10th-July-Azure-DBMS-Python-Questions

- ▼ 10th-July-Azure-DBMS-Python-Questions
  - 1. What is a primary key in sql
  - ▼ 2. What is a join in sql?
    - 1. INNER JOIN
    - 2. LEFT JOIN (or LEFT OUTER JOIN)
    - 3. RIGHT JOIN (or RIGHT OUTER JOIN)
    - 4. FULL OUTER JOIN (or FULL JOIN)
  - 3. Suppose I want to combine two tables, including all columns from both. What should I do?
  - 4. What is the difference between delete and truncate
  - ▼ 5. What is foreign key, difference between primary and foreign key
    - Difference Between Primary Key and Foreign Key
  - 6. What is python?
  - 7. What are the common datatypes in python
  - 8. In ADF what is copy activity
  - ▼ 9. What is a data flow?
    - Types of Data flow
    - Components of a Mapping Data Flow:
  - ▼ 10. For executing pipeline, what techniques to use.
    - 1. Manual Execution (Trigger Now)
    - ▼ 2. Trigger-Based Execution
      - a. Schedule Trigger
      - b. Tumbling Window Trigger
      - c. Event-Based Trigger (Storage Event Trigger, Custom Event Trigger)
    - 3. Programmatic Execution (APIs)
    - 4. Execute Pipeline Activity (within another pipeline)
  - 11. What are the major key components in ADF
  - ▼ 12. Difference between linked service and dataset.

- Example
- ▼ 13. What is an integration runtime? Which case we have to use that?
  - Types of integration runtimes
  - The Default Integration Runtime
- ▼ 14. What type of data sources does ADF support?
  - General Categories of Data Sources Supported by ADF:
- 15. Can we secure data in ADLS?
- 16. How to handle incremental loading in ADF?
- 17. How do you monitor all jobs?
- 18. What is CI/CD in azure
- ▼ 19. What is the difference between debug and trigger
  - 1. Debug Run
  - 2. Triggered Run (including "Trigger Now" / Manual Trigger)
- 20. What is lookup activity in ADF
- ▼ 21. What is a pipeline in azure?
  - ▼ Types of ADF Pipelines
    - Data Ingestion/Copy Pipelines:
    - Data Transformation Pipelines (ETL/ELT)
    - Orchestration/Control Flow Pipelines:
- 22. Examples of linked service
- 23. Can we copy all data from ADF to any other source
- 24. What is the difference between dataset and pipeline
- 25. Difference between where and having
- 26. What is cross join in sql
- 27. What is variable in python
- 28. Write a loop in python (for loop, while loop)
- 29. Double and single difference?
- ▼ 30. Staging in copy activity, What happens when its disabled
  - Example Scenario for Disabling Staging
  - Why data staging?
  - Why would you disable staging?
- ▼ 31. What is the difference between triggers and pipelines
  - Pipeline

1. What is a primary key in sql

- A Primary key
    - Uniquely identifies each row in a table.
    - Cannot contain duplicate values.
    - Cannot be empty (NULL).
    - Helps link tables together.

2. What is a join in sql?

- In SQL, a **JOIN** is used to combine rows from two or more tables based on a related column between them.
- Imagine you have separate lists of information, and you want to bring specific details from one list to another based on something they have in common. That's what a join does.

Here are the main types of joins:

1. INNER JOIN

- Returns only the rows where there's a **match in both tables** based on the specified join condition.
- Use When you only want to see data where a relationship exists in *both* tables.
- **Example:** If you have a `Customers` table and an `Orders` table, an `INNER JOIN` would show you only the customers who have placed orders, and their corresponding order details. Customers without orders and orders without a matching customer would be excluded.

```
SELECT C.CustomerID, C.CustomerName, O.OrderID
FROM Customers C
INNER JOIN Orders O ON C.CustomerID = O.CustomerID;
```

2. LEFT JOIN (or LEFT OUTER JOIN)

- Returns **all rows from the left table**, and the **matching rows from the right table**.
- If there's no match in the right table, it will still include the row from the left table but show `NULL` values for the columns from the right table.
- Use When you want to see all entries from one main table, even if they don't have related entries in another table.
- **Example:** Using `Customers` (left table) and `Orders` (right table), a `LEFT JOIN` would show *all* customers, even those who haven't placed any orders. For customers without orders, the `OrderID` column (from the `Orders` table) would be `NULL`.

```
SELECT C.CustomerID, C.CustomerName, O.OrderID
FROM Customers C
LEFT JOIN Orders O ON C.CustomerID = O.CustomerID;
```

3. RIGHT JOIN (or RIGHT OUTER JOIN)

- This is the opposite of a `LEFT JOIN`. It returns **all rows from the right table**, and the **matching rows from the left table**.
- If there's no match in the left table, it will include the row from the right table but show `NULL` values for the columns from the left table.

- **When to use it:** When you want to see all entries from one main table (the right one), even if they don't have related entries in another table.
- **Example:** Using `Customers` (left table) and `Orders` (right table), a `RIGHT JOIN` would show *all* orders, even if an order somehow doesn't have a matching customer.
- If a customer placed an order, their `CustomerID` and `CustomerName` would appear. If an order had no matching customer, those customer columns would be `NULL`.

```
SELECT C.CustomerID, C.CustomerName, O.OrderID
FROM Customers C
RIGHT JOIN Orders O ON C.CustomerID = O.CustomerID;
```

4. FULL OUTER JOIN (or FULL JOIN)

- Returns **all rows when there is a match in either the left or the right table**.
- It combines the results of both `LEFT JOIN` and `RIGHT JOIN`. If there's no match in one table, `NULL` values are shown for columns from that table.
- **When to use it:** When you want a complete picture of all data from both tables, regardless of whether a direct relationship exists.
- - **Example:** Using `Customers` and `Orders`, a `FULL OUTER JOIN` would show:
    - Customers with orders + their order details.
    - Customers without orders (with `NULL` for order details).
    - Orders without customers (with `NULL` for customer details).

```
SELECT C.CustomerID, C.CustomerName, O.OrderID
FROM Customers C
FULL OUTER JOIN Orders O ON C.CustomerID = O.CustomerID;
```

3. Suppose I want to combine two tables, including all columns from both. What should I do?

- You should use a FULL OUTER JOIN.
- Why Full Outer Join? A FULL OUTER JOIN is designed to return all rows when there's a match in either the left table or the right table.

