1. Write a PySpark query to find the top 3 highest-paid employees from a DataFrame with columns (id, name, salary).
2. from pyspark.sql import SparkSession
3. from pyspark.sql.functions import col
4. # Initialize Spark session
5. spark = SparkSession.builder.appName("example").getOrCreate()
6. # Sample data
7. data = [
8. (1, "Alice", 50000),
9. (2, "Bob", 60000),
10. (3, "Charlie", 70000),
11. (4, "David", 80000),
12. (5, "Eve", 90000)
13. ]
14. # Create DataFrame
15. df = spark.createDataFrame(data, ["id", "name", "salary"])
16. # Find the top 3 highest-paid employees
17. top\_3\_employees = df.orderBy(col("salary").desc()).limit(3)
18. # Show the result
19. top\_3\_employees.show()

 Read a CSV file in PySpark, filter out rows with null values in the price column, and save the result as a Parquet file

from pyspark.sql import SparkSession

# Initialize Spark session

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

# Read the CSV file into a DataFrame

df = spark.read.csv("path/to/your/file.csv", header=True, inferSchema=True)

# Filter out rows with null values in the 'price' column

df\_filtered = df.filter(df.price.isNotNull())

# Save the result as a Parquet file

df\_filtered.write.parquet("path/to/save/filtered\_data.parquet")

In this project, we designed and implemented a comprehensive data integration and analytics solution using Azure services. Our primary objective was to seamlessly ingest, process data from various on-premises sources to support downstream systems such as business intelligence (BI) reporting and ML

We started by utilizing Azure Data Factory's Copy Activity to read data from multiple on-premises sources, including SQL Server and other databases.

This ingested data was then loaded into Azure Data Lake Storage (ADLS), providing a scalable and secure storage solution for both raw and processed data. To process this data, we mounted the ADLS storage in Azure Databricks and implemented the Medallion Architecture, organizing the data into Bronze (raw), Silver (cleaned), and Gold (aggregated) layers. We leveraged Delta Lake's capabilities for ACID transactions, scalable metadata handling, and efficient data processing by storing the data in Delta format. The processed data in Delta tables was then prepared for BI reporting,

Throughout the project, Azure Data Factory was used to orchestrate the entire data pipeline, ensuring smooth and automated data flow from source to destination.

1. **Daily Stand-Up Meeting**: Start the day with a quick team meeting to discuss what everyone is working on, any challenges, and plans for the day. This helps keep the team aligned and informed.
2. **Checking Pipeline Status**: Review the status of data pipelines that ran overnight. Ensure all scheduled jobs in Azure Data Factory completed successfully. If there are any failures, prioritize investigating and resolving them.
3. **Collaboration**: Meet with senior data engineers, data scientists, or business analysts to understand their data requirements. Provide support by preparing datasets, answering queries, or helping with data-related issues.
4. **Documentation**: Document the data pipelines, transformations, and any changes made during the day. This helps maintain transparency and makes it easier for others to understand and maintain the system.
5. **Learning and Development**: Spend some time learning new tools, technologies, or best practices in data engineering.

**Ad-Hoc Support**: Handle any ad-hoc tasks or support requests that come up during the day. This could include debugging data issues, optimizing existing pipelines,

**What methodology did u follow?**

"We follow Agile methodology by breaking the project into sprints, holding daily stand-up meetings, and delivering features incrementally. We prioritize tasks, continuously integrate and test new features, and conduct sprint reviews and retrospectives to gather feedback and improve. This approach helps us stay flexible, collaborate effectively, and deliver high-quality results."

**How can you optimize the performance of your Azure Data Factory pipeline?**

* **Answer**: To optimize performance, you can:
  + Use parallel execution to process multiple tables simultaneously.
  + Increase the Data Integration Units (DIUs) for Copy activities to allocate more resources.
  + Use partitioning to break down large datasets into smaller chunks.

DIU- In Azure Data Factory, a **Data Integration Unit (DIU)** is a measure of the processing power allocated to a Copy activity. It represents a combination of CPU, memory, and network resources. The DIU determines how much power is available to perform data movement and transformation tasks.

