IMPLEMENTANDO DATA QUALITY CON DATABRICKS

¿Quiénes Somos?

Jose Manuel Garcia Gimenez

- Senior Data engineer en Plain Concepts
- Databricks Solution Architect Champion
- +6 años de experiencia en Azure y Databricks
- Diseño, implementación y gobernanza de plataformas de datos, especialmente lakehouse

plain concepts

¿Quiénes Somos?

Antonio Aliaga Cortés

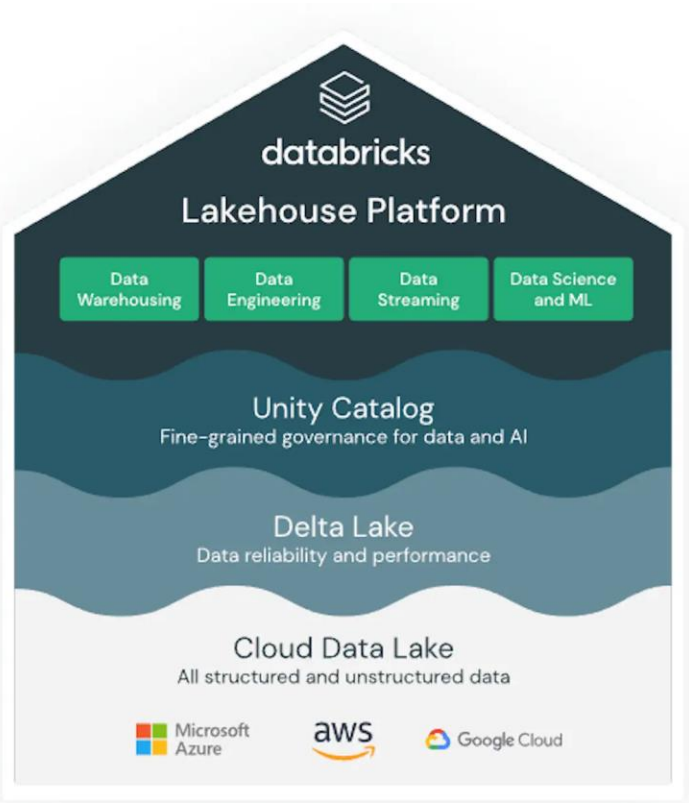
- Senior Data engineer - Intellias
- Databricks Solution Architect Champion
- +6 años de experiencia en Azure y Databricks
- Plataformas de datos

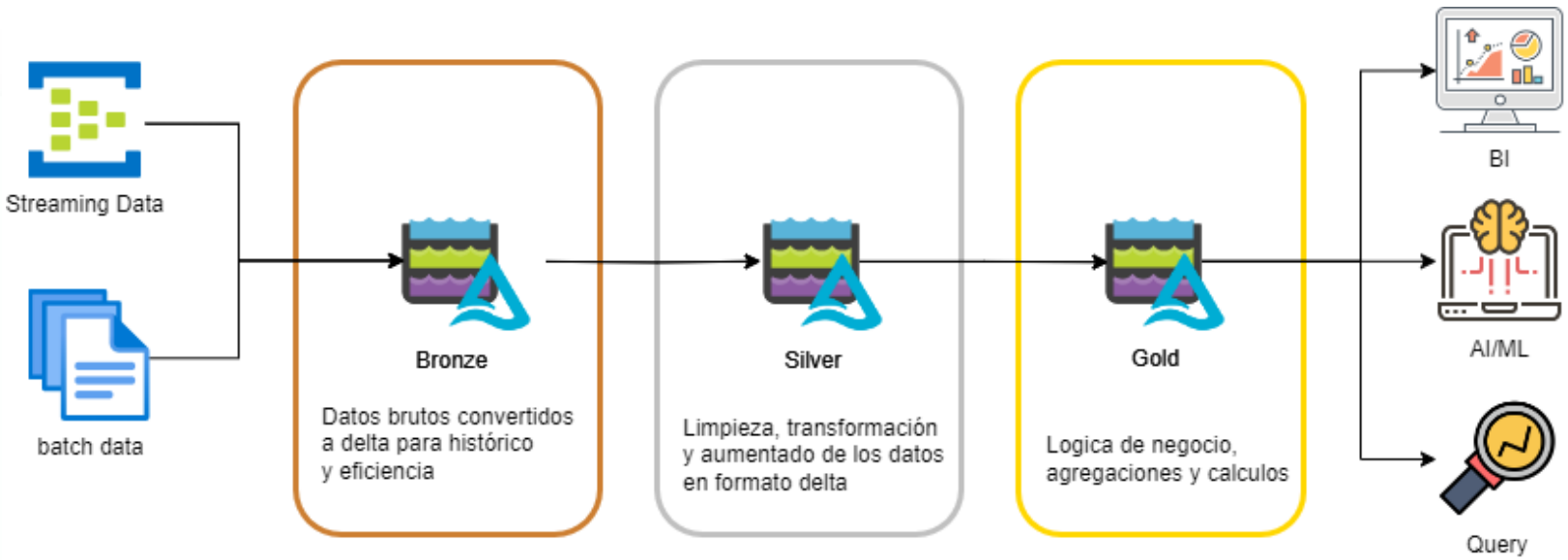
intellias
Intelligent Software Engineering