- If a row in one table doesn't have a match in the other, it will still be included, and the columns from the non-matching table will show NULL values. This ensures you get "all" the data from both tables.

```
SELECT
    Table1.A1,
    Table1.A2,
    Table2.B1,
    Table2.B2
FROM
    Table1
FULL OUTER JOIN
    Table2 ON Table1.CommonID = Table2.CommonID;
```

4. What is the difference between delete and truncate

DELETE and TRUNCATE are both SQL commands used to remove data from tables, but they operate very differently and are suited for different scenarios.

- 1. **What they remove**:
  - **DELETE:** Removes **rows** from a table. You can remove specific rows using a `WHERE` clause, or all rows if no `WHERE` clause is specified.
  - **TRUNCATE:** Removes **all rows** from a table. It cannot be used to delete specific rows.
- 2. **How they work**
  - **DELETE:** This is a **Data Manipulation Language (DML)** command. It removes rows one by one. This means it records each deleted row in the transaction log.
  - **TRUNCATE:** This is a **Data Definition Language (DDL)** command.
- 3. **Speed/Performance:**
  - **DELETE:** Generally **slower**, especially for large tables, because it processes deletions row by row and logs each one.
  - **TRUNCATE:** Significantly **faster** for large tables because it deallocates entire data pages, which is a much quicker operation than individual row deletions.
- 4. **Transaction Logging & Rollback:**
  - **DELETE:** Fully logged in the transaction log. This means you can **ROLLBACK** a `DELETE` operation if it's part of an active transaction.

    - **TRUNCATE:** Minimally logged. In most database systems, a `TRUNCATE` operation **cannot be rolled back** because it's a DDL command that often involves an implicit commit. Some modern SQL databases (like SQL Server and PostgreSQL) do allow `TRUNCATE` to be transactional and thus roll backable, but this is an exception rather than the rule for TRUNCATE.
- **5. `WHERE` clause:**
    - **DELETE: Supports** a `WHERE` clause to specify which rows to delete.
    - **TRUNCATE: Does NOT support** a `WHERE` clause. It's an "all or nothing" operation.

5. What is foreign key, difference between primary and foreign key

- A foreign key is a concept in relational databases that establishes a link between two tables.
- It's how you define relationships between different pieces of data.
- A foreign key is a column or a set of columns in one table that refers to the primary key of another table\

Example:

Consider two tables:

1. `Customers` Table (Parent Table)

CustomerID (PK)	CustomerName
1	Alice
2	Bob

2. `Orders` Table (Child Table)

OrderID (PK)	CustomerID (FK)	OrderDate
101	1	2024-01-10
102	1	2024-01-15
103	2	2024-01-20

In the `Orders` table, `CustomerID` is a **foreign key** that references the `CustomerID` (which is a **primary key**) in the `Customers` table.

Difference Between Primary Key and Foreign Key

Feature	Primary Key (PK)	Foreign Key (FK)
Purpose	**Uniquely identifies** each row within its own table.	Establishes a **link/relationship** to a primary key in another table.
Uniqueness	**Must be unique** across all rows in the table.	**Can contain duplicate values** (multiple orders can belong to one customer).
NULL Values	**Cannot be NULL**. Every row must have a PK value.	**Can be NULL** (if a relationship is optional, e.g., a customer might not have an order yet).
Number per Table	Only **one** primary key per table.	A table can have **multiple** foreign keys.
Role	The "identifier" of a record.	The "connector" or "pointer" to another table's record.
Referenced By	Can be referenced by foreign keys in other tables.	**References** a primary key (or unique key) in another table.

6. What is python?

- Python is a high-level, interpreted, general-purpose programming language.
- Python is designed to be easy for humans to read and write. You don't have to worry about low-level details like memory management
- **Interpreted:** This means that Python code is executed line by line by an interpreter, rather than being compiled into machine code beforehand.
- General-purpose
  - Web development: (e.g., Django, Flask)
  - Data analysis and machine learning: (e.g., Pandas, NumPy, scikit-learn, TensorFlow, PyTorch)
  - Automation and scripting: Automating repetitive tasks, system administration
  - Desktop applications: (e.g., PyQt, Tkinter)
- Some data types

- **int (Integers):** Whole numbers (e.g., 5, -100, 0).
- **float (Floating-point numbers):** Numbers with a decimal point (e.g., 3.14, -0.5, 2.0).
- **bool:** Represents truth values. Only two possible values: True or False.
- **str (Strings):** Sequences of characters, used for text. Enclosed in single or double quotes (e.g., "hello", 'Python').
- **list:** Ordered, mutable (changeable), and allows duplicate elements. Enclosed in square brackets [] (e.g., [1, 2, 'apple', True]).
- **tuple:** Ordered, immutable (unchangeable), and allows duplicate elements. Enclosed in parentheses () (e.g., (10, 20, 'banana')).
- **set:** Unordered, mutable, and does not allow duplicate elements. Enclosed in curly braces {} (e.g., {1, 2, 3, 'a'}).
- **dict (Dictionary):** Unordered collection of key-value pairs. Keys must be unique and immutable. Values can be of any type. Enclosed in curly braces {} (e.g., {'name': 'Alice', 'age': 30}).

7. What are the common datatypes in python

1. **Numbers:**
   - `int` (Integers): Whole numbers (e.g., `10`, `-5`, `0`).
   - `float` (Floating-point numbers): Numbers with a decimal point (e.g., `3.14`, `-0.5`)
2. **Booleans:**
   - `bool`: Represents truth values; either `True` or `False`
3. **Sequences:** (Ordered collections)
   - `str` (Strings): Sequences of characters, used for text (e.g., `"hello"`, `'Python'`).
   - `list`: Ordered, mutable (changeable) collections of items (e.g., `[1, "apple", True]`).
   - `tuple`: Ordered, immutable (unchangeable) collections of items (e.g., `(10, 20, "banana")`).
4. **Sets:** (Unordered collections of unique items)
   - `set`: Unordered, mutable, and contains only unique elements (e.g., `{1, 2, 3}`).
   - `frozenset`: An immutable version of a set.
5. **Mappings:**

- `dict` (Dictionary): An unordered (or insertion-ordered from Python 3.7+) collection of key-value pairs (e.g., `{'name': 'Alice', 'age': 30}`).

