Build Data Pipelines with Delta Live Tables

Module 04

©2023 Databricks Inc. — All rights reserved

Agenda

Build Data Pipelines with Delta Live Tables

The Medallion Architecture

Introduction to Delta Live Tables

DE 4.1 – DLT UI Walkthrough

DE 4.1A – SQL Pipelines

DE 4.1B – Python Pipelines

DE 4.2 – Python vs SQL

DE 4.3 – Pipeline Results

DE 4.4 – Pipeline Event Logs

©2023 Databricks Inc. — All rights reserved

The Medallion Architecture

©2023 Databricks Inc. — All rights reserved

Medallion Architecture in the Lakehouse

kafka

Kinesis

CSV,
JSON, TXT…

Data Lake

APACHE Spark

aws

BRONZE	SILVER	GOLD
Raw ingestion and history	Filtered, cleaned, augmented	Business-level aggregates

Data Quality & Governance

©2023 Databricks Inc. — All rights reserved

Multi-Hop in the Lakehouse

Bronze Layer

Typically just a raw copy of ingested data

Replaces traditional data lake

Provides efficient storage and querying of full, unprocessed history of data

©2023 Databricks Inc. — All rights reserved

Multi-Hop in the Lakehouse

Silver Layer

Reduces data storage complexity, latency, and redundancy

Optimizes ETL throughput and analytic query performance

Preserves grain of original data (without aggregations)

Eliminates duplicate records

Production schema enforced

Data quality checks, corrupt data quarantined

©2023 Databricks Inc. — All rights reserved

Multi-Hop in the Lakehouse

Gold Layer

Powers ML applications, reporting, dashboards, ad hoc analytics

Refined views of data, typically with aggregations

Reduces strain on production systems

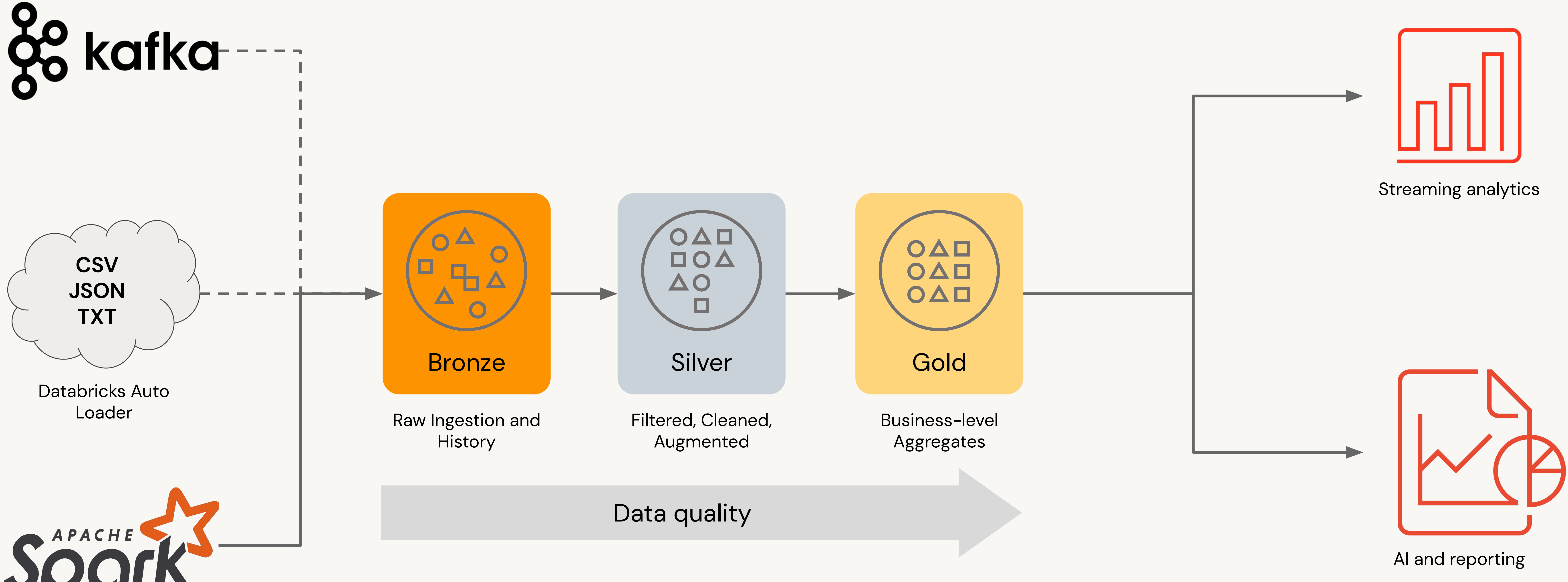
Optimizes query performance for business-critical data

©2023 Databricks Inc. — All rights reserved

Introduction to Delta Live Tables

©2023 Databricks Inc. — All rights reserved

Multi-Hop in the Lakehouse



©2023 Databricks Inc. — All rights reserved

The Reality is Not so Simple

Bronze

Silver

Gold

©2023 Databricks Inc. — All rights reserved

Large scale ETL is complex and brittle

Complex pipeline development

Hard to build and maintain table **dependencies**

Difficult to switch between **batch** and **stream** processing

Data quality and governance

Difficult to monitor and enforce **data quality**

Impossible to trace data **lineage**

Difficult pipeline operations

Poor **observability** at granular, data level

Error handling and **recovery** is laborious

©2023 Databricks Inc. — All rights reserved

Introducing Delta Live Tables

Make reliable ETL easy on Delta Lake

Operate with agility

Declarative tools to build batch and streaming data pipelines

Trust your data

DLT has built-in declarative quality controls

Declare quality expectations and actions to take

Scale with reliability

Easily scale infrastructure alongside your data

©2023 Databricks Inc. — All rights reserved

What is a LIVE TABLE?

©2023 Databricks Inc. — All rights reserved

What is a Live Table?

Live Tables are materialized views for the lakehouse.

A live table is:

- Defined by a SQL query
- Created and kept up-to-date by a pipeline

```
CREATE OR REFRESH LIVE TABLE report
AS SELECT sum(profit)
FROM prod.sales
```

Live tables provides tools to:

- Manage dependencies
- Control quality
- Automate operations
- Simplify collaboration
- Save costs
- Reduce latency

©2023 Databricks Inc. — All rights reserved

What is a Streaming Live Table?

Based on Spark™ Structured Streaming

A streaming live table is "stateful":

- Ensures exactly-once processing of input rows
- Inputs are only read once

```
CREATE STREAMING LIVE TABLE report
AS SELECT sum(profit)
FROM cloud_files(prod.sales)
```

- Streaming Live tables compute results over append-only streams such as Kafka, Kinesis, or Auto Loader (files on cloud storage)
- Streaming live tables allow you to reduce costs and latency by avoiding reprocessing of old data.