Agenda

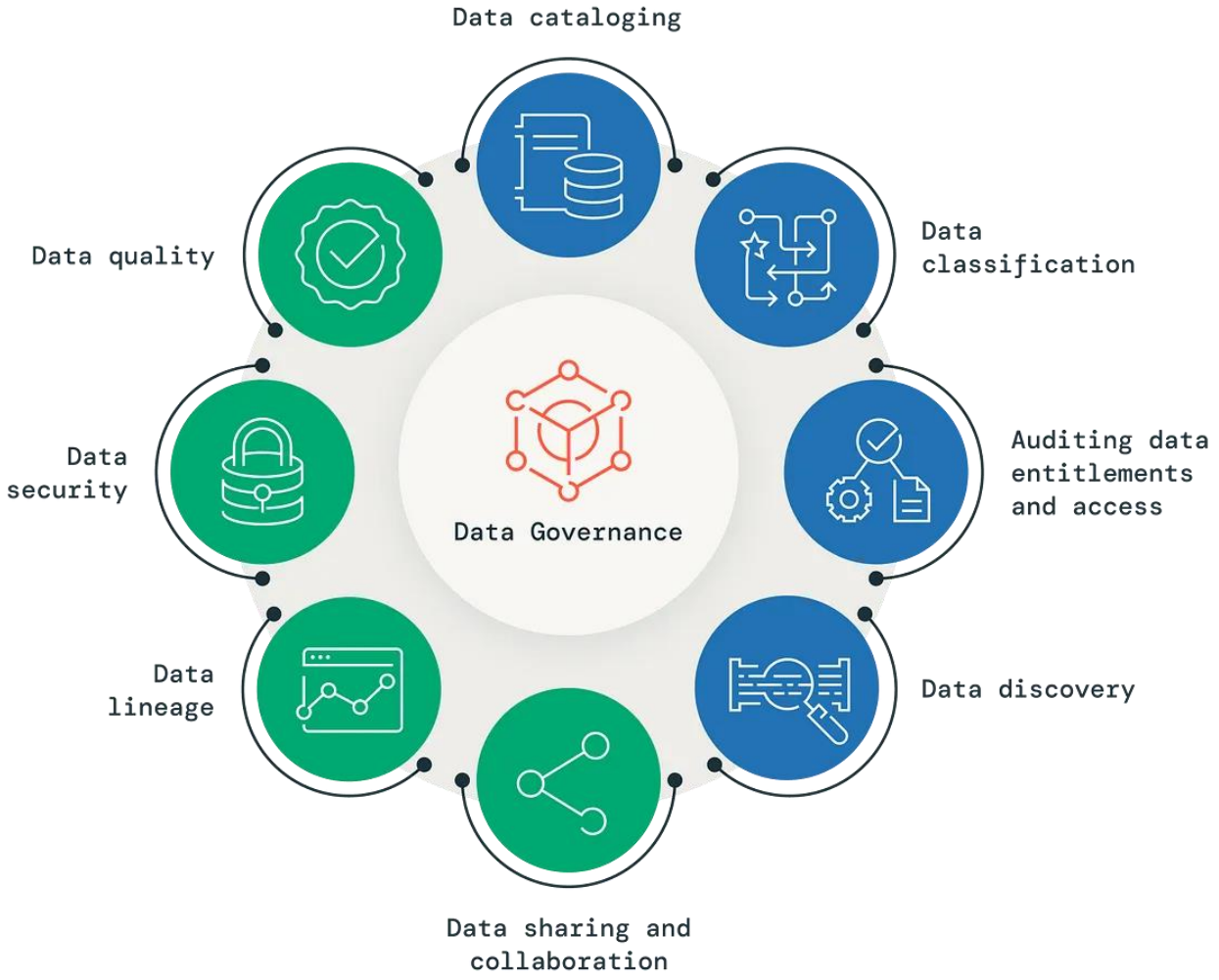
- Plataformas de datos
- Importancia del Data Quality
- 6 dimensiones de Data Quality
- Delta Live Tables
- Lakehouse Monitoring
- Demo