6. **None Type:**
   - `NoneType`: Represents the absence of a value. The only value is `None`.

8. In ADF what is copy activity

- **What is Copy Activity**
  - In Azure Data Factory (ADF), the Copy Activity is a fundamental and widely used activity within a pipeline.
  - Its primary purpose is to copy data from a source data store to a destination (sink) data store.
- **What it does**
  - Reads Data from Source: The Copy Activity connects to your specified source data store (e.g., an Azure SQL Database, Azure Blob Storage etc).
  - Performs Data Transformation (Optional/Basic)
    - Schema Mapping: Mapping columns from the source to different column names in the sink.
    - Data Type Conversion: Converting data types between source and sink if needed.
    - Converting between different data formats (e.g., CSV to Parquet, JSON to Avro).
  - Writes Data to Sink: It then writes the data to your specified destination data store.
- **Integration Runtime (IR): The Copy Activity executes on an Integration Runtime (IR).**
  - Azure IR: For copying data between publicly accessible cloud data stores.
  - Self-Hosted IR: For connecting to data stores located on-premises or within a private network (like an Azure Virtual Network).

9. What is a data flow?

- A Data Flow is a visually designed data transformation activity that allows you to build complex ETL (Extract, Transform, Load) logic without writing any code
- It's built on Apache Spark, which provides a scalable and performant engine for data processing.

- Instead of writing custom code (like Python, Scala, or SQL scripts), you use a drag-and-drop interface to create a sequence of transformations that visually represent how your data flow

Types of Data flow

There are primarily two types of Data Flows in ADF:

- **Mapping Data Flow:**
  - The most common and powerful type, designed for building scalable data transformation logic.
- **Wrangling Data Flow:**
  - Built on the Power Query experience, it's more geared towards data preparation, exploration, and cleansing, often by data analysts.

Components of a Mapping Data Flow:

- **1. Source Transformation:**
  - Purpose: This is the starting point of your data flow. It defines where your data originates.
  - Configuration: You specify the dataset, schema drift settings, and other source-specific options.
  - Examples: Azure Blob Storage, Azure Data Lake Storage Gen2, Azure SQL Database, Azure Synapse Analytics etc.
- **2. Sink Transformation:**
  - Purpose: This is the final destination for your transformed data. It defines where the output of your data flow will be written.
  - Configuration: You specify the dataset for the destination, how the data should be written
  - Examples: Same as sources, like Azure SQL Database, Azure Data Lake Storage, Azure Synapse Analytics, etc.
- **3. Transformations (The Core Logic):**
  - These are the building blocks that manipulate your data as it flows from source to sink. You connect them sequentially to form your transformation pipeline
  - **Select**
    - Purpose: To choose a subset of columns, rename columns, reorder columns, or drop unwanted columns from your data stream.

        - Example: Selecting only FirstName, LastName, and Email from a customer table, and renaming Email to ContactEmail.
    - **Filter:**
        - Purpose: To filter rows based on a specified condition. Only rows that meet the condition pass through.
        - Example: Filtering out all customers whose Region is not 'USA', or orders with a TotalAmount less than 100.
    - **Aggregate:**
        - Purpose: To perform aggregations like SUM, AVG, COUNT, MIN, MAX, grouped by one or more columns (similar to GROUP BY in SQL).
        - Example: Calculating the total sales per product category or the average order value per customer.
    - **Join:**
        - Purpose: To combine data from two different streams (tables) based on matching column values (like INNER, LEFT, RIGHT, FULL OUTER joins in SQL).
        - Example: Joining Customers data with Orders data on CustomerID.
    - **Conditional Split:**
        - Purpose: To route rows into different streams based on specified conditions (like an IF-ELSE IF-ELSE statement).
        - Example: Splitting customer data into 'HighValueCustomers' and 'StandardCustomers' streams based on their total purchase amount.
    - **Union:**
        - Purpose: To combine rows from two or more streams vertically, where columns align (like UNION in SQL).
        - Example: Combining sales data from different regions into a single stream.
    - **Sort:**
        - Purpose: To sort the data within a stream based on one or more columns.
    - **Pivot/Unpivot:**
        - Purpose: To transform data from rows to columns (Pivot) or columns to rows (Unpivot). Useful for reshaping data for reporting or analysis.
    - **Window:**
        - Purpose: To perform windowing functions (like RANK, DENSE_RANK, NTILE, LAG, LEAD) for analytics over a defined window of data.
- **How they work?**

- When you build a Data Flow in ADF, you are visually defining a series of transformations.
- ADF then translates this visual representation into Apache Spark code. When you execute a pipeline containing a Data Flow activity, ADF provisions and manages a Spark cluster to run that Spark code, processing your data at scale.

10. For executing pipeline, what techniques to use.

1. Manual Execution (Trigger Now)

- This is the simplest way to run a pipeline immediately. You go into the ADF Studio, open your pipeline, and click the "Trigger now" button.
- **Use Cases:**
  - **Testing and Debugging:** Ideal for quickly testing changes to your pipeline during development.
  - **Ad-hoc Runs:** For one-off data processing tasks that don't need a recurring schedule.
  - **Initial Loads:** When you need to load historical data once.
- **Characteristics:**
  - Immediate execution.
  - No scheduling involved.
  - Often used with parameters to specify input for a particular run.

2. Trigger-Based Execution

- Triggers are the core of automation in ADF. They define *when* a pipeline should execute automatically.
- ADF supports several types of triggers:

a. Schedule Trigger

- Invokes a pipeline on a schedule (e.g., hourly, daily, weekly, monthly, or specific times/days).
- **Use Cases:**
  - Running a pipeline every morning to process data from the previous day.
  - **Hourly data refreshes:** Updating dashboards or reports every hour.

