databricks

Build Data Pipelines with Lakeflow Declarative Pipelines

Databricks Academy

Course Learning Objectives

- Understand the core concepts and components of Lakeflow Declarative Pipelines, including the function and differences between streaming tables, materialized views, and temporary views.
- Identify and configure Lakeflow pipeline settings, such as compute, data assets, trigger modes, and advanced options.
- Develop a functional Lakeflow Declarative Pipeline using the new pipeline editor and SQL-based syntax.
- Incorporate data quality expectations into a pipeline to validate and enforce data integrity.
- Analyze event logs and pipeline metrics to understand the full execution and lifecycle of a Lakeflow Declarative Pipeline.
- Design and implement a Change Data Capture (CDC) to a pipeline using APPLY CHANGES INTO to handle slowly changing dimensions (SCD).

Agenda

Course Sections
• Introduction to Data Engineering in Databricks
• Lakeflow Declarative Pipeline Fundamentals
• Building Lakeflow Declarative Pipelines

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Databricks Workspaces
- Apache Spark
- Delta Lake and the Medallion Architecture
- Unity Catalog

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Databricks Workspaces
- Apache Spark
- Delta Lake and the Medallion Architecture
- Unity Catalog

Knowledge of Ingesting Raw Data into a Table

- Familiarity with ingesting raw data source files using the **read_files** SQL function
  - CSV
  - JSON
  - TXT
  - Parquet

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Databricks Workspaces
- Apache Spark
- Delta Lake and the Medallion Architecture
- Unity Catalog

Knowledge of Ingesting Raw Data into a Table

- Familiarity with ingesting raw data source files using the **read_files** SQL function
  - CSV
  - JSON
  - TXT
  - Parquet

Experience Transforming Data with SQL

- Experience in writing **intermediate-level queries** using SQL
- Basic knowledge of **SQL Joins**

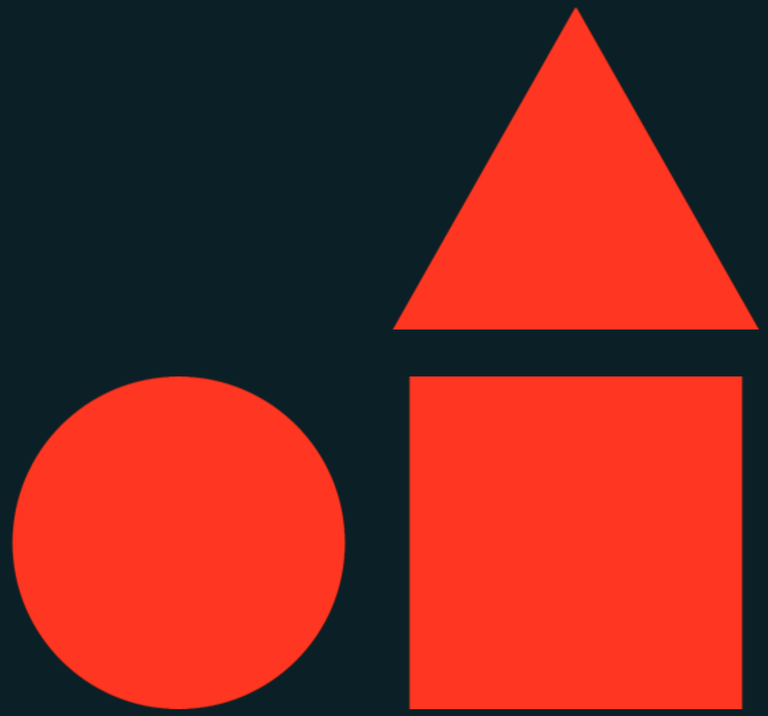
Lab Exercise Environment

Technical Details

- Your lab environment is provided by Vocareum.
- It will open in a new tab.
- It has been configured with the permissions and resources required to accomplish the tasks outlined in the lab exercise.
- Third party cookies must be enabled in your browser for Vocareum's user experience to work properly.
- Make sure to enable pop ups!

databricks

Introduction to Data Engineering in Databricks



Build Data Pipelines with Lakeflow Declarative Pipelines

Section Learning Objectives

- Understand the components of Data Engineering with Lakeflow in Databricks with Connect, Declarative Pipelines, and Jobs.
- Explain how Lakeflow Declarative Pipelines incrementally processes data using Streaming Tables and Materialized Views in batch or streaming workloads.
- Demonstrate how to create a Lakeflow Declarative Pipelines project in a Databricks Workspace.

Agenda

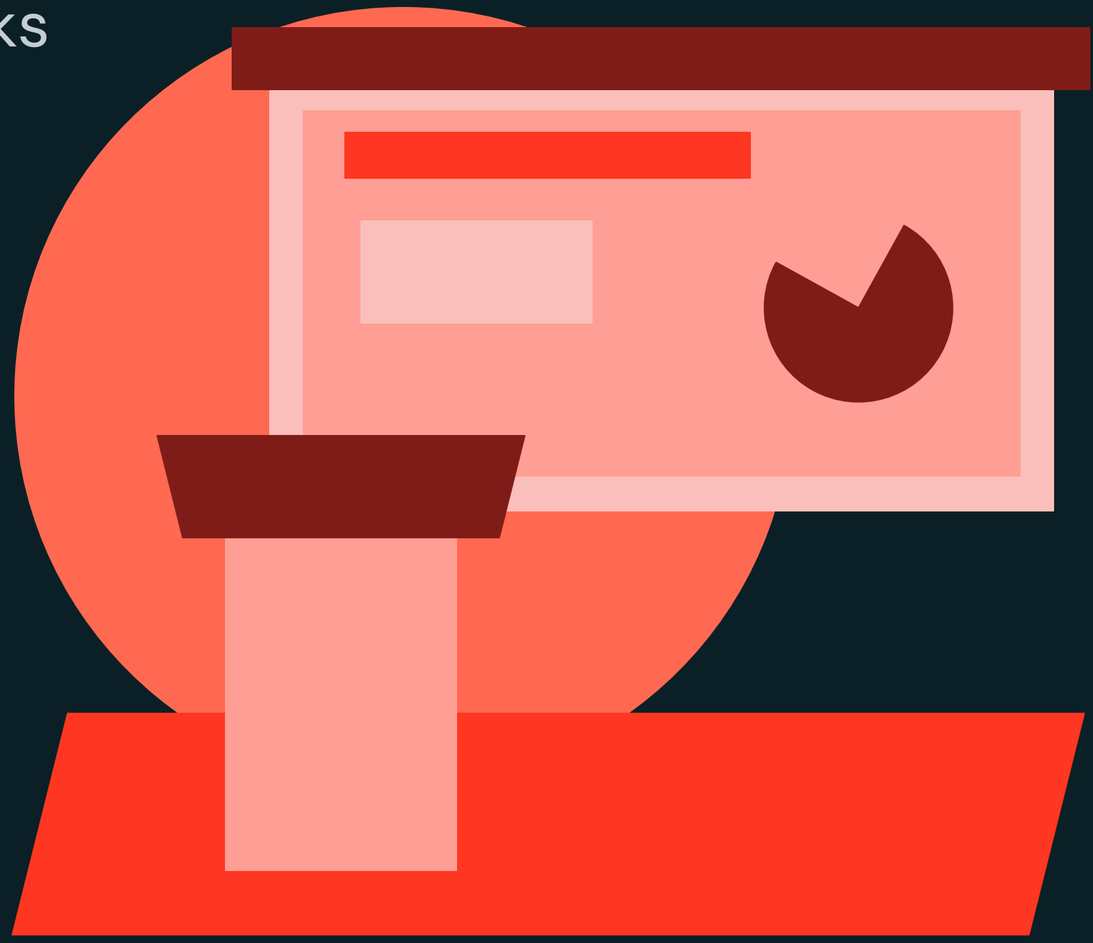
Section Overview – Introduction to Data Engineering in Databricks
• Data Engineering in Databricks
• What are Lakeflow Declarative Pipelines?
• Course Setup and Creating a Pipeline
• Course Project Overview

databricks

Introduction to Data Engineering in Databricks

LECTURE

Data Engineering in Databricks



Data Engineering in Databricks

databricks LAKEFLOW

UNIFIED DATA ENGINEERING FOR THE DATA INTELLIGENCE PLATFORM

CONNECT	DECLARATIVE PIPELINES (Previously DLT)	JOBS (Previously Workflows)
 Efficient ingestion connectors	 Accelerated ETL development	 Reliable orchestration for analytics and AI

INDUSTRY LEADING DATA PROCESSING ENGINE
(Apache Spark + Structured Streaming)

UNIFIED GOVERNANCE	Unity Catalog
OPTIMIZED STORAGE	DELTA LAKE Parquet ICEBERG

Data Engineering in Databricks

Simplifying batch and streaming ETL with automated reliability, optimizations and built-in data quality with **Lakeflow Declarative Pipelines**!

databricks LAKEFLOW

UNIFIED DATA ENGINEERING FOR THE DATA INTELLIGENCE PLATFORM

CONNECT

Efficient ingestion connectors

DECLARATIVE PIPELINES
(Previously DLT)

Accelerated ETL development

JOBS
(Previously Workflows)

Reliable orchestration for analytics and AI

INDUSTRY LEADING DATA PROCESSING ENGINE
(Apache Spark + Structured Streaming)

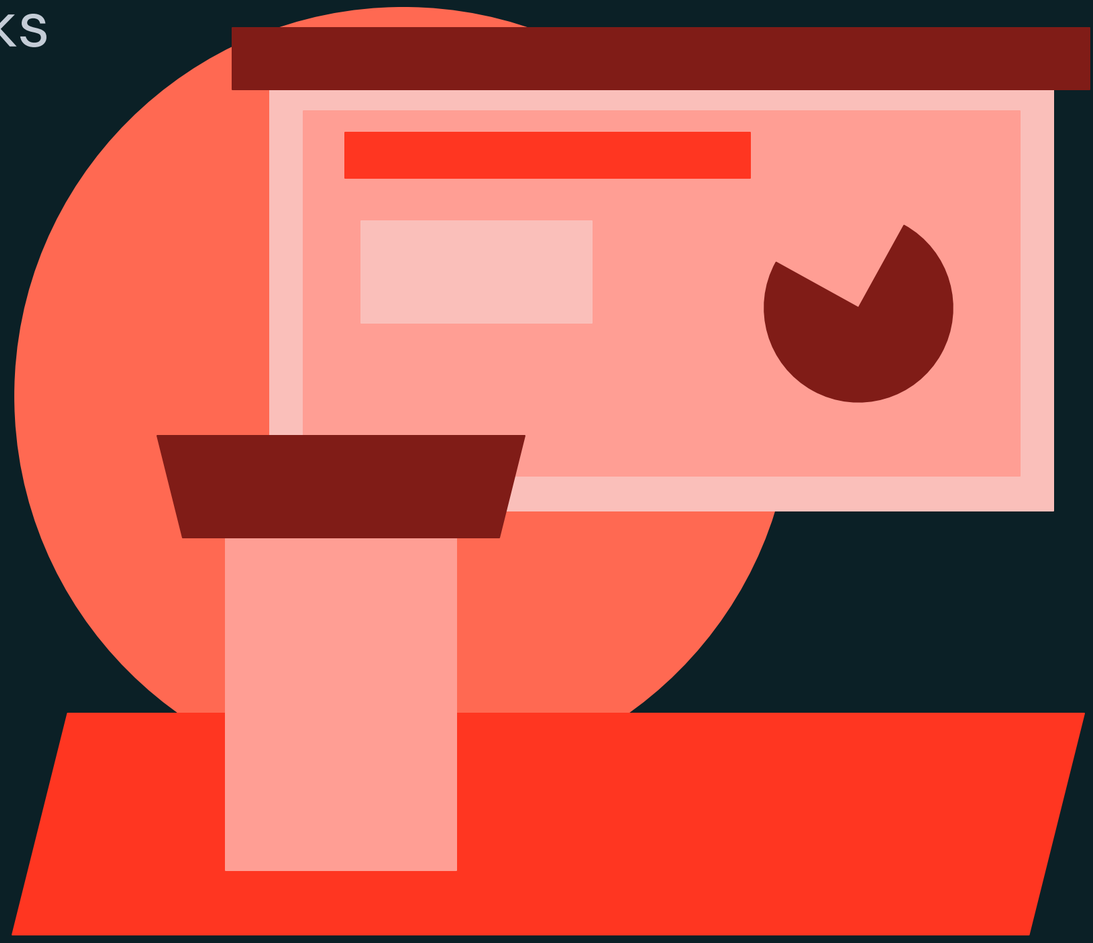
UNIFIED GOVERNANCE	Unity Catalog
OPTIMIZED STORAGE	DELTA LAKE Parquet ICEBERG

databricks

Introduction to Data Engineering in Databricks

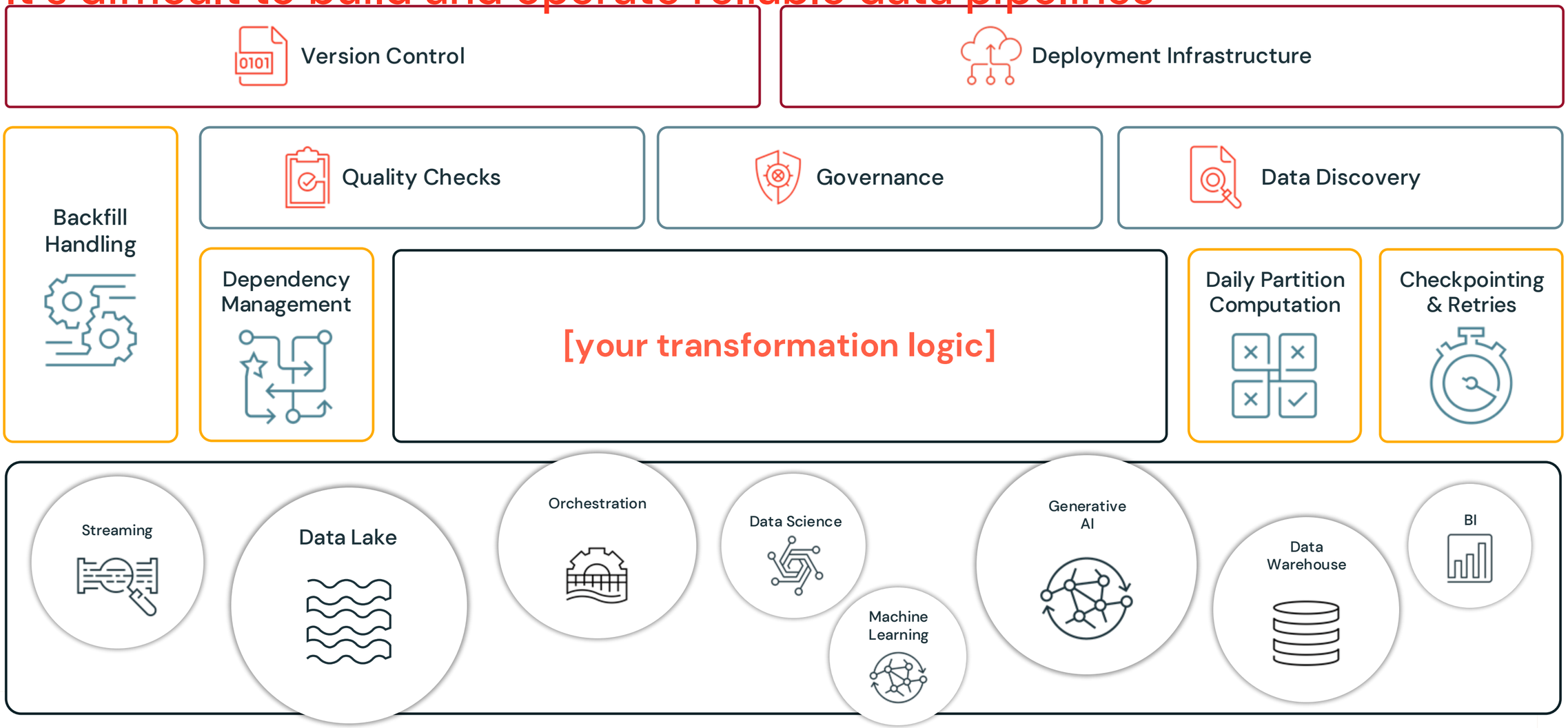
LECTURE

What are Lakeflow Declarative Pipelines?