Plataforma de datos





Gobernanza de datos



https://www.databricks.com/discover/data-governance

Que vamos a ver? Data quality

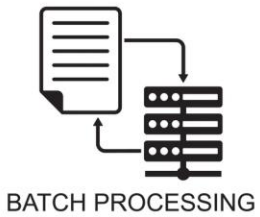
- Garbage in -> Garbage out
  - IA
  - Toma de decisiones
- Ahorro de cómputo y dinero
- Cumplimiento, regulación y auditoría
- Asegurar una única fuente de verdad

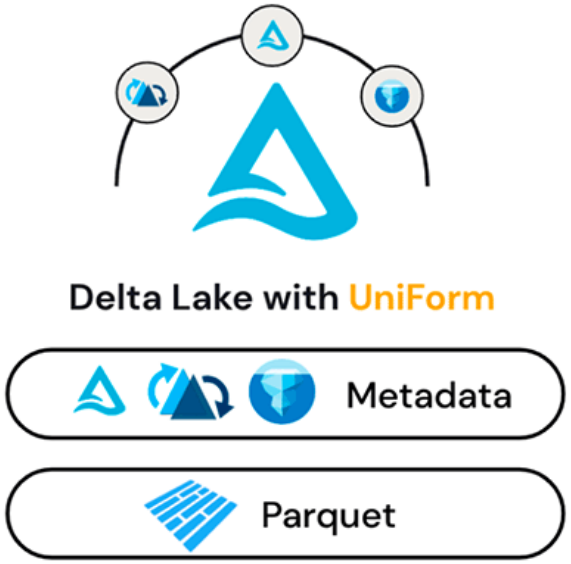
Las 6 Dimesiones de Data Quality

La calidad de los datos se refiere al grado en que los datos son precisos, completos, consistentes, válidos y están actualizados, lo que garantiza su fiabilidad para la toma de decisiones y el análisis.

Consistencia en los Datos

Los datos en la plataforma no deben entrar en conflicto otros datos. Debe existir una única fuente de verdad para la información