©2023 Databricks Inc. — All rights reserved

When should I use streaming?

©2022 Databricks Inc. — All rights reserved

Using Spark Structured Streaming for ingestion

Easily ingest files from cloud storage as they are uploaded

```
CREATE STREAMING LIVE TABLE raw_data
AS SELECT *
FROM cloud_files("/data", "json")
```

This example creates a table with all the json data stored in "/data":

- `cloud_files` keeps track of which files have been read to avoid duplication and wasted work
- Supports both listing and notifications for arbitrary scale
- Configurable schema inference and schema evolution

©2022 Databricks Inc. — All rights reserved

Using the SQL STREAM( ) function

Stream data from any Delta table

```
CREATE STREAMING LIVE TABLE mystream
  AS SELECT *
  FROM STREAM(my_table)
```

Pitfall: `my_table` must be an append-only source.

e.g. it may not:

- be the target of `APPLY CHANGES INTO`
- define an aggregate function
- be a table on which you've executed DML to delete/update a row (see GDPR section)

- `STREAM(my_table)` reads a stream of new records, instead of a snapshot
- `Streaming` tables must be an append-only table
- Any append-only delta table can be read as a stream (i.e. from the live schema, from the catalog, or just from a path).

©2023 Databricks Inc. — All rights reserved

How do I use DLT?

©2023 Databricks Inc. — All rights reserved

Creating Your First Live Table Pipeline

SQL to DLT in three easy steps...

Write create live table

- Table definitions are written (but not run) in notebooks
- Databricks Repos allow you to version control your table definitions.

```
1  CREATE LIVE TABLE daily_stats
2  AS SELECT sum(rev) - sum(costs) AS profits
3  FROM prod_data.transactions
4  GROUP BY day
```

Create a pipeline

- A Pipeline picks one or more notebooks of table definitions, as well as any configuration required.

Delta Live Tables

Click start

- DLT will create or update all the tables in the pipelines.

©2023 Databricks Inc. — All rights reserved

BEST PRACTICE

Development vs Production

Fast iteration or enterprise grade reliability

Development Mode

- Reuses a **long-running cluster** running for fast iteration.
- **No retries** on errors enabling faster debugging.

Production Mode

- Cuts costs by **turning off clusters** as soon as they are done (within 5 minutes)
- **Escalating retries**, including cluster restarts, ensure reliability in the face of transient issues.

In the Pipelines UI:

Development	Production

©2023 Databricks Inc. — All rights reserved

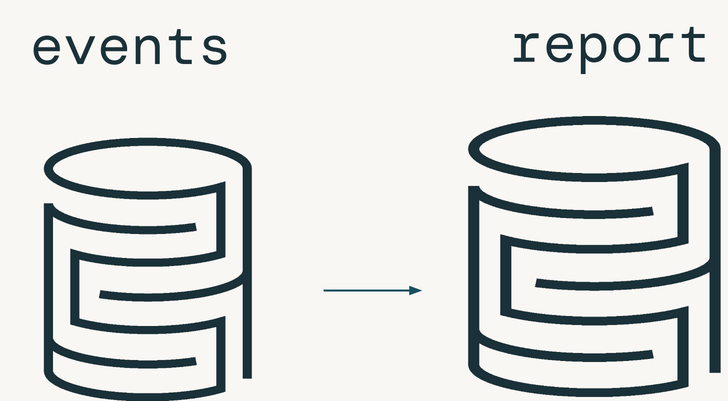
What if I have dependent tables?

©2023 Databricks Inc. — All rights reserved

Declare LIVE Dependencies

Using the LIVE virtual schema.

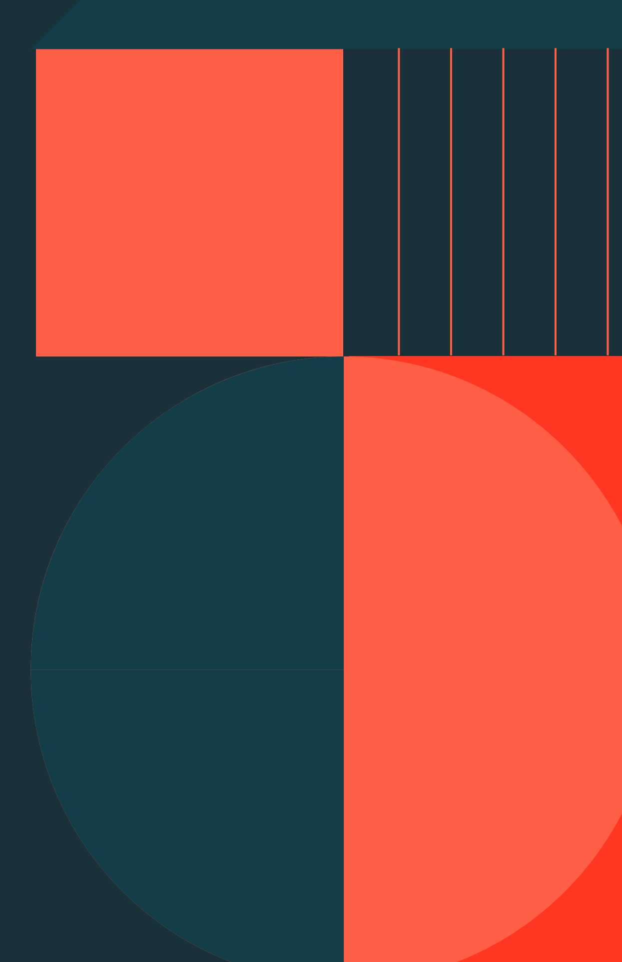
```
CREATE LIVE TABLE events
AS SELECT … FROM prod.raw_data


CREATE LIVE TABLE report
AS SELECT … FROM LIVE.events
```



- Dependencies owned by other producers are just read from the catalog or spark data source as normal.
- LIVE dependencies, from the **same pipeline**, are read from the LIVE schema.
- DLT detects LIVE dependencies and executes all operations in **correct order**.
- DLT handles parallelism and captures the lineage of the data.

©2023 Databricks Inc. — All rights reserved

How do I ensure Data Quality?



