**🧩 1. Data Architecture & Systems Design**

Understand how data systems fit together end-to-end.

* **Data Lifecycle** – ingestion → storage → processing → serving
* **Batch vs Stream Processing** – latency, throughput, use cases
* **ETL vs ELT architectures**
* **Lambda vs Kappa architecture**
* **Data Lake vs Data Warehouse vs Data Lakehouse**
* **Data Mesh & Data Fabric** – modern distributed governance models
* **Data Lineage & Observability** – tracking data flow across systems
* **Data Orchestration Principles** – DAGs, dependencies, retries, idempotency
* **Schema Evolution** – managing schema drift safely
* **Distributed System Concepts:**
  + Partitioning
  + Replication
  + Consistency models (CAP theorem)
  + Eventual consistency
  + Data locality and sharding

**⚙️ 2. Data Modeling Theory**

Goes beyond SQL schema design — this is about *how to represent data for analytics.*

* **Normalization & Denormalization** (1NF–3NF, star/snowflake trade-offs)
* **Dimensional Modeling** (Kimball)
* **Fact and Dimension Tables**
* **Slowly Changing Dimensions (SCD Types 1–6)**
* **Data Vault Modeling (for large-scale warehouses)**
* **Entity-Relationship Modeling (ERD basics)**
* **Surrogate vs Natural Keys**
* **Grain Definition** – how to choose the correct level of detail for facts

**🧮 3. Data Warehousing & Storage Concepts**

Learn how different storage systems shape data workflows.

* **Columnar vs Row-based Storage**
* **File Formats & Compression** – Parquet, ORC, Avro, Delta, JSON
* **Partitioning & Bucketing** strategies
* **Z-Order & Clustering in Snowflake/Databricks**
* **Table Types** – external, managed, transient, temporary
* **Indexing & Caching** in SQL Server and cloud warehouses
* **Query Optimization Internals** – cost-based optimization, stats, join order
* **Data Retention & Archival Strategies**

**🧠 4. Distributed Data Processing Fundamentals**

Even if you don’t use Spark daily, you should understand the foundations.

* **MapReduce paradigm**
* **Shuffle and Partition mechanics**
* **Data Skew & Load Balancing**
* **Resilient Distributed Datasets (RDDs)** in Spark
* **Spark SQL Catalyst Optimizer**
* **Lazy evaluation vs Eager evaluation**
* **Job execution DAGs** (conceptually same in Spark & Airflow)
* **Parallelism vs Concurrency**

**🌊 5. Data Ingestion & Streaming Theory**

Learn how real-time data pipelines work and why.

* **Change Data Capture (CDC)** – how it works conceptually
* **Event-driven architecture**
* **Message Queues vs Streams (Kafka, Kinesis, Pub/Sub)**
* **At-least-once vs Exactly-once processing semantics**
* **Event Time vs Processing Time**
* **Watermarks & Late Data Handling**
* **Schema Registry & Avro evolution**
* **Batch-Streaming Unification (Structured Streaming concepts)**

**🧱 6. Data Quality, Governance & Management**

Every professional engineer cares about *trustworthy data.*

* **Data Quality Dimensions:** accuracy, completeness, timeliness, consistency
* **Data Validation Frameworks** (Great Expectations, dbt tests)
* **Metadata Management** – active metadata, catalogs (Glue, DataHub, Alation)
* **Master Data Management (MDM)**
* **Data Lineage & Provenance**
* **Data Privacy & Compliance (GDPR, HIPAA, etc.)**
* **PII detection & anonymization**
* **Data Governance Models** – centralized vs federated ownership

**🧰 7. Workflow, Versioning, and CI/CD for Data**

How to build **reliable, reproducible, and testable** data systems.

* **DataOps principles** – version control + automation for data
* **Data pipeline CI/CD** – testing SQL, schema diffs, data validation
* **Git-based workflows** (branching, versioning for transformations)
* **Idempotency & Retry logic**
* **Backfills, Restarts & Reprocessing**
* **Monitoring & Alerting** – success/failure metrics, data lag detection

**☁️ 8. Cloud Data Engineering Concepts (Vendor-Agnostic)**

Know the ideas behind AWS/GCP/Azure data stacks.

* **Object Storage (S3, ADLS, GCS)**
* **Data Warehouses (Snowflake, BigQuery, Redshift)**
* **Serverless Processing (Lambda, Cloud Functions, Azure Functions)**
* **Infra as Code (Terraform, CloudFormation basics)**
* **Networking concepts for data pipelines (VPC, Subnets, NAT)**
* **Secrets & Credentials Management** (AWS Secrets Manager, KMS)
* **Scalability & Cost Optimization Strategies**

**🔐 9. Security, Privacy & Access Control**

Data engineers must understand the *risk* side too.

* **Role-based access control (RBAC)**
* **Data masking & encryption at rest/in transit**
* **Tokenization & key rotation**
* **Row-level and column-level security in SQL Server/Snowflake**
* **Audit logs & activity monitoring**

**📊 10. Analytical Concepts & Data Consumption**

Understand what analysts and dashboards expect from your data.

* **OLTP vs OLAP** workloads
* **Star schema design for BI tools (Power BI, Looker, Tableau)**
* **Aggregate tables, materialized views, and summary tables**
* **KPI design and metric consistency**
* **Time series modeling in SQL**
* **Dimensional hierarchies (Year → Month → Day)**

**⚡ 11. Performance, Scaling & Cost Efficiency**

You’ll need these principles when you run data systems at scale.

* **Horizontal vs Vertical scaling**
* **Data partitioning for performance**
* **Caching strategies (query, result-set, materialized)**
* **Query profiling & telemetry**
* **Cost optimization in warehouses (credits, slots, compute hours)**

**🧠 12. Mathematical & Statistical Underpinnings (Optional but Valuable)**

This separates data engineers from pipeline plumbers.

* **Data distributions & outlier detection**
* **Sampling techniques (simple, stratified, reservoir)**
* **Basic statistics for aggregation and quality checks**
* **Understanding variance, skew, kurtosis**
* **Time-based aggregations (rolling, moving average logic)**

**🎓 Bonus: Data Engineering Mindset**

These are not tools — they’re *habits*.

* Think in **pipelines**, not queries.
* Prefer **idempotent** operations — reruns don’t break pipelines.
* Always **track data freshness** and **quality metrics**.
* Keep your transformations **modular** and **traceable**.
* Use **metadata-driven** ETL — parameterized, not hardcoded.
* Write SQL that can **self-document** through naming conventions.