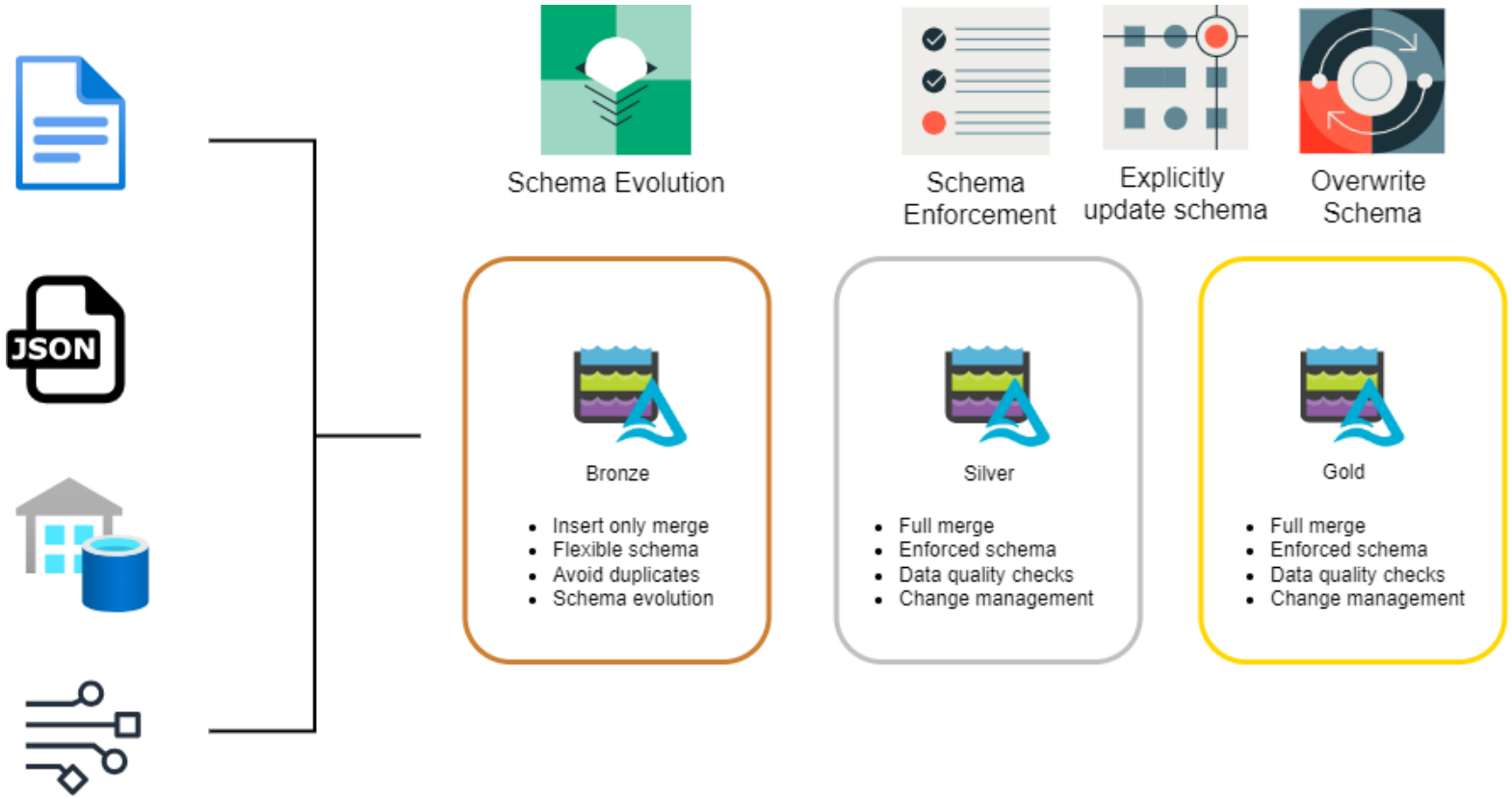




- Transacciones ACID
- Multiples lecturas/escrituras
- Unificar Stream y Batch
- Repositorio central de datos

Validez de los datos

Los datos deben estar en un esquema y formato válido y usable y que admita modificaciones.



Datos actualizados

Los datos deben estar actualizados. La tolerancia es muy importante y dependerá de los casos de uso de la organización

- Analizar caso de uso
  - ¿Cuando necesitamos realmente los datos?
  - Diferentes paises/zonas horarias
- Implicaciones en el coste y en la implementacion

Completitud de los datos

Los datos deben tener toda la información necesario para el caso de uso especifico así como los metadatos necesarios.

