution: Increase executor memory, optimize joins using broadcast, or increase partition sizes.

10. What is a DynamicFrame in PySpark, and how does it differ from a DataFrame?
- DynamicFrame is a higher-level abstraction used in AWS Glue, providing additional ETL capabilities like schema reconciliation, whereas DataFrame is optimized for performance.

11. How would you use groupBy and reduce to aggregate data in PySpark?

```
from pyspark.sql.functions import sum
df.groupBy("column").agg(sum("value_column")).show()
```

12. What are some common techniques you use to optimize performance in PySpark?
- Techniques include caching DataFrames, using broadcast joins, tuning parallelism ('spark.sql.shuffle.partitions'), and avoiding unnecessary shuffles.

Here's a quick rundown:

1. '**capitalize**()` - Capitalizes the first letter.
 Example: "**akshay kumar**".**capitalize**() ➔ '**Akshay kumar**'

2. '**casefold**()' - Converts to lowercase, stronger than 'lower()'.
 Example: "**Akshay Kumar**".**casefold**() ➔ '**akshay kumar**'

3. '**count**(**substring**)' - Counts occurrences of a substring.
 Example: "**Akshay Kumar**".**count**("**a**") ➔ '2'

4. '**find**(**substring**)' - Finds the first occurrence of a substring.
 Example: "**Akshay Kumar**".**find**("**Kumar**") ➔ '7'

5. '**index**(**substring**)' - Finds the index of the first occurrence (raises an error if not found).
 Example: "**Akshay Kumar**".**index**("**Akshay**") ➔ '0'

6. '**isalnum**()' - Checks if all characters are alphanumeric.
 Example: "**Akshay**123".**isalnum**() ➔ '**True**'

Note: "**Akshay** 123".**isalnum**() ➔ **False** (**because of the space**).

7. '**isalpha**()' - Checks if all characters are alphabetic.
Example: "**Akshay**123".**isalpha**() ➔ '**False**'
Note: "**Akshay**123".**isalpha**() ➔ **False** (**because of the numbers**).

8. '**isascii**()' - Checks if all characters are ASCII.
Example: "**Akshay Kumar**".**isascii**() ➔ '**True**'

9. '**isdecimal**()' - Checks if all characters are decimal.
Example: "12345".**isdecimal**() ➔ '**True**'
Note: "123.45".**isdecimal**() ➔ **False** (**because of the dot**).

10. '**isdigit**()' - Checks if all characters are digits.
Example: "12345".**isdigit**() ➔ '**True**'

11. '**isidentifier**()' - Checks if the string is a valid Python identifier.
Example: "**Akshay**".**isidentifier**() ➔ '**True**'
Note: "123**Akshay**".**isidentifier**() ➔ **False** (**identifiers cannot start with numbers**).

12. '**islower**()' - Checks if all characters are lowercase.
Example: "**akshay**".**islower**() ➔ '**True**'

13. '**isnumeric**()' - Checks if all characters are numeric.
Example: "12345".**isnumeric**() ➔ '**True**'

14. '**isprintable**()' - Checks if all characters are printable.
Example: "**Akshay Kumar**".**isprintable**() ➔ '**True**'

1. Explain all types of joins.
   - INNER JOIN: Returns rows with matching values in both tables, filtering out non-matching rows.
   - LEFT JOIN: Returns all rows from the left table and matched rows from the right. If there's no match, it returns NULL for the right table's columns.
   - RIGHT JOIN: Opposite of LEFT JOIN, it returns all rows from the right table and matched rows from the left.
   - FULL JOIN: Combines LEFT and RIGHT JOIN, returning all rows when there is a match in one of the tables, with NULLs where there are no matches.
   - CROSS JOIN: Produces a Cartesian product, combining every row of the first table with every row of the second table.

   2. What are subqueries and CTEs, and which is preferable?
   - Subqueries are nested queries inside a main query, often used for filtering,

calculations, or transforming data within the same SQL statement.
- CTEs are Defined using WITH clauses, CTEs are temporary result sets that can be referenced multiple times within a query, making the SQL easier to read and maintain.
- CTEs are often better for readability, debugging, and reusability, especially in complex queries.

3. Write a query to get a list of top 10 titles that are not in the user titles table.

```
SELECT title
FROM all_titles
WHERE title NOT IN (SELECT title FROM user_titles)
LIMIT 10;
```

4. What is an index in SQL and why is it useful?
- An index is a database object that improves the speed of data retrieval operations on a table at the cost of additional writes and storage space. By indexing columns used frequently in searches or joins, SQL queries can locate rows faster, making data access much more efficient.

5. How do you use HAVING and WHERE in an SQL query?
- WHERE: Filters rows before any grouping, working on individual rows of data.
- HAVING: Filters groups after the aggregation is applied, useful for conditions on grouped data like sums or averages.