©2023 Databricks Inc. — All rights reserved

BEST PRACTICE

Ensure correctness with Expectations

Expectations are tests that ensure data quality in production

```
CONSTRAINT valid_timestamp
EXPECT (timestamp > '2012-01-01')
ON VIOLATION DROP
```

```
@dlt.expect_or_drop(
  "valid_timestamp",
  col("timestamp") > '2012-01-01')
```

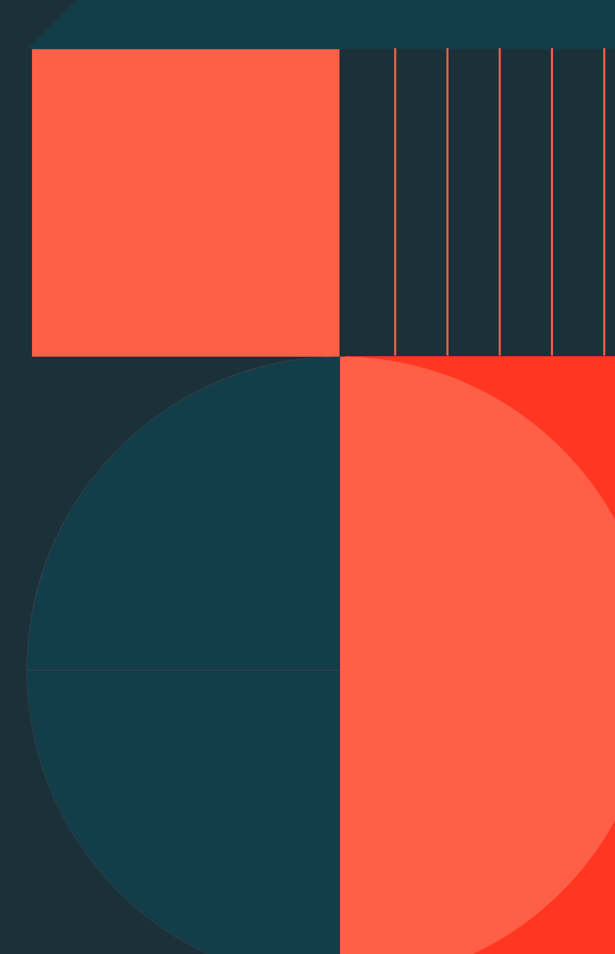
Expectations are true/false expressions that are used to validate each row during processing.

DLT offers flexible policies on how to handle records that violate expectations:

- Track number of bad records
- Drop bad records
- Abort processing for a single bad record

©2023 Databricks Inc. — All rights reserved

What about operations?