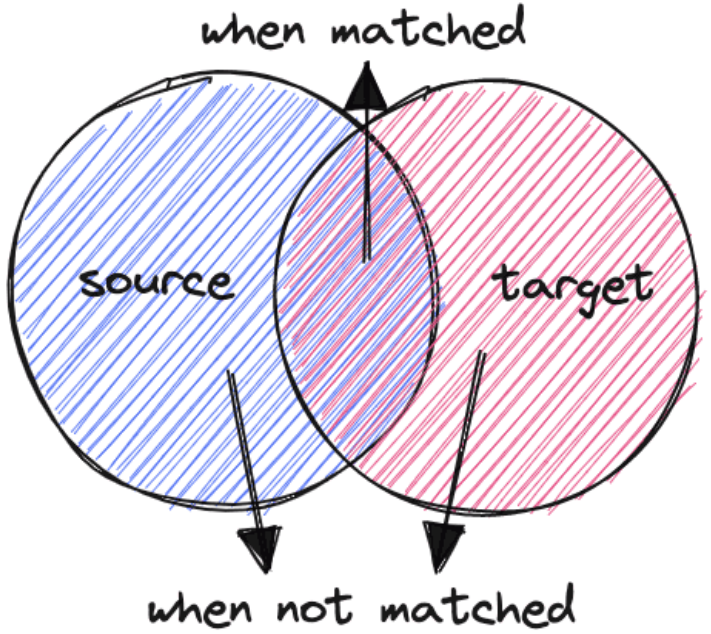
- Atomicidad en las transacciones
- Enriquecer los datos
- Metadatos con Unity Catalog.

```python
def add_metadata(df: DataFrame) -> DataFrame:
  patern_date = "[0-9]{4}\/[0-9]{2}\/[0-9]{2}"
  return (df
      .withColumn("_sourcefile", F.input_file_name())
      .withColumn("datekey",  F.date_format((F.to_timestamp(F.reg
      .withColumn("_process_timestamp", F.current_timestamp())
  )
```

Singularidad de los datos

No debe haber duplicados en los datos que puedan dar lugar a información errónea.

- Actualizar registros
  - Dimensiones tipo 1, tipo 2
- Eliminar duplicados



Precision de los Datos

Los datos deben representar información veraz sobre el negocio que proviene de distintas fuentes de información