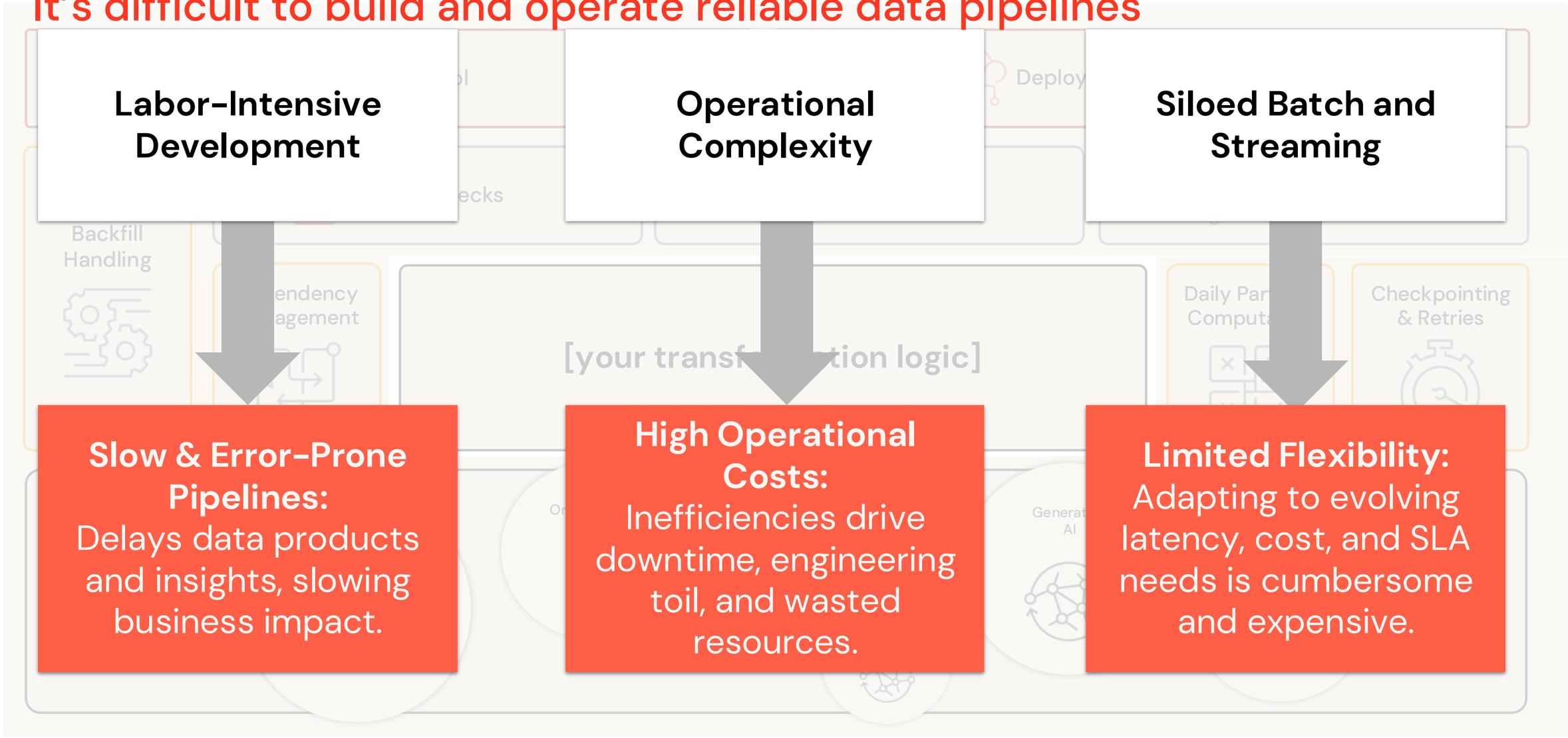
What are Lakeflow Declarative Pipelines?

It's difficult to build and operate reliable data pipelines



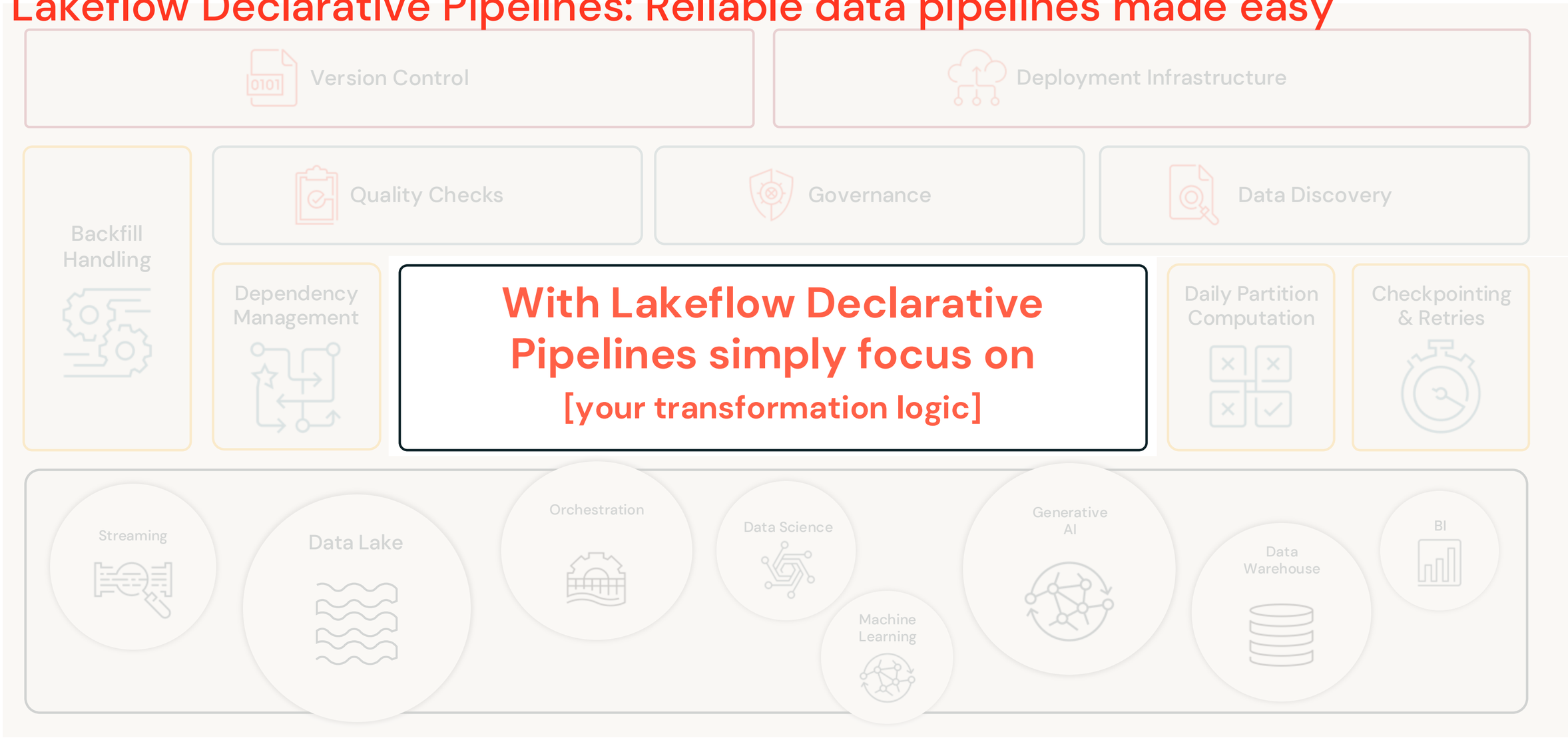
What are Lakeflow Declarative Pipelines?

It’s difficult to build and operate reliable data pipelines



What are Lakeflow Declarative Pipelines?

Lakeflow Declarative Pipelines: Reliable data pipelines made easy



What are Lakeflow Declarative Pipelines?

Introducing Declarative Pipelines: Reliable data pipelines made easy

- **Simplified Pipeline Authoring**

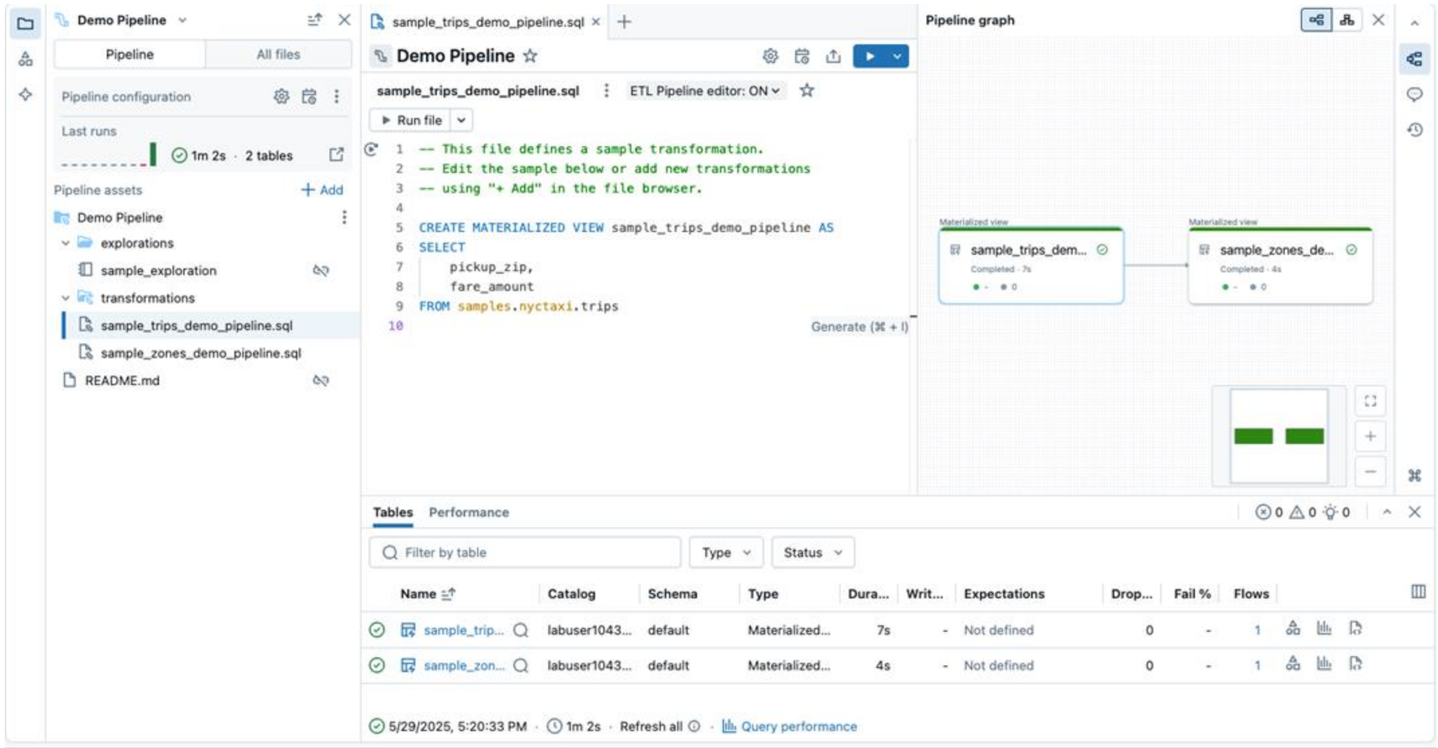
  Easily declare data ingestion and transformations with **SQL or Python,** and let Lakeflow Declarative Pipelines handle the rest!

- **Intelligent Optimization at Scale**

  Automated scaling and recovery improve reliability and reduce maintenance.

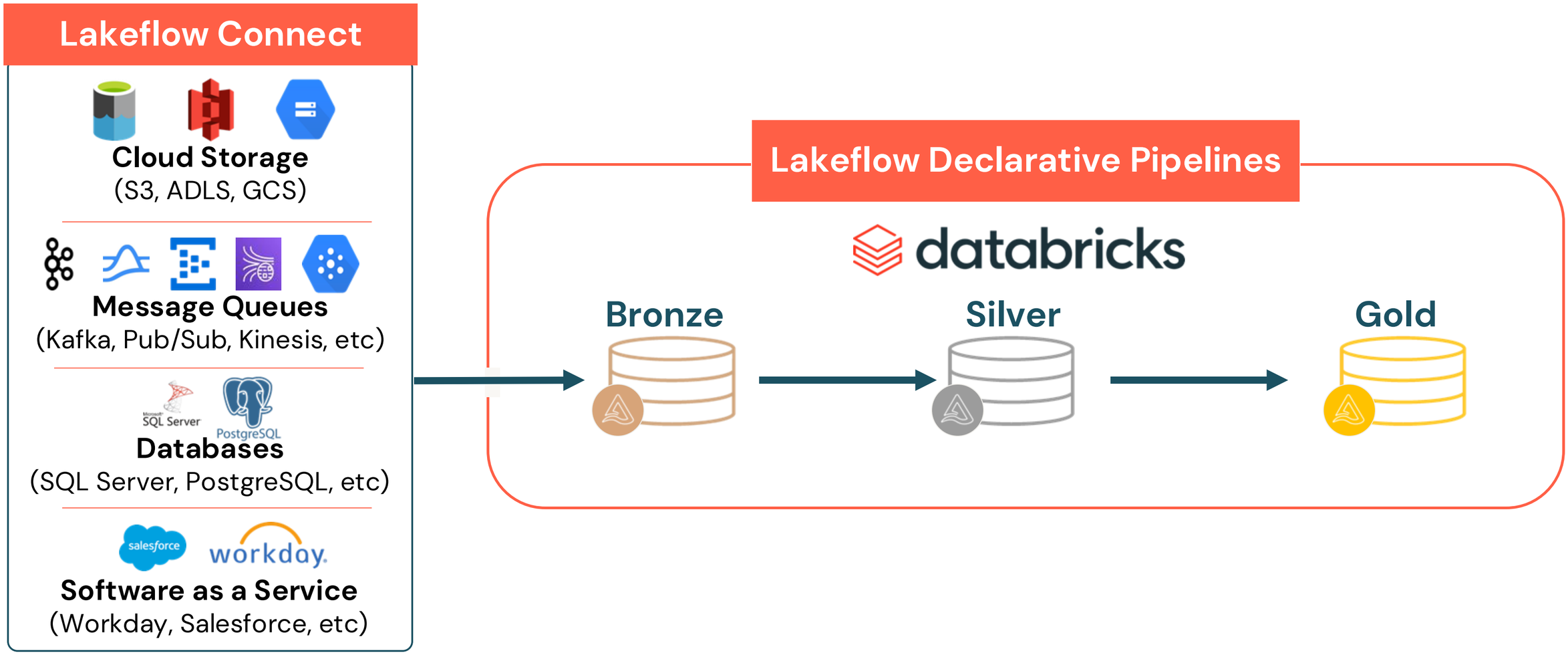
- **Unified Batch and Streaming**

  Pipelines seamlessly adapt to **both near real-time and batch workloads,** optimizing performance and cost.



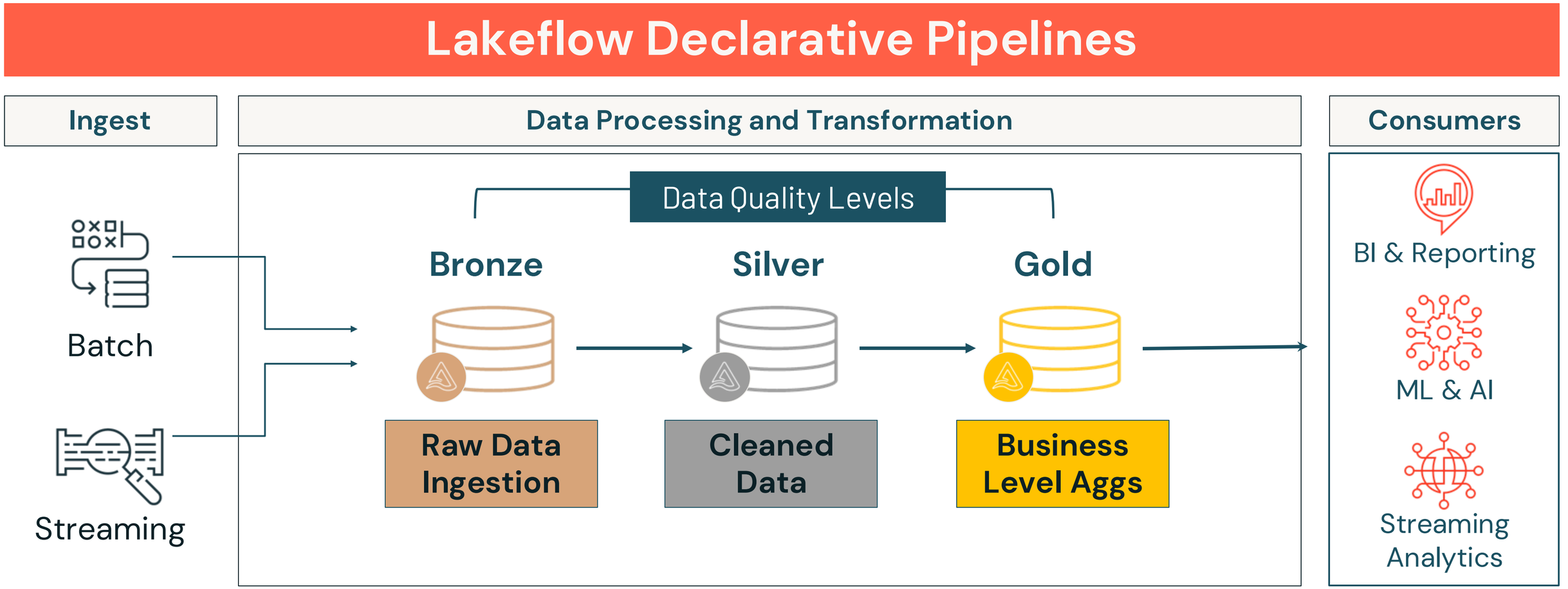
What are Lakeflow Declarative Pipelines?

Connecting to Data Sources



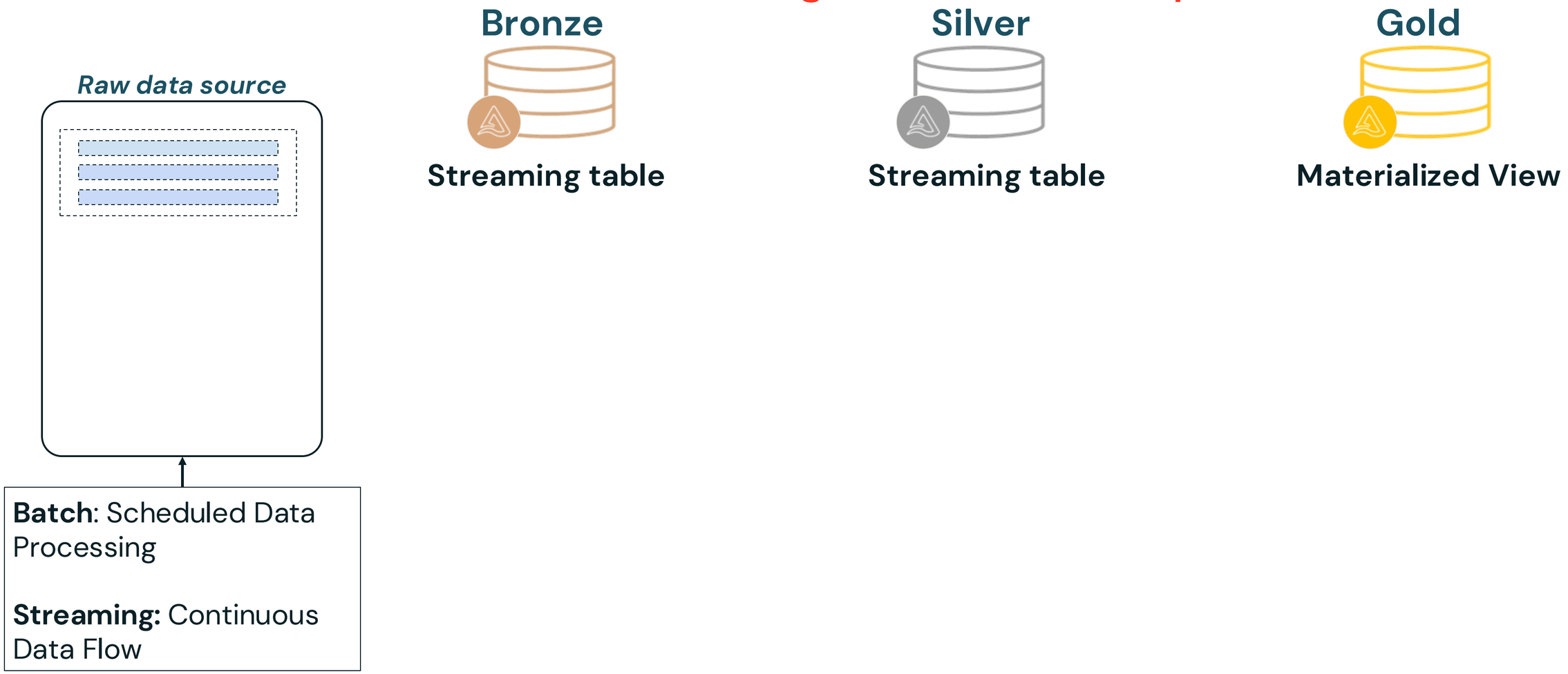
What are Lakeflow Declarative Pipelines?