Here are some key points about DIUs:

* **Range**: The DIU can range from 4 to 256 units, depending on the complexity and size of the data being processed
* **Auto Scaling**: If you select the "Auto" option, Azure Data Factory dynamically adjusts the DIU based on the source and sink data stores and the data pattern

**Use parallel execution to process multiple tables simultaneously.**

**how we can do this**

1. **Create a Lookup Activity**:
   * Use a **Lookup** activity to retrieve the list of tables you want to process. This activity will fetch the table names from a source, such as a database or a configuration file.
2. "SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'your\_schema'"
3. **Add a ForEach Activity**:
   * Add a **ForEach** activity to your pipeline. This activity will iterate over the list of tables retrieved by the Lookup activity.
4. **Configure Parallel Execution**:
   * In the **ForEach** activity settings, set the **Batch Count** property to the number of parallel executions you want. For example, setting the Batch Count to 10 will process 10 tables simultaneously
5. **Add Activities Inside ForEach**: adding copy activity
6. **Pass Table Names as Parameters** – Use table names retrieved from lookup
7. **What strategies can you use to process large volumes of data efficiently in Azure Databricks?**
   * **Answer**: Strategies include:
     + Using Delta Lake for efficient data storage and processing.
     + Implementing the Medallion Architecture (Bronze, Silver, Gold layers) to organize data.
     + Leveraging Spark's built-in optimizations like Catalyst and Tungsten.
     + Scaling up or out Databricks clusters based on workload requirements.

**2)How to avoid tables which are already processed?**

1. **Checkpointing**: Implement checkpointing logic in your pipeline to keep track of which tables have been processed. You can use a control table or a status flag in your source or destination to mark the completion of each table.
2. **Conditional Execution**: Use the "If Condition" activity to check the status of each table before processing. This way, you can skip the tables that have already been processed.

**For AWS Glue:**

1. **Job Bookmarking**:
   * Enable job bookmarking in AWS Glue to keep track of processed data. This feature allows Glue jobs to process only new or changed data since the last successful run
   * Configure your Glue job to use bookmarks by setting the --enable-job-bookmark parameter to true.

**For AWS EMR:**

1. **Checkpointing**:
   * Implement checkpointing in your Spark jobs to keep track of processed tables. You can use HDFS or S3 to store checkpoint files.
   * Modify your Spark job to read the checkpoint file and skip already processed tables.

**DELTA LAKE**- Delta Lake is used as an optimized storage layer that brings several key benefits to data processing and management. Here are some of the main reasons why Delta Lake is used:

1. **ACID Transactions**:
   * Delta Lake provides ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data consistency and reliability even in the presence of concurrent writes and reads
2. **Schema Enforcement and Evolution**:
   * It supports schema enforcement to prevent bad data from being written into your tables. Additionally, it allows schema evolution, enabling you to update your table schema without rewriting data using **megeSchema** option
3. df.write.option**("mergeSchema", "true"**).mode("append").format("delta").save("path/to/delta-table")
4. **option("multiLine", "true"):** This option allows Spark to handle rows that span multiple lines.
5. **Time Travel**:
   * Delta Lake allows you to access and query historical versions of your data, making it possible to perform audits, rollbacks, and reproduce experiments
6. **Unified Batch and Streaming**:
   * It supports both batch and streaming data processing, allowing you to use a single copy of data for both types of workloads
7. **Scalable Metadata Handling**:
   * Delta Lake efficiently handles metadata, making it scalable for large datasets. This is

particularly useful for managing big data environments

**3)** **For suppose we have a blob storage in blob storage in 1 container we have so many files. So now Design a pipeline where it will check based on current date and will Delete files which are more than 30 days.**

1. **Create a Data Factory**:
   * Go to the Azure portal.
   * Create a new Data Factory instance.
2. **Create a Pipeline**:
   * In the Data Factory, create a new pipeline.
3. **Add Activities to the Pipeline**:
   * **Get Metadata Activity**:
     + Use this activity to list all the files in the blob storage container.
     + Configure it to get the last modified date of each file.
   * **ForEach Activity**:
     + Use this activity to iterate over the list of files obtained from the Get Metadata activity.
   * **If Condition Activity**:
     + Inside the ForEach activity, add an If Condition activity to check if the file is older than 30 days.
     + Use an expression to compare the current date with the file's last modified date.
   * **Delete Activity**:
     + If the condition is true (file is older than 30 days), use the Delete activity to delete the file from the blob storage.

