Databricks Lakeflow

Co nowego w pipelines i jak to zmienia Data Engineering

Agenda

Wprowadzenie do Lakeflow Pipelines

Definicja i podejście deklaratywne

Składnia i podstawy

CREATE OR REFRESH, Streaming Tables vs Materialized Views

Expectations

Jakość danych, typy reakcji, dobre praktyki

Porównanie z innymi rozwiązaniami

Lakeflow vs Standardowy Spark, Automatyzacja

Compute

Serverless vs Classic Clusters

Lakeflow UI

Widok DAG, Monitoring

Demo

Praktyczne zastosowanie

Krzysztof Burejza

Azure Data Engineer

Inżynier danych z 10-letnim doświadczeniem w pracy z danymi, od SQL Server po chmurowe platformy takie jak Azure, Databricks i Microsoft Fabric. Skupia się na przekształcaniu surowych danych w wartościowe rozwiązania, budując pipeline'y, hurtownie danych i lakehouse'y. Entuzjasta nowych technologii z pogranicza danych i AI, które sprawiają, że inżynieria danych staje się jeszcze bardziej fascynująca.

Lakeflow Pipelines

Nowoczesny ETL

Co to jest Lakeflow Pipelines?

Lakeflow (wcześniej Delta Live Tables) to framework do **deklaratywnego budowania potoków danych** w Databricks. Zamiast pisać kod orkiestracji, definiujemy **ostateczny wynik** – framework zarządza wykonaniem, monitorowaniem i jakością danych.

Tradycyjne podejście

- Ręczne zarządzanie stanem
- Niestandardowa logika ponownych prób
- Niestandardowe punkty kontrolne
- Ręczne śledzenie zależności
- Oddzielne narzędzia do kontroli jakości

Lakeflow Pipelines

- Deklaratywne definicje tabel
- Automatyczne ponawianie i obsługa błędów
- Wbudowane checkpointing
- Automatyczny graf pochodzenia
- Wbudowane Oczekiwania (zasady jakości)

Co się stało z Delta Live Tables (DLT)?

Delta Live Tables (DLT) zostało zaktualizowane do **Lakeflow Spark Declarative Pipelines (SDP)**. Istniejący kod DLT jest w pełni kompatybilny, lecz zaleca się aktualizację do nowych nazw API dla przyszłych funkcji i kompatybilności z Apache Spark™ Declarative Pipelines (od Spark 4.1).

Kluczowe zmiany w Python API:

~~@dlt~~ → @dp

Zastąpienie odwołania do biblioteki.

~~@table~~ (ogólnie) → @table (streaming) / @materialized_view

Wprowadzono dedykowany dekorator dla widoków zmaterializowanych, zwiększając czytelność.

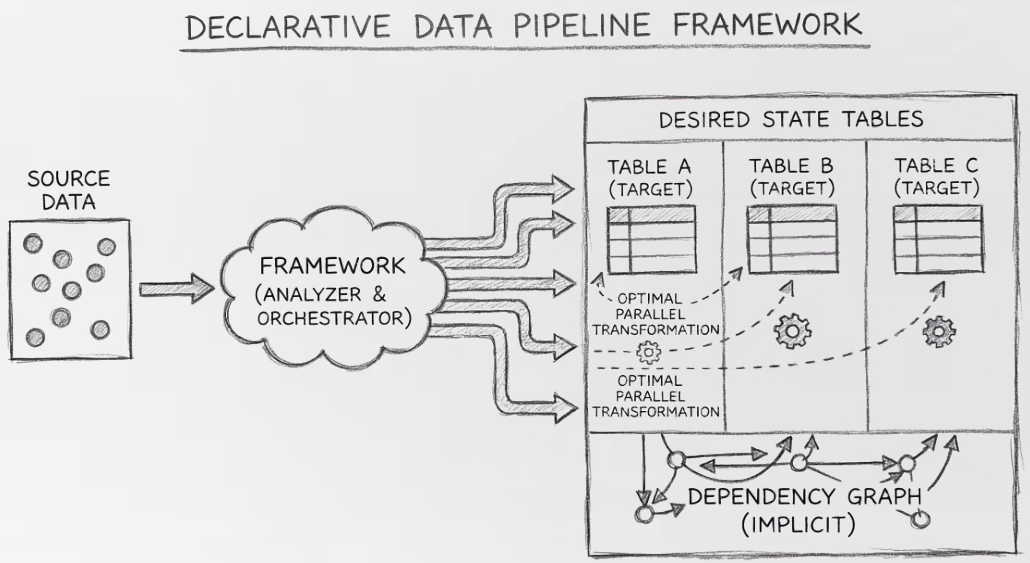
~~@view~~ → @temporary_view

Jasne rozróżnienie dla widoków tymczasowych.

