**Python**

**exception handling try,except,raise.**

Exception handling in Python helps manage errors gracefully without crashing the program.

**The key components are:**

1. **try Block**: The code that might raise an exception is placed inside a try block.
2. **except Block**: If an exception occurs, the except block catches and handles it.
3. **raise Statement**: Used to manually raise an exception.

**Multithreading and multiprocessing:**

* **Multithreading**: Runs multiple tasks at the same time within a single program.
* **Multiprocessing**: Runs multiple tasks as separate programs, best for heavy calculations like data processing or image editing.

**Lazy Evaluation:**

Lazy Evaluation means Spark does not execute transformations immediately. Instead, it builds a logical execution plan and only runs computations when an action is called.

**Transformations:**

Transformations only define computations, but Spark doesn't execute them immediately.  
Examples: filter(), map(), select(), groupBy(), join().

**Actions:**

Actions force Spark to execute all previous transformations.  
Examples: show(), collect(), count(), write().

**Difference between dropduplicates and distinct:**

dropDuplicates : Removes duplicates based on specific columns, keeping the first occurrence.

Distinct:Removes fully identical rows across all columns.

**Persist:**

Persist is used to store a DataFrame in memory and/or disk to avoid recomputation and improve performance.

**Databricks Architecture:**

Databricks follows a two-layered architecture: Control Plane and Data Plane, ensuring efficient data processing, security, and scalability.

**Control Plane (Managed by Databricks):**

The Control Plane is responsible for managing infrastructure, job scheduling, and security policies. It does not process or store user data. Instead, it handles metadata, notebooks, and configurations.

* Manages notebooks, jobs, and cluster configurations.
* Schedules and monitors jobs.
* Handles security policies like access control (Unity Catalog).
* Stores notebook source code and metadata.

**Data Plane (Runs in Your Cloud Account):**

The Data Plane is where actual data processing happens. It runs in the user's cloud environment (AWS, Azure, GCP) and interacts with cloud storage (S3, ADLS, GCS).

* Executes Apache Spark jobs on Databricks clusters.
* Reads and writes data from cloud storage (Delta Lake, Parquet, CSV, etc.).
* Processes machine learning and ETL workloads.

**Upsert:**

Upsert (Update + Insert) in PySpark is performed using the merge operation in Delta Lake.

It updates existing records if a match is found and inserts new records if no match exists.

**Different write mode types in pyspark:**

**Append Mode**

* Adds new data to the existing dataset without modifying or deleting any existing records.
* **Use Case**: When we need to insert new records continuously, such as log data or incremental loads.

df.write.mode("append").parquet("path")

**2. Overwrite Mode**

* Replaces the existing data at the destination.
* **Use Case:** When you need to refresh the entire dataset.
* **Options:**

overwriteSchema=True: Updates schema as well when overwriting.

df.write.mode("overwrite").parquet("path")

**3. ErrorIfExists Mode (Default)**

* Fails if the target file or table already exists.
* **Use Case:** Prevents accidental overwrites.

df.write.mode("errorIfExists").parquet("path”)

**4. Ignore Mode**

* If the target exists, it does nothing and does not throw an error.
* **Use Case:** Ensures no accidental overwrites or errors in scheduled jobs.

df.write.mode("ignore").parquet("path/")

**Left Anti Join:**

Returns only the records from the left DataFrame that do not have a match in the right DataFrame.

**Left Semi Join:**

Returns only the records from the left DataFrame where a match exists in the right DataFrame.

**replaceWhere:**

replaceWhere is used with Delta Lake in PySpark to overwrite only specific partitions instead of the entire table.