6. Write a query to get the average renewal time for each user, using start date and end date columns.

```
SELECT user_id,
AVG(DATEDIFF(end_date, start_date)) AS avg_renewal_days
FROM renewals
GROUP BY user_id;
```

7. Is A LEFT JOIN B the same as B RIGHT JOIN A?
- Yes, both produce the same result, but the columns' order will reflect the order of the tables as specified in the query.

8. Explain RANK, DENSE_RANK, ROW_NUMBER with examples.
- RANK: Assigns ranks with gaps in case of ties (e.g., 1, 2, 2, 4).
- DENSE_RANK: Similar to RANK but without gaps between tied ranks (e.g., 1, 2, 2, 3).
- ROW_NUMBER: Assigns a unique row number to each row, regardless of any ties.

1. How do you flatten a deeply nested list?

```
from collections.abc import Iterable
def flatten(lst):
for item in lst:
if isinstance(item, Iterable) and not isinstance(item, str):
```

```
yield from flatten(item)
else:
yield item

nested_list = [1, [2, 3, [4, 5]], 6]
flat_list = list(flatten(nested_list))
print(flat_list)

Output: [1, 2, 3, 4, 5, 6]
```

2. How to find all indices of a specific value in a list?

```
lst = [10, 20, 10, 30, 10, 40]
indices = [i for i, x in enumerate(lst) if x == 10]
print(indices)

Output: [0, 2, 4]
```

3. How can you transpose a list of lists (matrix)?

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
transposed = list(map(list, zip(*matrix)))
print(transposed)

Output: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

4. How do you rotate a list by n positions?

```
def rotate(lst, n):
return lst[-n:] + lst[:-n]

lst = [1, 2, 3, 4, 5]
rotated_lst = rotate(lst, 2)
print(rotated_lst)

Output: [4, 5, 1, 2, 3]
```

5. How can you find duplicates in a list?

```
from collections import Counter
```

```
lst = [1, 2, 2, 3, 4, 4, 5]
duplicates = [item for item, count in Counter(lst).items() if count > 1]
print(duplicates)

Output: [2, 4]
```

6. How to split a list into evenly sized chunks?

```
def chunk(lst, n):
for i in range(0, len(lst), n):
yield lst[i:i + n]

lst = [1, 2, 3, 4, 5, 6, 7, 8]
chunks = list(chunk(lst, 3))
print(chunks)

Output: [[1, 2, 3], [4, 5, 6], [7, 8]]
```

7. How do you remove consecutive duplicates from a list?

```
from itertools import groupby
lst = [1, 2, 2, 3, 3, 3, 4, 4, 5]
result = [key for key, _ in groupby(lst)]
print(result)

Output: [1, 2, 3, 4, 5]
```

Tiger Analytics AWS Data Engineer Interview Questions:

1. What is the difference between RDD and DataFrame in Spark?
2. How do you optimize for memory and cost in ETL jobs in Spark?
3. What are wide and narrow transformations in Spark? How do they impact performance?
4. How would you find the second-highest salary in SQL using window functions?
5. What are the key differences between ROW_NUMBER(), RANK(), and DENSE_RANK() in SQL?
6. How can you write unit test cases for PySpark code locally?
7. What is the role of coalesce() and repartition() in Spark DataFrames? How are they different?
8. How do you deploy a PySpark application on an Apache Spark cluster or AWS services like EMR or Glue?
9. After deploying a PySpark job in production, how do you check if the changes have been successfully made?
10. How do you create data products in AWS? What are the key steps involved?

11. How does SparkSession work in AWS Glue, and what is the use of GlueContext?
12. How does distribution work in Spark, and how can you optimize partitioning?
13. What are DynamicFrames in AWS Glue, and how do they differ from DataFrames?

Here are some of the most essential PySpark questions you need to master:

1. What are some strategies for optimizing Spark jobs, and why are they important?

2. How do repartition and coalesce functions differ in Spark, and when should each be used?

3. What are the steps to run a Python file containing PySpark code on AWS EC2?

4. What is the difference between a Job, Stage, and Task in Apache Spark, and how do they work together?

5. What are the advantages of DataFrames over RDDs in PySpark?

6. What is a Spark session, and what are its key functions?

7. What is the difference between Bucketing and Partitioning in Spark, and how does each impact performance?

8. Why is data persistence important in Spark, and what are the common persistence mechanisms?

9. How do Spark Context and Streaming Context differ, and when do you use each?

10. What role does the DAG Scheduler play in executing jobs in Spark?

11. What is a Catalyst Optimizer in Spark, and why is it essential for query optimization?

12. What distinguishes wide transformations from narrow transformations in Spark?

13. How do you execute an anti-join operation in Spark?

14. What are semi joins in Spark, and when should you use them?

15. What are the differences between cache and checkpoint in Spark, and when should you use each?

16. Why is the Directed Acyclic Graph (DAG) critical to Spark's architecture?

17. What is speculative execution in Spark, and how does it help in job execution?

18. What happens if you forget to uncache data in Spark? How does it impact performance?

19. What are the main components of Spark architecture, and how do they interact?

1. Write a PySpark code snippet to filter out customers who are older than 30 years and show the resulting DataFrame.

Solution :