Uwaga: Wciąż można spotkać odwołania do nazwy DLT w niektórych miejscach Databricks (np. SKU klasyczne, schematy logów zdarzeń). Dotychczasowe Python API z nazwami DLT są nadal wspierane, ale rekomendowane jest przejście na nowe nazewnictwo.

Podejście Declarative – "CO" zamiast "JAK"

W podejściu deklaratywnym opisujemy pożądany stan danych (co chcemy osiągnąć), zamiast imperatywnych kroków (jak to zrobić). Lakeflow samodzielnie analizuje zależności i optymalizuje wykonanie transformacji.



Co składa się na Lakeflow Pipelines?

Lakeflow Pipelines to framework integrujący kluczowe aspekty inżynierii danych, umożliwiający budowanie niezawodnych i wydajnych potoków w oparciu o paradygmat Lakehouse.

Deklaratywne Definicje

Skoncentruj się na **"CO"** chcesz osiągnąć z danymi, a nie **"JAK"** to zaimplementować. Lakeflow automatycznie zarządza logiką wykonania.

Automatyczna Orkiestracja

Framework samodzielnie zarządza kolejnością zadań, zależnościami, ponownymi próbami w przypadku błędów i monitorowaniem postępu.

Wbudowana Jakość Danych

Definiuj reguły jakości danych bezpośrednio w kodzie potoku, zapewniając automatyczną walidację i kontrolę nad danymi.

Streaming i Widoki Zmaterializowane

Ujednolicone podejście do przetwarzania danych strumieniowych i batchowych w jednym potoku.

Uproszczony Compute

Opcje Serverless minimalizują potrzebę zarządzania infrastrukturą, automatycznie skalując zasoby zależnie od obciążenia.

Składnia CREATE OR REFRESH

Tabela strumieniowa

Dla źródeł append-only (logi, zdarzenia, transakcje):

```
CREATE OR REFRESH STREAMING TABLE bronze_orders
AS SELECT *, current_timestamp() as ingestion_time
FROM read_files(
  "/Volumes/raw/orders/", format=>"csv",
header=>true, inferSchema=>true
)
```

Tabela strumieniowa automatycznie przetwarza nowe pliki.

Zmaterializowany widok

Dla transformacji z pełnym odświeżeniem lub aktualizacją inkrementalną:

```
CREATE OR REFRESH MATERIALIZED VIEW silver_orders
AS SELECT
  order_id, customer_id, order_date, total_amount,
items
FROM STREAM(bronze_orders)
WHERE order_status != 'cancelled'
```

`STREAM()` oznacza przetwarzanie inkrementalne.

Kiedy używać Streaming, a kiedy Materialized?

Streaming Table

- Dane są typu append-only (zdarzenia, logi)
- Wymagana semantyka exactly-once
- Dane napływają w czasie rzeczywistym
- Warstwa Bronze/Raw - zachowanie wszystkich danych
- CDC z baz danych transakcyjnych

Przykłady: logi aplikacji, strumienie Kafka, CDC z PostgreSQL

Materialized View

- Wymagane UPDATE/DELETE (SCD2, deduplikacja)
- Agregacje i złożone połączenia
- Warstwa Gold - metryki biznesowe
- Tabele Snapshot - tylko bieżący stan
- Tabele cech (Feature Tables) dla ML

Przykłady: Obliczanie CLV, dziennej liczby aktywnych użytkowników, funkcji rekomendacji produktów

Przepływy Danych (Flows) w Lakeflow Pipelines

Przepływ to połączenie zapytania i tabeli docelowej, przetwarzające dane wsadowo lub strumieniowo, inkrementalnie w potoku Lakeflow Spark Declarative Pipelines.