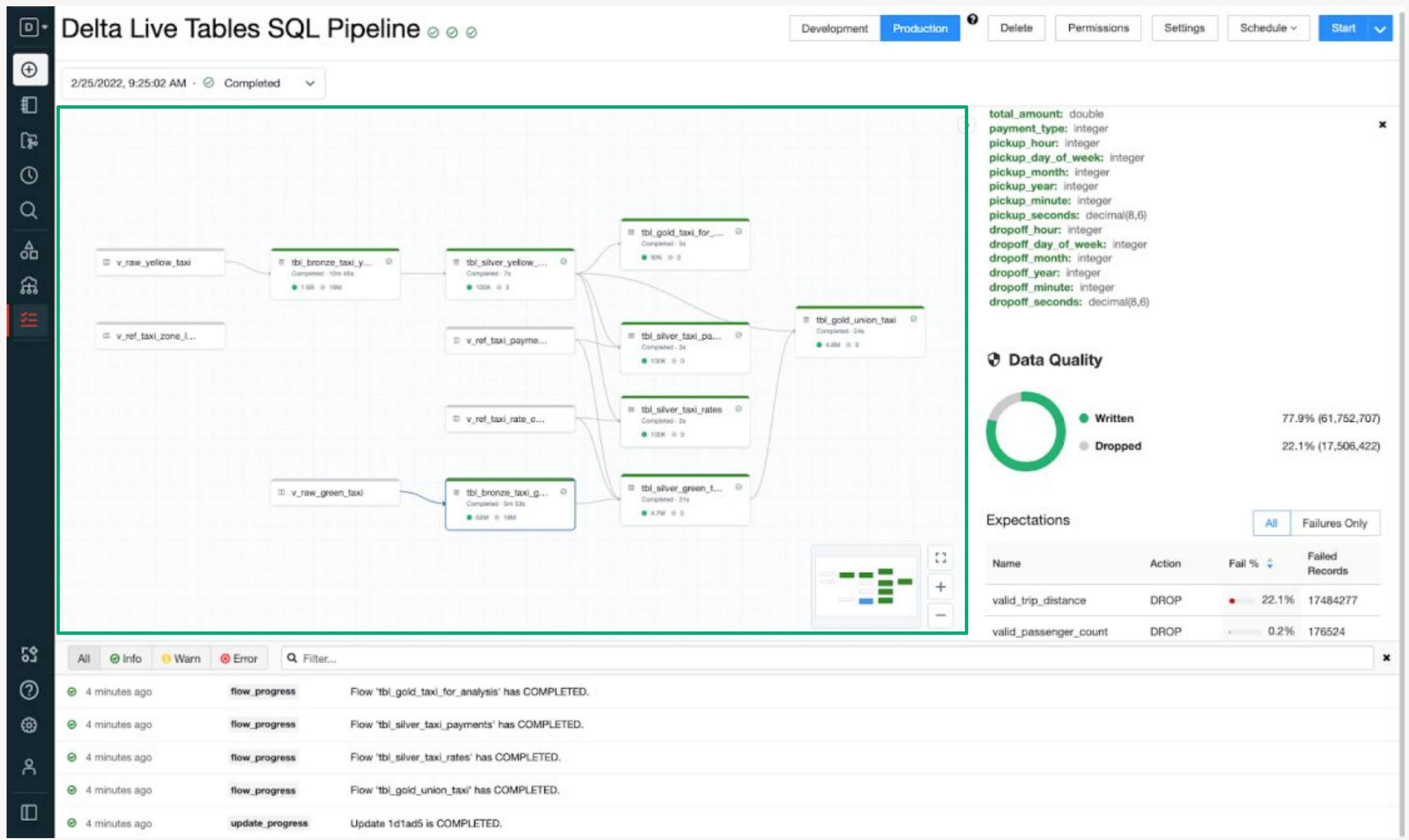


©2023 Databricks Inc. — All rights reserved

Pipelines UI (1 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables

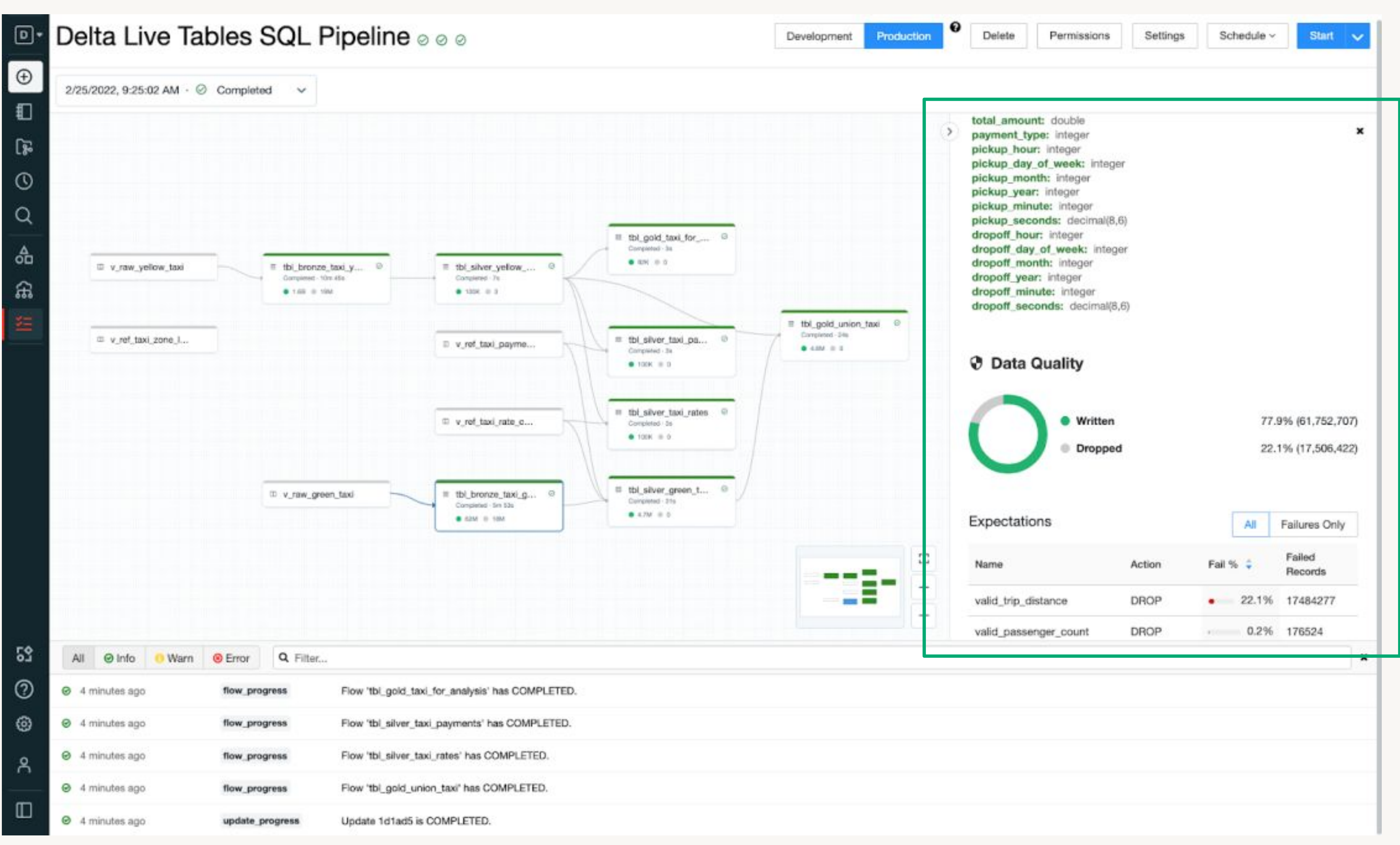


©2023 Databricks Inc. — All rights reserved

Pipelines UI (2 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables
- Discover metadata and quality of each table