Simplifying Batch and Streaming ETL in the Medallion Architecture



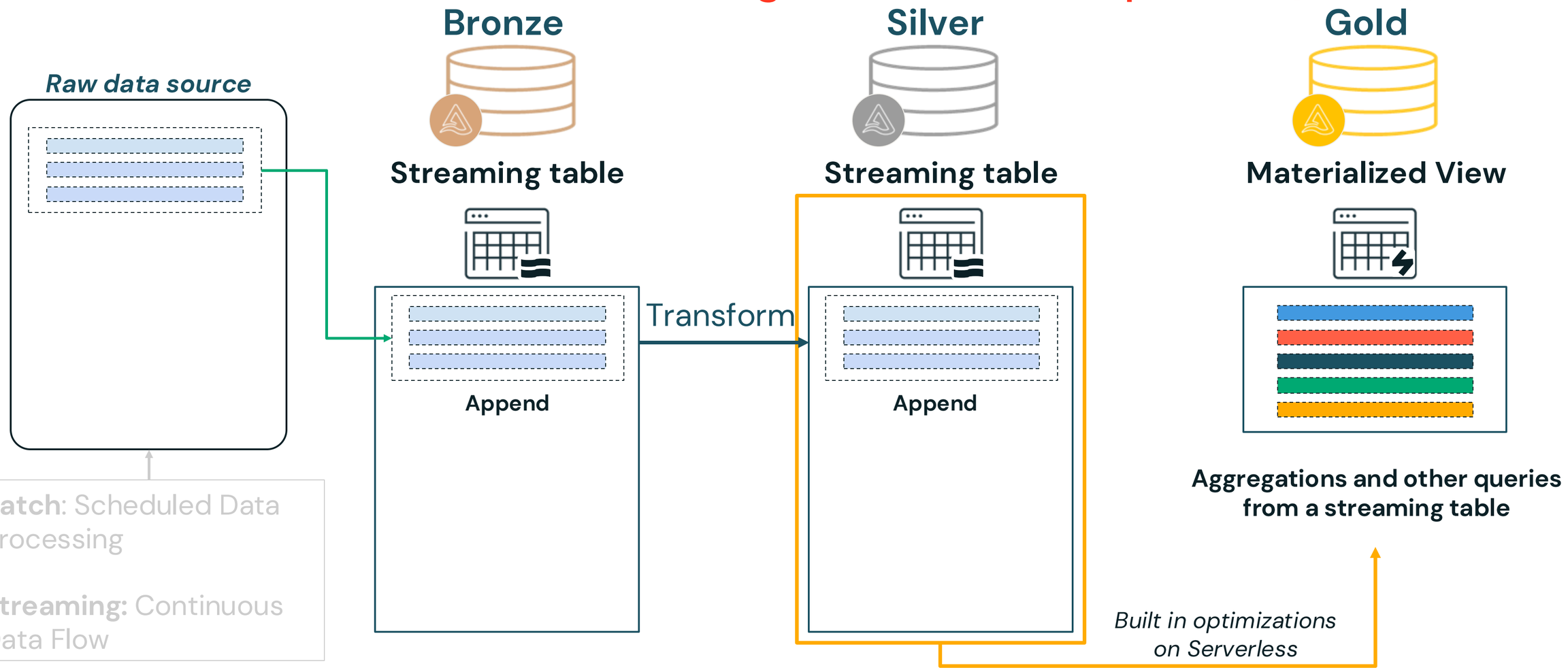
What are Lakeflow Declarative Pipelines?

Overview of Incremental Processing in Declarative Pipelines



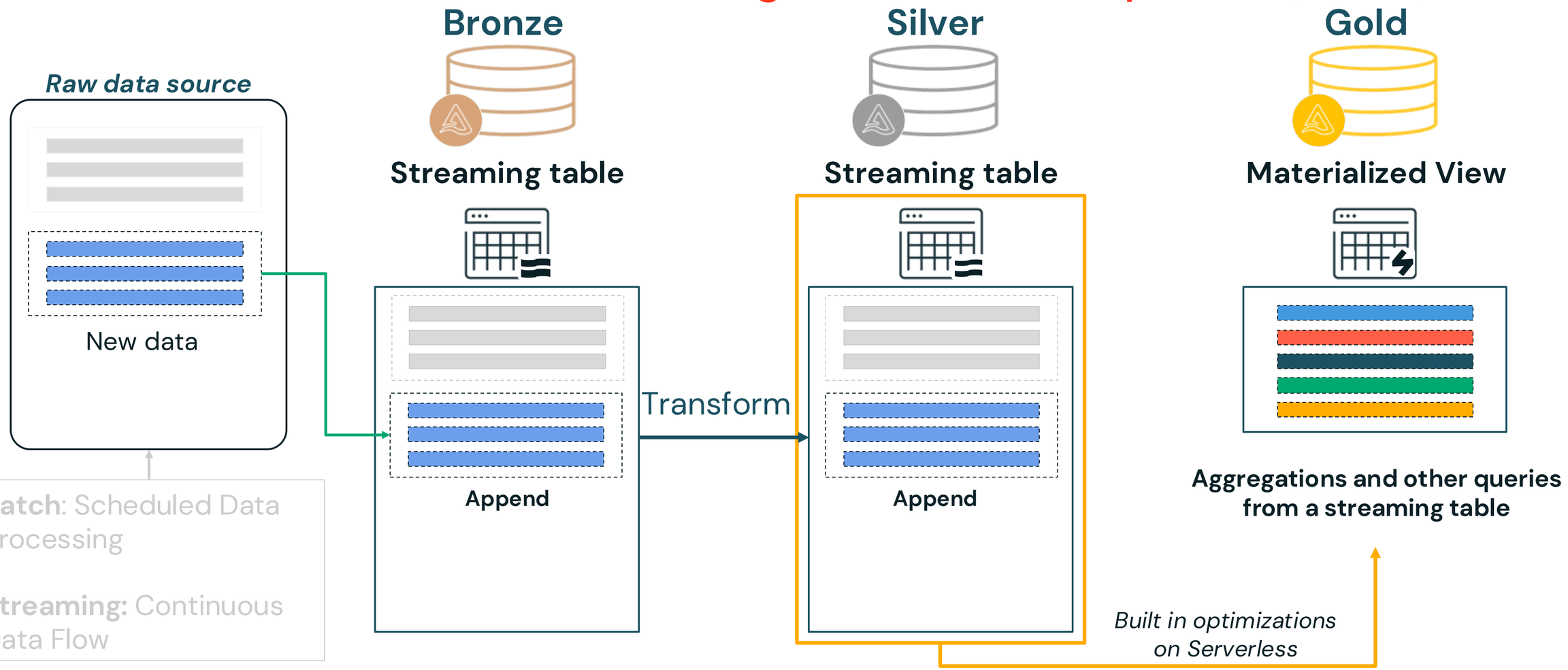
What are Lakeflow Declarative Pipelines?

Overview of Incremental Processing in Declarative Pipelines (Run 1)

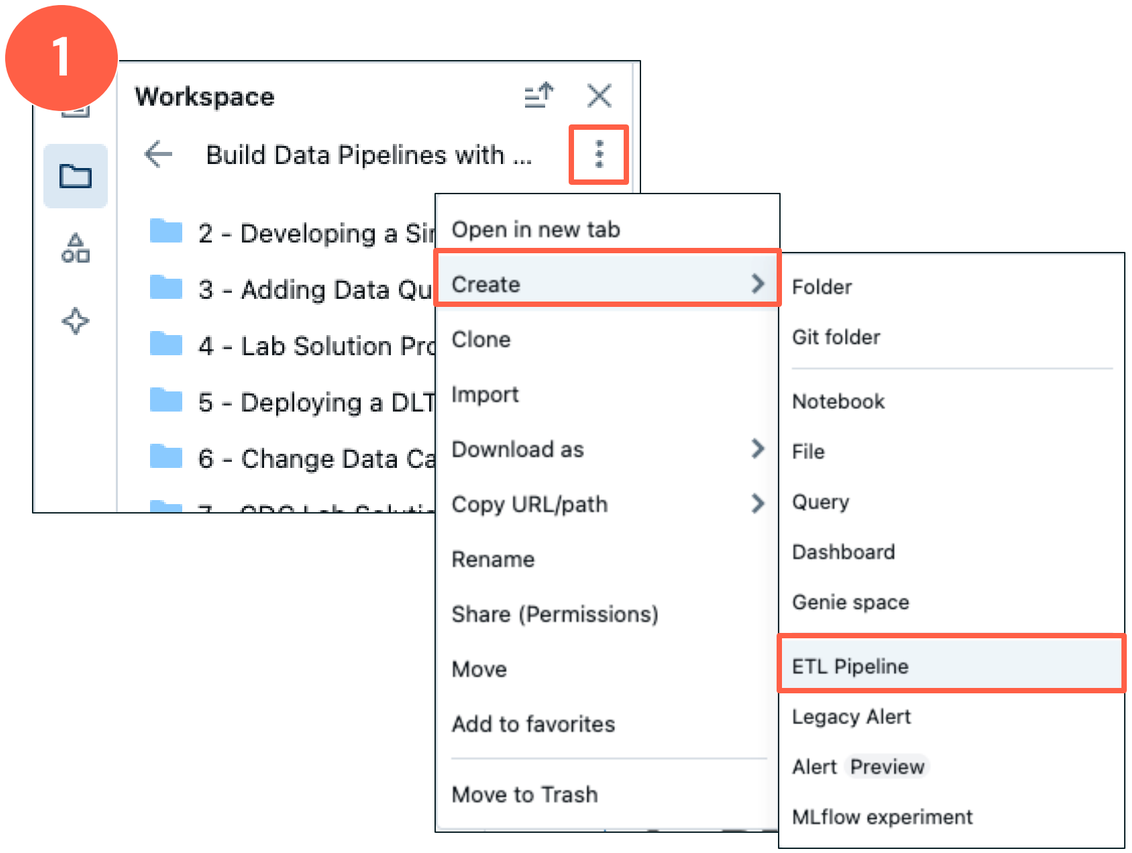


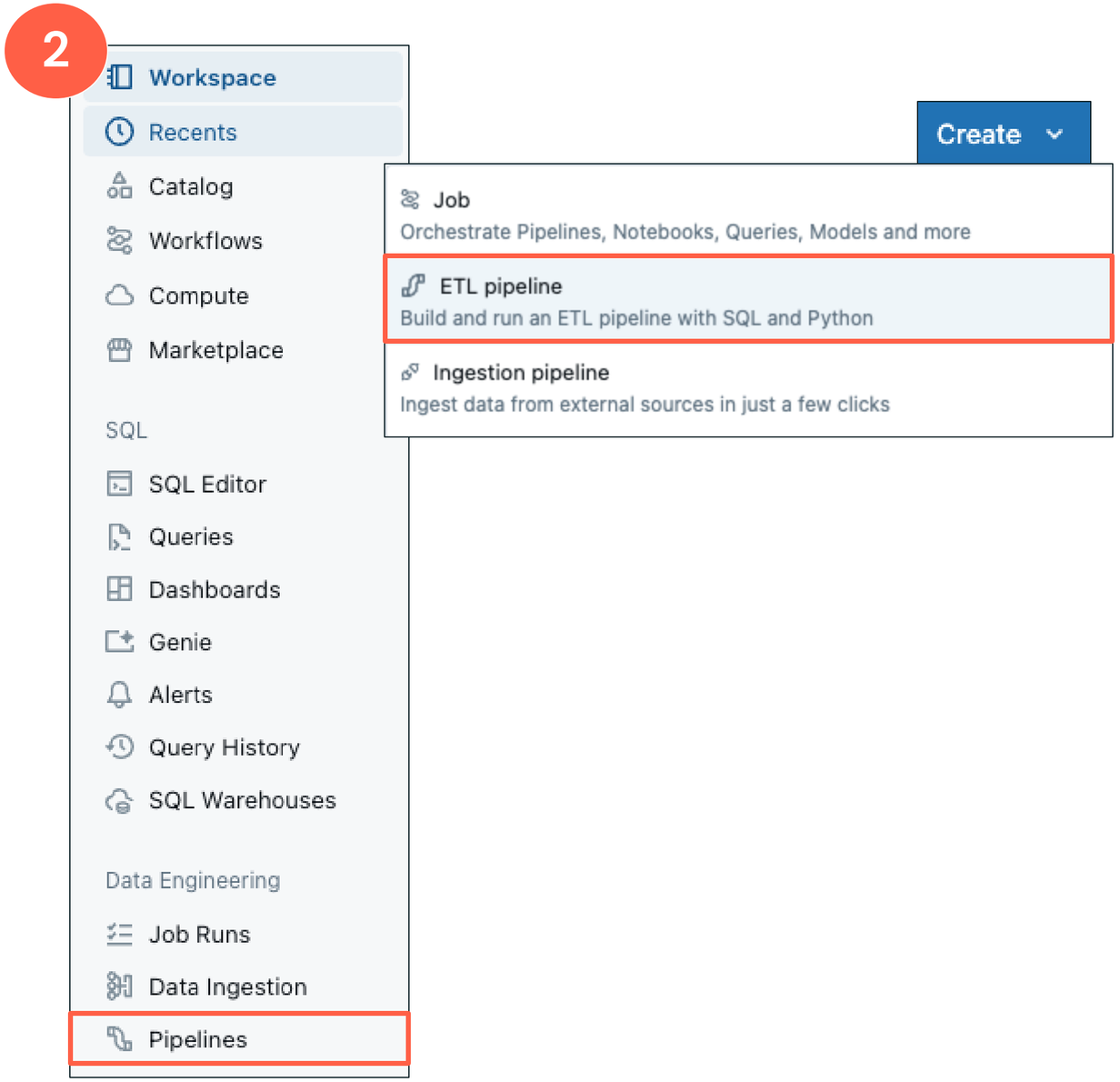
What are Lakeflow Declarative Pipelines?

Overview of Incremental Processing in Declarative Pipelines (Run 2)