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col


spark = SparkSession.builder.appName("FilterCustomers").getOrCreate()


df = spark.read.csv("customers.csv", header=True, inferSchema=True)

filtered_df = df.filter(col("age") > 30)

filtered_df.show()
```

2. Write a PySpark code to read the sales log, compute the total sales amount for each 'sale_type' and 'date', aggregate the sales data by 'sale_type' and 'date', and write the aggregated sales data to a separate table.

Solution:-

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import *


spark = SparkSession.builder.appName("AggregateSales").getOrCreate()

sales_df = spark.read.csv("sales_log.csv", header=True, inferSchema=True)

aggregated_df = sales_df.groupBy("sale_type", "date")\

.agg(sum("sales_amount").alias("total_sales"))

aggregated_df.write.mode("overwrite").saveAsTable("aggregated_sales")
```

3. Write a PySpark code to detect columns with JSON log entries, infer the schema, convert JSON columns to fields, and persist the updated DataFrame.

Solution:-

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

spark = SparkSession.builder.appName("JsonColumns").getOrCreate()

schema = StructType([
StructField("field1", StringType(), True),
StructField("field2", IntegerType(), True)
])

df = spark.read.json("logs.json")
df = df.withColumn("json_column", from_json(col("json_column"), schema))
updated_df = df.select("json_column.*", *[col(c) for c in df.columns if c != "json_column"])
updated_df.write.parquet("output_path")
```

4. Write a PySpark code to read a position-based text file and store it in Delta format. The file contains lines like "Sam 12 Mumbai", "Rahul 19 Nashik", "John 21 Pune".

Solution:-

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import substring

spark = SparkSession.builder.appName("PositionBasedFile").getOrCreate()

df = spark.read.text("input.txt")
df = df.withColumn("name", substring("value", 1, 4)) \
.withColumn("age", substring("value", 6, 2).cast("int")) \
.withColumn("city", substring("value", 9, 10))
```

```
df.write.format("delta").save("delta_output")
```

1. Write code for a palindrome without reversing the list.

   Solution:-
   ```
   def is_palindrome(s):
   for i in range(len(s) // 2):
   if s[i] != s[-(i + 1)]:
   return False
   return True
   ```

   2. Reverse a string without using string slicing.

   Solution:-
   ```
   def reverse_string(s):
   result = ""
   for char in s:
   result = char + result
   return result
   ```

   3. Count the number of words in a string variable.

   Solution:-
   ```
   def count_words(s):
   return len(s.split())
   ```

   4. Write a function in Python to check for duplicate letters in words.

   Solution:-
   ```
   def has_duplicate_letters(s):
   words = s.split()
   for word in words:
   if len(set(word)) != len(word):
   return True
   return False
   ```

   5. For a given linked list of N nodes, check if it has a loop.

   Solution:-
   ```
   class ListNode:
   ```

```
def __init__(self, value=0, next=None):
self.value = value
self.next = next

def has_cycle(head):
slow = fast = head
while fast and fast.next:
slow = slow.next
fast = fast.next.next
if slow == fast:
return True
return False
```

6. Write code to implement a function for checking if a string is a valid parenthesis expression.

Solution:-

```
def is_valid_parenthesis(s):
stack = []
mapping = {")": "(", "}": "{", "]": "["}
for char in s:
if char in mapping:
top_element = stack.pop() if stack else '#'
if mapping[char] != top_element:
return False
else:
stack.append(char)
return not stack
```

7. Find the second largest element in a list.

Solution:-

```
def second_largest(nums):
first, second = float('-inf'), float('-inf')
for n in nums:
if n > first:
first, second = n, first
elif first > n > second:
second = n
return second
```

1. Write a Python function to return the result string which should not contain digits and duplicate characters:

Solution:

```
def remove_digits_and_duplicates(input_string):
result = ""
seen = set()
for char in input_string:
if char.isdigit():
continue
if char not in seen:
seen.add(char)
result += char
return result
```

2. Write a Python function to rotate an array by K:

Solution:-

```
def rotate_array(arr, k):
n = len(arr)
k = k % n
return arr[-k:] + arr[:-k]
```