**What are the different types of clusters in Databricks?**

* **Answer**: The main types of clusters are All-Purpose Clusters, Job Clusters, and High-Concurrency Clusters. All-Purpose Clusters are used for interactive analysis, Job Clusters for running scheduled jobs, and High-Concurrency Clusters for concurrent task execution

1. **Cluster Size**:
   * **Driver Node**: Choose an instance type with sufficient memory and CPU to handle the coordination of tasks. For example, Standard\_DS3\_v2 (4 cores, 14 GB RAM).
   * **Worker Nodes**: Select instance types based on your workload requirements. For example, Standard\_DS3\_v2 or Standard\_DS4\_v2 (8 cores, 28 GB RAM) for moderate workloads. Scale up to Standard\_DS5\_v2 (16 cores, 56 GB RAM) for more intensive tasks.
2. **Auto-scaling**:
   * Enable auto-scaling to dynamically adjust the number of worker nodes based on the workload. This ensures efficient resource utilization and cost management
3. **Photon Acceleration**:
   * Enable Photon acceleration if available, as it can significantly improve the performance of your data processing tasks
4. **Databricks Runtime**:
   * Use the latest Databricks Runtime version for all-purpose compute to ensure you have the latest optimizations and compatibility.
   * For job compute running operational workloads, consider using the Long Term Support (LTS) Databricks Runtime version
5. **Configuration Example**:
   * **Driver Node**: Standard\_DS3\_v2
   * **Worker Nodes**: Standard\_DS4\_v2
   * **Auto-scaling**: Enabled, with a minimum of 2 workers and a maximum of 10 workers.
   * **Databricks Runtime**: Latest version or LTS version for stability.

Databricks clusters use **Spark's Standalone cluster manager**.

**Example Calculation**

Suppose you have a workload that requires:

* 64 cores
* 256 GB of RAM

You can choose between different instance types:

* **Option 1**: Standard\_DS4\_v2 (8 cores, 28 GB RAM)
  + Number of instances: 8
  + Total cores: 8 \* 8 = 64
  + Total RAM: 8 \* 28 GB = 224 GB (slightly less than required, may need to adjust)
* **Option 2**: Standard\_DS5\_v2 (16 cores, 56 GB RAM)

**Driver Node**

The driver node is responsible for orchestrating the execution of tasks. It manages the SparkContext, schedules tasks, and collects results from the worker nodes.

**Worker Node**

Worker nodes are responsible for executing the tasks assigned by the driver node. They perform the actual data processing and computation.

* **Cluster Name**: Unique identifier.
* **Cluster Mode**: Standard, High Concurrency, or Single Node.(Job cluster, all purpose)
* **Databricks Runtime Version**: Select the appropriate version.
* **Worker Type and Number**: Configure worker nodes.(that are cluster nodes only cpu and memory depend on instance type)
* **Driver Type**: Configure the driver node.
* **Autoscaling**: Enable and configure autoscaling.
* **Advanced Options**: Set Spark configurations, environment variables, and init scripts.
* **Libraries**: Install necessary libraries

**Common Instance Types and Specifications**

1. **Standard\_DS3\_v2**:
   * **CPU**: 4 cores
   * **Memory**: 14 GB RAM
2. **Standard\_DS4\_v2**:
   * **CPU**: 8 cores
   * **Memory**: 28 GB RAM
3. **Standard\_DS5\_v2**:
   * **CPU**: 16 cores
   * **Memory**: 56 GB RAM
4. **Standard\_D8s\_v3**:
   * **CPU**: 8 cores
   * **Memory**: 32 GB RAM
5. **Standard\_D16s\_v3**:
   * **CPU**: 16 cores
   * **Memory**: 64 GB RAM
6. **Standard\_D32s\_v3**:
   * **CPU**: 32 cores
   * **Memory**: 128 GB RAM