Creating a Pipeline



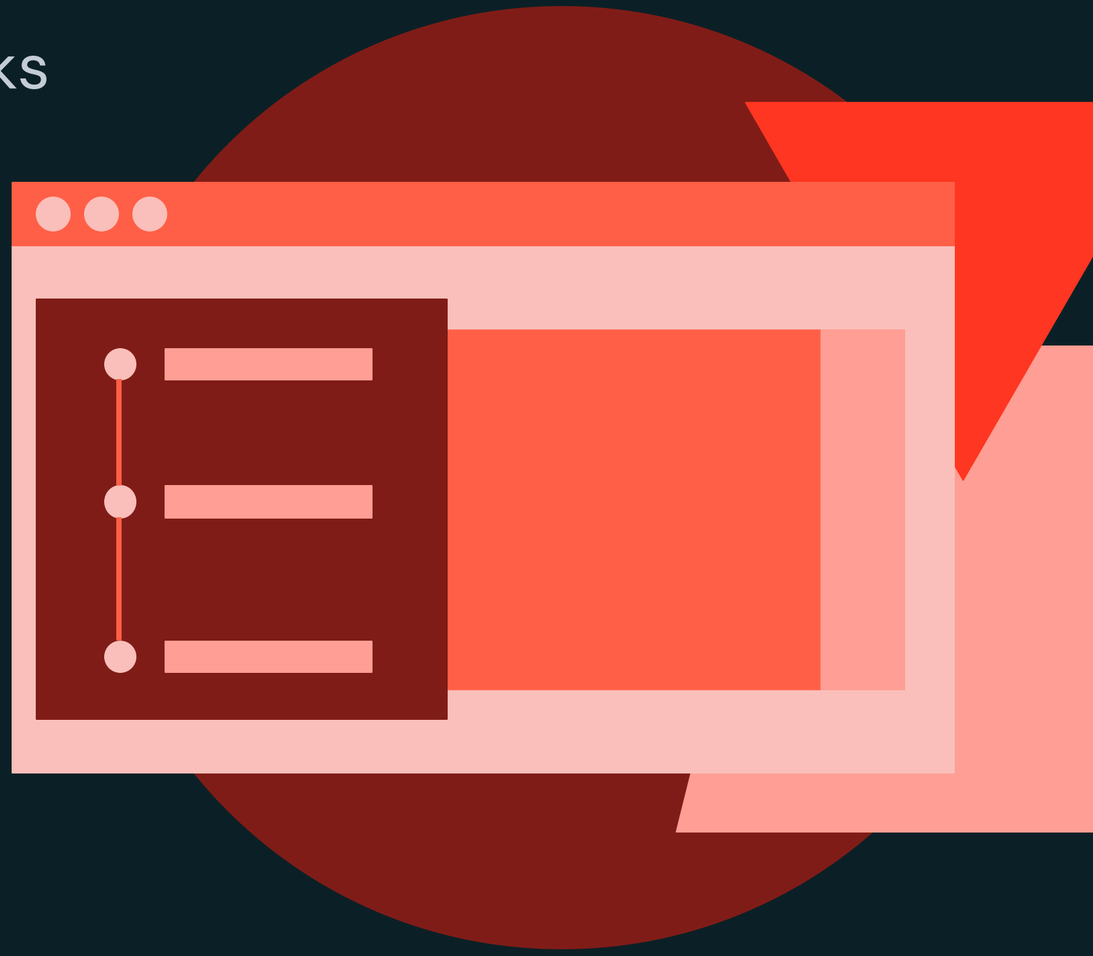


databricks

Introduction to Data Engineering in Databricks

DEMONSTRATION

REQUIRED – Course Setup and Creating a Pipeline



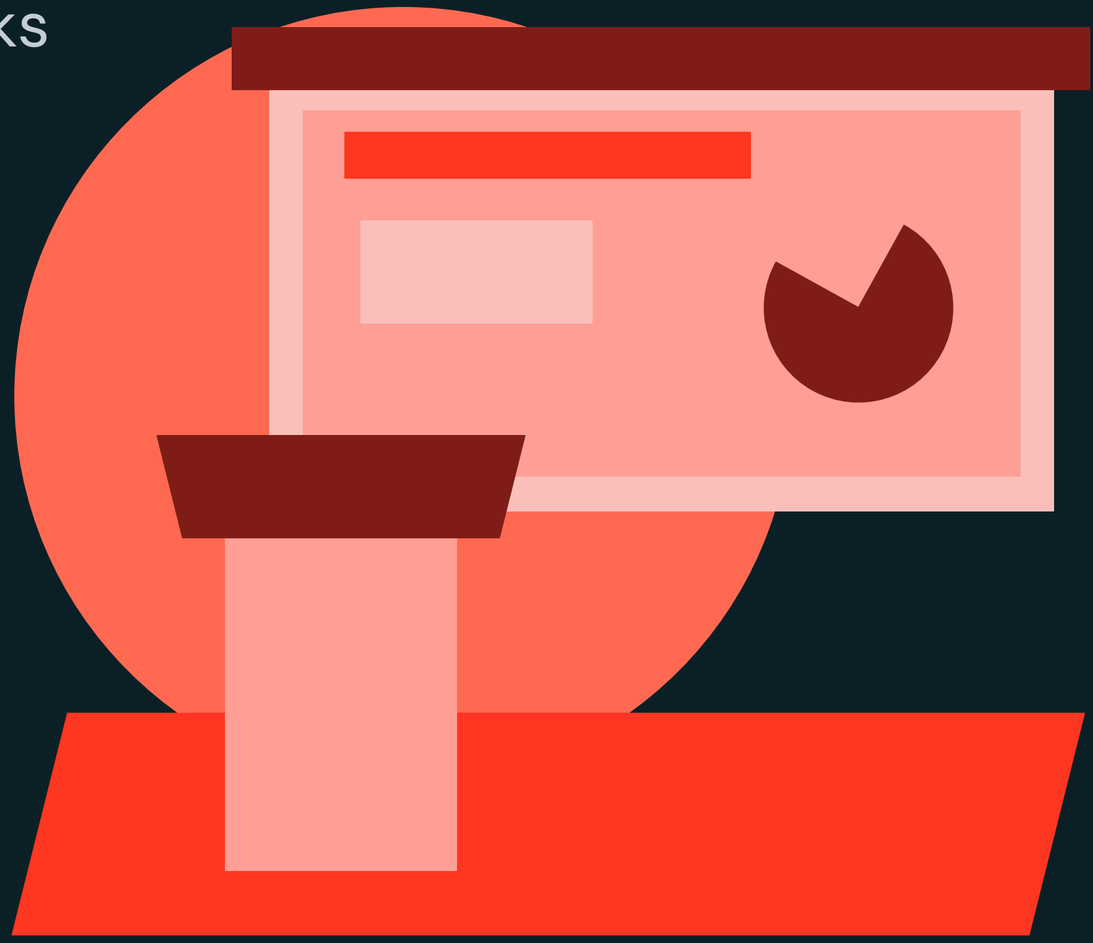
Notebook: 1 - REQUIRED - Course Setup and Creating a Pipeline

databricks

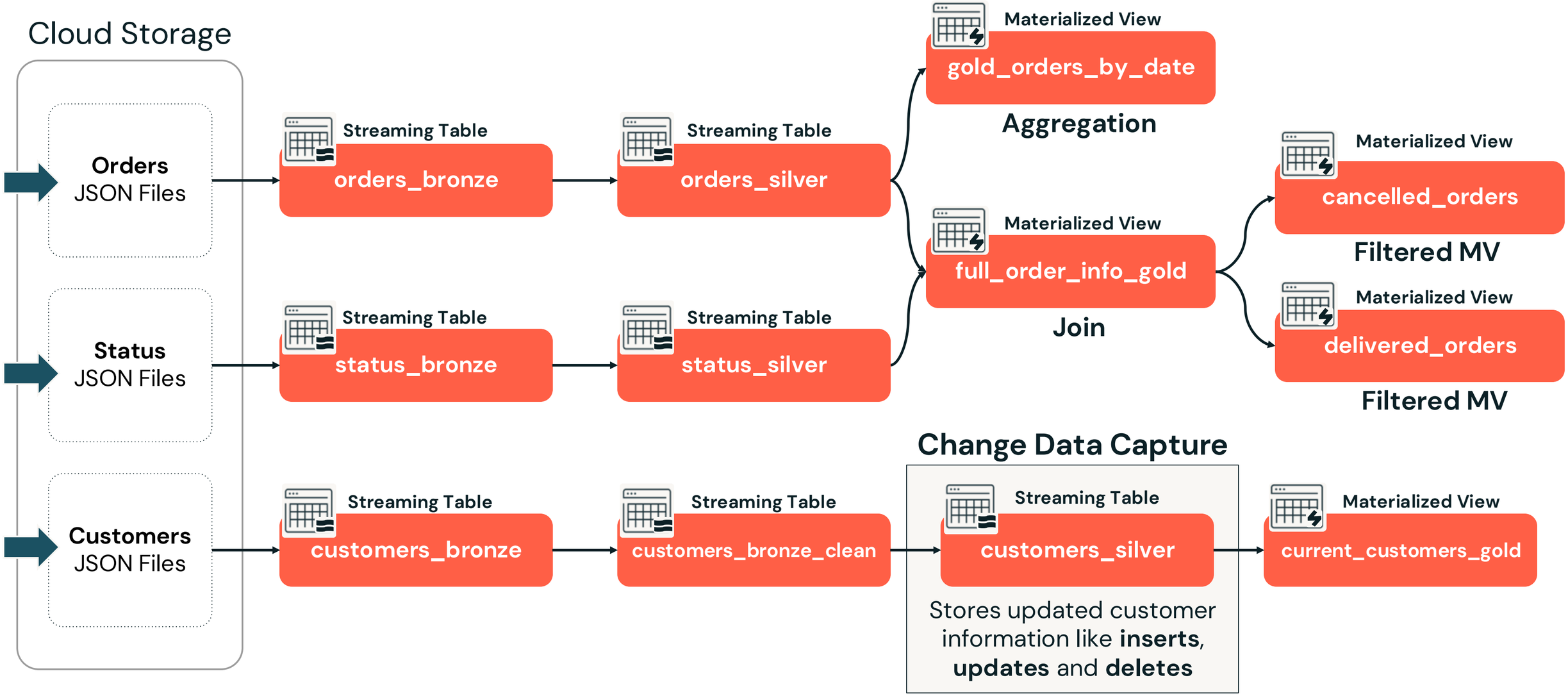
Introduction to Data Engineering in Databricks

LECTURE

Course Project Overview



Course Project Overview



databricks

Lakeflow Declarative Pipeline Fundamentals

Build Data Pipelines with Lakeflow Declarative Pipelines

Section Learning Objectives

- Understand the core concepts and components of Lakeflow Declarative Pipelines pipelines, including how streaming tables, materialized views, and temporary views function and differ.
- Identify and configure pipeline settings such as compute, data assets, trigger modes, and advanced options.
- Develop a functional Declarative Pipeline using the new pipeline editor and SQL-based syntax for the pipeline.
- Incorporate data quality expectations into a Declarative Pipeline pipeline to validate and enforce data integrity.

Agenda

Section Overview – Pipeline Fundamentals
• Dataset Types Overview
• Simplified Pipeline Development
• Common Pipeline Settings
• Developing a Simple Pipeline
• Ensure Data Quality with Expectations

databricks

Lakeflow Declarative Pipeline Fundamentals

LECTURE

Dataset Types Overview

Dataset Types Overview

How streaming tables, materialized views, and views process data

STREAMING TABLE (ST)

A table with support for streaming or incremental data processing, **only processing new data**

MATERIALIZED VIEW (MV)

VIEWS

Dataset Types Overview

STREAMING TABLE Overview

STREAMING TABLE

- Support for **batch** or **streaming** for **incremental** data processing (exactly **once processing** of the files)
- Each time an streaming table is refreshed, data added to the source tables is **appended** to the **streaming table**
- SQL code that creates streaming tables:
  `CREATE OR REFRESH STREAMING TABLE`
- In SQL, you must use `FROM STREAM read_files()` to **invoke Auto Loader functionality** for incremental streaming reads and checkpoints

Dataset Types Overview

Create a STREAMING TABLE with SQL

Create a STREAMING TABLE named **orders_bronze** from **JSON files**

```
CREATE OR REFRESH STREAMING TABLE 1_bronze_db.orders_bronze AS
SELECT
  *,
  current_timestamp() AS processing_time,
  _metadata.file_name AS source_file
FROM STREAM read_files(
    "/Volumes/dbacademy/ops/labuser/orders",
    format => 'JSON');
```

Note: The query is using the default catalog

Creates a streaming table named **orders_bronze** in the **1_bronze_db** schema

Use the SELECT clause to specify the columns to read into the streaming table from your data source

- The **STREAM** keyword indicates to use streaming read for the source files (JSON)
- The **read_files()** function reads files under a provided location and returns the data in tabular form

Dataset Types Overview

Create a STREAMING TABLE with SQL

Create a STREAMING TABLE named **orders_silver** that reads from the **STREAMING orders_bronze** table

```
CREATE OR REFRESH STREAMING TABLE 2_silver_db.orders_silver AS
SELECT
    order_id,
  timestamp(order_timestamp) AS order_timestamp,
  customer_id,
  notifications
FROM STREAM 1_bronze_db.orders_bronze;
```

Specify your SQL transformations for the streaming data from the bronze table

Use **STREAM** keyword with the name of the STREAMING TABLE you are reading from to transform new rows

Note: The query is using the default catalog

Dataset Types Overview

How streaming tables, materialized views, and views process data

STREAMING TABLE (ST)

A table with support for streaming or incremental data processing, **only processing new data**

MATERIALIZED VIEW (MV)

- Records are processed as required to return accurate results for the **current data state**
- Used for data processing tasks such as:
  - Transformations
  - Aggregations
  - pre-computing slow queries
  - frequently used computations

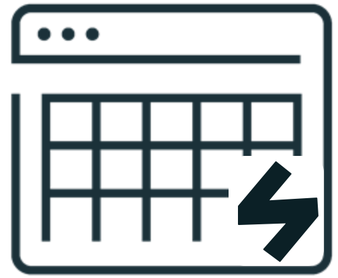
VIEWS

Dataset Types Overview

MATERIALIZED VIEW Overview

MATERIALIZED VIEW

- Each time a materialized view is updated, **query results are recalculated to reflect changes in upstream datasets**
- Created and **kept up-to-date** by pipeline
- Use the `CREATE OR REFRESH MATERIALIZED VIEW` syntax
- Can be used **anywhere in your pipeline**, not just in the gold layer
- **Where applicable,** results are **incrementally refreshed for materialized views**, avoiding the need to completely rebuild the materialized view when new data arrives **(Serverless Only)**
- Incremental refresh for materialized views is a cost-based optimizer to power fast and efficient transformations for materialized views on Serverless compute



Dataset Types Overview

Create a MATERIALIZED VIEW with SQL

Create a MATERIALIZED VIEW named **gold_orders_by_date** that summarizes the **orders_silver** table

```
CREATE OR REFRESH MATERIALIZED VIEW 3_gold_db.gold_orders_by_date AS
SELECT
 date(order_timestamp) AS order_date,
 count(*) AS total_daily_orders
FROM 2_silver_db.orders_silver
GROUP BY date(order_timestamp);
```

Create the materialized view **gold_orders_by_date** in the **3_gold_db** schema (database)

Specify the the source table **without the STREAM** keyword to utilize the source STREAMING table

NOTE: Where possible, queries will be incrementally refreshed

Note: The query is using the default catalog

Dataset Types Overview

How streaming tables, materialized views, and views process data

STREAMING TABLE (ST)

A table with support for streaming or incremental data processing, **only processing new data**

MATERIALIZED VIEW (MV)

- Records are processed as required to return accurate results for the **current data state**
- Used for data processing tasks such:
  - Transformations
  - Aggregations
  - pre-computing slow queries
  - frequently used computations.

VIEWS

Constructs a virtual table with no physical data based on the query in your Declarative Pipelines

- TEMPORARY VIEW
- VIEW

Dataset Types Overview

TEMPORARY VIEW Overview

TEMPORARY VIEW

- Temporary views are only **persisted across the lifetime** of the pipeline and **private to the defining pipeline**
- They are **not registered** as an object to Unity Catalog
- Use the `CREATE TEMPORARY VIEW` statement
- TEMPORARY VIEWS are useful as **intermediate queries** that **are not exposed** to end users

Dataset Types Overview

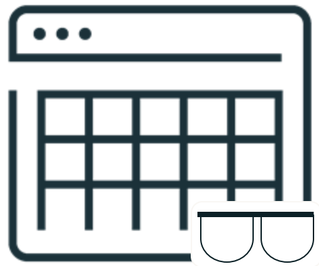
VIEW Overview

VIEW

- Constructs a virtual table with no physical data based on the result-set of a SQL query in your pipelines
- They are **registered** as an object to Unity Catalog
- Use the `CREATE VIEW` statement

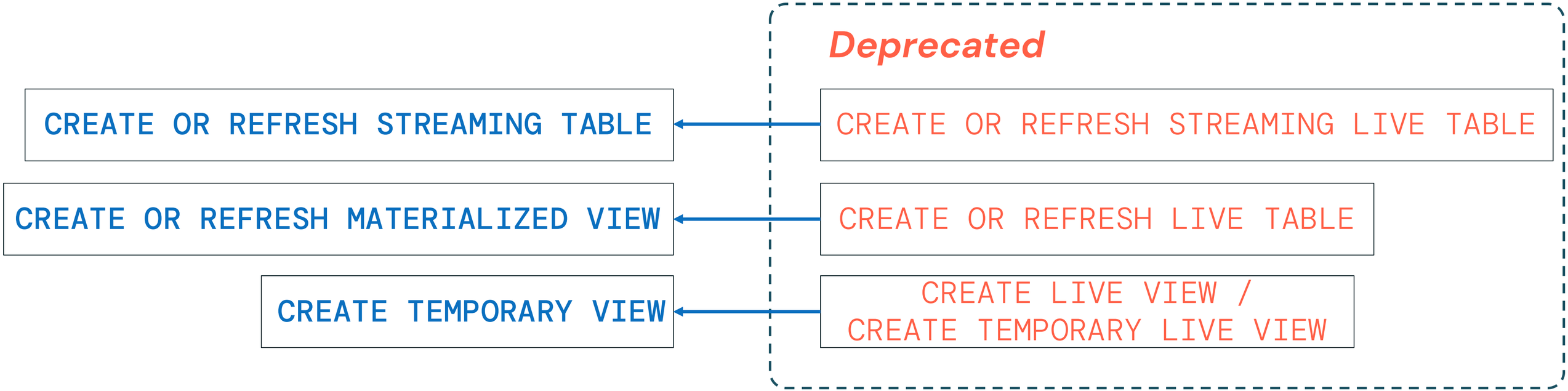
LIMITATIONS

- The pipeline must be a Unity Catalog pipeline
- Views cannot have streaming queries, or be used as a streaming source for a pipeline



Dataset Types Overview

Existing users of DLT will notice that we've evolved the names



The **semantics remain the same**, and **we'll support the old syntax** for compatibility.
Our goal is to simplify the syntax and match other systems

Dataset Types Overview

Pipeline dependencies are parsed automatically, code order is not required

Raw JSON -> Bronze

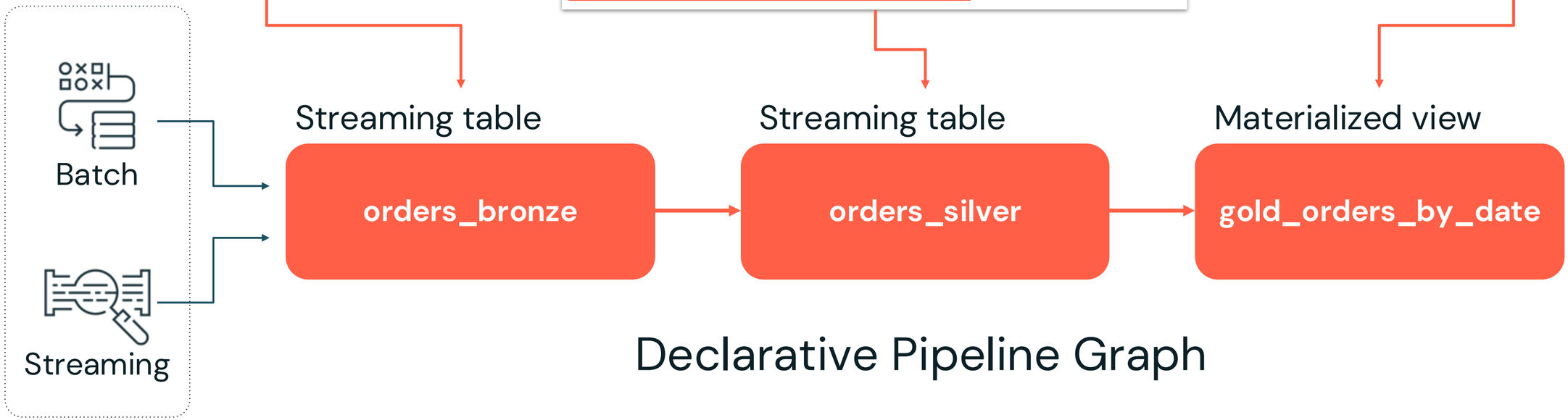
```
CREATE OR REFRESH STREAMING TABLE
1_bronze_db.orders_bronze AS
SELECT
 *,
 current_timestamp() AS processing_time,
 _metadata.file_name AS source_file
FROM STREAM read_files(
   "/Volumes/dbacademy/ops/labuser/orders",
   format => 'JSON');
```