- **Regular maintenance tasks:** Performing cleanup or archival operations periodically.

b. Tumbling Window Trigger

- Fires at a periodic interval from a specified start time, operating on fixed-sized, non-overlapping, and contiguous time intervals (windows).
- **Use Cases:**
  - **Processing time-series data:** When you need to process data for specific, continuous time slices (e.g., hourly sales data, daily sensor readings).
  - **Historical data backfills:** Efficiently processing data for past time windows.
- **Characteristics:**
  - Fixed-size windows (e.g., 1 hour, 24 hours).
  - Can configure dependencies on other tumbling window triggers (e.g., ensure pipeline B runs only after pipeline A completes for the same window).

c. Event-Based Trigger (Storage Event Trigger, Custom Event Trigger)

- **Storage Event Trigger:**
  - Fires a pipeline when a specific event occurs in an Azure Storage account (Blob Storage or ADLS Gen2), such as a file being created, deleted, or reaching a certain size.
- **Custom Event Trigger:**
  - Fires a pipeline in response to custom events published to Azure Event Grid.
- **Use Cases:**
  - **Real-time data ingestion:** Automatically processing a file as soon as it lands in a blob container.
  - **Responding to data changes:** Triggering a pipeline when a specific database event occurs (via Event Grid).

3. Programmatic Execution (APIs)

- You can programmatically initiate pipeline runs using APIs.

4. Execute Pipeline Activity (within another pipeline)

- This is a control activity within an ADF pipeline itself.
- It allows one pipeline (the parent) to invoke another pipeline (the child) as an activity.

- **Use Cases:**
  - **Modularization:** Breaking down large, complex data flows into smaller, reusable child pipelines.
  - **Orchestration of workflows:** Creating a master pipeline that controls the execution order and dependencies of several sub-pipelines.
- **Characteristics:**
  - Allows passing parameters between parent and child pipelines.

11. What are the major key components in ADF

- **1. Pipelines:**
  - A pipeline is a logical grouping of activities that perform a unit of work. Think of it as your overall workflow.
  - They define the sequence and dependencies of various data movement and transformation steps.
  - A pipeline might involve copying data from Blob Storage, then transforming it using a Data Flow, and finally loading it into a data warehouse.
- **2. Activities:**
  - Activities are the individual processing steps within a pipeline. Each activity performs a specific action.
  - **Types:**
    - **Data Movement Activities:** Like the **Copy Activity**, which moves data between data stores.
    - **Data Transformation Activities:** Like **Data Flows**, Azure Databricks Notebook activity, Stored Procedure activity, etc., which transform data.
    - **Control Flow Activities:** These manage the flow of the pipeline, such as `For Each` loops, `If Condition`, `Wait`, `Execute Pipeline`, `Web` activity, etc.
- **3. Datasets:**
  - Datasets are named references to the data you want to use in your activities. They define the structure, format, and location of the data.
  - They act as inputs and outputs for activities, abstracting the underlying data storage details.

- **Example:** A dataset could represent a CSV file in Azure Blob Storage, a table in an Azure SQL Database, or a specific REST API endpoint.

- **4. Linked Services:**
  - Linked services are like connection strings or credentials that define the connection information ADF needs to access external resources.
  - They link your ADF to external data stores (like databases, storage accounts)
  - **Example:** A linked service for Azure SQL Database contains the server name, database name, authentication details
- 5. **Integration Runtimes (IRs):**
  - Integration Runtimes are the compute infrastructures that ADF uses to execute activities.
  - **Types:**
    - **Azure Integration Runtime (Azure IR):** A fully managed, serverless compute environment in Azure. Used for data movement between cloud data stores.
    - **Self-Hosted Integration Runtime (SHIR):** An agent you install on an on-premises machine or a VM in a private network. Used for copying data between on-premises data stores and cloud data stores, or for running activities that access private network resources.
    - **Azure-SSIS Integration Runtime:** A dedicated cluster to natively execute SQL Server Integration Services (SSIS) packages in Azure.
  - **Purpose:** They provide the necessary compute power and network connectivity for ADF activities to run.
- 6. **Triggers:**
  - Triggers determine when a pipeline execution should be initiated automatically.
  - **Purpose:** They automate your data pipelines based on schedules, events, or cascading windows.
  - **Types:**
    - **Schedule Triggers:** For executing pipelines at specific times or recurring intervals (e.g., daily at 2 AM).
    - **Tumbling Window Triggers:** For executing pipelines over fixed-size, non-overlapping, contiguous time intervals, often used for historical data processing.
    - **Event-Based Triggers:** For reacting to specific events, like a new file arriving in Azure Blob Storage.

12. Difference between linked service and dataset.

- Linked Services and Datasets are two distinct but interconnected components that work together to enable data movement and transformation.
- Linked Service = The Connection Information (How to connect)
  - A Linked Service is essentially a connection string that defines how Azure Data Factory can connect to an external resource.
  - It holds all the necessary connection details and credentials to authenticate with a data store or a compute service.
- Dataset = The Data Structure and Location (What data to use)
  - A Dataset is a named reference to a specific piece of data within a data store that is already connected via a Linked Service.
  - It defines the structure (schema), format, and specific location of the data
- You first tell ADF how to get to a data source (Linked Service), and then you tell it exactly what data you want to use from that source (Dataset).

Example

Scenario: Copying Sales Data

- Imagine you have daily sales data arriving as a CSV file in an Azure Blob Storage container.
- You want to copy this data into a SalesData table in your Azure SQL Database for reporting and analysis.
- Step 1: Define the Linked Services (The Connections)
  - First, you need to tell ADF how to connect to both your Azure Blob Storage and your Azure SQL Database.
- Step 2: Define the Datasets (The Specific Data)
  - describe the CSV file in Blob Storage that you want to read

13. What is an integration runtime? Which case we have to use that?

- An Integration Runtime (IR) in Azure Data Factory (ADF) is the compute infrastructure that ADF uses to perform all its data integration capabilities.
- It's the bridge between your ADF pipelines (which define what to do) and your linked services (which define how to connect to data stores). The IR provides the environment

where the actual work of reading, processing, and writing data happens

Types of integration runtimes

- Azure Integration Runtime (Azure IR)
- Self-Hosted Integration Runtime (SHIR)
- Azure-SSIS Integration Runtime

The Default Integration Runtime

- When you create a new Azure Data Factory instance, ADF automatically provisions a default Azure Integration Runtime.
- It's named AutoResolveIntegrationRuntime by default.