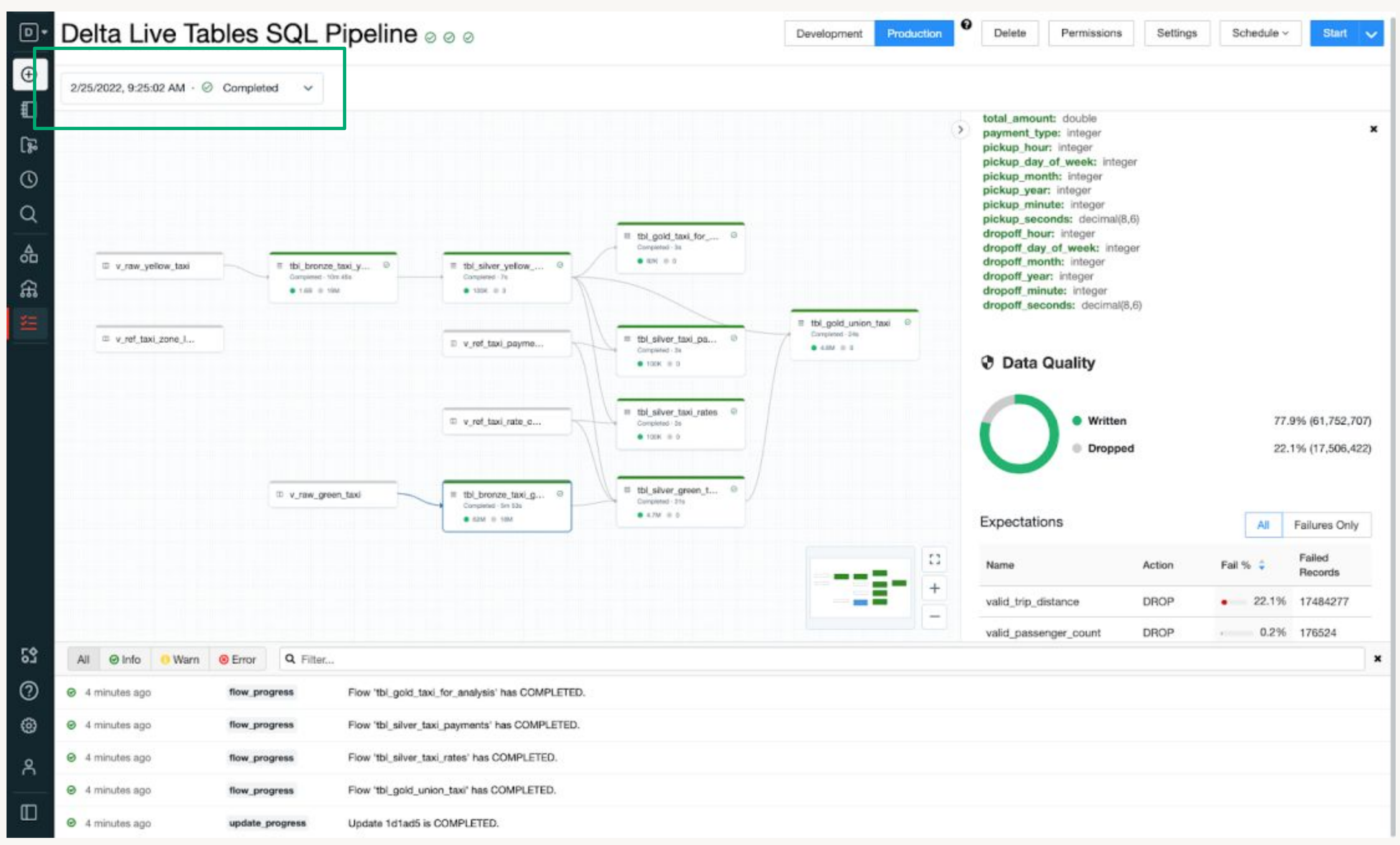


©2023 Databricks Inc. — All rights reserved

Pipelines UI (3 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates

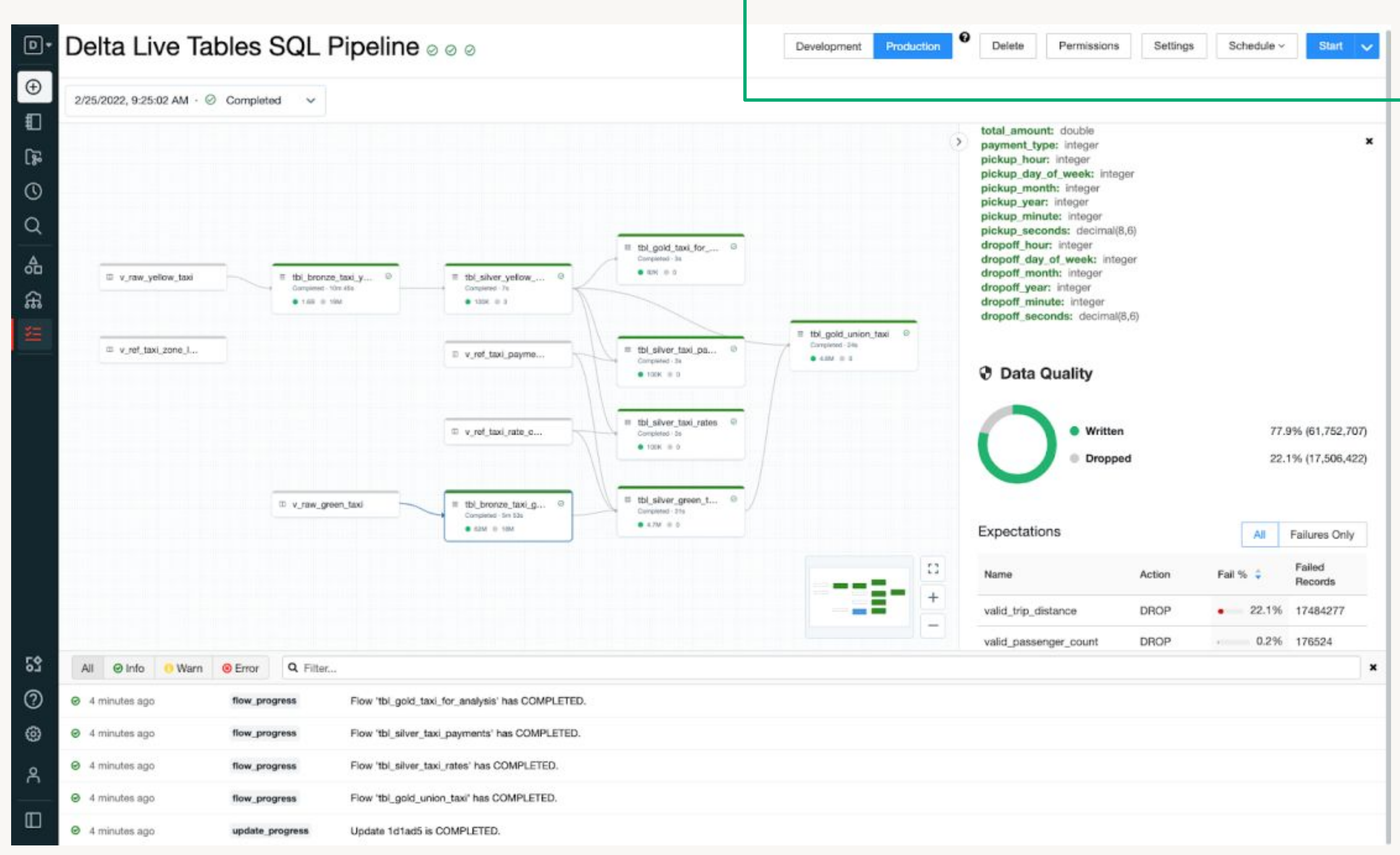


©2023 Databricks Inc. — All rights reserved

Pipelines UI (4 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates
- Control operations



©2023 Databricks Inc. — All rights reserved

Pipelines UI (5 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates
- Control operations
- Dive deep into events

Delta Live Tables SQL Pipeline

Development | Production | Delete | Permissions | Settings | Schedule | Start

2/25/2022, 9:25:02 AM · Completed

v_raw_yellow_taxi
tbl_bronze_taxi_y...
tbl_silver_yellow_...
tbl_gold_taxi_for_...
v_ref_taxi_zone_l...
v_ref_taxi_payme...
tbl_silver_taxi_pa...
tbl_gold_union_taxi
v_ref_taxi_rate_c...
tbl_silver_taxi_rates
v_raw_green_taxi
tbl_bronze_taxi_g...
tbl_silver_green_t...

total_amount: double
payment_type: integer
pickup_hour: integer
pickup_day_of_week: integer
pickup_month: integer
pickup_year: integer
pickup_minute: integer
pickup_seconds: decimal(8,6)
dropoff_hour: integer
dropoff_day_of_week: integer
dropoff_month: integer
dropoff_year: integer
dropoff_minute: integer
dropoff_seconds: decimal(8,6)

Data Quality

Written 77.9% (61,752,707)
Dropped 22.1% (17,506,422)

Expectations

All | Failures Only

Name	Action	Fail %	Failed Records
valid_trip_distance	DROP	22.1%	17484277

All | Info | Warn | Error | Filter...

4 minutes ago	flow_progress	Flow 'tbl_gold_taxi_for_analysis' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_silver_taxi_payments' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_silver_taxi_rates' has COMPLETED.
4 minutes ago	flow_progress	Flow 'tbl_gold_union_taxi' has COMPLETED.
4 minutes ago	update_progress	Update 1d1ad5 is COMPLETED.