Definicja i typy przepływów

- Przepływy są tworzone **automatycznie** podczas definiowania tabeli lub widoku (np. `CREATE OR REFRESH STREAMING TABLE`).
- Można je definiować **jawnie** (`CREATE FLOW` / `@dp.append_flow`) dla złożonych scenariuszy.
- Standardowe przepływy (append flows) **dodają nowe wiersze**, idealne do pozyskiwania i transformacji danych.
- Przepływy **Auto CDC** służą do ingestowania danych ze zmianami (Change Data Capture).

Zastosowania i korzyści

- Każda aktualizacja potoku uruchamia przepływ odświeżający tabele.
- Może to być **odświeżanie inkrementalne** (tylko nowe rekordy) lub **pełne**.
- Umożliwia **łączenie danych z wielu źródeł** do jednej tabeli docelowej (np. dodawanie nowych regionów) bez pełnego odświeżania.
- Pozwala na **uzupełnianie danych historycznych** (backfilling) przy użyciu składni `INSERT INTO ONCE`.

Wiele Przepływów (Flows) do Jednego Celu

Tworzenie tabeli i przypisywanie przepływu

Można najpierw utworzyć tabelę strumieniową, a następnie zdefiniować do niej przepływ. Poniższy przykład pokazuje, jak niezależnie utworzyć tabelę i przypisać pojedynczy przepływ, uzyskując te same rezultaty co w przypadku domyślnego tworzenia przepływu.

```
-- utwórz tabelę strumieniową
CREATE OR REFRESH STREAMING TABLE customers_silver;

-- dodaj przepływ do tabeli customers_silver
CREATE FLOW customers_silver_flow
AS INSERT INTO customers_silver BY NAME
SELECT * FROM STREAM(customers_bronze);
```

Łączenie danych z wielu źródeł

Wykorzystanie wielu przepływów dopisujących (`append_flow` w Python lub `CREATE FLOW...INSERT INTO` w SQL) jest idealne do łączenia danych z różnych źródeł w jednej tabeli strumieniowej. Zapewnia to inkrementalne aktualizacje, bardziej efektywne niż użycie klauzuli `UNION`.

```
-- utwórz tabelę strumieniową
CREATE OR REFRESH STREAMING TABLE customers_us;

-- dodaj pierwszy przepływ dopisujący z danych z zachodniego USA
CREATE FLOW append_us_west
AS INSERT INTO customers_us BY NAME
SELECT * FROM STREAM(customers_us_west);

-- dodaj drugi przepływ dopisujący z danych ze wschodniego USA
CREATE FLOW append_us_east
AS INSERT INTO customers_us BY NAME
SELECT * FROM STREAM(customers_us_east);
```

Ważne: Ograniczenia jakości danych (Expectations) należy definiować na **tabeli docelowej** (np. `customers_us`), a nie w definicji przepływu `@dp.append_flow`.

Przepływy są identyfikowane przez nazwę, służącą do zarządzania punktami kontrolnymi strumieniowania. Zmiana nazwy istniejącego przepływu jest traktowana jako utworzenie nowego, resetując jego punkty kontrolne. Ponadto, tej samej nazwy przepływu nie można użyć ponownie w jednym potoku.

Backfilling Danych Historycznych

Lakeflow Pipelines ułatwia uzupełnianie danych historycznych (backfilling) w istniejących potokach. Jest to kluczowe, gdy trzeba przetworzyć dane z okresów nieuwzględnionych w początkowej konfiguracji potoku.

Definiowanie zakresu

Stwórz nowy przepływ, np. poprzez usunięcie lub zmianę `modifiedAfter`.

Jednorazowe uruchomienie

Użyj składni `INSERT INTO ONCE` do jednorazowego wykonania i uniknięcia duplikatów.