**1)Your Databricks job is running slower than expected. How would you identify bottlenecks and optimize it?**

**Identifying Bottlenecks**

1. **Monitor Job Metrics**: Check CPU, memory, and disk I/O usage in databricks UI.
2. **Analyze Spark UI**: Identify stages with high latency and task stragglers.
3. **Check Data Skew**: Look for uneven workload distribution.

**Optimizing Performance**

1. **Cluster Configuration**: Increase cluster size, enable auto-scaling, and use Photon.
2. **Spark Configurations**: Adjust spark.sql.shuffle.partitions and tune executor memory and cores.
3. **Delta Lake Features**: Use Delta caching and run OPTIMIZE and VACUUM commands.
4. **Code Optimization**: Minimize wide transformations and use broadcast joins for small tables.
5. **Caching Delta -**Delta caching, also known as disk caching, is a feature in Databricks that accelerates data reads by creating copies of remote Parquet data files in the local storage of nodes.

**2) A scheduled Databricks job fails randomly due to memory issues. What strategies would you apply to prevent this?**

1. **Optimize Cluster Configuration**:
   * **Increase Memory per Executor**: Choose instance types with more memory or increase the memory allocated to each executor
   * **Enable Auto-scaling**: Allow the cluster to scale up or down based on workload demands
2. **Optimize Spark Configurations**:
   * **Adjust Shuffle Partitions**: Increase the number of shuffle partitions to distribute data more evenly
   * **Tune Garbage Collection**: Use garbage collection settings like spark.executor.extraJavaOptions to optimize memory management
3. **Data Skew Management**:
   * **Check for Data Skew**: Ensure data is evenly distributed across partitions. Use Adaptive Query Execution (AQE) to automatically handle skew
4. **Code Optimization**:
   * **Avoid Large Broadcast Variables**: Minimize the use of large broadcast variables that can consume excessive memory
   * **Efficient Data Processing**: Optimize your code to reduce memory usage, such as using map-side reductions and avoiding wide transformations
5. **Delta Lake Features**:
   * **Delta Caching**: Use Delta caching to speed up read operations and reduce memory usage
   * **Optimize and Vacuum**: Regularly run OPTIMIZE and VACUUM commands on Delta tables to compact files and remove old data

**3)** **How do you handle small files and optimize performance in Databricks?**

**1. Delta Lake Format**

Using Delta Lake as the storage format helps manage small files efficiently. Delta Lake supports ACID transactions and unifies batch and streaming data, which simplifies data management and improves performance

**2. Auto Compaction**

Databricks provides an auto-compaction feature that automatically combines small files within Delta table partitions. This reduces the overhead associated with managing numerous small files

**3. Optimize Command**

The **OPTIMIZE** command in Databricks can be used to compact small files into larger ones. This command helps improve query performance by reducing the number of files that need to be read

NOTE- Delta tables are accessed and queried like traditional tables with rows and columns. The underlying data is stored in files, but when you query a Delta table, you interact with it using SQL or DataFrame APIs, just like any other table.

**4) Can you explain the difference between Job Clusters and Interactive Clusters?**

| **Feature** | **Job Clusters** | **Interactive Clusters** |
| --- | --- | --- |
| **Purpose** | Automated jobs and workflows | Collaborative data analysis and development |
| **Usage** | Scheduled jobs, batch processing, production | Running notebooks, ad-hoc queries, exploration |
| **Creation** | Automatically created for each job run | Manually started and stopped |
| **Termination** | Terminates after job completion | Can be restarted |
| **User Access** | Single job isolation | Multiple users can share |
| **Cost** | Cost-effective (active only during job run) | Generally, more expensive (kept running) |
| **Best For** | Production-level tasks | Development and collaboration |

**1)How moved your pipeline to different environments?**

**Here are the steps to set up a CI/CD pipeline using GitHub and Azure DevOps in simple terms:**

**1. Set Up Your GitHub Repository**

* Create a Repository: Go to GitHub and create a new repository for your project.
* Push Code: Push your project code to this repository.