14. What type of data sources does ADF support?

General Categories of Data Sources Supported by ADF:

- **1. Azure Storage Services:**
  - **Azure Blob Storage:** For unstructured data, often used as a landing zone for raw data.
  - **Azure Data Lake Storage Gen2 (ADLS Gen2):** It **natively supports hierarchical namespaces**, which is built on top of Azure Blob Storage. ADF fully leverages this.
  - Azure Files
  - Azure Table Storage
- **2. Azure Relational Databases:**
  - Azure SQL Database
  - Azure SQL Managed Instance
  - Azure Synapse Analytics (formerly Azure SQL Data Warehouse)
  - Azure Database for MySQL
  - Azure Database for PostgreSQL
  - Azure Database for MariaDB
- **3. Azure NoSQL Databases & Analytics Services:**
  - Azure Cosmos DB (for NoSQL, MongoDB, Cassandra, Gremlin, Table APIs)
  - Azure Data Explorer (Kusto)

- **4. Generic Protocols and Formats:**
  - **Generic HTTP/REST:** For connecting to almost any web-based API.
  - **Generic ODBC/OLE DB:** For connecting to various databases that support these standards.
  - **FTP/SFTP:** For file transfers.
  - **Data Formats:** ADF also supports various file formats within these storage accounts:
    - Delimited Text (CSV, TSV, etc.)
    - JSON
    - Parquet
    - ORC
    - Avro
    - XML
    - Binary (for copying files as-is)
    - Excel

15. Can we secure data in ADLS?

- Here are the key ways you can secure data in ADLS Gen2:
  - **Authentication:**
    - **Microsoft Entra ID (formerly Azure Active Directory) Integration**
      - This is the most recommended method. You can authenticate users, groups, and service principals (for applications) from your Microsoft Entra ID. This allows you to manage identities centrally and leverage features like Multi-Factor Authentication (MFA), Conditional Access, etc.
    - **Shared Access Signatures (SAS):**
      - You can create a SAS token that grants limited permissions (e.g., read-only access to a specific container for a limited time).
      - This is useful for providing temporary, scoped access without sharing your full account keys
  - **Authorization (Access Control):**
    - Azure Role-Based Access Control (Azure RBAC)
    - This provides coarse-grained access control at the storage account and container (file system) level

16. How to handle incremental loading in ADF?

- Incremental loading is a crucial technique in data warehousing and ETL processes to efficiently update target systems with only new or changed data, rather than reloading the entire dataset every time.
- Azure Data Factory (ADF) provides several robust ways to handle incremental loading
  - **1. Change Data Capture (CDC) - Native ADF Feature**
    - ADF offers a native Change Data Capture (CDC) resource
    - Configure Source & Sink: You specify your source (e.g., Azure SQL Database, SQL Server, certain file types) and your target (e.g., ADLS Gen2, Azure SQL Database).
    - Automatic Mapping & Application: The CDC resource automatically detects and applies inserts, updates, and deletes from source to target. You configure latency (e.g., real-time, 15 minutes, hourly).
  - **2. Delta lakes**
    - Source Data in Delta Lake: If your source is already a Delta Lake table, ADF's Data Flow can directly read changes from it because Delta Lake tables maintain transaction logs.
    - You can specify "Incremental Load" in the Data Flow Source transformation

17. How do you monitor all jobs?

- **1. The Monitor Tab in ADF Studio**
  - The "Monitor" tab (or "Monitor hub") in your Azure Data Factory Studio is your primary go-to place for overseeing all pipeline and activity runs.
  - Key Sections and Features:
    - **Triggered Runs:** This is where you see all pipeline runs initiated by triggers (schedule, tumbling window, event-based) or through manual "Trigger now" actions.
    - **Debug Runs:** This section displays runs initiated when you "Debug" a pipeline directly from the canvas. These runs do not count towards billing and are purely for testing and development.
    - Activity Runs:

        - Status of Integration Runtimes
- **2.Proactive Monitoring (Alerts):**
    - Metric Alerts:
        - You can set up alerts based on various ADF metrics (e.g., "Failed Pipeline Runs," "Failed Activity Runs," "DIU Usage," "Integration Runtime CPU Usage").
        - You define a threshold (e.g., "If Failed Pipeline Runs count is greater than 0 in the last 5 minutes").
        - When the condition is met, an Action Group is triggered, which can send email notifications, SMS messages, push notifications to the Azure mobile app, trigger Azure Functions, Logic Apps, or webhooks.

18. What is CI/CD in azure

- CI/CD in Azure refers to the practices of Continuous Integration (CI) and Continuous Delivery (CD) (or Continuous Deployment) within the Microsoft Azure cloud ecosystem
- **1. Continuous Integration (CI)**
    - CI is a development practice where developers frequently integrate their code changes into a shared version control repository (like Azure Repos Git or GitHub).
    - Every time code is committed or merged, an automated process is triggered to build the application and run automated tests (unit tests, integration tests)
- **2. Continuous Delivery (CD) / Continuous Deployment (CD)**
    - What it is: CD extends CI by automating the process of releasing the validated code.
    - Continuous Delivery: After successful CI, the software is automatically built, tested, and prepared for a release to production.
    - Continuous Deployment: This goes a step further. Every change that successfully passes all automated tests and quality gates in the pipeline is automatically deployed to the production environment without any manual intervention

19. What is the difference between debug and trigger

In Azure Data Factory (ADF), "Debug" and "Trigger" refer to two distinct ways of executing a pipeline, primarily differing in their purpose, behavior, and the environment they run against.

1. Debug Run

- Development and Testing:
  - Debug runs are specifically designed for iterative development, testing, and troubleshooting of pipelines before publishing them to the live Data Factory service.
- Key Characteristics:
  - You don't need to publish your changes for a debug run.
  - For Mapping Data Flows, there's a dedicated "Debug Mode" that spins up an interactive Spark cluster, allowing you to visually inspect data transformations at each step without actually writing to a sink.

