**Core Spark & Databricks Concepts**

* **RDD (Resilient Distributed Dataset)**  
  Fundamental Spark data structure; an immutable, distributed collection of objects partitioned across the cluster. Supports fault tolerance via *lineage*.
* **DAG (Directed Acyclic Graph)**  
  A graph structure representing computation flow. Spark builds a DAG of stages and tasks when executing jobs, ensuring no cyclic dependencies.
* **Stage:** A set of tasks that can be executed in parallel, separated by shuffle boundaries.
* **Task:** The smallest unit of execution in Spark, processing one partition of data.
* **Job:** A complete execution triggered by an action (collect, count, save, etc.). A job is broken into stages and then tasks.
* **Executor:** Worker process on a cluster node that runs tasks and stores data in memory or disk.
* **Driver:** The main process that coordinates Spark jobs, runs the user program, and schedules tasks on executors.

**Transformations & Actions**

* **Narrow Transformation:** Each input partition maps to exactly one output partition (e.g., map, filter).
* **Wide Transformation:** Requires shuffle across partitions (e.g., groupByKey, reduceByKey, join).
* **Action:** Operation that triggers job execution (e.g., count, collect, saveAsTextFile).
* **Transformation:** Operation that defines a new RDD/DataFrame but is lazily evaluated (e.g., map, filter).

**Data Abstractions**

* **DataFrame:** Distributed collection of data organized into named columns (like a table). Optimized via Catalyst optimizer.
* **Dataset:**Type-safe, object-oriented representation on top of DataFrames (more common in Scala/Java).
* **Schema:** Definition of the structure of a DataFrame (column names, types).

**Optimization & Execution**

* **Catalyst Optimizer**  
  Spark SQL’s query optimizer that applies rules (analysis, logical optimization, physical planning) to improve execution plans.
* **Tungsten Engine:** Spark’s in-memory computation engine for efficient code generation and memory management.
* **Shuffle:** Costly data re-distribution step across partitions/nodes, triggered by wide transformations.
* **AQE (Adaptive Query Execution)**  
  Spark 3.0+ feature that dynamically adjusts query plans (e.g., changing number of shuffle partitions at runtime).
* **Speculative Execution:** Re-runs slow/stuck tasks on different executors to mitigate stragglers.

**Delta Lake & Databricks-Specific Terms**

* **Delta Lake**  
  Open-source storage layer that adds ACID transactions, schema enforcement, and versioning to data lakes.
* **Transaction Log (\_delta\_log)**  
  JSON + Parquet files that store metadata and versions of Delta tables for reliability.
* **Time Travel:** Delta feature allowing queries on historical snapshots of a table using version or timestamp.
* **Z-Ordering:** Multi-dimensional clustering technique in Delta Lake to optimize read queries.
* **Optimize:** Delta command that compacts small files into larger ones for efficient reads.
* **Vacuum:** Command that physically removes old, unneeded data files (post-retention).

**Cluster & Resource Management**

* **Cluster:** A set of nodes (driver + executors) that run Spark/Databricks jobs.
* **Cluster Manager:** Manages cluster resources; Databricks uses its own manager but can also integrate with YARN, Kubernetes, or Mesos.
* **Dynamic Allocation:** Feature that scales executors up/down automatically based on workload.
* **Photon:** Databricks’ vectorized query engine (C++), optimized for speed in SQL workloads.

**Streaming & Ingestion**

* **Structured Streaming**  
  Declarative API for real-time streaming built on DataFrames. Provides exactly-once semantics.
* **Watermarking:** Mechanism in streaming queries to handle late data with event-time processing.
* **Trigger:** Defines how often streaming queries are executed (e.g., micro-batch every 5s, or continuous mode).
* **Autoloader:** Databricks ingestion feature that efficiently loads data from cloud storage into Delta with schema inference and evolution.

**Governance & Catalog**

* **Unity Catalog:** Centralized governance layer in Databricks for data access, lineage, and permissions.
* **Metastore:** Stores metadata (schemas, tables, views). Unity Catalog is the next-gen metastore for multi-workspace governance.

**Common Utility Concepts**

* **Accumulator:** Write-only variables used for aggregating information across tasks (e.g., counters).
* **Broadcast Variable:** Read-only variable cached on each executor for efficient joins or lookups.
* **Checkpoints:** Persisted intermediate states (esp. in streaming) to ensure recovery.
* **Lineage:** Metadata tracking how an RDD/DataFrame was derived, used for fault tolerance.