**2. Connect GitHub to Azure DevOps**

* Create a Project: In Azure DevOps, create a new project.
* Service Connection: Set up a service connection to link your GitHub repository with Azure DevOps.

**3. Create a Build Pipeline (CI)**

* Pipeline Setup: In Azure DevOps, go to Pipelines and create a new pipeline.
* Select GitHub: Choose GitHub as the source and select your repository.
* Configure Pipeline: Use the wizard or YAML file to define the build steps (e.g., install dependencies, run tests, build the project).

**4. Create a Release Pipeline (CD)**

* Release Setup: In Azure DevOps, go to Releases and create a new release pipeline.
* Add Artifacts: Link the build artifacts from your build pipeline.
* Define Stages: Add stages for different environments (e.g., UAT, Production).
* Deploy: Configure deployment tasks to deploy your application to the target environment (e.g., Azure services).

**5. Automate and Monitor**

* Triggers: Set up triggers to automatically run the build pipeline when code is pushed to GitHub.
* Monitoring: Use Azure DevOps to monitor the pipeline runs and set up alerts for any failures.

**Summary**

1. GitHub: Create a repository and push your code.
2. Azure DevOps: Create a project and connect it to GitHub.
3. Build Pipeline: Set up a pipeline to build and test your code.
4. Release Pipeline: Set up a pipeline to deploy your code to different environments.
5. Automation: Automate the process and monitor for issues.

**3)For different or inconsistent records**

**Data Validation**

* **Validation Activity:** Use the Validation Activity to check the integrity and structure of the data before copying. This can help identify records that don't meet the expected criteria
* It helps automate data quality checks, ensuring that the data meets specific criteria before further processing.

**Error Handling**

* **Fault Tolerance**: Configure fault tolerance settings in the Copy Activity to skip inconsistent or erroneous records and log them for further analysis

**Key Features of Fault Tolerance**

1. **Skip Incompatible Data**:
   * **Automatic Skipping**: If the source data has rows that are incompatible with the sink (e.g., data type mismatches, primary key violations), the Copy Activity can skip these rows instead of failing the entire operation
   * **Logging**: You can enable logging to keep track of the skipped rows, which helps in diagnosing and resolving issues later

**Activities in ADF**-

**1. Data Movement Activities**

* **Copy Activity**: The most common activity used to copy data from a source to a destination. It supports a wide range of data stores and formats

**2. Data Transformation Activities**

* **Mapping Data Flow**: Allows you to perform data transformations at scale using a visual interface.
* **Data Flow Activity**: Executes data flows that transform data within ADF.
* **Stored Procedure Activity**: Executes stored procedures in databases.
* **Databricks Notebook/JAR/Python Activity**: Runs notebooks, JAR files, or Python scripts on an Azure Databricks cluster.
* **Azure Function Activity**: Executes Azure Functions for custom processing.
* **HDInsight Activities**: Runs Hive, Pig, MapReduce, or Spark jobs on an HDInsight cluster

**3. Control Activities**

* **ForEach Activity**: Iterates over a collection of items and executes specified activities for each item.
* **If Condition Activity**: Executes activities based on a condition.
* **Wait Activity**: Pauses the pipeline for a specified duration.
* **Validation Activity**: Validates the existence and properties of datasets before proceeding.
* **Set Variable Activity**: Sets the value of a pipeline variable.
* **Execute Pipeline Activity**: Invokes another pipeline within the current pipeline