2. Triggered Run (including "Trigger Now" / Manual Trigger)

- **Production Execution:** Triggered runs are intended for running pipelines in a production or scheduled environment.
- **Automation and Orchestration:** They are the mechanism for automating pipeline execution based on schedules, events, or dependencies.
- Key Characteristics:
  - Published Code:
    - Triggered runs always execute the last published version of your pipeline in the ADF service.
    - If you have unsaved or unpublished changes on your canvas, a triggered run will not use those changes.
    - You must explicitly "Publish" your changes for them to be picked up by triggers.
  - Associated with Triggers: Triggered runs are initiated by:
    - Scheduled Triggers: At specific times (e.g., daily, hourly).
    - Tumbling Window Triggers: Over fixed, non-overlapping time windows.
    - Event-Based Triggers: In response to events (e.g., file arrival in Blob Storage).
    - Manual Trigger ("Trigger Now"):

20. What is lookup activity in ADF

- The Lookup Activity in Azure Data Factory (ADF) is a control flow activity used to retrieve a dataset from any of the data sources supported by ADF

21. What is a pipeline in azure?

A Pipeline in ADF is a logical grouping of activities that together perform a unit of work or a complete data-driven workflow. It defines the sequence, control flow, and orchestration of tasks. It's the "how" of your data integration process.

Types of ADF Pipelines

Data Ingestion/Copy Pipelines:

- Primarily focused on moving data from various sources to a destination, often a data lake or staging area.
- Common Activities:
  - Copy Activity

Data Transformation Pipelines (ETL/ELT)

- Purpose: Focused on cleaning, reshaping, enriching, and aggregating data.
- Common activities
  - Data Flow Activity: For visual, code-free data transformations at scale using Spark.
  - Stored Procedure Activity: To execute SQL stored procedures in databases.
  - Azure Function Activity: To run custom code for specific transformation logic.

Orchestration/Control Flow Pipelines:

- To manage the flow of other pipelines or activities, implement complex logic, and handle dependencies.
- Common Activities:
  - **Execute Pipeline Activity:** To call and execute other (child) pipelines. This is crucial for modularity.
  - **For Each Activity:** To iterate over a list of items (e.g., tables, files) and execute a set of activities for each item.
  - **If Condition Activity:** To execute different branches of activities based on a condition.
  - **Lookup Activity:** To fetch configuration data or metadata to drive control flow.
  - **Web Activity:** To make calls to REST APIs.
  - **Wait Activity:** To pause pipeline execution for a specified duration.
  - **Set Variable Activity:** To set the value of pipeline variables.

22. Examples of linked service

- **1. Azure Blob Storage Linked Service**
  - This Linked Service provides the full connection string, including the storage account name and access key, allowing ADF to read from or write to any container or blob within that specific storage account.
- **2. Azure Data Lake Storage Gen2 Linked Service**
- **3. Azure SQL Database Linked Service**

23. Can we copy all data from ADF to any other source

- Yes
- **Universal Connector Support:**
  - ADF boasts a vast library of connectors for various data stores, both in the cloud and on-premises. This includes:
    - Cloud Data Stores:
    - On-premises Data Stores
- **Dataset Definition:**
  - For both your source and your destination, you'll define a Dataset. The dataset specifies the location, format, and structure of the data.
- **Flexible Data Formats:**
  - ADF's Copy Activity can handle various data formats and even convert between them during the copy process (serialization/deserialization):
  - Delimited Text (CSV, TSV)
  - JSON
  - Parquet
  - XML etc

24. What is the difference between dataset and pipeline

- Dataset

    - A Dataset in ADF is a named view or a logical representation of the data you want to use within your Data Factory.
    - It essentially defines the structure, format, and location of data within a Linked Service. It's the "what" and "where" of your data.
- Pipeline
    - A Pipeline in ADF is a logical grouping of activities that together perform a unit of work or a complete data-driven workflow. It defines the sequence, control flow, and orchestration of tasks. It's the "how" of your data integration process.
- **Linked Services define connections**, **Datasets define data structures and locations within those connections**, and **Pipelines define the sequence of operations** that act upon those datasets using various Activities.

25. Difference between where and having

- WHERE
    - Purpose: Filters individual rows from the table(s) before any grouping or aggregation takes place.
- HAVING
    - Purpose: Filters groups of rows (or aggregated results) after the grouping and aggregation have occurred.

26. What is cross join in sql

- A CROSS JOIN in SQL is a type of join that produces the Cartesian product of two or more tables.
- This means that every row from the first table is combined with every row from the second table.
- If there are 'm' rows in the first table and 'n' rows in the second table, a CROSS JOIN will result in 'm * n' rows
- No Join Condition: Unlike INNER JOIN, LEFT JOIN, or RIGHT JOIN, a CROSS JOIN does not require an ON clause or any join condition. It inherently combines all rows.

```
SELECT
    C.ColorName,
    S.SizeName
FROM
    Colors AS C
CROSS JOIN
    Sizes AS S;
```

27. What is variable in python

- A variable is essentially a named storage location for a value. It's a way to label data with a meaningful name, so you can refer to it later in your code.
- Python is a dynamically typed language. This means you don't need to declare the data type of a variable before you use it.
- Python automatically infers the data type based on the value you assign to it at runtime.
- The single equals sign (=) is the assignment operator. It's used to assign a value to a variable.
- Variables have a scope, which defines where in your code they can be accessed. Common scopes include:
    - Local: Defined inside a function; only accessible within that function.
    - Global: Defined at the top level of a module; accessible throughout the module.
    - Enclosing (Nonlocal): In nested functions, accessible in the inner function from the outer function's scope
    - Built-in: Predefined names in Python (e.g., print, len).

28. Write a loop in python (for loop, while loop)

```
for item in iterable:
    # code to be executed for each item
```

```
while condition:
    # code to be executed as long as the condition is True
```

```
    # (make sure to include code that eventually makes the condition False
to avoid infinite loops)
```

29. Double = and single= difference?

- **1. = (Single Equals Sign): The Assignment Operator**
- **2. == (Double Equals Sign): The Equality Comparison Operator**

30. Staging in copy activity, What happens when its disabled

- In ADF's Copy Activity, staging refers to an intermediate storage location (usually an Azure Blob Storage or Azure Data Lake Storage Gen2 account) that ADF uses to facilitate data movement
- When staging is enabled, the Copy Activity follows a two-step process:
  - Step 1 (Staging):
    - Data is first read from the source and written (staged) to the intermediate storage account.
  - Step 2 (Loading):
    - Data is then loaded from the intermediate storage to the final sink. ADF typically cleans up the staged data after successful completion.