Integracja

Nowe dane historyczne zostaną bezproblemowo zintegrowane z tabelą docelową.

```
-- create the streaming table
CREATE OR REFRESH STREAMING TABLE registration_events_raw;

-- original incremental flow (e.g., from 2025 onwards)
CREATE FLOW registration_events_raw_incremental
AS INSERT INTO registration_events_raw BY NAME
SELECT * FROM STREAM read_files(
  "/Volumes/gc/demo/apps_raw/event_registration/*",
  format => "json",
  modifiedAfter => "2024-12-31T23:59:59.999+00:00"
);

-- one-time backfill for 2024
CREATE FLOW registration_events_raw_backfill_2024
AS INSERT INTO ONCE registration_events_raw BY NAME
SELECT * FROM read_files(
  "/Volumes/gc/demo/apps_raw/event_registration/year=2024/*",
  format => "json"
);
```

CDC APPLY w Lakeflow Pipelines (SCD Type 1 i 2)

Czym jest AUTO CDC INTO?

`AUTO CDC INTO` automatyzuje zarządzanie zmianami danych, zastępując ręczną logikę `MERGE INTO`. Upraszcza integrację danych z systemów transakcyjnych, gdzie śledzenie ewolucji rekordów jest kluczowe.

`KEYS (kolumna)`

Określa jedną lub więcej kolumn stanowiących unikalny klucz rekordu.

`APPLY AS DELETE WHEN ...`

Definiuje warunek logiczny, kiedy rekord źródłowy powinien zostać potraktowany jako usunięcie.

`APPLY AS TRUNCATE WHEN ...`

Definiuje warunek, który powoduje usunięcie wszystkich rekordów z tabeli docelowej przed zastosowaniem zmian.

`SEQUENCE BY kolumna`

Służy do obsługi zdarzeń poza kolejnością, określając najnowszą zmianę.

`COLUMNS * EXCEPT (kolumny)`

Wskazuje wszystkie kolumny źródłowe do uwzględnienia, z wyjątkiem wymienionych.

`STORED AS SCD TYPE 1`

Zmiany aktualizują rekordy, nadpisując stare wartości (tylko najnowsza wersja).

`STORED AS SCD TYPE 2`

Zmiany są przechowywane jako nowe rekordy z kolumnami `__START_AT` i `__END_AT` do śledzenia historii.

SCD Type 1: Zachowanie Najnowszych Danych

W SCD Type 1 aktualizacje danych nadpisują istniejące rekordy w tabeli docelowej, zapewniając zawsze najnowszą wersję danych. Lakeflow Pipelines automatycznie zarządza tą logiką.

```
CREATE OR REFRESH STREAMING TABLE target;

CREATE FLOW flowname AS AUTO CDC INTO target
  FROM stream(cdc_data.users)
  KEYS (userId)
  APPLY AS DELETE WHEN operation = "DELETE"
  APPLY AS TRUNCATE WHEN operation = "TRUNCATE"
  SEQUENCE BY sequenceNum
  COLUMNS * EXCEPT (operation, sequenceNum)
  STORED AS SCD TYPE 1;
```

Po zastosowaniu aktualizacji SCD Type 1, tabela docelowa zawiera najnowsze rekordy:

userId	name	city
124	Raul	Oaxaca
125	Mercedes	Guadalajara
126	Lily	Cancun

SCD Type 2: Śledzenie Historii Zmian

SCD Type 2 śledzi wszystkie zmiany, zachowując historyczne wersje rekordów. Każda zmiana tworzy nowy rekord z nowym okresem ważności (`__START_AT` i `__END_AT`), co jest kluczowe dla analizy zmian w czasie.

```
CREATE OR REFRESH STREAMING TABLE target;

CREATE FLOW target_flow AS AUTO CDC INTO target
  FROM stream(cdc_data.users)
  KEYS (userId)
  APPLY AS DELETE WHEN operation = "DELETE"
  SEQUENCE BY sequenceNum
  COLUMNS * EXCEPT (operation, sequenceNum)
  STORED AS SCD TYPE 2;
```

Po zastosowaniu aktualizacji SCD Type 2, tabela docelowa zawiera historyczne rekordy:

userId	name	city	__START_AT	__END_AT
123	Isabel	Monterrey	1	5
123	Isabel	Chihuahua	5	6

Zautomatyzowane Zadania Konserwacji

Lakeflow Pipelines autonomicznie wykonuje kluczowe zadania konserwacyjne na zarządzanych tabelach, wykorzystując zaawansowaną optymalizację predykcyjną. Zapewnia to lepszą wydajność zapytań i redukcję kosztów poprzez usuwanie przestarzałych danych i ich porządkowanie.

