**Topics Covered:**

* Schema Evolution
* Schema Enforcement & How It Works
* Code Optimization Techniques
* Handling Bad Records from Source
* Features of Delta Lake
* How Joins Work in Databricks
* Ensuring No Data Loss When Loading to Staging
* Testing Data in ETL Pipelines
* Ensuring Data Quality After Loading to Staging Layer
* Cost Optimization Strategies
* Importance of Surrogate Keys
* Security of Surrogate Keys
* Steps to Follow Before Starting Development
* Security of Gold Table
* Why Load Data from Gold Table to Power BI?
* Handling Bad Records
* Incremental Data Loading & Use Case
* SCD Types

**Schema Evolution**

Schema evolution allows changes in a table’s schema over time, such as adding, modifying, or deleting columns without breaking existing data. In Delta Lake, schema evolution is enabled using MERGE and ALTER TABLE commands.

**Schema Enforcement & How It Works**

Schema enforcement also known as schema validation ensures that incoming data follows the predefined schema. If the schema doesn't match, the write operation fails. In Delta Lake, this prevents accidental data corruption. It works by comparing the schema of the incoming data with the table schema before insertion.

**Code Optimization Techniques**

To optimize the code, we can:

* Use efficient joins (e.g., Broadcast Joins for small datasets).
* Partition data correctly to avoid full-table scans.
* Optimize queries using Z-Ordering for fast data retrieval.
* Enable Data Skipping to read only relevant data.
* Use caching in Spark to store frequently used DataFrames.

**Handling Bad Records from Source**

Bad records are handled by:

* Using bad record handling options in Auto Loader (Databricks).
* Applying Try-Catch blocks in Spark to handle parsing errors.
* Writing invalid records to a quarantine table for later analysis.
* Using dropMalformed, permissive, and failFast options in Spark.

**Features of Delta Lake**

* ACID Transactions for data consistency.
* Schema Evolution & Enforcement to manage changes.
* Time Travel to query previous versions of data.
* Data Skipping & Z-Ordering for performance optimization.
* Merge & Upsert Support for handling changing data.

**How Joins Work in Databricks**

Joins in Databricks work by combining records from two or more tables based on a related column. Common join types:

* Inner Join – Returns matching rows from both tables.
* Left Join – Returns all rows from the left table, with matching data from the right.
* Right Join – Returns all rows from the right table, with matching data from the left.
* Full Outer Join – Returns all rows from both tables.
* Left Semi Join – Returns only left table rows that have a match in the right table.
* Left Anti Join – Returns left table rows that do NOT have a match in the right table.
* Broadcast Join: Optimizes joins by broadcasting smaller tables across nodes.

**Ensuring No Data Loss When Loading to Staging**

* Using checkpointing to track processed data.
* Implementing idempotent loads to avoid duplicate inserts.
* Using Delta Lake’s ACID transactions to ensure reliability.
* Logging & monitoring to detect missing data.

**Testing Data in ETL Pipelines**

Testing includes:

* Unit Testing: Testing individual transformations using pytest or unittest.
* Integration Testing: Ensuring ETL jobs work end-to-end.
* Data Quality Checks: Using Great Expectations or custom validation scripts.
* Regression Testing: Validating data consistency over time.

**Ensuring Data Quality After Loading to Staging Layer**

* Duplicate Checks: Removing duplicate records.
* Null & Constraint Validation: Ensuring required fields are populated.
* Row Count Validation: Comparing source vs. staging counts.
* Business Rule Validation: Checking if data meets expected patterns.

**Cost Optimization Strategies**

* Optimized Cluster Size: Used auto-scaling clusters to minimize idle resources.
* Z-Ordering & Data Skipping: Reduced unnecessary scans.
* Spot Instances: Leveraged lower-cost spot instances in the cloud.
* Table Caching: Used Databricks caching to improve query performance.

**Importance of Surrogate Keys**

* Provides unique identification for records.
* Helps maintain historical changes without affecting natural keys.
* Improves join performance compared to natural keys.

**Security of Surrogate Keys**

surrogate keys are secure because they do not expose business-sensitive information like natural keys (e.g., Social Security Numbers or Emails). They also help maintain data integrity in a data warehouse.

**Steps to Follow Before Starting Development**

* Understanding Business Requirements: Analyzing data needs.
* Designing Schema: Planning data storage format.
* Performance Considerations: Choosing the right partitioning and indexing strategy.
* Security Planning: Implementing access control and encryption.
* Defining Testing Strategy: Setting up validation checks.

**Security of Gold Table**

* Role-Based Access Control (RBAC) to restrict unauthorized access.
* Data Masking & Encryption for sensitive columns.
* Audit Logging to track data modifications.

**Why Load Data from Gold Table to Power BI?**

* Gold tables contain aggregated and cleaned data, ready for analysis.
* Reduces query complexity and improves performance in Power BI.
* Ensures consistent and accurate reporting across dashboards.

**Handling Bad Records**

* Storing in a quarantine table for review.
* Using try-except blocks in PySpark transformations.
* Applying schema validation before writing to target.

**Incremental Data Loading & Use Case**

Incremental loading updates only the new or changed data instead of reloading everything.  
**Use Case**:

* Ingesting daily sales data without reprocessing the entire dataset.
* Using watermarking to track processed records.
* Using Merge in Delta Lake to update existing records efficiently.

**SCD Types:**

SCD Type 0 (Fixed Dimension) – No changes are allowed. The data remains static.

SCD Type 1 (Overwrite) – Old data is overwritten with new data, losing historical changes.

SCD Type 2 (Versioning with History Tracking) – Keeps history by adding a new record with versioning (Start Date, End Date, Is\_Current flag).

SCD Type 3 (Limited History with Columns) – Stores limited history by maintaining separate columns for old and new values.

SCD Type 4 (Historical & Current Table) – Maintains a separate historical table while keeping only current data in the main table.