Proactivo

Constraints

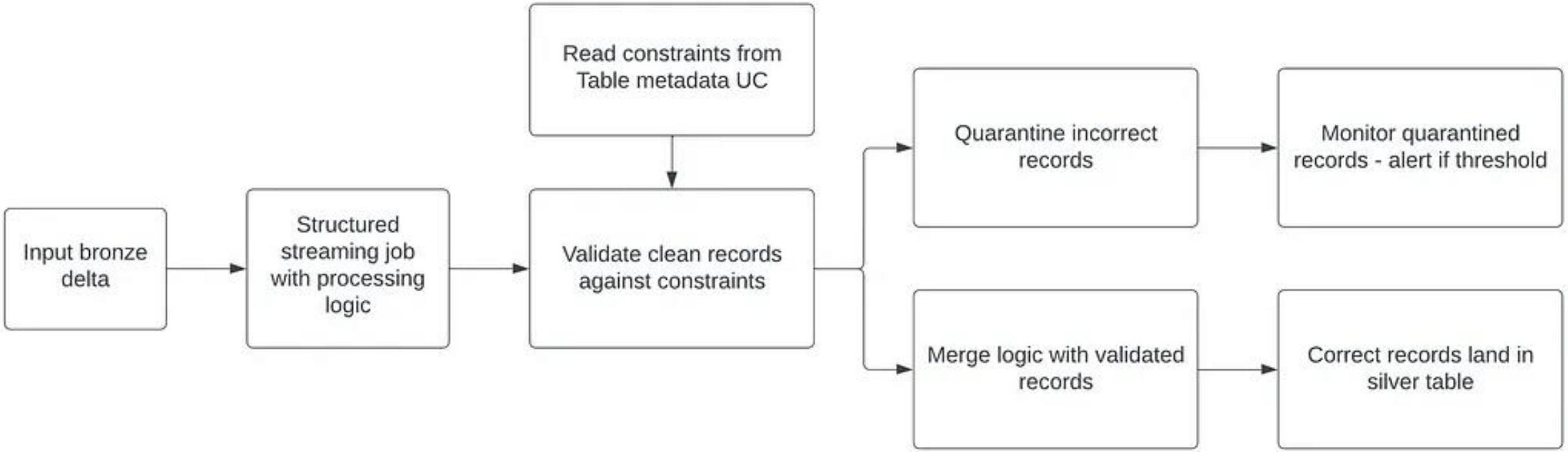
Quarentena

Reactivo

Monitorización

Alerts

Diseño logico Data Quality



Demo

Delta Live Tables - Expectations

- Seguimiento de registros incorrectos

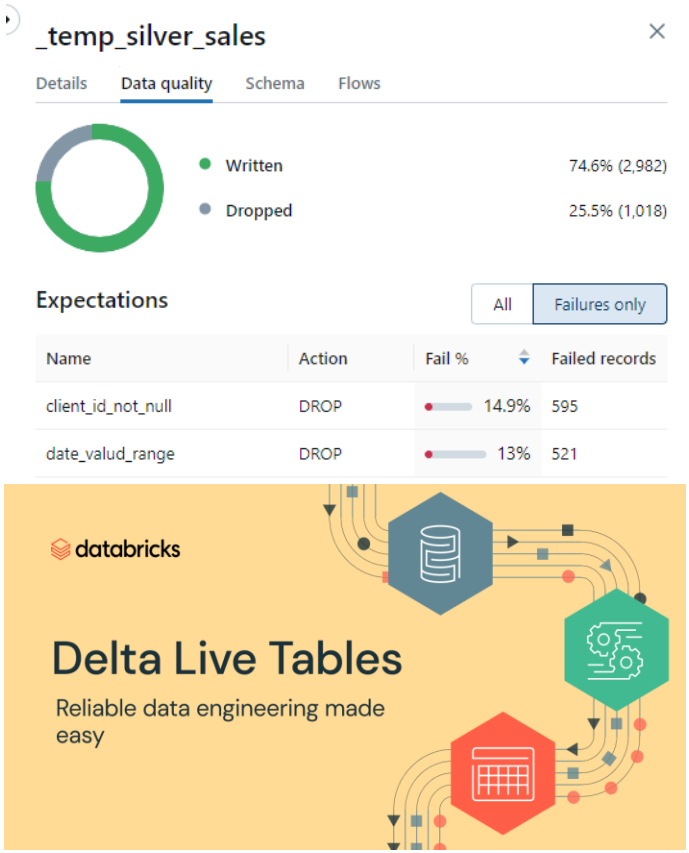
```
@dlt.expect("valid timestamp",
"timestamp > '2012-01-01'")
```

- Eliminar registros incorrectos

```
@dlt.expect_or_drop("valid_current_page",
"current_page_id IS NOT NULL AND
current_page_title IS NOT NULL")
```

- Cancelar/Abortar el procesamiento

```
@dlt.expect_or_fail("valid_count", "count > 0")
```



Implementación reactiva - Lakehouse monitoring

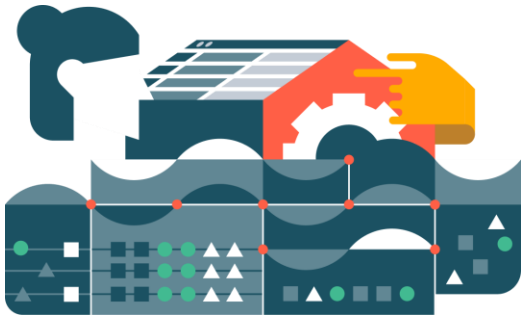
Profile type

- Snapshot
- Time series
- Model inference

Optional baseline table

Known “good” data for comparison

Databricks monitor
You define what type of metrics are relevant for your table



Profile metrics table

Profile statistics

Drift table

Changes in profile metrics compared to baseline or previous time window

Databricks SQL Dashboard

Inspect metrics and create custom visualizations

Databricks alerts

Configurable automatic notifications when changes are detected

Demo

Conclusiones

- Data quality debe incluirse en la estrategia de Data governance
- Importancia del Data quality desde el inicio del diseño
- Aprovechar features de delta y metadatos
- Adaptar la implementacion segun el caso de uso