©2023 Databricks Inc. — All rights reserved

The Event Log

The event log automatically records all pipelines operations.

Operational Statistics

Time and current status, for all operations

Pipeline and cluster configurations

Row counts

Provenance

Table schemas, definitions, and declared properties

Table-level lineage

Query plans used to update tables

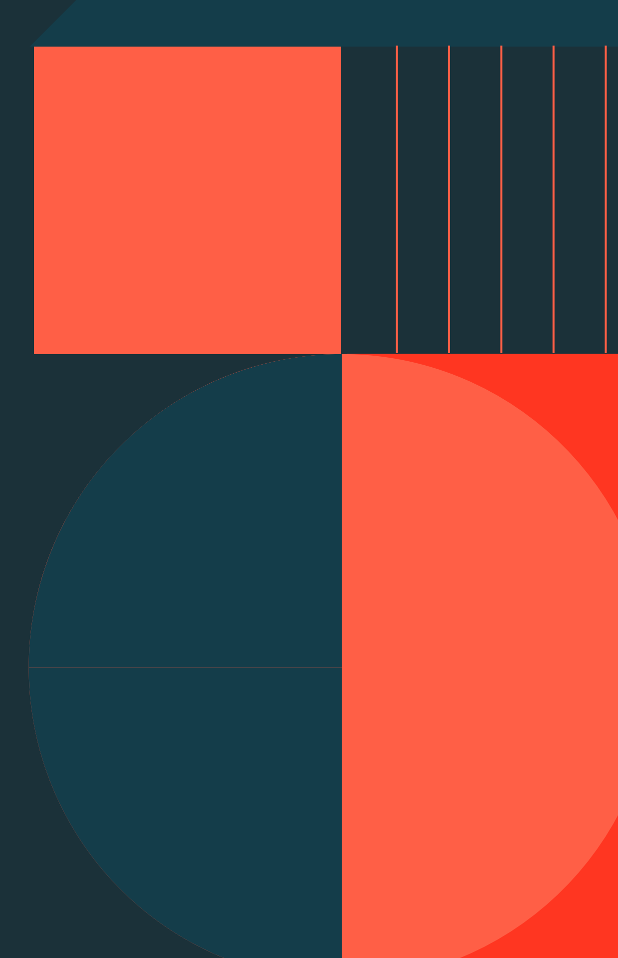
Data Quality

Expectation pass / failure / drop statistics

Input/Output rows that caused expectation failures

©2023 Databricks Inc. — All rights reserved

How can I use parameters?



©2022 Databricks Inc. — All rights reserved

Modularize your code with configuration

Avoid hard coding paths, topic names, and other constants in your code.

A pipeline's configuration is a map of key value pairs that can be used to parameterize your code:

- Improve code readability/maintainability
- Reuse code in multiple pipelines for different data

Configuration

my_etl.input_path	s3://my-data/json/

Add configuration

```
CREATE STREAMING LIVE TABLE data AS
SELECT * FROM cloud_files("${my_etl.input_path}", "json")
```

```
@dlt.table
def data():
  input_path = spark.conf.get("my_etl.input_path")
  spark.readStream.format("cloud_files").load(input_path)
```

©2023 Databricks Inc. — All rights reserved

How can I do change data capture (CDC)?

©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

{UPDATE}
{DELETE}
{INSERT}

Up-to-date Snapshot

©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A target for the changes to be applied to.

©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A source of changes, currently this has to be a stream.

city_updates

```
{"id": 1, "ts": 1, "city": "Bekerly, CA"}
```

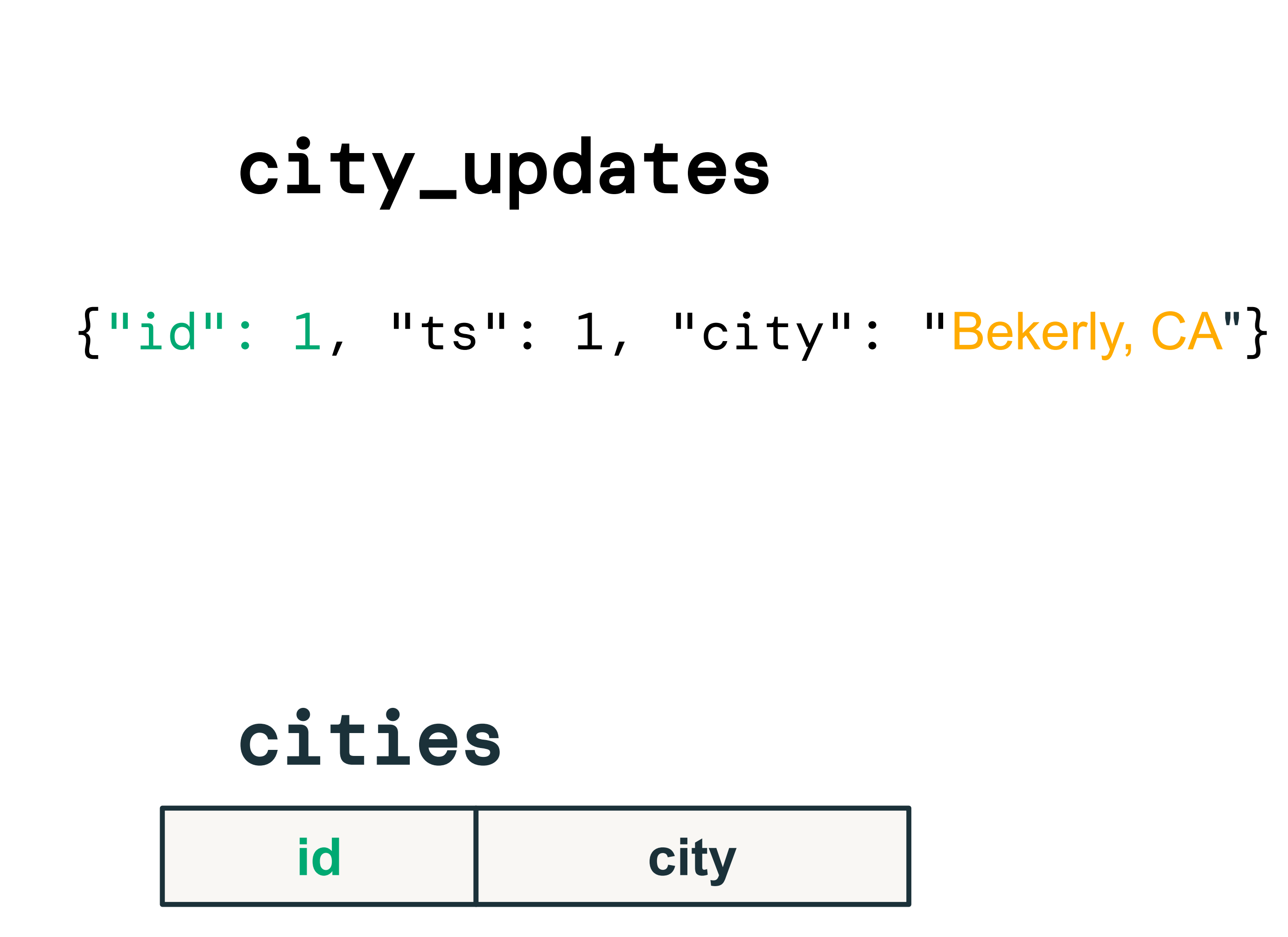
©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A unique key that can be used to identify a given row.



