Data Ingestion

Real-Time Data Ingestion:

- **Do you need low-latency data streaming?**
  - **Yes → Tool**: **Apache Kafka / Pulsar**
    - **Technique**:
      - **Event-Driven Architecture**: Use Kafka's publish-subscribe model to decouple producers and consumers.
      - **Backpressure Handling**: Manage data flow through Kafka's ability to handle spikes without overwhelming the system.
      - **Stream Partitioning**: Use partitioning for scalability and parallelism.
  - **No → Tool**: **AWS Kinesis / Google Pub/Sub**
    - **Technique**:
      - **Load Shedding**: Handle oversaturation by shedding load or buffering with a dedicated queuing system.
      - **Sharding**: Distribute data into multiple streams to ensure better resource utilization.

Batch Data Ingestion:

- **Are you dealing with large datasets that don't need real-time processing?**
  - **Yes → Tool**: **Apache Spark**
    - **Technique**:
      - **Partitioning**: Split data into manageable chunks for parallel processing, improving performance.
      - **Data Pipelines**: Use Spark's ability to process large volumes of data in distributed environments, applying transformations and aggregations.
      - **Checkpointing**: Save intermediate states to recover from failures.
  - **No → Tool**: **Airflow with ETL**
    - **Technique**:
      - **Task Dependency Management**: Chain tasks in Airflow to manage complex dependencies between steps.
      - **Data Orchestration**: Use Airflow's DAG (Directed Acyclic Graph) to define and schedule ETL workflows.
      - **Error Handling & Retries**: Configure retries and alerting to ensure robustness.

Data Storage

Structured vs. Unstructured Data:

- **Structured (Relational)**:
    - **Do you need ACID-compliant transactions and complex queries?**
        - **Yes → Tool**: **PostgreSQL / MySQL / Google BigQuery**
            - **Technique**:
                - **Normalization**: Organize data to reduce redundancy and improve integrity.
                - **Indexing**: Use indexing to optimize query performance.
                - **Transactional Consistency**: Use ACID properties (Atomicity, Consistency, Isolation, Durability) to maintain data integrity.
        - **No → Tool**: **AWS RDS**
            - **Technique**:
                - **Managed Backup and Recovery**: Automate backups and use RDS for fault tolerance and failover.
- **Unstructured Data (Big Data/Non-relational)**:
    - **Do you need to store large amounts of raw data with no schema requirements?**
        - **Yes → Tool**: **AWS S3 / Google Cloud Storage**
            - **Technique**:
                - **Object Storage**: Store large amounts of unstructured data in the form of files (images, videos, logs).
                - **Versioning**: Enable version control for object storage to track changes and maintain historical data.
        - **No → Tool**: **MongoDB / Cassandra / DynamoDB**
            - **Technique**:
                - **Schema-less Design**: Use flexible schema definitions to allow dynamic data structure adjustments.
                - **Indexing**: Optimize query performance by indexing frequently accessed fields.
                - **Replication**: Ensure high availability by replicating data across multiple nodes.
- **Do you need to store transactional data with ACID properties?**
    - **Yes → Tool**: **Delta Lake / Apache Hudi**
        - **Technique**:
            - **ACID Transactions**: Ensure data consistency with features like transaction logs and versioning.
            - **Time Travel**: Query historical data and rollback changes if needed.

Data Processing

Real-Time Processing vs. Batch Processing:

- **Real-Time Processing:**
    - **Do you need to perform complex, low-latency transformations or computations?**
        - **Yes → Tool**: **Apache Flink / Apache Beam**
            - **Technique**:
                - **Windowing**: Break streams into time-bound windows for aggregate operations.
                - **Stateful Stream Processing**: Retain state between events for more complex transformations.
                - **Fault Tolerance**: Implement checkpoints and restore to consistent states on failures.
        - **No → Tool**: **Spark Streaming**
            - **Technique**:
                - **Micro-batching**: Split streams into small batches for processing, balancing low-latency with throughput.
                - **Backpressure Management**: Handle incoming data rate variations smoothly.
- **Batch Processing:**
    - **Are you dealing with large datasets that require distributed computation?**
        - **Yes → Tool**: **Apache Spark**
            - **Technique**:
                - **Data Partitioning**: Divide datasets for parallel processing to speed up computations.
                - **Lazy Evaluation**: Spark's transformations are not executed until an action is performed, optimizing resource usage.
        - **No → Tool**: **AWS Lambda**
            - **Technique**:
                - **Event-Driven Execution**: Trigger computations via events without managing infrastructure.
                - **Stateless Functions**: Ensure functions are independent for scalability and fault tolerance.

Data Transformation

- **Do you need to transform data into a clean, analytics-ready state?**
    - **Yes → Tool**: **dbt**
        - **Technique**:
            - **SQL-based Transformations**: Use dbt to model, test, and document data transformations.
            - **Version Control**: Track changes and collaborate on SQL models.
            - **Testing and Documentation**: Automate testing and generate documentation for data models.

- **No → Tool**: **AWS Glue / Apache NiFi**
  - **Technique**:
    - **ETL Pipelines**: Glue simplifies ETL processes with automatic schema discovery and job scheduling.
    - **Data Flow Design**: Use NiFi for easy drag-and-drop flow design and real-time data routing.

Data Orchestration

- **Do you need to automate workflows or manage complex dependencies between tasks?**
  - **Yes → Tool**: **Apache Airflow**
    - **Technique**:
      - **Task Dependencies**: Define complex dependencies between tasks to ensure correct execution order.
      - **Dynamic DAGs**: Dynamically generate DAGs for flexible workflows.
      - **Retry Logic**: Implement retry mechanisms and failure handling for robustness.
  - **No → Tool**: **AWS Step Functions**
    - **Technique**:
      - **Serverless Workflow**: Use Step Functions for lightweight, serverless orchestration without managing infrastructure.
      - **Simple Task Coordination**: Coordinate simple tasks without the overhead of full workflow engines.

Data Quality & Testing

- **Do you need to enforce data validation rules throughout your pipeline?**
  - **Yes → Tool**: **Great Expectations**
    - **Technique**:
      - **Data Profiling**: Define and track data expectations (e.g., ranges, formats) to catch issues early.
      - **Automated Validation**: Run validation checks automatically as data flows through the pipeline.
  - **No → Tool**: **Pytest**
    - **Technique**:
      - **Unit Testing**: Test individual components of the data pipeline to ensure correctness.
  -