IMPLEMENTANDO DATA QUALITY CON DATABRICKS

Medium

Repositorio

```python
# Step 1: Dedupe input batch, use max_col if provided
if max_col:
    logger.info("Deduplication using max column: %s", max_col)
    w = Window.partitionBy(ids).orderBy(col(max_col).desc())
    df_dedupe = (
        batch_df
        .withColumn("rank", F.rank().over(w))
        .filter(col("rank") == 1)
        .drop("rank")
    )
else:
    logger.info("Deduplication using primary keys: %s", ids)
    df_dedupe = batch_df.dropDuplicates(ids)
```

Eliminar duplicados

```python
# Step 2: Check if Delta table exists
if spark.catalog.tableExists(f"{table_schema}.{table_name}"):
    logger.info("Delta table exists: %s.%s", table_schema, table_name)
    delta = DeltaTable.forName(spark, f"{table_schema}.{table_name}")
else:
    # Step 3: Create Delta table if it doesn't exist
    logger.info("Creating Delta table: %s.%s", table_schema, table_name)
    delta = (DeltaTable.create(spark)
             .tableName(f"{table_schema}.{table_name}")
             .addColumns(df_dedupe.schema)
             .execute())
```

Crear tabla si no existe y añadir restricciones

```python
# Step 5: Quarantine invalid records based on table constraints
constraints_conditions = get_table_constraints_conditions(table_schema, table_name)

quarantine_records = df_dedupe.filter(constraints_conditions)
valid_records = df_dedupe.filter(~constraints_conditions)

# Write quarantine records to the specified quarantine schema
quarantine_records.write.mode("append").saveAsTable(f"{table_schema}_{quarantine_schema}.

logger.info("Total Records: %d", df_dedupe.count())
logger.info("Records moved to quarantine: %d", quarantine_records.count())
logger.info("Valid Records: %d", valid_records.count())

# Step 6: Create merge condition based on primary keys (ids)
condition = " AND ".join(f"l.{c} = r.{c}" for c in ids)
logger.info("Merge condition: %s", condition)

# Step 7: Perform the merge operation into Delta table
merge = (
    delta.alias("l")
    .merge(valid_records.alias("r"), condition)
    .whenMatchedUpdateAll()
    .whenNotMatchedInsertAll()
)

merge.execute()
logger.info("Merge operation completed for table: %s.%s", table_schema, table_name)
```