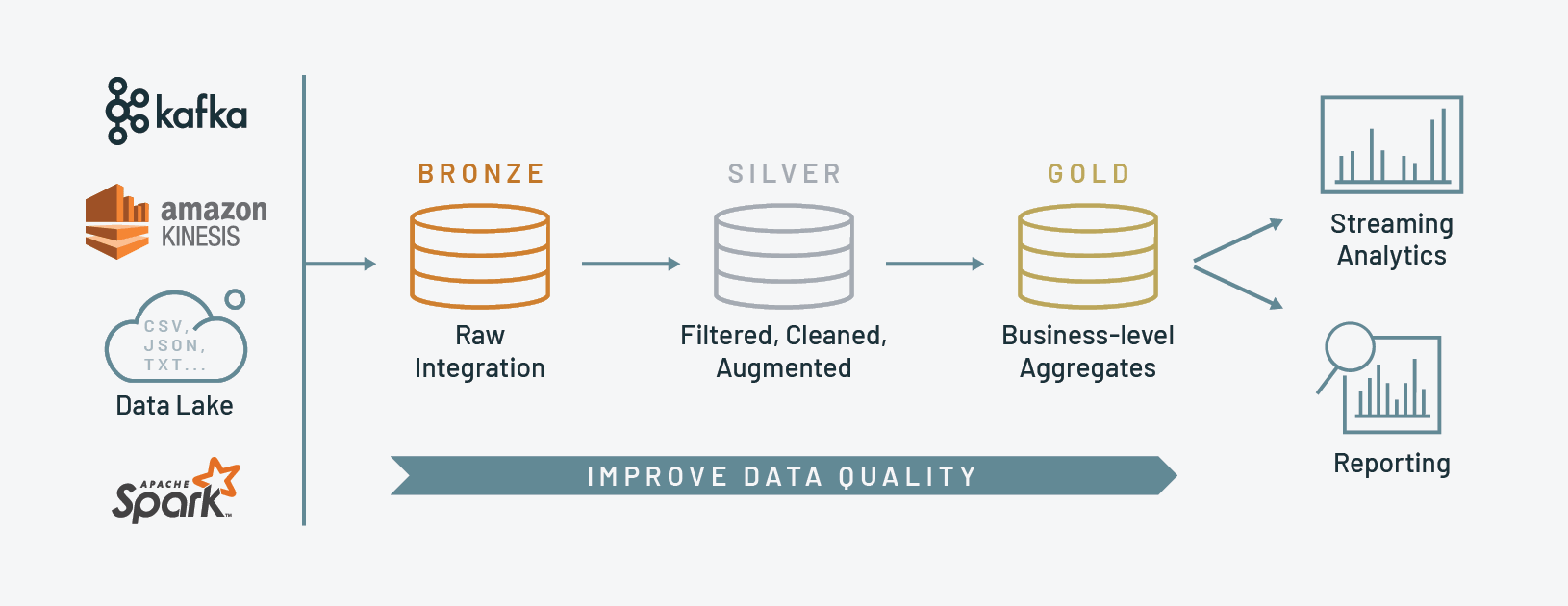
**Partition:**

Partitioning is a technique used in Delta Lake and Spark to improve query performance by dividing large datasets into smaller, manageable parts based on a specific column.

**Medallion architecture:**

Medallion architecture is a data design pattern used to logically organize data in Lakehouse. Improving the structure and quality of data as it each layer of the architecture (Bronze 🡪 Silver 🡪 Gold)

Medallion architecture is also called ‘mutli-hop’ architecture.



**Bronze Layer:**

Bronze layer also known as the Staging layer; data is extracted from its original sources (raw data).

**Silver layer:**

Data from the Bronze layer undergoes a transformation in the silver layer. like filtered, cleaned, and validated the data.

**Gold layer:**

In gold layer, aggregates and prepares data for business use. It applies data quality standards and business logic to present data.

**SQL:**

**Write one query using group by and having**

SELECT dept, count(dept) as dept\_count

FROM employee

GROUP BY dept

HAVING count(dept) > 30;

**Difference between rank and dense rank:**

* RANK() skips numbers when there are duplicates.
* DENSE\_RANK() assigns consecutive ranks without gaps

**Difference between groupby and window function:**

Group by: Groups data and returns one result per group, removing row-level details.

Window Functions: Groups data but keeps all rows, adding extra computed values like totals or rankings.

**Window function - row number, rank, dense rank:**

row\_number() → Gives a unique number to each row, even if values are the same.

rank() → Gives the same rank for ties, but skips numbers after a tie.

dense\_rank() → Gives the same rank for ties, but does not skip numbers after a tie.

**where and having clause:**

**Where** is used to filter out the record level based on the condition

**Having** is used to filter out the group level based on the condition

**Difference Between Primary Key and Unique Key:**

**Primary Key:**

A primary key is a column of a table that uniquely identifies each tuple (row) in that table. Only one primary key is allowed to be used in a table. The primary key does not accept any duplicate and NULL values.

**Unique Key**

Unique Key constraints also identify an individual tuple uniquely in a relation or table. A table can have more than one unique key, unlike a primary key. Unique key constraints can accept only one NULL value for the column.

**ADF:**

Until Activity: Runs repeatedly until a condition is met (like a loop with a stop condition).

ForEach Activity: Iterates over a collection of items and executes logic for each item.

**GDAI:**

**what is unity catalog:**

* Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Databricks workspaces.
* Unity Catalog offers a single place to administer data access policies that apply across all workspaces.