Kluczowe Operacje

Potoki wykonują operacje **OPTIMIZE** (kompakcja plików) i **VACUUM** (usuwanie starych danych), zapewniając optymalne wykorzystanie pamięci masowej.

Inteligentne Harmonogramowanie

Zadania konserwacyjne są planowane przez mechanizm optymalizacji predykcyjnej i uruchamiane tylko po aktualizacji potoku.

Korzyści

Automatyczna konserwacja utrzymuje tabele w optymalnym stanie, skutkując zwiększoną wydajnością zapytań, mniejszym zużyciem zasobów i obniżonymi kosztami przechowywania.

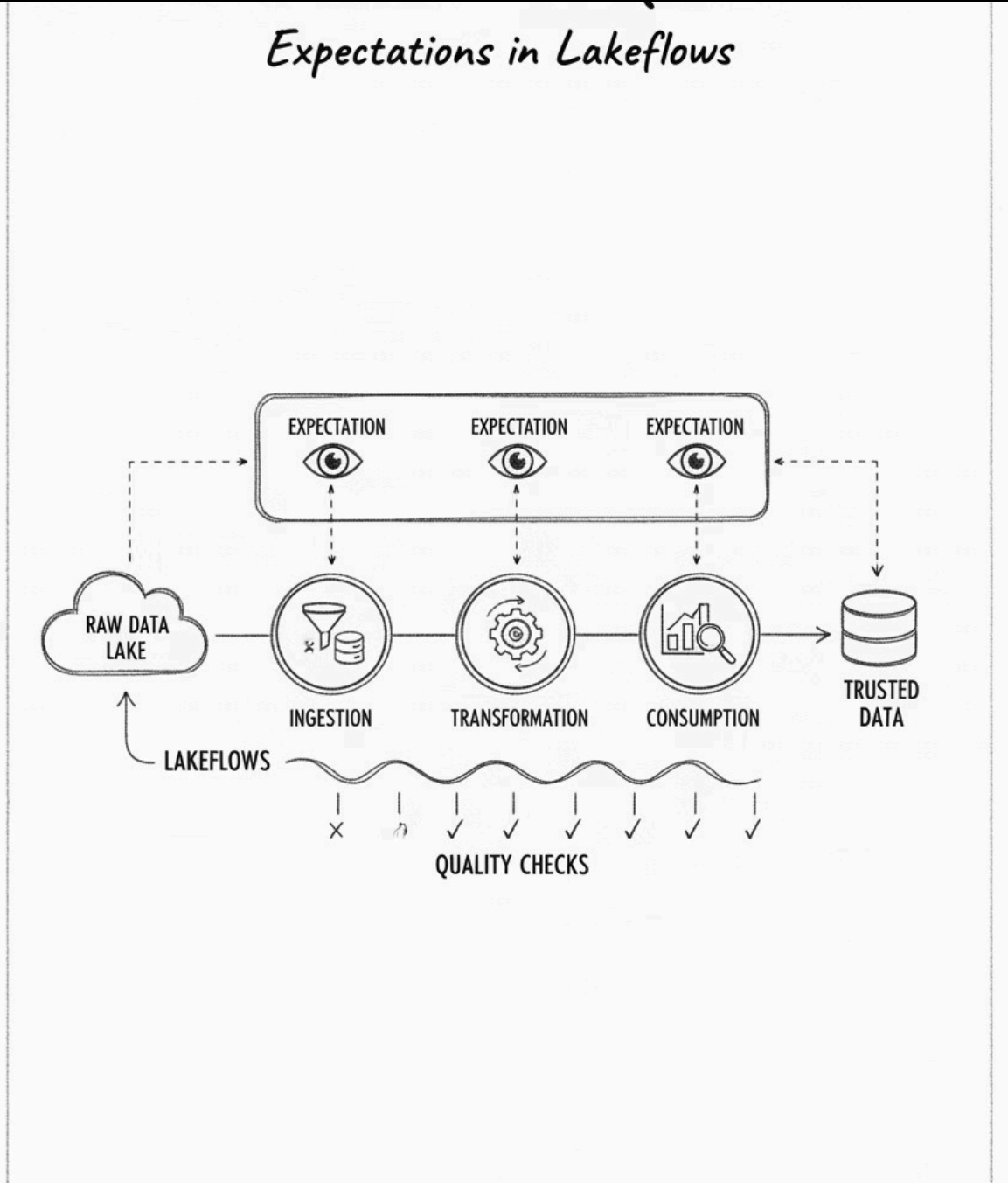
Aby dowiedzieć się więcej o częstotliwości działania optymalizacji predykcyjnej i związanych z nią kosztach, zapoznaj się z **tabelą systemową optymalizacji predykcyjnej**.