Bronze -> Silver

```
CREATE OR REFRESH STREAMING TABLE 2_silver_db.orders_silver
AS
SELECT
  order_id,
  timestamp(order_timestamp) AS order_timestamp,
  customer_id,
  notifications
FROM STREAM 1_bronze_db.orders_bronze;
```

Silver -> Materialized View

```
CREATE OR REFRESH MATERIALIZED VIEW 3_gold_db.gold_orders_by_date
AS
SELECT
 date(order_timestamp) AS order_date,
 count(*) AS total_daily_orders
FROM 2_silver_db.orders_silver
GROUP BY date(order_timestamp);
```



Declarative Pipeline Graph

databricks

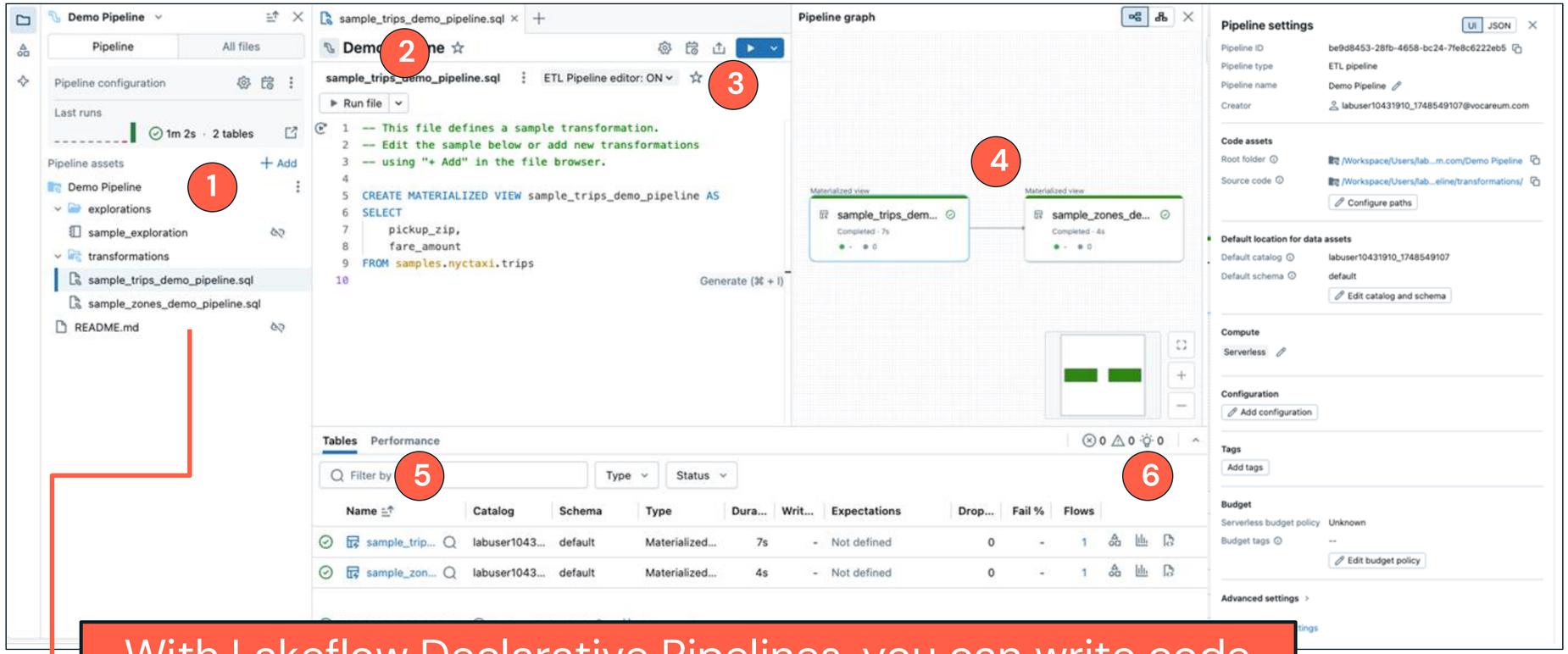
Lakeflow Declarative Pipeline Fundamentals

LECTURE

Simplified Pipeline Development

Simplified Pipeline Development

Multi-file editor in Lakeflow Declarative Pipelines



With Lakeflow Declarative Pipelines, you can write code using **.sql or .py files (recommended)**, or use notebooks.

What's New

1. Pipeline asset browser
2. Multi-file code editor with features for step-by-step development
3. Pipeline-specific toolbars
4. Interactive DAG
5. Data previews
6. Execution insights panels
   - Easier debugging
   - Faster validation with dry run

As of Data & AI Summit 2025

databricks

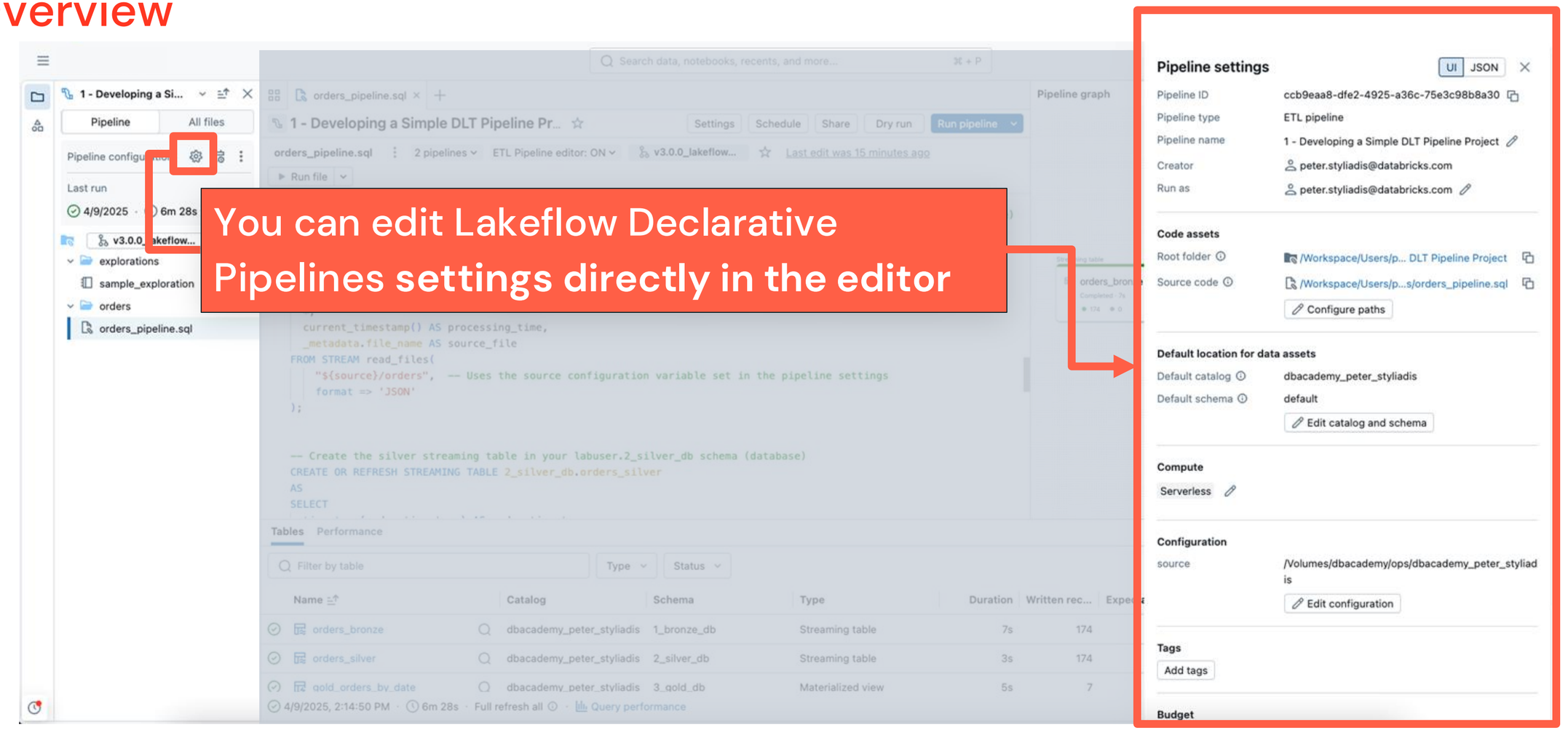
Lakeflow Declarative Pipeline Fundamentals

LECTURE

Common Pipeline Settings

Common Pipeline Settings

Overview



Common Pipeline Settings

Compute

Code Assets

Configuration
(parameters)

Common Pipeline Settings

Compute

Code Assets

Configuration
(parameters)

SERVERLESS

- Databricks **recommends** developing new pipelines using serverless
- **Optimizes** cost while **maintaining** performance
- **Enables** you to focus on code rather than managing infrastructure
- Optional **Serverless Performance optimized** setting for time-sensitive workloads

- Serverless compute includes **Incremental refresh for materialized views**
- A cost-based optimizer to power fast and efficient transformations for materialized views on Serverless compute

Common Pipeline Settings

Compute

Code Assets

Configuration
(parameters)

SERVERLESS

- Databricks **recommends** developing new pipelines using serverless
- **Optimizes** cost while **maintaining** performance
- **Enables** you to focus on code rather than managing infrastructure
- Optional **Serverless Performance optimized** setting for time-sensitive workloads

CLASSIC (fixed size)

- Users need **permission** to create compute for Declarative Pipelines
- Workspace admins **can configure cluster policies** to provide users with access to compute resources for Declarative Pipelines

AUTOSCALING

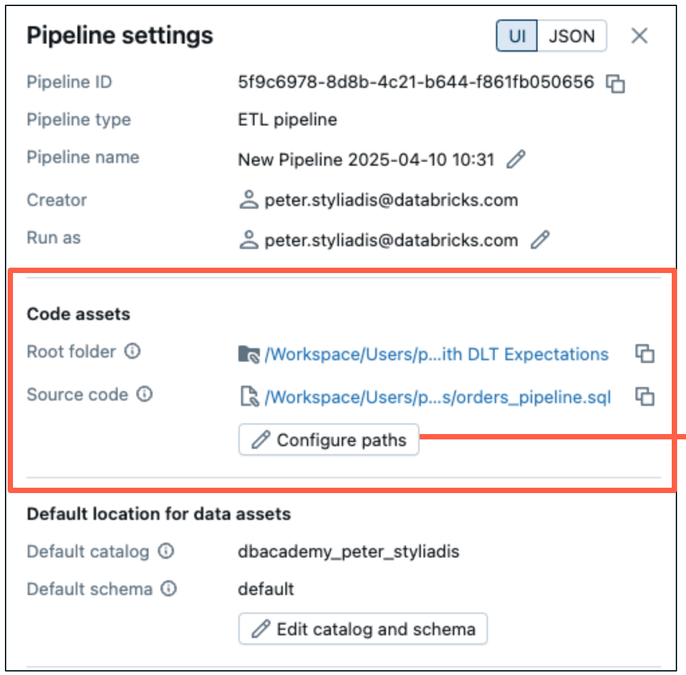
- Enhanced autoscaling **is enabled by default** for all new pipelines.
- **Optimizes cluster utilization** by automatically allocating cluster resources based on workload volume

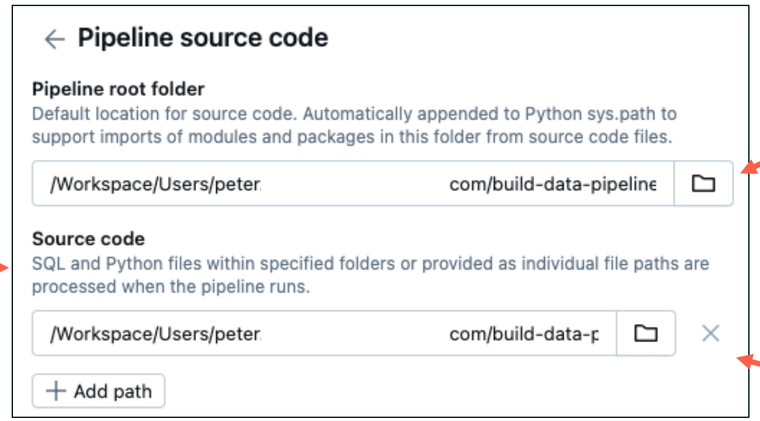
Common Pipeline Settings

Compute

Code Assets

Configuration
(parameters)





The **Pipeline root folder** automatically includes all relevant files within the folder for the pipeline project (can be a Git folder)

The **source code** section references include subfolders (or files) for the pipeline **(.py, .sql, notebooks)**

Common Pipeline Settings

Compute

Code Assets

Configuration
(parameters)

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

- Improve code readability/maintainability
- Reuse common parameters in multiple pipelines files


```
CREATE STREAMING TABLE bronze
SELECT *
FROM STREAM read_files(
   "${source}/orders",
   format => 'JSON');
```

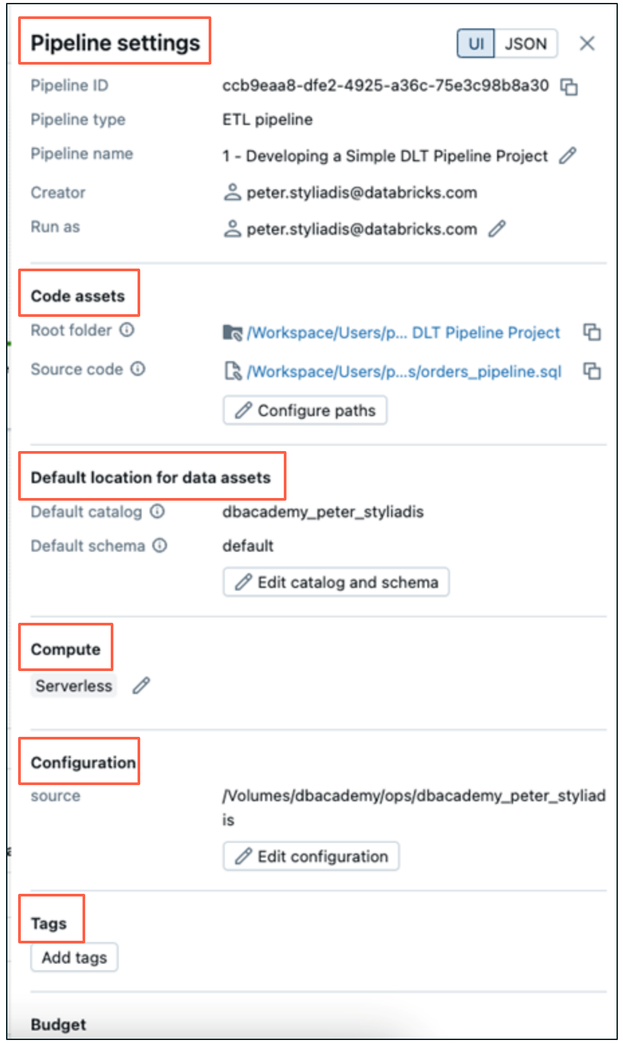
In SQL reference the key's value using the **${key-name}** syntax

Common Pipeline Settings

Common Settings

There are **various other settings** we will explore throughout the course, including:

- Pipeline settings
- Code assets
- Default location for data assets
- Compute
- Configuration
- Environment
- Tags
- Budget
- Advanced settings

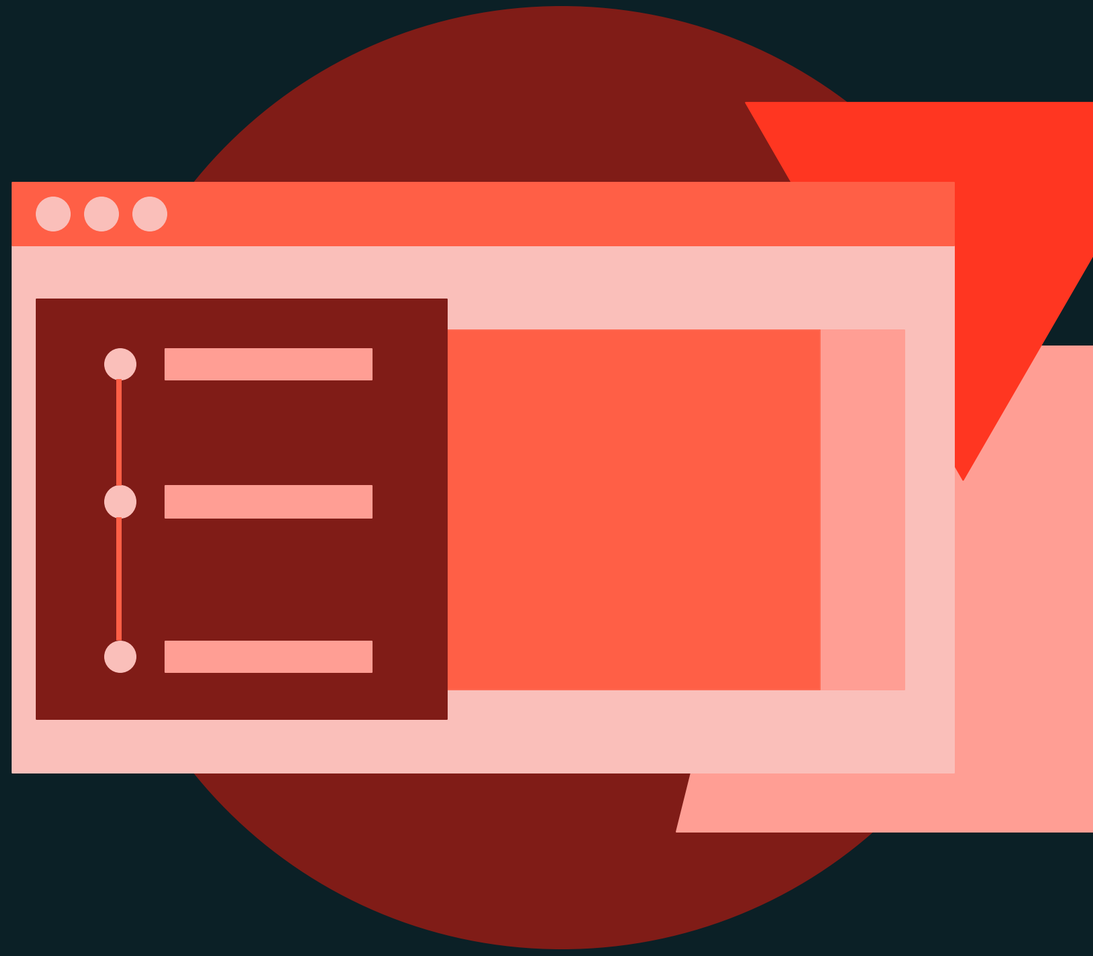


databricks

Lakeflow Declarative Pipeline Fundamentals

DEMONSTRATION

Developing a Simple Pipeline



Notebook: 2 - Developing a Simple Pipeline

databricks

Lakeflow Declarative Pipeline Fundamentals

LECTURE

Ensure Data Quality with Expectations

Ensure Data Quality with Expectations

Creating Expectations

With Lakeflow Declarative Pipelines you can apply **data quality rules** to validate rows during ETL processes to gain insights into data integrity using the following

```
CONSTRAINT constraint-name EXPECT (column condition) [ON VIOLATION action]
```

