**Error Handling in Redpanda Connect**

**Error Handling in Redpanda Connect (and similar data pipelines)**. When processing streams, some messages fail due to bad data, connectivity issues, or downstream errors. These mechanisms help ensure reliability and controlled recovery. Let’s go through each point:

**Error Flagging**

* Failed messages are marked with **metadata flags**.
* These flags indicate what went wrong and allow the system to decide: retry, reroute, or log.
* Example: If a message couldn’t reach Snowflake, it is flagged for reprocessing.

**Catch Processor**

* Acts like a **try/catch block** in programming.
* Intercepts failed messages, attempts recovery, and clears error flags once fixed.
* Example: If message transformation fails, Catch Processor can apply a fallback transformation.

**Switch Processor**

* Routes errored messages based on **failure details**.
* Example:
  + If error = "timeout" → retry path
  + If error = "schema mismatch" → send to validation topic

**Logging**

* Uses error() function to log failures.
* Can also **append error details into the payload** for later debugging.
* Example: A failed record may include "error\_code": "SCHEMA\_INVALID" in logs.

**Dropping**

* Errored messages are dropped (acknowledged as handled but not processed further).
* Prevents blocking the pipeline due to bad data.
* Example: Invalid sensor data that cannot be parsed is safely dropped.

**Rejecting**

* Failed messages are explicitly **rejected (nack)** instead of acknowledged.
* These can be routed to special rejection outputs.
* Example: Messages that violate compliance rules are rejected.

**Dead-Letter Queues (DLQ)**

* Common in Kafka & Redpanda.
* Failed messages are redirected into a **special topic (DLQ)** for later inspection, reprocessing, or manual debugging.
* Example: Messages failing schema validation go to topic.orders.DLQ.

**Conditional Outputs**

* Routes messages based on **error type or content**.
* Example:
  + Errors containing "timeout" → retry topic
  + Errors containing "invalid\_json" → DLQ