©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A sequence that can be used to order changes:

- Log sequence number (lsn)
- Timestamp
- Ingestion time

city_updates

```
{"id": 1, "ts": 100, "city": "Bekerly, CA"}
```

cities

id	city

©2023 Databricks Inc. — All rights reserved

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

city_updates

```
{"id": 1, "ts": 100, "city": "Bekerly, CA"}
{"id": 1, "ts": 200, "city": "Berkeley, CA"}
```

cities

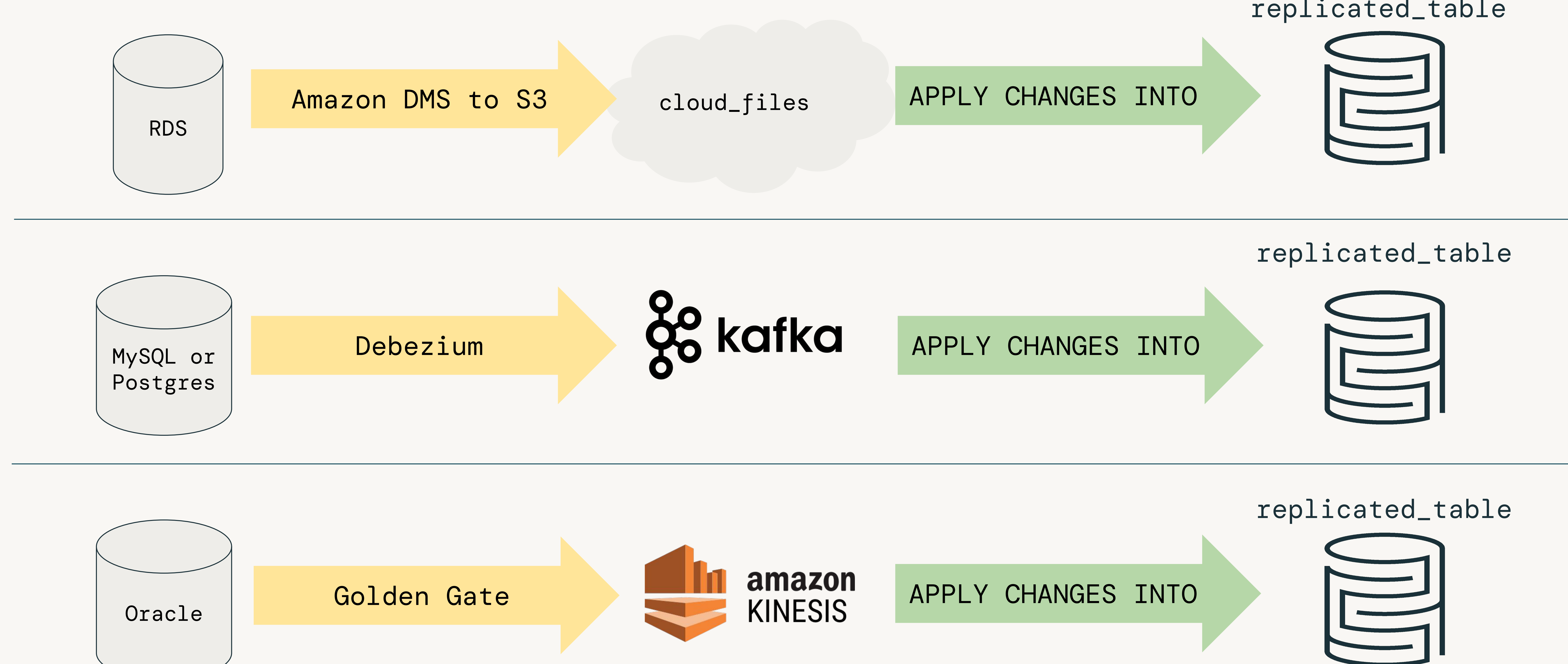
id	city
1	~~Bekerly, CA~~ Berkeley, CA

©2023 Databricks Inc. — All rights reserved

REFERENCE ARCHITECTURE

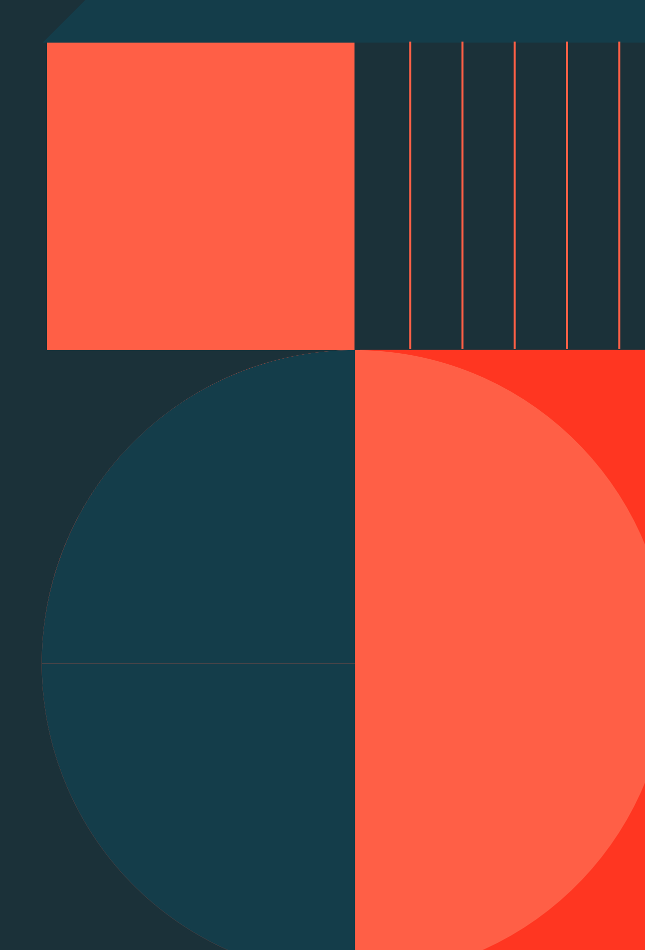
Change Data Capture (CDC) from RDBMS

A variety of 3rd party tools can provide a streaming change feed



©2023 Databricks Inc. — All rights reserved

What do I *no longer* need to manage with DLT?