Example Scenario for Disabling Staging

- Imagine you are copying data from an Azure SQL Database directly to Azure Blob Storage (as a CSV file).
- Scenario A: Staging Enabled (Default/Recommended for some sources/sinks)
  - Source: Azure SQL Database
  - Staging: Azure Blob Storage Account A (temporary staging)
  - Sink: Azure Blob Storage Account B (final destination)
- Scenario B: Staging Disabled (Direct Copy)
  - Source: Azure SQL Database
  - Sink: Azure Blob Storage (final destination)

Why data staging?

- It needs to optimize performance for specific data stores
    - For example, when copying data into Azure Synapse Analytics (SQL Pool) or Azure SQL Database, using PolyBase or the bulk insert API often requires the data to first be staged in Blob Storage.
- It needs to perform format conversions:
    - If the source and sink formats are incompatible or require complex conversion, ADF might stage the data to convert it into an intermediate format (like Parquet) before loading it into the final destination.
- It needs to handle compression:
    - If the source data is compressed or the sink requires compression, staging might be used

Why would you disable staging?

You would typically disable staging (or choose not to enable it in the first place) in the following cases:

- Simplicity and Directness:
- Cost Optimization:
    - Staging involves an additional write operation (to the staging area)
    - potentially an additional read operation (from the staging area to the sink).
    - This can incur extra storage transaction costs. Disabling staging eliminates these extra transactions.
    - It also reduces the total copy duration for simple scenarios, which can translate to lower Data Integration Unit (DIU) consumption costs.

31. What is the difference between triggers and pipelines

Pipeline

- A pipeline is the fundamental unit of execution in ADF.
- It's a logical grouping of activities that together perform a specific data-driven task or workflow.
- **Container for Activities:** A pipeline is composed of one or more Activities.

    - Activities are the individual steps (e.g., Copy Data Activity, Data Flow Activity, Stored Procedure Activity, Get Metadata Activity, Web Activity).
- **Parameters:** Pipelines can be parameterized, allowing you to pass dynamic values into them at runtime, making them reusable and flexible.
- **Not Self-Executing:** A pipeline, by itself, doesn't "run" automatically. It's a definition of a workflow that needs to be initiated.

Trigger

- A trigger is the mechanism that initiates (or "kicks off") a pipeline run based on a specified schedule or event. It's the "when" and "why" a pipeline starts.
- **Initiator of Pipeline Runs:** When a trigger fires, it creates a new "pipeline run" instance. Each run has a unique Run ID.
- **Schedule Trigger:** Runs pipelines on a recurring, wall-clock schedule (e.g., every day at 2 AM, every hour, every Monday).
- **Tumbling Window Trigger:** Fires at a periodic interval from a specified start time, creating non-overlapping, fixed-size, contiguous time windows. Ideal for historical data processing or time-series data where each slice needs to be processed exactly once.
- **Event-Based Trigger:** Initiates pipelines in response to specific events happening in Azure Storage, such as a file being created, deleted, or updated in a Blob Storage container or ADLS Gen2.
- **Manual Trigger:** (Not a true "automation" trigger) Allows you to start a pipeline run on demand from the ADF UI or via API calls.

32. What is the control flow?

- Control flow refers to the order in which individual instructions, statements, or activities are executed.
- It dictates the path that the program or workflow takes based on conditions, loops, or sequences.
- ADF provides a rich set of Control Flow Activities that enable complex orchestration logic:
    - **Sequencing and Dependencies:**
        - Activity Dependencies: This is the most basic form of control flow. You connect activities with arrows to define their execution order. An activity can be configured to run:

        - **On Success:** Runs if the preceding activity succeeds.
        - **On Failure:** Runs if the preceding activity fails.
        - **On Completion:** Runs regardless of whether the preceding activity succeeds or fails.
        - **On Skipped:** Runs if the preceding activity was skipped.
- **Conditional Execution (Branching):**
    - **If Condition Activity:**
        - This activity allows you to define a logical expression.
        - If the expression evaluates to true, one set of activities is executed; if false, a different set of activities is executed.
- **Looping (Iteration):**
    - **For Each Activity:**
        - Used to iterate over a collection of items (e.g., a list of file paths, table names, or database connections).
        - For each item in the collection, a defined set of activities inside the "For Each" loop will be executed.
- **Parameterization and Dynamic Behavior:**
    - **Get Metadata Activity:** Retrieves metadata about a data object
    - **Set Variable Activity:** Sets the value of a pipeline variable,
    - **Lookup Activity:**
        - Fetches data from a source. The output can then be used to drive subsequent control flow activities (e.g., the input for a For Each loop or the condition for an If Condition).

33. How do you handle schema drifting in ADF

- Schema drifting refers to the dynamic changes in the schema of your source data. This means that over time, columns might be added, removed, renamed, or their data types might change.
- **Schema Drift Setting:** In the Source transformation of a Data Flow, you'll find a setting called "Schema drift".
    - When enabled Data Flow can automatically detect new columns (and removed columns) in the incoming data stream even if they weren't present in the initial schema

you defined for the source dataset.

- It allows you to process these new columns without your pipeline failing.

34. How do you optimize mapping data flows for performance. For example source to sink you are partitioning data

- **Optimize Data Flow Design:**
  - **Filter Early:** Push down filters to the source whenever possible. This reduces the amount of data read and processed by the Spark cluster.
  - **Select Only Necessary Columns:** Project only the columns you need as early as possible. Less data means less I/O and less memory consumption.
  - **Minimize Transformations:** Every transformation adds overhead.
  - **Avoid Unnecessary Sorts:** Sort transformations are expensive as they require a full shuffle of data across the cluster.
- **Choose right Integration Runtime**
  - Choose the Right Integration Runtime (IR) Size:
  - **Core Count:** Start with a moderate number of cores (e.g., 16-32) and scale up as needed. More cores mean more parallelism, but also higher cost.
  - **Time To Live (TTL):** Set an appropriate TTL for your IR. A higher TTL (e.g., 30-60 minutes) reduces cluster startup time for frequently executed data flows. A lower TTL saves cost if data flows are infrequent.
- **Partitioning Strategies (Source to Sink)**
  - Partitioning is a key strategy to leverage Spark's parallel processing capabilities. It involves dividing your data into smaller, manageable chunks that can be processed independently across the cluster's nodes.
  - If data already partitioned
    - "Use Current Partitioning"
  - "Set Partitioning" (Custom Partitioning at Source):
    - If your source data is not physically partitioned, or if you want to re-partition it as it's read, you can explicitly set the partitioning method.