Action	SQL Syntax Example	Result
WARN (default)	`CONSTRAINT valid_notification EXPECT (notifications IN ('Y','N'))`	• Invalid rows **are still written** to the target • Logs include **counts of valid vs. invalid records** and other metrics
DROP	`CONSTRAINT valid_date EXPECT (order_timestamp > "2021-01-01")` `ON VIOLATION DROP ROW`	• Invalid rows **dropped** from table • The count is logged alongside other metrics
FAIL	`CONSTRAINT valid_id EXPECT (customer_id IS NOT NULL)` `ON VIOLATION FAIL UPDATE`	• Causes a **failure of a single flow** and does not cause other flows in your pipeline to fail. • **Manual** intervention is required

Ensure Data Quality with Expectations

Adding Data Quality Expectations Example

```
CREATE OR REFRESH STREAMING TABLE 2_silver_db.orders_silver
 (
   CONSTRAINT valid_notifications EXPECT (notifications IN ('Y','N')),
   CONSTRAINT valid_id EXPECT (customer_id IS NOT NULL) ON VIOLATION DROP ROW,
   CONSTRAINT valid_date EXPECT (order_timestamp > "2021-01-01") ON VIOLATION FAIL UPDATE
 )
AS
SELECT
      Order_id,
   timestamp(order_timestamp) AS order_timestamp,
   Customer_id,
   notifications
FROM STREAM 1_bronze_db.orders_bronze;
```

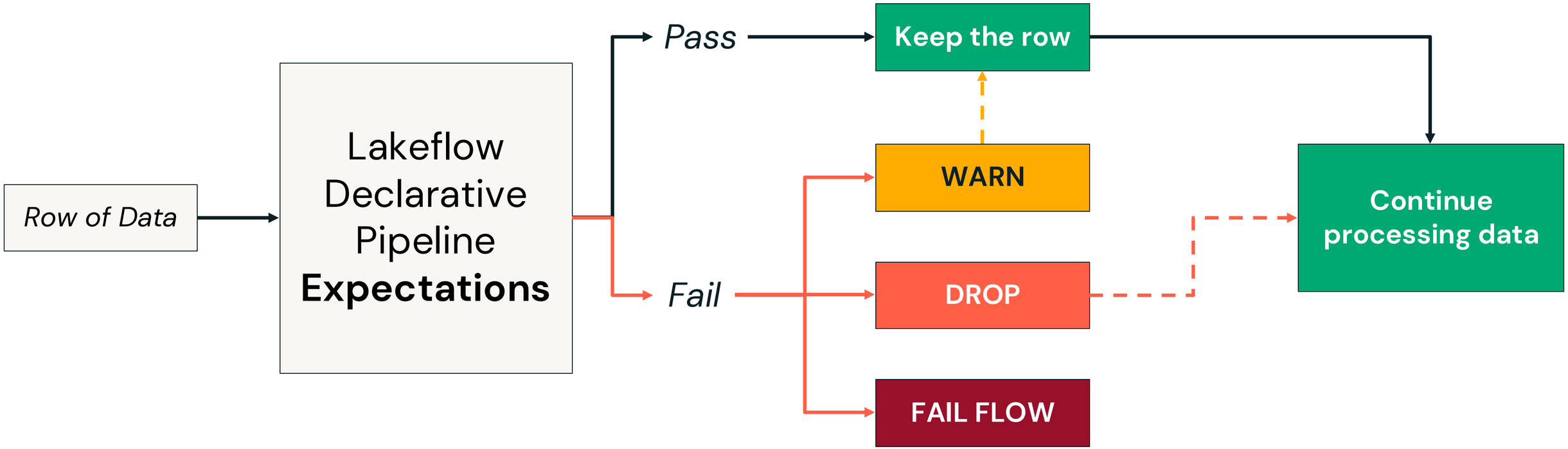
WARN in metrics

DROP the row

FAIL THE PIPELINE

Ensure Data Quality with Expectations

Actions Overview



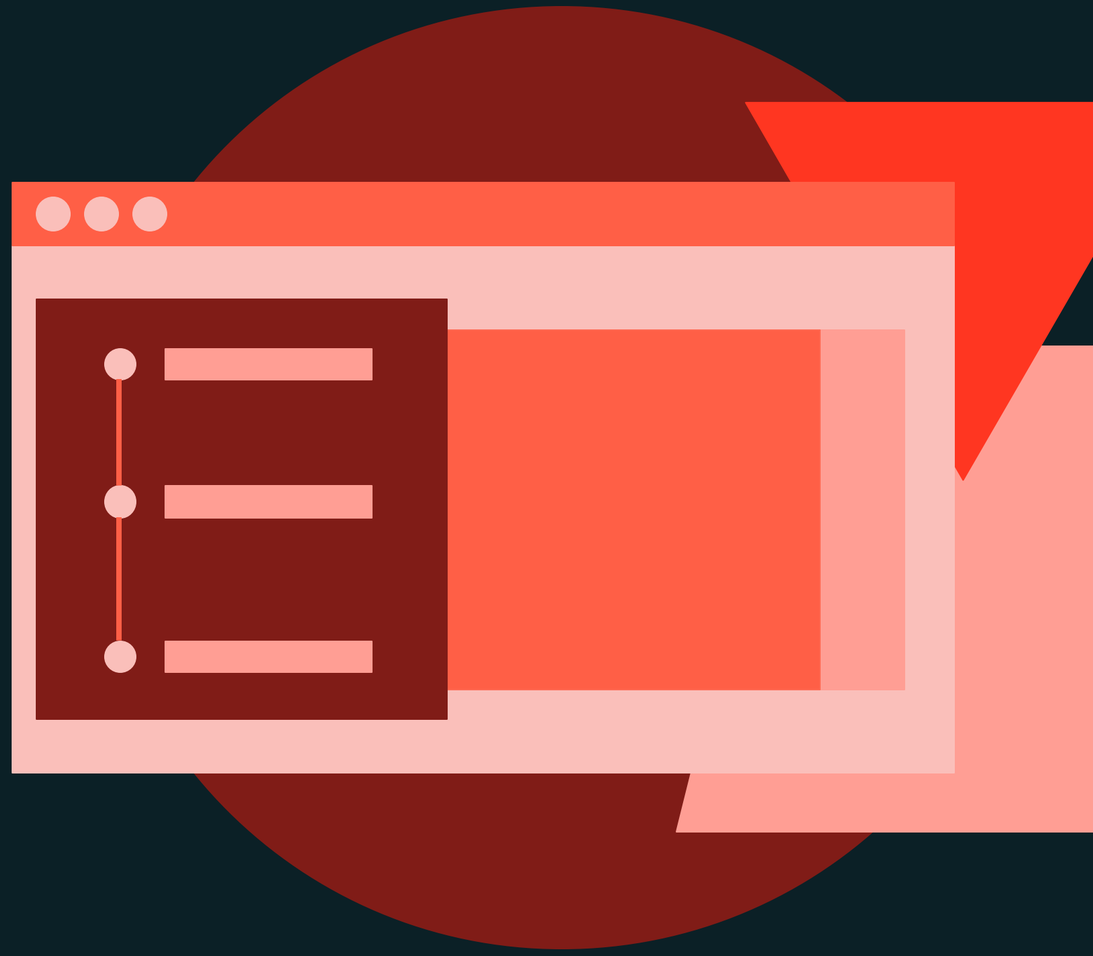
As of 2025Q2 materialized views that use **expectations** are **always fully refreshed**

databricks

Lakeflow Declarative Pipeline Fundamentals

DEMONSTRATION

Adding Data Quality Expectations



Notebook: 3 - Adding Data Quality Expectations

databricks

Lakeflow Declarative Pipeline Fundamentals

LAB EXERCISE

Create a Pipeline

Notebook: 4 Lab - Create a Pipeline

databricks

Building Lakeflow Declarative Pipelines

Build Data Pipelines with Lakeflow Declarative Pipelines

Section Learning Objectives

- Deploy a Lakeflow Declarative Pipeline in production by modifying configuration options like mode, schedule, email notifications and more.
- Analyze event logs and pipeline metrics to examine the entirety of a pipeline.
- Design and implement a Change Data Capture (CDC) Declarative Pipeline using APPLY CHANGES INTO to handle slowly changing dimensions (SCD).

Agenda

Section Overview – Building Pipelines
• Streaming Joins Overview
• Deploying a Pipeline to Production
• Change Data Capture (CDC) Overview
• Change Data Capture with Apply CHANGE INTO
• Additional Features Overview

databricks

Building Lakeflow Declarative Pipelines

LECTURE

Streaming Joins Overview

Streaming Joins Overview

Stream-Snapshot Join Overview

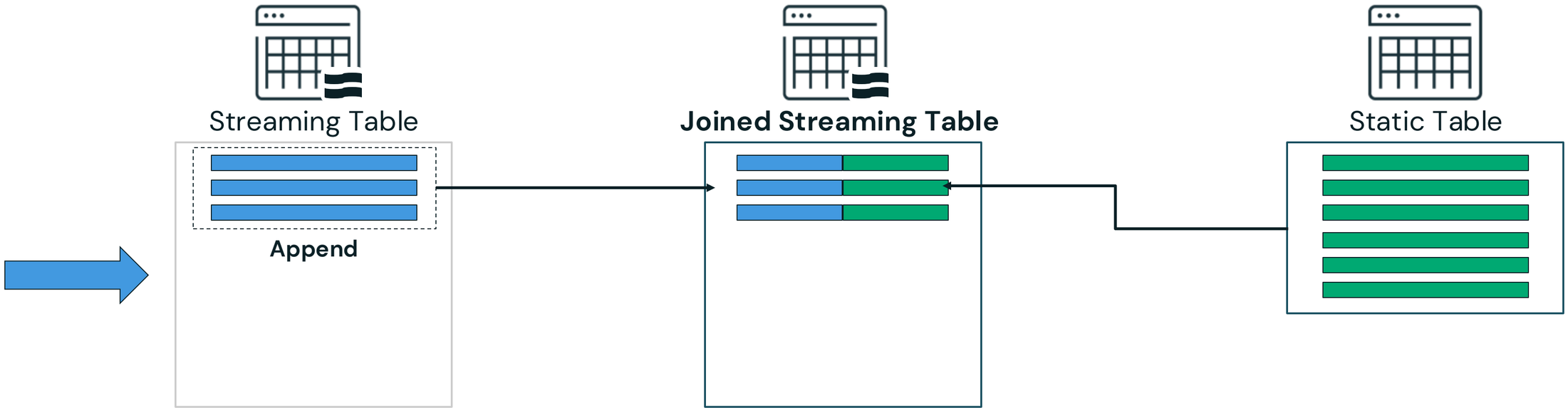
Streaming Table to **Static Table**

The goal is to **incrementally** join new data with a **static table**

Streaming Joins Overview

Stream-Snapshot Join Overview

Streaming Table to **Static Table**

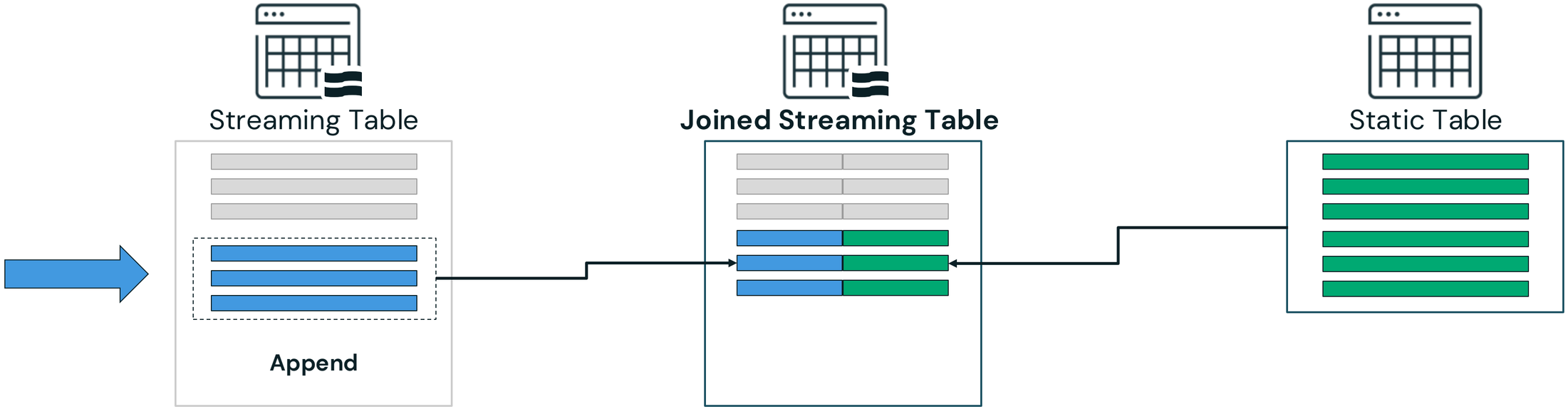


The **new data** in the streaming table JOINS using all of the **data in the lookup table**

Streaming Joins Overview

Stream-Snapshot Join Overview

Streaming Table to **Static Table**



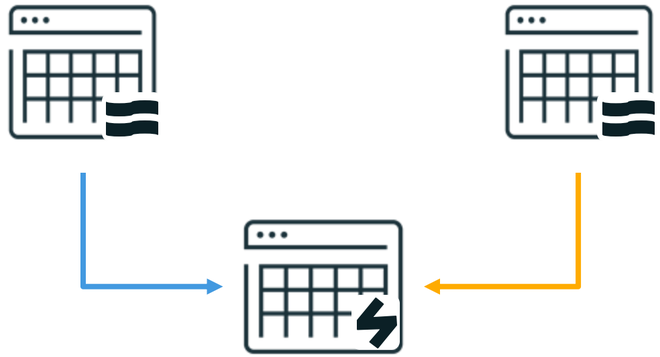
Only the new data in the streaming table JOINS using all of the **data in the lookup table**

Streaming Joins Overview

Joining Streaming Tables with a Materialized View

Streaming Table to Static Table

Streaming Table to **Streaming Table** with a **Materialized View**



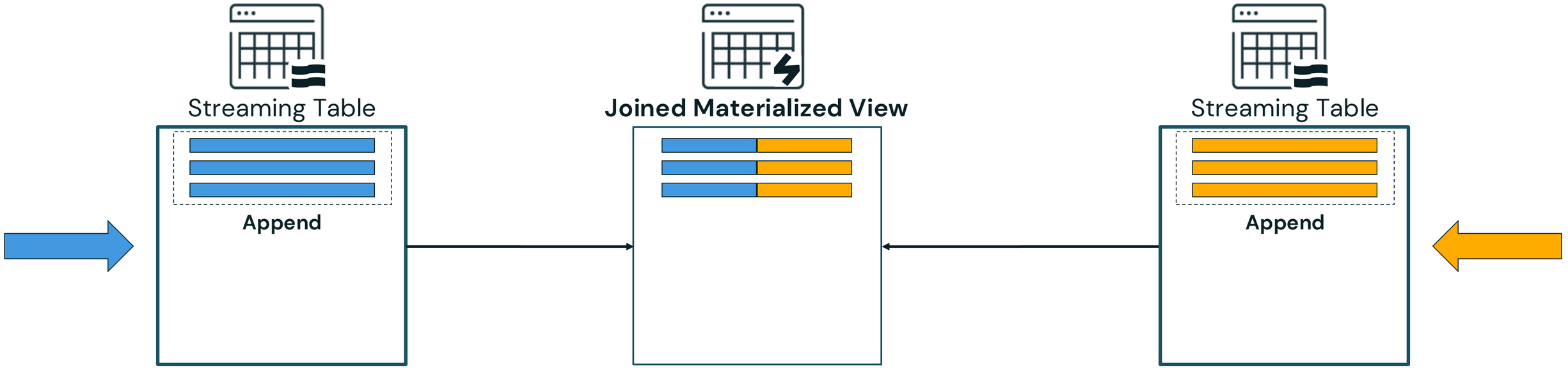
The goal is to take **all rows from two streaming tables** and join them together each time the pipeline is run

Streaming Joins Overview

Joining Streaming Tables with a Materialized View

Streaming Table to Static Table

Streaming Table to **Streaming Table** with a **Materialized View**



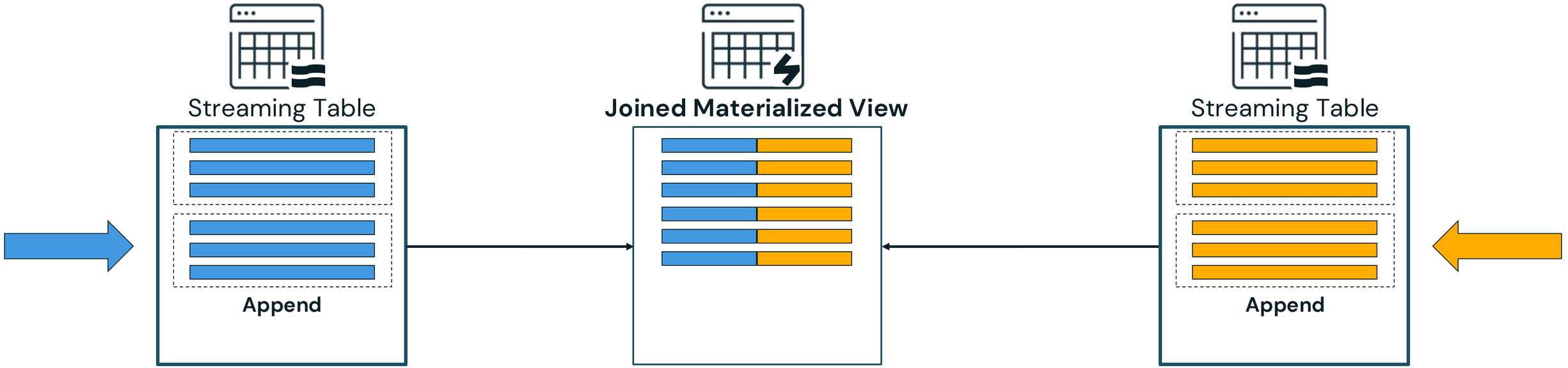
The materialized view **efficiently computes all the data** in the streaming tables and joins the rows.