**6)Difference between Control Flow and Data Flow?**

**Control Flow**

* **Purpose**: Orchestrates and manages the workflow of various activities in a pipeline.
* **Components**: Includes activities like Copy Activity, Execute Pipeline, ForEach, If Condition, Wait, and Validation.
* **Functionality**: Manages the sequence and dependencies of tasks, ensuring that activities are executed in the correct order and under the right conditions.
* **Use Case**: Ideal for coordinating the overall workflow, handling task dependencies, and implementing conditional logic

**Data Flow**

* **Purpose**: Performs data transformation operations.
* **Components**: Includes transformations like joins, aggregations, filtering, and row manipulations on **Mapping Data Flow**.
* **Functionality**: Transforms data at scale using a visual interface, running on a scaled-out Apache Spark cluster managed by ADF.
* **Use Case**: Best suited for transforming large datasets and implementing complex data transformation logic visually

**7)** **What is the difference between Retry Policy vs Fault Tolerance in ADF Copy Activity?**

**Retry Policy**

* **Purpose**: To handle transient errors by retrying the entire activity.
* **Functionality**: If an activity fails due to a transient issue (e.g., network glitches, temporary unavailability of a service), the Retry Policy will automatically retry the activity a specified number of times before marking it as failed.
* **Configuration**: You can configure the number of retry attempts and the interval between retries.
* **Use Case**: Ideal for scenarios where temporary issues can be resolved by simply retrying the operation

**Fault Tolerance**

* **Purpose**: To handle data inconsistencies and errors within the data being processed.
* **Functionality**: If the Copy Activity encounters incompatible data (e.g., data type mismatches, primary key violations), Fault Tolerance allows the activity to skip the problematic rows and continue processing the rest of the data. It can also log the skipped rows for further analysis.
* **Configuration**: You can enable fault tolerance settings to skip incompatible rows and configure logging to capture details of the skipped data.
* **Use Case**: Useful for scenarios where data quality issues are expected, and you want to ensure the pipeline continues processing without interruption

A **Tumbling Window Trigger** in Azure Data Factory is a type of trigger that fires at a periodic time interval from a specified start time, while retaining state

**Schedule Trigger**

1. **Recurrence**: Fires at regular intervals based on a specified schedule (e.g., hourly, daily).
2. **Future Loads**: Can only trigger future-dated loads.
3. **Many-to-Many Relationship**: Can be associated with multiple pipelines, and multiple triggers can be associated with a single pipeline.
4. **No Self-Dependency**: Does not have a built-in mechanism to ensure that the previous run is completed before starting the next one

**Tumbling Window Trigger**

1. **Fixed Intervals**: Fires at periodic, fixed-sized, non-overlapping, and contiguous time intervals (e.g., every hour from 2 PM to 3 PM, then 3 PM to 4 PM).
2. **Past and Future Loads**: Can be configured to initiate both past and future-dated loads.
3. **One-to-One Relationship**: Each tumbling window trigger is associated with a single pipeline.
4. **Self-Dependency**: Ensures that the next window does not start until the previous window has successfully completed. This is useful for scenarios where consecutive pipeline runs depend on each other

**) if your pipeline failes at one stage/activity , what you will do?**

1. **Monitor the Pipeline**:
   * Go to the **Monitor** tab in Azure Data Factory.
   * Find the pipeline run that failed and click on it to view the details.
2. **Identify the Failed Activity**:
   * In the pipeline run details, locate the activity that failed. You can see the error message and logs to understand the cause of the failure.
3. **Fix the Issue**:
   * Based on the error message, fix the underlying issue. This might involve correcting data, adjusting configurations, or resolving connectivity issues.
4. **Rerun the Pipeline from the Failed Activity**:
   * In the pipeline run details, you have the option to **Rerun from Failed Activity**. This will restart the pipeline from the point of failure, skipping the activities that were already completed successfully.

