**Schema Evolution & Schema Enforcement in Databricks:**

**Schema Enforcement also known as Strict Schema Validation.**

Schema Enforcement ensures that incoming data matches the defined schema of a table. If a new column or incompatible data type appears, the write operation fails.

**How It Works in Databricks**

* When writing data to a Delta table, Databricks validates column names, data types, and order.
* If an incoming Data Frame has a different schema than the existing table, it throws an error.

Exxample:

Df.write.format(“delta”).mode(“overwrite”).save(path)

**Schema Evolution (Automatic Updates):**

Schema Evolution is allows adding new columns dynamically without failing the write operation. Helps when data schema changes over time from a sources.

**How It Works in Databricks**

When enable mergeShema option, Delta Lake automatically updates the table schema if new columns appear.

Example:

Df,write.format(“delta”).mode(“append”).option(“mergeShema” , True).save(path)

**How do you handle bad records that comes from source.**

We can handle bad records/corrupt records in PySpark using different modes.

**Load Correct Records & Capture Bad Records (PERMISSIVE Mode)**

* Keeps valid records, replaces corrupt values with null
* Stores bad records separately using badRecordsPath

Options:

.option(“mode”, “PERMISSIVE”)

.option(“badRecoredsPath”, “location”)

If any bad records come from the source, Spark replaces the corrupt value with null and stores the bad records separately for later analysis.

**Ignore Bad Records (DROPMALFORMED Mode)”**

* Removes entire rows that contain bad records. Does not log or capture dropped records.

.option("mode", "DROPMALFORMED")

If any bad records come from the source, Spark automatically drops the entire row from the dataset and loads only valid records.

**Stop Processing on First Bad Record (FAILFAST Mode)**

* Stops execution immediately when a bad record is encountered. It will throws an error.

.option("mode", "FAILFAST")

If any bad record is found in the source data, Spark fails the job immediately, preventing further processing.

**Performance optimization:**

**Optimizing File Formats**

* Prefer Parquet or ORC over CSV for better compression and predicate pushdown.
* Use Delta Lake for ACID transactions and better performance.

**Partitioning Optimization**

Repartition or coalesce to optimize data distribution (df.repartition(n) or df.coalesce(n)).

**Bucketing:**

Use bucketing for improved performance in shuffle-intensive operations.

**Caching and Persistence**

Use .cache() or .persist(storageLevel) for frequently accessed DataFrames.

**Broadcast Joins for Small Tables**

We can use broadcast(df) from pyspark.sql.functions for **small dimension tables** to reduce shuffle operations.

**Partitioning (partitionBy)**

Partitioning helps distribute data across multiple directories based on a specified column. It improves query performance by reducing data scans.

**z-order:**

Z-Order is used for while doing optimize command files are merge but the files merge unorderly (default size is 1GB) But using z-order technique the files merge orderly it ensures the query performance.