Streaming Joins Overview

Joining Streaming Tables with a Materialized View

Streaming Table to Static Table

Streaming Table to **Streaming Table** with a **Materialized View**



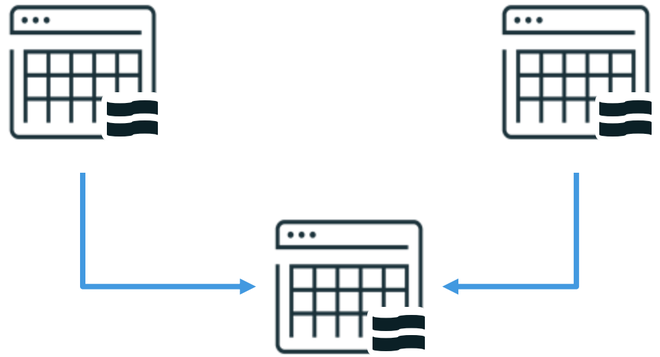
As new data is added to the streaming tables, the materialized view will again **efficiently compute all the data** in the streaming tables and joins the rows

Streaming Joins Overview

Stream-Stream Joins

Outside the scope of the course

Streaming Table to Static Table	Streaming Table to Streaming Table with a Materialized View	**Incremental** Stream to Stream Joins



The goal is to **incrementally** join new data from two tables, **past data is not used**

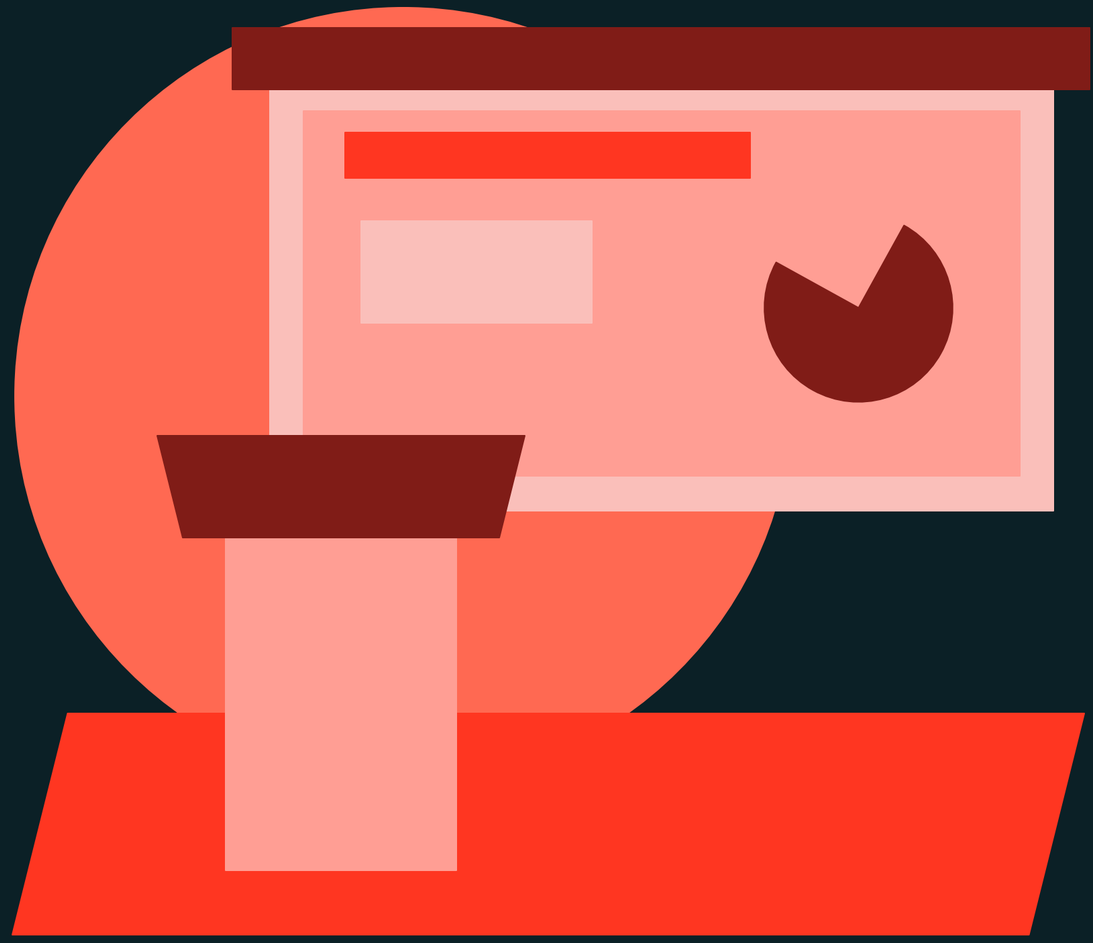
Databricks Streaming and DLT (Advanced) Course

databricks

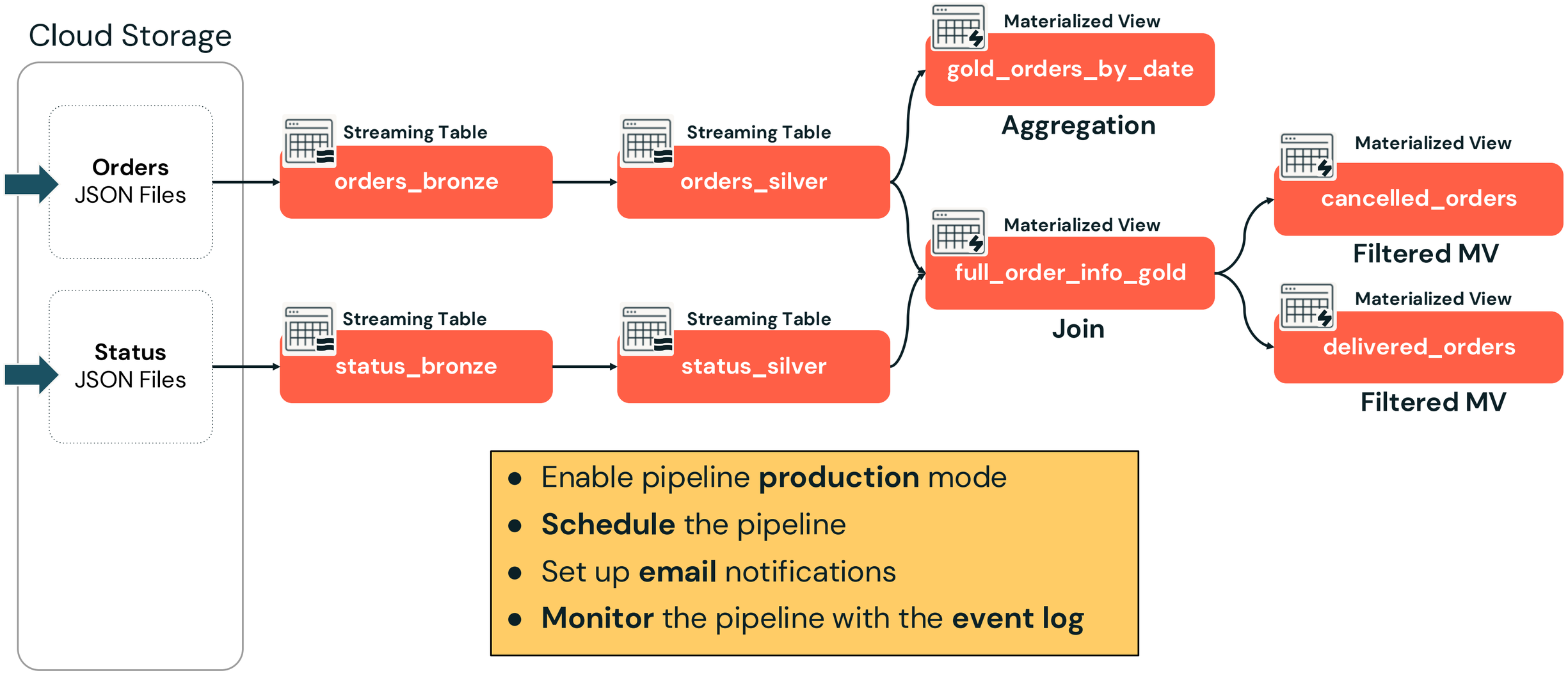
Building Lakeflow Declarative Pipelines

LECTURE

Deploying a Pipeline to Production



Deploying a Pipeline to Production



- Enable pipeline **production** mode
- **Schedule** the pipeline
- Set up **email** notifications
- **Monitor** the pipeline with the **event log**

Deploying a Pipeline to Production

Common Production Tasks

Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

Deploying a Pipeline to Production

Common Production Tasks

Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

DEVELOPMENT Mode

- **Cluster Reuse** - Avoids overhead of restarts
- **Disables Pipeline Retries** - Immediate error detection and fixing
- **Optimized for Debugging** - Quick feedback loop for development

PRODUCTION Mode

Deploying a Pipeline to Production

Common Production Tasks

Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

DEVELOPMENT Mode

- **Cluster Reuse** - Avoids overhead of restarts
- **Disables Pipeline Retries** - Immediate error detection and fixing.
- **Optimized for Debugging** - Quick feedback loop for development.

PRODUCTION Mode

- **Cluster Restart** - Restarts for recoverable errors, including memory leaks and stale credentials.
- **Pipeline Retries** - Retries execution on specific errors (cluster startup failure).
- **Optimized for Reliability** - Ensures robustness and fault tolerance in production.

Deploying a Pipeline to Production

Common Production Tasks

Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

Triggered

- Can be triggered **manually** or run on a **schedule**
- **Refreshes** selected tables using data available at the start, then stops once the update is complete

Continuous

- **Continuously** processes new data to keep streaming tables and materialized views up to date in **near real time**
- Monitors dependencies and **updates only** when source data changes

Deploying a Pipeline to Production

Common Production Tasks

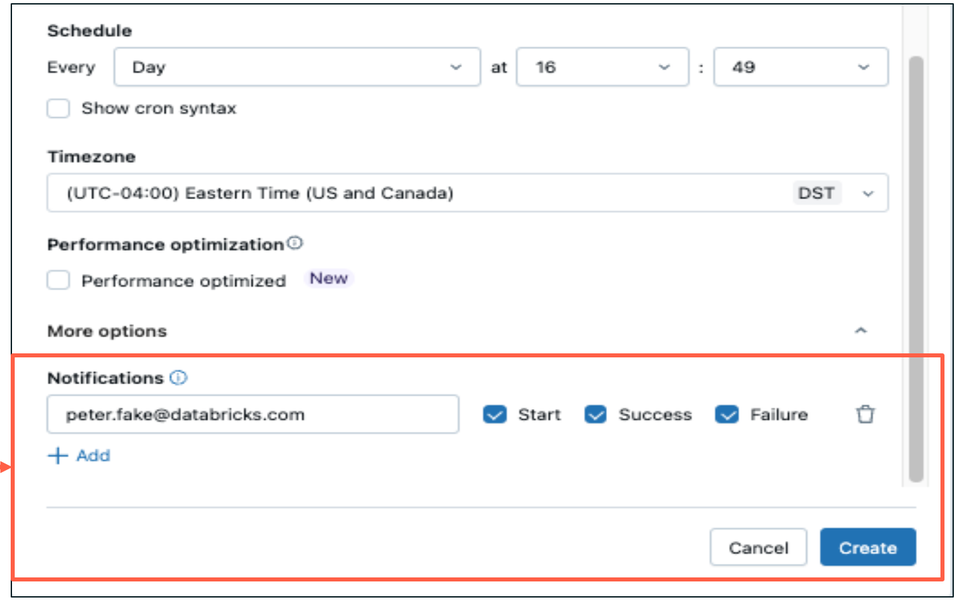
Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

You can configure **email notifications** when **scheduling** the pipeline



Deploying a Pipeline to Production

Common Production Tasks

Pipeline Modes

Scheduling

Email Notifications

Monitor pipelines

- The **pipeline event log** captures all key information about the pipeline:
  - Audit logs
  - Data Quality Checks
  - Pipeline Progress
  - Data Lineage

Deploying a Pipeline to Production

Querying the Declarative Pipeline Event Log

- **Publish the event log** as a **Delta table** using the **advanced settings**
  - Specify the table **location** (catalog, schema)
  - Define the **table** name

Query the pipeline event log table

```
SELECT * FROM <catalog>.<schema>.<event_log_table_name>
```

Deploying a Pipeline to Production

Querying the Declarative Pipeline Event Log

- **Publish the event log** as a **Delta table** using the **advanced settings**
  - Specify the table **location** (catalog, schema)
  - Define the **table** name

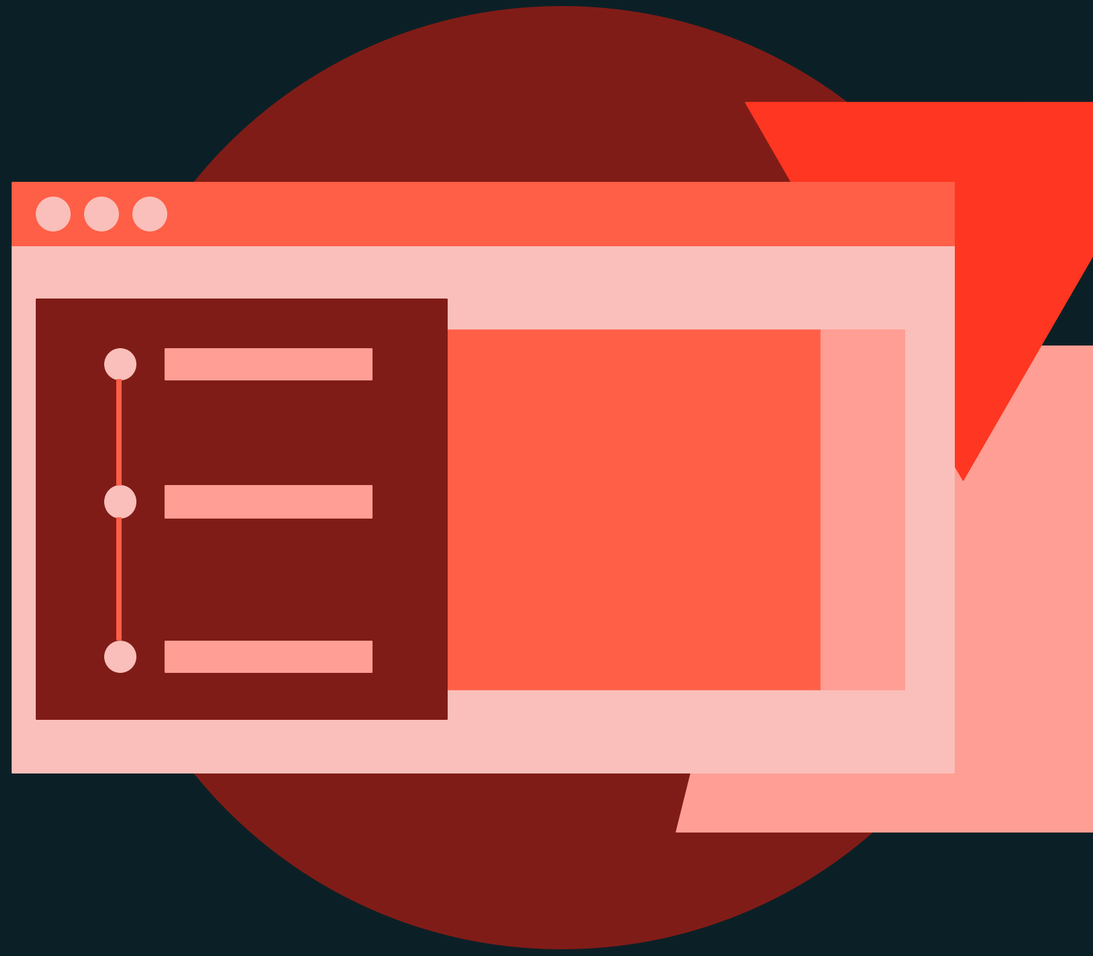
- **(DEFAULT)** The Event Log is written as a **hidden Delta table**
  - Located in the pipeline **default catalog** and **schema**
  - View the query the event log documentation to access the hidden event log