Comprobar restricciones

Quarentenar records

Merge idempotente

_temp_bronze_clean

Details | Data quality | Schema | Flows

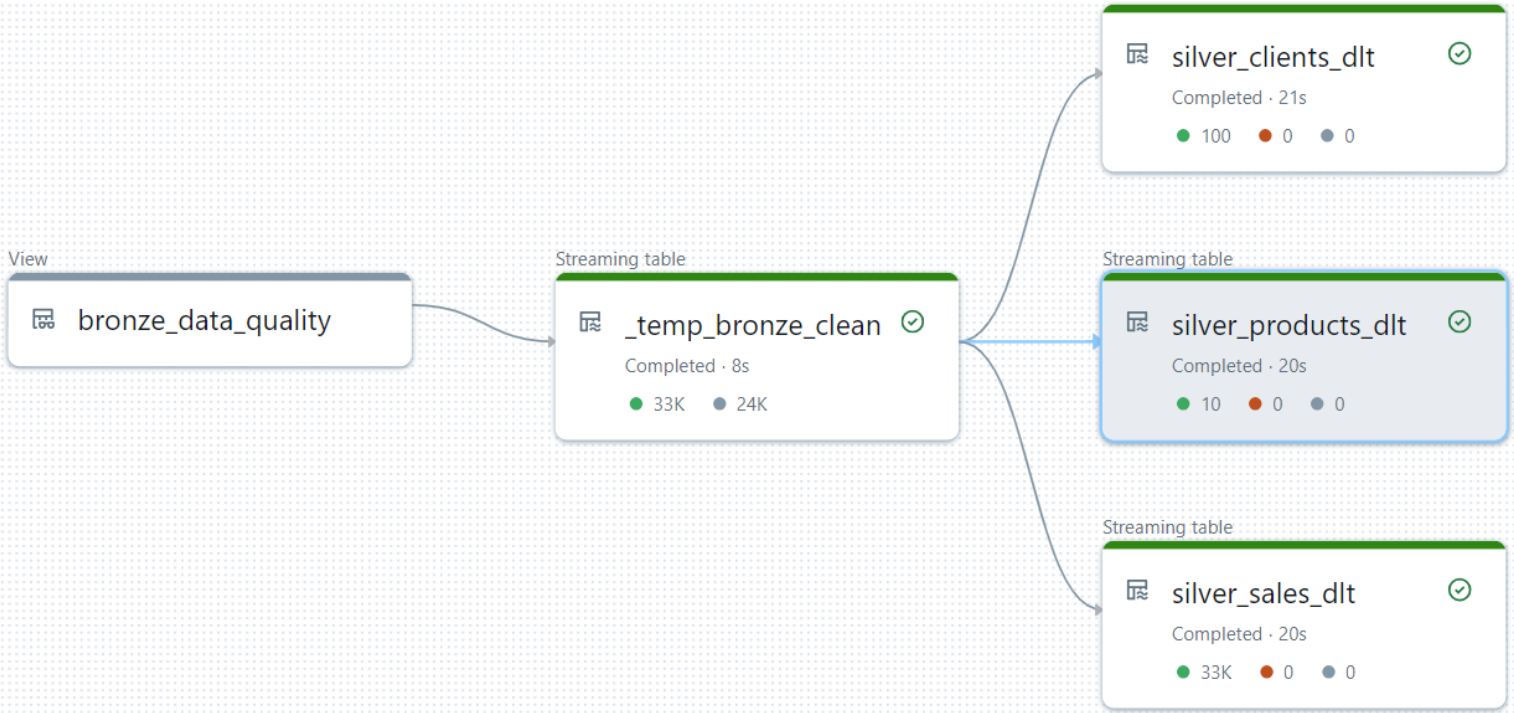
Written

Dropped

Expectations

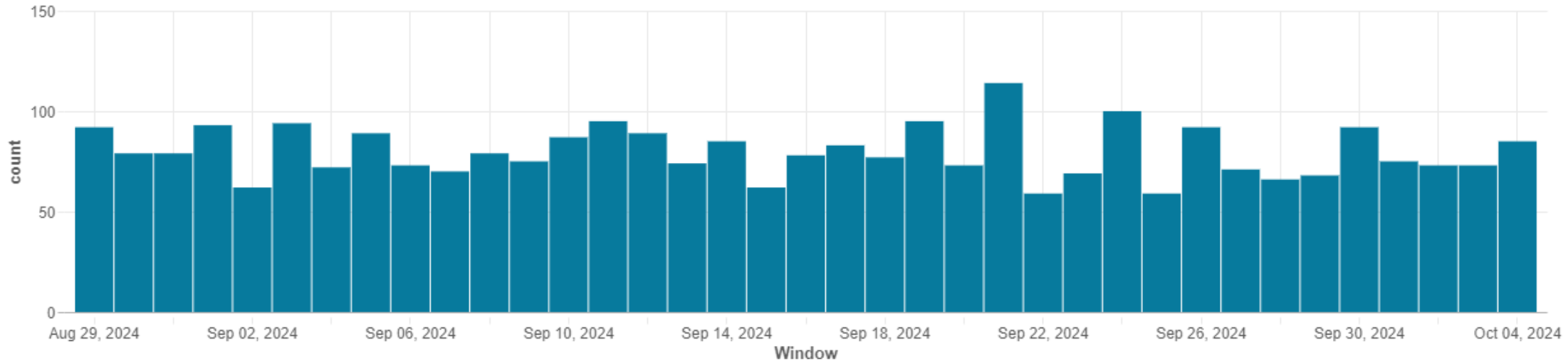
All

Name	Action	Fail %	Failed records
valid_client_id	DROP	29%	16550
valid_product_id	DROP	19.2%	10967
valid_address	ALLOW	10.5%	5969
valid_date	ALLOW	4.1%	2332
valid_quantity	ALLOW	0%	0
valid_sales_id	DROP	0%	0



Row Count Over Time

Track how the volume of data has been changing over time.



Filter by Column Name

All

% NULLs Over Time

Compare the percentage of NULLs over time