©2023 Databricks Inc. — All rights reserved

Automated Data Management

DLT automatically optimizes data for performance & ease-of-use

Best Practices

What:

DLT encodes Delta best practices automatically when creating DLT tables.

How:

DLT sets the following properties:

- `optimizeWrite`
- `autoCompact`
- `tuneFileSizesForRewrites`

Physical Data

What:

DLT automatically manages your physical data to minimize cost and optimize performance.

How:

- runs vacuum daily
- runs optimize daily

You still can tell us how you want it organized (ie ZORDER)

Schema Evolution

What:

Schema evolution is handled for you

How:

Modifying a live table transformation to add/remove/rename a column will automatically do the right thing.

When removing a column in a streaming live table, old values are preserved.

©2023 Databricks Inc. — All rights reserved

DE 4.1 - Using the Delta Live Tables UI

Deploy a DLT pipeline

Explore the resultant DAG

Execute an update of the pipeline

©2023 Databricks Inc. — All rights reserved

DE 4.1.1 - Fundamentals of DLT Syntax

Declaring Delta Live Tables

Ingesting data with Auto Loader

Using parameters in DLT Pipelines

Enforcing data quality with constraints

Adding comments to tables

Describing differences in syntax and execution of live tables and streaming live tables

©2023 Databricks Inc. — All rights reserved

DE 4.1.2 - More DLT SQL Syntax

Processing CDC data with APPLY CHANGES INTO

Declaring live views

Joining live tables

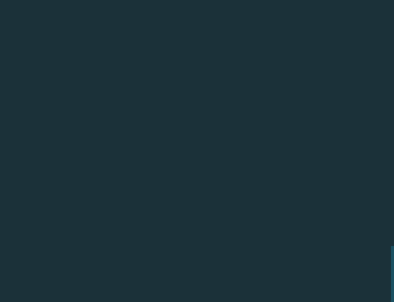
Describing how DLT library notebooks work together in a pipeline

Scheduling multiple notebooks in a DLT pipeline

©2023 Databricks Inc. — All rights reserved

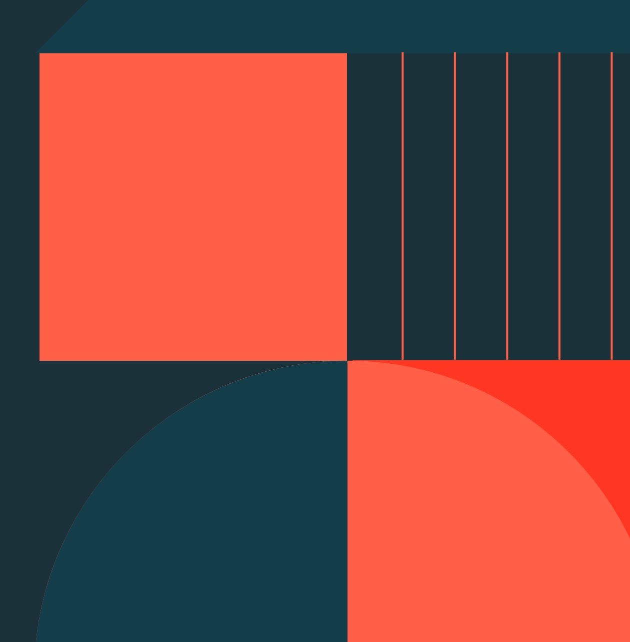
DE 4.2 - Delta Live Tables: Python vs SQL

Identify key differences between the Python and SQL implementations of Delta Live Tables



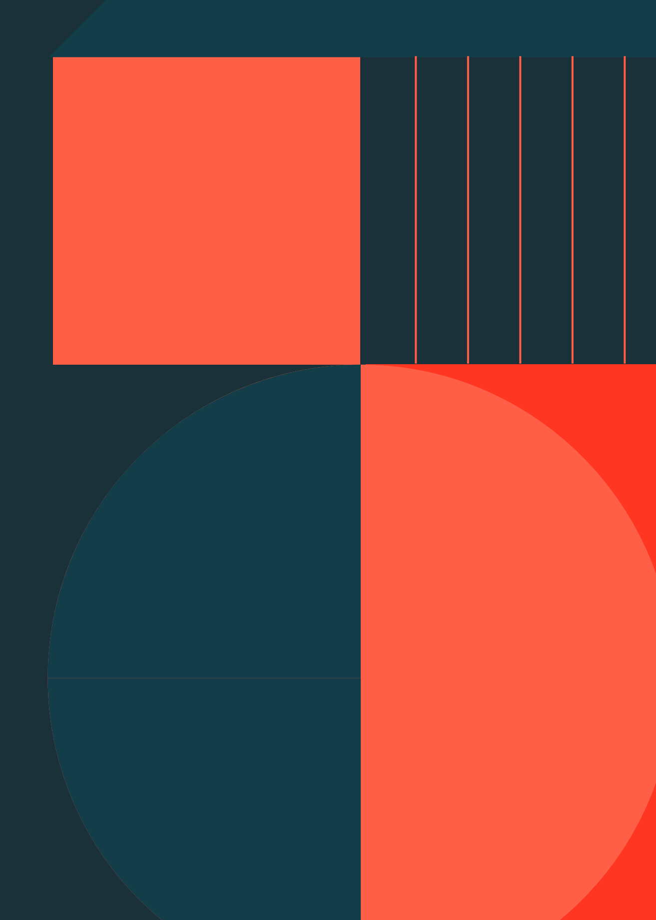
©2023 Databricks Inc. — All rights reserved

DE 4.3 - Exploring the Results of a DLT Pipeline



©2023 Databricks Inc. — All rights reserved

DE 4.4 - Exploring the Pipeline Events Logs



©2023 Databricks Inc. — All rights reserved

DE 4.1.3 - Troubleshooting DLT Syntax Lab

Identifying and troubleshooting DLT syntax

Iteratively developing DLT pipelines with notebooks

©2023 Databricks Inc. — All rights reserved

databricks

©2023 Databricks Inc. — All rights reserved