databricks

Building Lakeflow Declarative Pipelines

DEMONSTRATION

Deploying Your Pipeline to Production



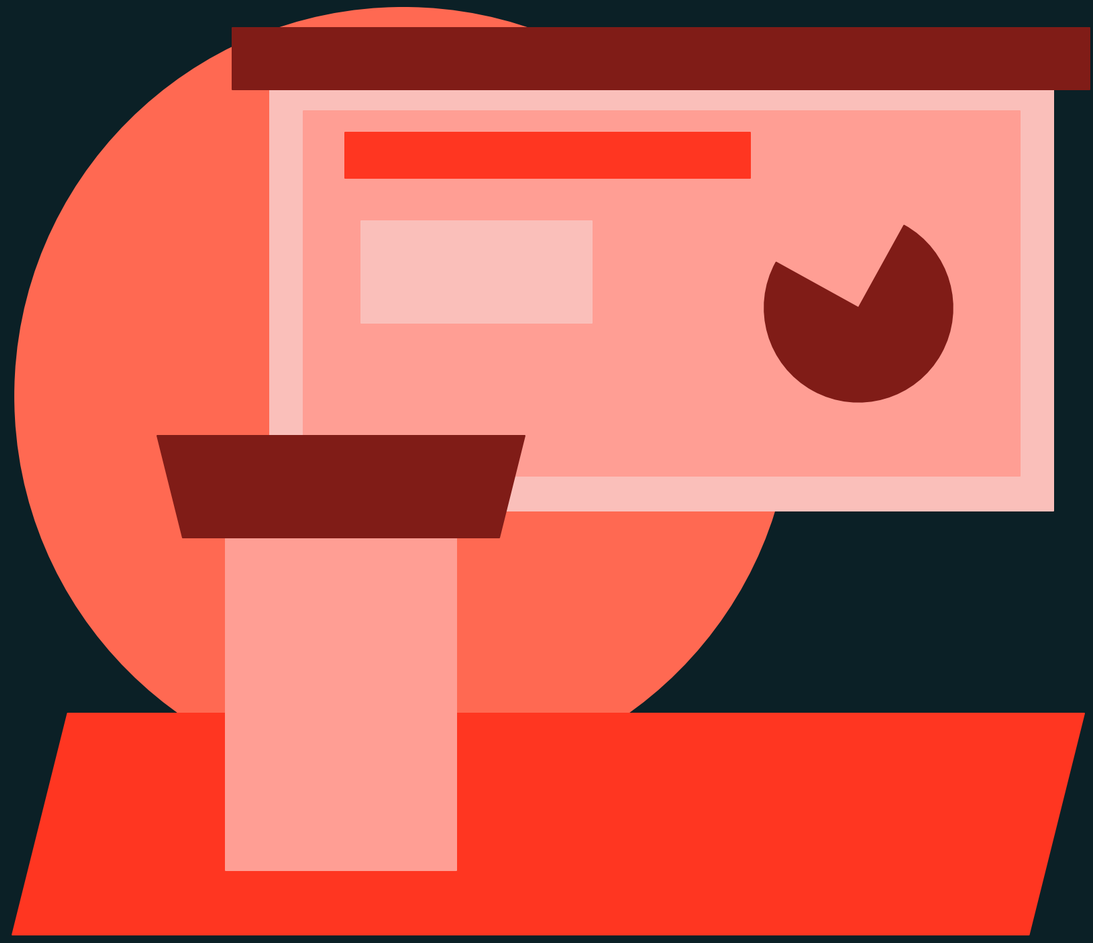
Notebook: 5 - Deploying Your Pipeline to Production

databricks

Building Lakeflow Declarative Pipelines

LECTURE

Change Data Capture (CDC) Overview



Change Data Capture (CDC) Overview

What is Change Data Capture?

- Change Data Capture (CDC) is a technique **used to track and capture changes in a data source** (such as a database, lakehouse or data warehouse).
- Then **applying those changes** into the Lakehouse or data warehouse

Source Data

INSERTS

UPDATES

DELETES

A Source table of changes, currently this must be streaming table

CDC is categorized into different types of **Slowly Changing Dimensions (SCD)**, which represent how **historical changes are handled**

Main types

SCD Type 1

SCD Type 2

Target Table

Table for changes to be applied

Change Data Capture (CDC) Overview

Slowing Changing Dimensions (SCD) Type 1 (1 of 2)

SCD Type 1 (Overwrite Target with Latest Values)

- When a record updates, the **previous record is simply overwritten** by its **key** with the new value.
- When a record is deleted by its **key**, the **record is removed**
- There is **no tracking of old keys (rows),** only the **current data** is retained.

customers (*target table with all customers*)

CustomerID	Name	Address	ProcessDate
1	Peter	1 Blue Rd.	5/1/2025
2	Samarth	22 Front St	5/1/2025

*Update the **customers** table with the **new customer** information*

updates (*source table with updates*)

CustomerID	Name	Address	Operation	ProcessDate
1	Peter	**123 Main St.**	**UPDATE**	5/20/2025
2	null	null	**DELETE**	5/20/2025
3	Kostas	100 Athens Dr.	**NEW**	5/20/2025

Change Data Capture (CDC) Overview

Slowing Changing Dimensions (SCD) Type 1 (2 of 2)

Update the row

Delete the row

Insert the row

customers (*target table with all customers*)

CustomerID	Name	Address	ProcessDate
1	Peter	**123 Main St.**	5/1/2025
~~2~~	~~Samarth~~	~~22 Front St~~	~~5/1/2025~~
3	**Kostas**	**100 Athens Dr.**	**5/20/2025**

updates (*source table with updates*)

CustomerID	Name	Address	Operation	ProcessDate
1	Peter	123 Main St.	**UPDATE**	5/20/2025
2	null	null	**DELETE**	5/20/2025
3	Kostas	100 Athens Dr.	**NEW**	5/20/2025

Used **SCD Type 1** when only the **current data matters** and historical values are irrelevant

Change Data Capture (CDC) Overview

Slowing Changing Dimensions (SCD) Type 2

SCD Type 2 (Historical Tracking/Versioning)

- *When a record changes:*
  - The **old record is preserved** with an additional column indicating its validity period (start date, end date, or a current flag) and a **new record is inserted** with the updated information.
  - When a **record is marked as deleted,** the record is kept and a column indicates the record is **inactive**.
- Used when **historical data is important**, and the system needs to track **how attributes change over time** (like tracking changes in a customer's address or status).

Example customers target table with SCD Type 2

Columns indicate **active** and **inactive** rows

CustomerID	Name	Address	ProcessDate	__START_AT	__END_AT
1	Peter	123 Main St.	5/20/2025	5/20/2025	null
1	Peter	1 Blue Rd.	5/1/2025	5/1/2025	5/20/2025
2	Samarth	22 Front St	5/1/2025	5/1/2025	5/20/2025
3	Kostas	100 Athens Dr.	5/20/2025	5/20/2025	null

Null values indicate the **current row**

Non null values indicate **inactive (historic)** rows

Change Data Capture (CDC) Overview

Using APPLY CHANGES INTO to Perform SCD in Declarative Pipelines

customers (*target table with all customers*)

CustomerID	Name	Address	ProcessDate
1	Peter	1 Blue Rd.	5/1/2025
2	Samarth	22 Front St	5/1/2025

updates (source table with updates)

CustomerID	Name	Address	Operation	ProcessDate
1	Peter	123 Main St.	**UPDATE**	5/20/2025
2	Samarth	22 Front St	**DELETE**	5/20/2025
3	Kostas	100 Athens Dr.	**NEW**	5/20/2025

```
APPLY CHANGES INTO customers
 FROM STREAM updates
 KEYS (CustomerID)
 APPLY AS DELETE WHEN operation = "DELETE"
 SEQUENCE BY ProcessDate
 COLUMNS * EXCEPT (operation)
 STORED AS SCD TYPE 2;
```

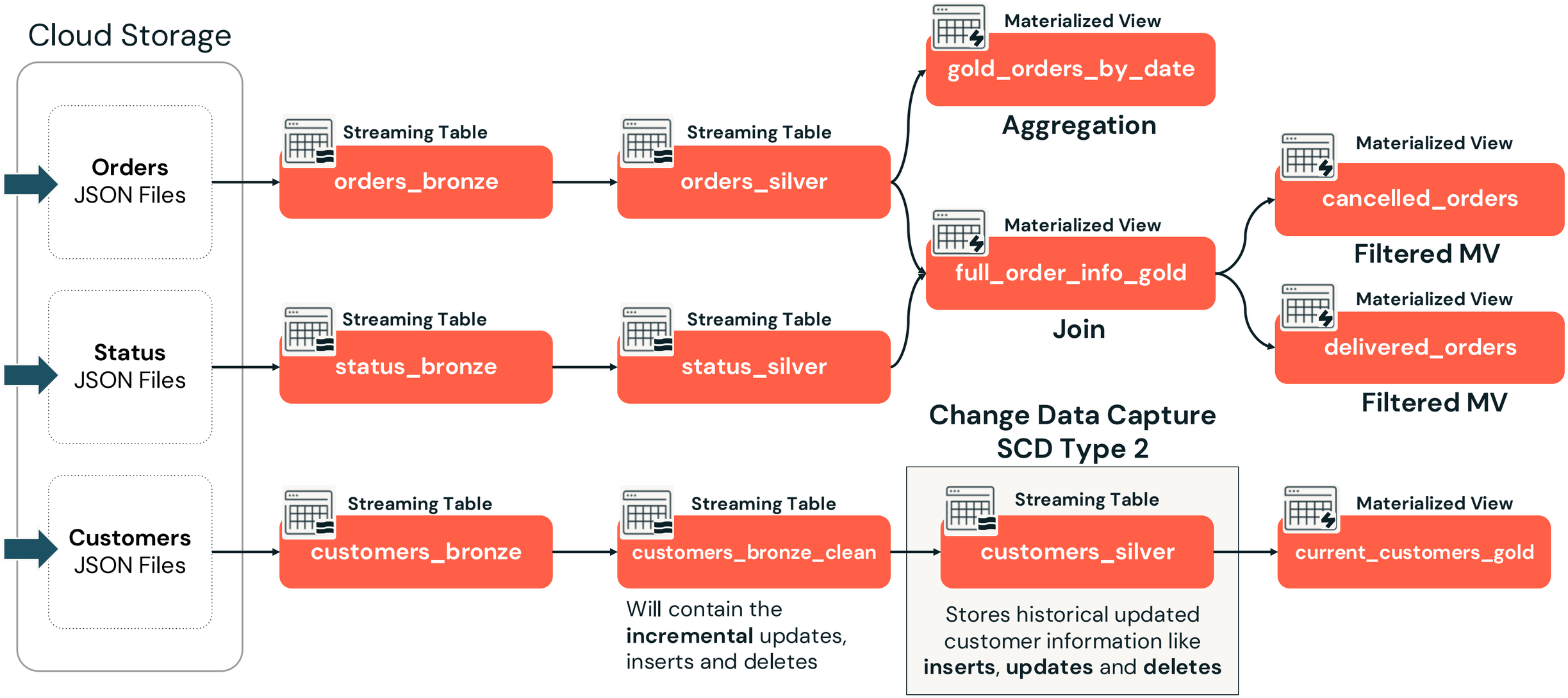
- Apply updates, inserts and deletes to the **target** table
- Source records to determine **updates, deletes and inserts**
- The column(s) that **uniquely identify a row** in the source and target tables
- Specifies when a CDC event should be treated as a **DELETE** rather than an upsert.
- Specifies the **logical order of CDC events** in the source data
- Specifies a **subset of columns** to include in the **target**
- Whether to store records as **SCD type 1 (default) or type 2**

*INSERTs and UPDATEs are implicitly implemented using the **KEY(s)**, no coding required*

Use **`APPLY CHANGES INTO`** instead of a traditional batch job with **`MERGE INTO`**

Use `APPLY CHANGES FROM SNAPSHOT` (Public Preview as of 2025Q2) to process changes in database snapshots.

Change Data Capture (CDC) Overview

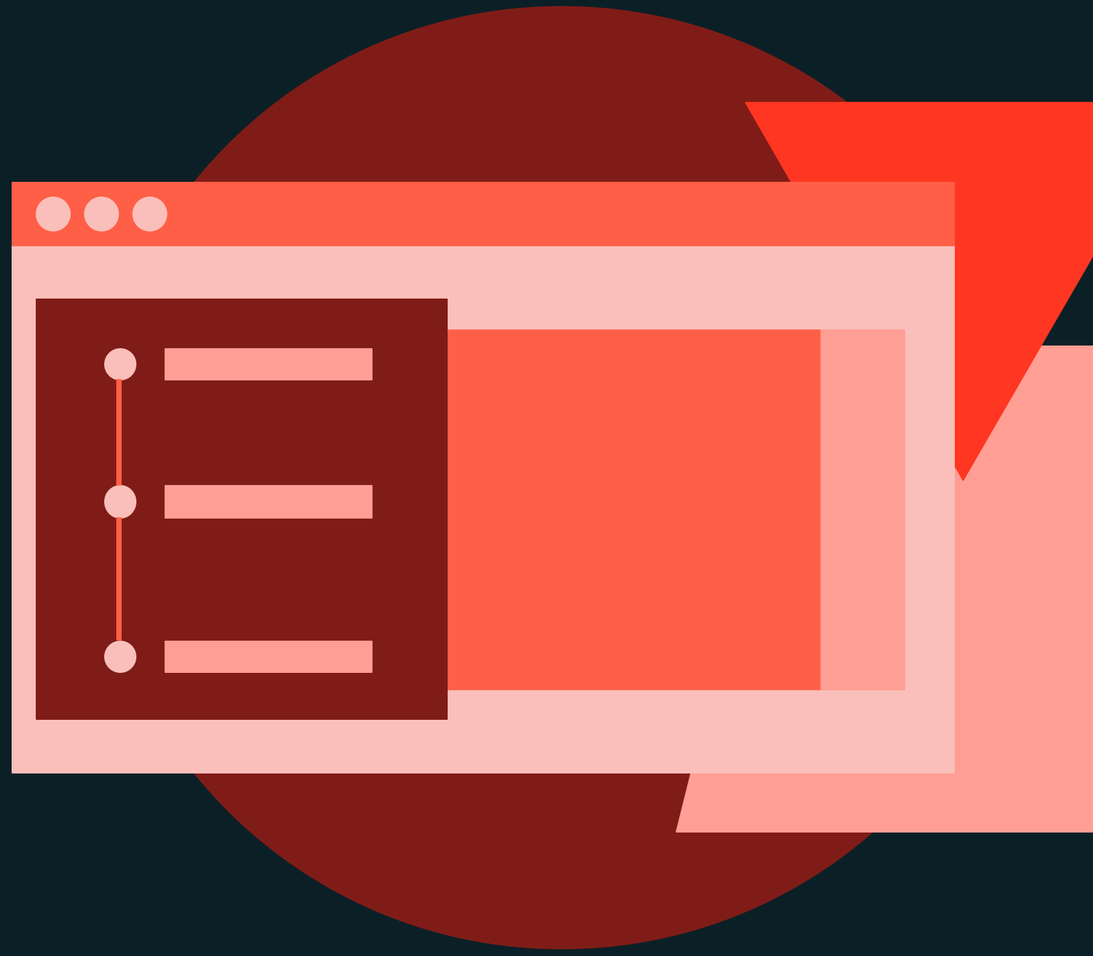


databricks

Building Lakeflow Declarative Pipelines

DEMONSTRATION

Change Data Capture with APPLY CHANGES INTO



Notebook: 6 - Change Data Capture with APPLY CHANGES INTO

databricks

Building Lakeflow Declarative Pipelines

LAB EXERCISE

BONUS – APPLY CHANGES INTO with SCD Type 1

If you are interested in performing CDC with Declarative Pipelines we have provided an extra lab you can complete after class.

Notebook: 7 BONUS Lab - APPLY CHANGES INTO with SCD Type 1

databricks

Building Lakeflow Declarative Pipelines

LECTURE

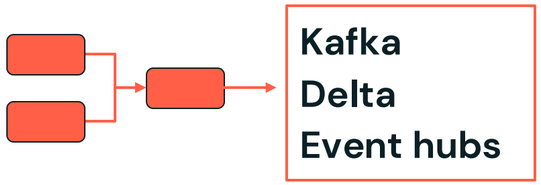
Additional Features Overview

Additional Features Overview

What's Next: Features Outside This Course

FLOWS

Add **flows** for the **same target**, creating a union of the two data sources (or tables)



SINKS

Sink API **enables writing to destinations** outside of the pipeline:

- Kafka
- Delta
- Event hubs

Full DELTA Support

Improved Delta Support for MVs/STs

- Liquid clustering
- Row-level security and column masking
- Change data feed (CDF) from STs

Additional Features Overview

What's Next: Features Outside This Course

Full UC Support

- Publish to multiple catalogs & schemas
- Read **STs** and **MVs** in Dedicated Access Mode
- Create a Unity Catalog pipeline by cloning a Hive metastore pipeline

Better Performance

Improved performance throughout Lakeflow Declarative Pipelines!

- Serverless
- Incremental refresh for materialized views
- Photon

Databricks Asset Bundles (DABs)

DABs enable you to **programmatically** validate, deploy, and run Databricks resources such as pipelines for **CI/CD production workloads**

- Automated Deployment with Databricks Asset Bundles course

Summary and Next Steps

Course Learning Objective Recap

- Understand the core concepts and components of Lakeflow Declarative Pipelines, including the function and differences between streaming tables, materialized views, and temporary views.
- Identify and configure Lakeflow pipeline settings, such as compute, data assets, trigger modes, and advanced options.
- Develop a functional Lakeflow Declarative Pipeline using the new pipeline editor and SQL-based syntax.
- Incorporate data quality expectations into a pipeline to validate and enforce data integrity.
- Analyze event logs and pipeline metrics to understand the full execution and lifecycle of a Lakeflow Declarative Pipeline.
- Design and implement a Change Data Capture (CDC) to a pipeline using APPLY CHANGES INTO to handle slowly changing dimensions (SCD).

Next Steps

Additional resources for continuing the learning journey.

Data Engineering with Databricks

- Continue your learning through self-paced or instructor-led offerings
- The courses offer hands-on instruction in:
  - Databricks Data Science & Engineering Workspace
  - Databricks SQL
  - Declarative Pipelines
  - Databricks Repos
  - Databricks Task Orchestration
  - Unity Catalog

Data Engineer Associate Certification

- Validate your data and AI skills on Databricks by earning a Databricks credential
- Exam information and exam guide
- The exam covers:
  - Data Intelligence Platform
  - ELT With Spark SQL and Python
  - Incremental Data Processing
  - Production Pipelines
  - Data Governance