35. How do you pass parameters from one pipeline to another pipeline

- The Execute Pipeline activity allows you to call another existing pipeline from within your current pipeline.
- When you configure this activity, you can specify values for the parameters that the called (child) pipeline expects.

Example

Let's imagine you have two pipelines:

- Parent Pipeline (Caller): MainDataProcessingPipeline
- Child Pipeline (Called): ProcessIndividualFilePipeline
- The ProcessIndividualFilePipeline needs to know the fileName and fileSize to do its job.

Step 1: Define Parameters in the Child Pipeline

- Open the Child Pipeline
- Add Pipeline Parameters:
  - Click on the blank canvas of the pipeline.
  - In the "Parameters" tab of the properties pane at the bottom, click + New.
  - Create two new parameters:
    - Name: fileName
      - Type: String
    - Name: fileSize
      - Type: Int (or Float, depending on your needs)
- Use the Parameters: Inside ProcessIndividualFilePipeline, you can now use these parameters in any activity.
  - Eg: `@pipeline().parameters.fileName`

Step 2: Call the Child Pipeline from the Parent Pipeline

- Next, you'll configure the parent pipeline to call the child pipeline and pass the required parameter values.
- Open the Parent Pipeline (MainDataProcessingPipeline):
- Add an "Execute Pipeline" Activity:

    - From the "Activities" pane on the left, drag and drop an Execute Pipeline activity onto your canvas.
- Configure the "Execute Pipeline" Activity:
    - Go to the "Settings" tab in the properties pane.
    - Invoked pipeline: From the dropdown, select ProcessIndividualFilePipeline.
- Pass Parameters:
    - After selecting the invoked pipeline, the "Parameters" section will automatically populate with the parameters defined in the child pipeline (fileName, fileSize).

36. How do you handle incremental loading in ADF

- Incremental loading is a crucial technique in data warehousing and ETL processes where you only load new or changed data from your source system, instead of reloading the entire dataset every time. This significantly reduces processing time, resource consumption, and cost.

Ways to handle incremental loading

Change Data Capture (CDC) - Native Database Features

- For certain relational database sources, ADF can leverage the database's native CDC capabilities.
- This is often the most robust and accurate way to capture changes, including deletes.

Watermark Technique

- **Identify a Watermark Column:**
    - Choose a column in your source table that changes when data is updated or new data is inserted
- **Maintain a Watermark Value:**
    - Create a separate "watermark table"
    - Initialize this value to a very old date or a minimal ID.
- **Pipeline Steps:**
    - **Lookup Activity (Current Watermark):**
        - The first step in your pipeline is a Lookup activity that reads the last_successful_load_watermark_value from your watermark table.

    - **Lookup Activity (New Watermark):**
        - A second Lookup activity gets the current maximum watermark value from your source table. This will be the new watermark for the next run.
    - **Copy Activity: In the Copy Data activity (or a Data Flow source):**
        - Source: Use a query or filter condition (e.g., in the source dataset or Copy Activity source settings) to select only rows where the watermark column value is greater than the current_watermark_value retrieved from the first Lookup activity, and less than or equal to the new_watermark_value from the second Lookup activity.
- Scheduling: Schedule the pipeline to run periodically (e.g., hourly, daily) using a Schedule Trigger.

37. What is mapping in dataflow

- "mapping" primarily refers to the process of defining how columns from an input stream are connected to and potentially transformed into columns for an output stream or a subsequent transformation
- **Schema Enforcement and Transformation:**
    - Data coming from sources or intermediate transformations often doesn't have the exact column names, order, or even data types required for the next step or the final destination.
    - Mapping allows you to explicitly define these.
- Through mapping we can do
    - **Renaming Columns:** Change column names to be more descriptive, adhere to naming conventions, or match a target schema.
    - **Reordering Columns:** Arrange columns in a specific order for readability or to match a target schema
    - **Handling Schema Drift:** Data Flows are powerful in handling schema changes in source data. Mapping helps you explicitly decide how new, removed, or changed columns should be treated.

38. "is" in python

```
a = [1, 2, 3]
b = a  # b refers to the same list object as a
c = [1, 2, 3] # c creates a new list object, even if its content is the same

print(a is b)  # Output: True (a and b point to the exact same list)
print(a is c)  # Output: False (a and c are different list objects in
memory)
```

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]

print(a == b)  # Output: True (a and b have the same value)
print(a == c)  # Output: True (a and c have the same value, even if they are
different objects)
```

39. What is data drifting?

- Data drift refers to a change in statistical properties of data over time
- It means that the data you are receiving now is different from what your system was originally built or trained on, even if schema remains the same

40. Types of schema drifting?

- **Source schema drift**
  - Schema changes in the incoming data source
  - Examples
    - New column is added to a CSV or database table
    - Column is removed or renamed
- **Sink schema drift**
  - Schema changes when writing to destination
  - Example
    - You are writing to an SQL table and the new data has extra fields

41. 2 Types of staging layer

- In data warehousing and ETL (Extract, Transform, Load) processes, a staging layer (also known as a staging area or landing zone) is an intermediate storage area where raw data is temporarily held and prepared before being loaded into the final data warehouse or data mart
- **1. Persistent Staging Layer (PSA)**
  - Staging area that stores source system data for an extended period, often indefinitely, including its full change history
- **2. Non-Persistent Staging Layer (Temporary/Transient Staging)**
  - Intermediate storage area where data is held only for the duration of the current ETL/ELT process.
  - Once the data has been successfully processed and loaded into the target data warehouse or data mart, the staging area is typically cleared, truncated, or dropped.