**Steps to Handle Extra Columns Manually**

1. **Identify the Error**:
   * Go to the **Monitor** tab in ADF.
   * Identify the activity that failed and check the error message to confirm that the failure is due to extra columns.
2. **Fix the Source Schema**:
   * Open the pipeline in the **Author** tab.
   * Locate the activity that reads the source data (e.g., a Copy Data activity).
   * Edit the source dataset to adjust the schema. You can manually map the columns you want to include and ignore the extra columns.
3. **Modify the Mapping**:
   * In the Copy Data activity, go to the **Mapping** tab.
   * Ensure that only the required columns are mapped from the source to the target. You can exclude the extra columns by not mapping them.

**8)  What is Mapping Data Flow in ADF, and how does it differ from Copy Activity?**

**Mapping Data Flow** is a feature in Azure Data Factory that allows you to visually design data transformations without writing code. It provides a graphical interface where you can build complex data transformation logic using a series of transformations. These data flows are executed on ADF-managed Spark clusters, enabling scalable data processing

**Copy Activity** is used to copy data from a source to a destination. It supports a wide range of data stores and formats, and it can handle data movement between on-premises and cloud environments. The Copy Activity is typically used for data ingestion and simple data movement tasks

**9) How can you handle incremental data loads in ADF? Explain with an example.**

1. **Define Source and Destination**:
   * **Source**: Azure SQL Database
   * **Destination**: Azure Data Lake Storage
2. **Create a Watermark Column**:
   * Add a column in your source table to track the last modified timestamp (e.g., LastModifiedDate).
3. **Set Up ADF Pipeline**:
   * **Create a Pipeline**: In ADF, create a new pipeline.
   * **Add Activities**: Add the necessary activities to the pipeline, such as Lookup, Copy Data, and Stored Procedure activities.
4. **Lookup Activity**:
   * Use a Lookup activity to get the maximum value of the LastModifiedDate from the destination table. This will serve as the **watermark**.
5. **Copy Data Activity**:
   * Configure the Copy Data activity to copy data from the source to the destination.
   * Use a dynamic query to select only the rows that have a LastModifiedDate greater than the watermark value obtained from the Lookup activity.

SELECT \* FROM SourceTable

WHERE LastModifiedDate > @Watermark

**10) What are parameters in ADF, and how are they different from variables?**

**Parameters**

* **Definition**: Parameters are defined at the pipeline level and are used to make pipelines dynamic and configurable.
* **Usage**: They are set before the pipeline run and cannot be changed during the run.
* **Purpose**: Parameters are typically used to pass values such as file paths, table names, or connection strings into a pipeline

**Variables**

* **Definition**: Variables are also defined at the pipeline level but can be modified during the pipeline run.
* **Usage**: They can be set and updated using activities like Set Variable and Append Variable.
* **Purpose**: Variables are used to store and manipulate data during the pipeline execution, such as counters, intermediate results, or states.
* **Example**: You can initialize a variable called counter and update its value within a ForEach activity to keep track of iterations.

**11)  How can you monitor and debug pipeline failures in ADF?**

1. **Monitor Tab**: Navigate to the Monitor tab and locate the failed pipeline run.
2. **Activity Details**: Click on the failed Copy activity to view the error message and stack trace.
3. **Input and Output**: Check the JSON input and output to verify the data being copied.
4. **Retry**: If the error is transient, manually retry the activity or configure a retry policy.

**12) How can ADF be integrated with Azure Databricks? Explain using a real-world use case.**

Same our project by creating databricks linked service

**13) How do you optimize an ADF pipeline for performance and cost efficiency?**

1. **Parallel Copy**: Configure the Copy Data activity to use parallelism by setting the parallelCopies property.
2. **Data Partitioning**: Use the source settings to partition the data by a key column, enabling parallel processing.
3. **Integration Runtime**: Set the TTL for the Integration Runtime to reduce startup times for sequential jobs.