3. Write a Python function to find the length of the longest substring without repeating characters:

Solution:-

```
def longest_unique_substring(s):
start = 0
max_length = 0
used_char = {}
for i, char in enumerate(s):
if char in used_char and start <= used_char[char]:
start = used_char[char] + 1
else:
max_length = max(max_length, i - start + 1)
used_char[char] = i
return max_length
```

4. Write a Python program to get the length of the input strings and count each character occurrence excluding spaces:

Solution :-

```
def string_length_and_char_count(s):
char_count = {}
length = 0
for char in s.replace(" ", ""):
```

```
length += 1
char_count[char] = char_count.get(char, 0) + 1
return length, char_count
```

5. Output of this code:

Solution:-

```
var = 10
var1 = 5
var2 = 3
result = var // var1 + var1 % var2
print(result)
# Output: 4
```

6. What is wrong in this code snippet?

Solution:-

```
def concatenate_strings(str1, str2):
return str1 + " " + str2

result = concatenate_strings("Test", 3)
print(result)
# The str2 argument is an integer, and it needs to be converted to a string first.
```

7. Write a Python program to print spatial numbers (numbers where digits are repeating, like 444, 33, etc.) from 1 to a given number n:

Solution:-

```
def spatial_numbers(n):
spatial_nums = []
for num in range(1, n + 1):
num_str = str(num)
if all(d == num_str[0] for d in num_str):
spatial_nums.append(num)
return spatial_nums
```

1. Getting 10 titles that are not in user database:
SELECT title FROM titles_table WHERE title NOTIN (SELECT title FROM user_table) LIMIT 10;

2. Names of employees that are present only in table 2 and not in table 1 using join condition:
SELECT t2.name FROM table2 t2 LEFT JOIN table1 t1 ON t2.name = t1.name WHERE t1.name IS NULL;

3. Write an SQL Query to get the Number of Employees working in each department:

```
SELECT d.department_name, COUNT(e.employee_id)
FROM employees e
JOIN department d ON e.department_id = d.department_id
GROUP BY d.department_name;
```

4. Generate the following output using SQL:

```
A-B
A-C
A-D
B-C
B-D
C-D
```

Table :-

```
col1
A
B
C
D
```

```
SELECT t1.col1, t2.col1
FROM table t1
JOIN table t2 ON t1.col1 < t2.col1;
```

5. Write an SQL Query to find values that are not common across two tables:

```
SELECT c1 FROM table1
UNION
SELECT c2 FROM table2
EXCEPT
SELECT c1 FROM table1
INTERSECT
SELECT c2 FROM table2;
```

6. Write an SQL query to find the number of customers who have bought an iPhone from a store located in Mumbai in the last month using the following tables: fact_sales, dim_customer, dim_store, and dim_product:

```
SELECT COUNT(DISTINCT c.customer_id)
FROM fact_sales fs
JOIN dim_customer c ON fs.customer_id = c.customer_id
JOIN dim_store s ON fs.store_id = s.store_id
JOIN dim_product p ON fs.product_id = p.product_id
WHERE p.product_name = 'iPhone'
AND s.city = 'Mumbai'
AND fs.sale_date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);
```

Additional Question :-

7. Write an SQL Query to find the count of records for different types of joins between two tables with dissimilar numbers of records.

Table 1	Table 2
1	4
2	5
3	6
7	

This is especially useful for large datasets or pre-trained models that need to be accessed by multiple tasks.

Here's how to use broadcast variables:

Pyspark code:-

```
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("BroadcastExample").getOrCreate()

# Sample data
data = [("Alice", 1), ("Bob", 2), ("Charlie", 3)]
columns = ["Name", "ID"]

# Create DataFrame
df = spark.createDataFrame(data, columns)

# Create a broadcast variable
broadcast_var = spark.sparkContext.broadcast([1, 2, 3])

# Use broadcast variable
df_with_broadcast = df.withColumn("Broadcasted_ID",
df.ID.isin(broadcast_var.value))
df_with_broadcast.show()
```

Missing Data Handling in PySpark

Dealing with missing data is a common challenge in data processing. PySpark provides powerful methods to handle missing values efficiently. Here's a quick guide on how to use `dropna()`, `fillna()`, and `replace()`:

Pyspark code :-

```
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("MissingDataHandling").getOrCreate()

# Sample data
data = [("Alice", None), ("Bob", 25), (None, 30), ("David", 45)]
columns = ["Name", "Age"]

# Create DataFrame
df = spark.createDataFrame(data, columns)

# Drop rows with missing values
df_dropna = df.dropna()

# Fill missing values
df_fillna = df.fillna({"Name": "Unknown", "Age": 0})

# Replace missing values
df_replace = df.replace(to_replace=None, value="Unknown", subset=["Name"])

df_dropna.show()
df_fillna.show()
df_replace.show()
```