**14) What are the different ways to efficiently move large volumes of data in ADF?**

**Parallel Copy**

* **Description**: ADF supports parallel copying of data, which allows you to copy data from multiple sources simultaneously.
* **Implementation**: Configure the parallelCopies property in the Copy Data activity to specify the number of parallel copies

**2. Data Partitioning**

* **Description**: Partitioning large datasets into smaller chunks can significantly improve performance by enabling parallel processing.
* **Implementation**: Use the source and sink settings in the Copy Data activity to define partitions based on key columns

**3. Incremental Loading**

* **Description**: Instead of loading the entire dataset, incremental loading involves transferring only the new or changed data since the last load.
* **Implementation**: Implement watermarking techniques to track changes and load only the incremental data

**4. Integration Runtime Optimization**

* **Auto-Scaling**: Use the auto-scaling feature of the Azure Integration Runtime to dynamically adjust the number of nodes based on the workload
* **Time to Live (TTL)**: Set the TTL for the Integration Runtime to keep the cluster alive for a specified period, reducing startup time for sequential jobs

**Azure Data Lake Storage Gen1**:

* **Definition**: A standalone service designed specifically for big data analytics. It provides a hierarchical namespace for organizing data and is optimized for high-throughput analytics workloads.

**Azure Data Lake Storage Gen2**:

* **Definition**: A unified data lake solution that combines the capabilities of Azure Data Lake Storage Gen1 with Azure Blob Storage. It supports both hierarchical and flat namespaces and is optimized for both analytics and general-purpose storage.

1. **Pricing**:
   * **Gen1**: Pay-as-you-go pricing model.
   * **Gen2**: More flexible pricing options, including hot and cold storage tiers and capacity reservations.

**What are the four main types of Azure Storage services?**

* **Blob Storage**: For storing unstructured data like images, videos, and documents.
* **File Storage**: Provides fully managed file shares in the cloud using the SMB protocol.
* **Queue Storage**: Offers a messaging solution for asynchronous communication between application components.
* **Table Storage**: A NoSQL datastore for structured data with a schemaless design

**2. How does Azure Blob Storage handle concurrency scenarios?**

* **Concurrency**: Azure Blob Storage uses ETags and access conditions to manage concurrency. When a blob is read, an ETag value representing the version of the blob is returned. To update the blob, clients must provide the original ETag in their request as an access condition

**3. What is the difference between hot, cool, and archive access tiers in Azure Blob Storage?**

* **Hot Tier**: Optimized for data that is accessed frequently.
* **Cool Tier**: Suitable for data that is infrequently accessed and stored for at least 30 days.
* **Archive Tier**: For data that is rarely accessed and stored for at least 180 days. It offers the lowest storage cost but higher retrieval costs

**4. How does Azure Data Lake Storage Gen2 differ from Gen1?**

* **Gen1**: Standalone service optimized for big data analytics with a hierarchical namespace.
* **Gen2**: Unified with Azure Blob Storage, supporting both hierarchical and flat namespaces, and offering enhanced security and integration with other Azure services

**5. What are the security features available in Azure Storage?**

* **Encryption**: Data is encrypted at rest and in transit.
* **Access Control**: Role-based access control (RBAC) and shared access signatures (SAS) for fine-grained access control.
* **Private Endpoints**: Secure access to storage accounts over a private network

**6. How can you optimize the performance of Azure Storage?**

* **Performance Optimization**: Use appropriate access tiers, enable read-access geo-redundant storage (RA-GRS) for high availability, and optimize data partitioning and indexing

**7. What is Azure Storage Explorer?**

* **Azure Storage Explorer**: A standalone app that allows you to easily work with Azure Storage data on Windows, macOS, and Linux. It provides a graphical interface for managing storage accounts, blobs, files, queues, and tables

**1)How do you implement error handling in PySpark jobs?**

import logging

from pyspark.sql import SparkSession

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

# Configure logging

logging.basicConfig(level=logging.ERROR)

logger = logging.getLogger(\_\_name\_\_)

**try:**

data = [("Alice", 34), ("Bob", 45), ("Cathy", 29)]

columns = ["Name", "Age"]

df = spark.createDataFrame(data, columns)

df.show()

# Intentionally cause an error

df.filter(df["NonExistentColumn"] > 30).show()

**except Exception as e:**

logger.error(f"An error occurred: {e}") /or **print(f"An error occurred: {e}")**

**finally:**

spark.stop()