Expectations

W kontekście Lakeflow Pipelines, "Expectations" definiują zasady i testy jakości danych, które są automatycznie weryfikowane podczas przetwarzania. Pozwalają na zapewnienie wiarygodności danych i wczesne wykrywanie anomalii.

Jakość danych - Expectation

Oczekiwania to deklaratywne reguły jakości danych, wbudowane bezpośrednio w potok. Działają jak testy jednostkowe dla danych, automatycznie weryfikując warunki przy każdym przetwarzaniu. Blokują propagację błędnych danych do tabel podrzędnych.



Typy reakcji na naruszenia

USUŃ WIERSZ

Wiersze niespełniające warunku są automatycznie pomijane. Stosuj, gdy błędne dane mogą być bezpiecznie ignorowane.

```
CONSTRAINT valid_email
  EXPECT (email LIKE '%@%')
  ON VIOLATION DROP ROW
```

ANULUJ AKTUALIZACJĘ

Cała partia jest odrzucana, jeśli jakikolwiek wiersz narusza warunek. Stosuj do krytycznych reguł biznesowych.

```
CONSTRAINT positive_amount
  EXPECT (amount > 0)
  ON VIOLATION FAIL UPDATE
```

TYLKO ŚLEDZENIE (DOMYŚLNIE)

Naruszenie jest rejestrowane w metrykach, ale dane przepływają dalej. Idealne do monitorowania jakości bez blokowania potoku.

```
CONSTRAINT recent_data
  EXPECT (
    event_date >= current_date() -
7
  )
```

Metryki: Wszystkie naruszenia są automatycznie zapisywane w dzienniku zdarzeń potoku i dostępne w interfejsie użytkownika jako metryki dla każdej tabeli.

Dobre praktyki z expectations

Dlaczego to jest ważne?

Dobre praktyki ułatwiają stosowanie, aktualizację i audytowanie reguł expectations w wielu zbiorach danych i pipeline'ach, bez modyfikacji kodu. Zwiększają spójność walidacji, redukują koszty utrzymania i poprawiają przenośność rozwiązań.

Separacja definicji

Przechowuj definicje expectations oddzielnie od logiki pipeline'u

Tagowanie

Dodawaj własne tagi do grupowania powiązanych expectations

Reużywalność

Stosuj te same expectations w wielu pipeline'ach

Centralne repozytorium Expectations: Tabela Delta

Tabela `rules` centralizuje reguły walidacji, przechowując nazwę, constraint SQL i tag do filtrowania.

```
CREATE OR REPLACE TABLE rules AS
SELECT col1 AS name, col2 AS constraint, col3 AS tag
FROM (
  VALUES
    ("website_not_null",
     "Website IS NOT NULL",
     "validity"),
    ("fresh_data",
     "to_date(updateTime,'M/d/yyyy h:m:s a') > '2010-01-01'",
     "maintained"),
    ("social_media_access",
     "NOT(Facebook IS NULL AND Twitter IS NULL AND Youtube IS NULL)",
     "maintained")
)
```

Upraszcza zarządzanie regułami przez SQL, umożliwiając łatwe dodawanie nowych constraints i kategoryzację tagami biznesowymi.

Aplikowanie reguł w pipeline'ie Python

Funkcja `get_rules()` pobiera reguły walidacji z tabeli, dopasowując je do tagu. Dekorator `@dp.expect_all_or_drop()` automatycznie usuwa rekordy niespełniające tych kryteriów.

```
from pyspark import pipelines as dp
from pyspark.sql.functions import expr, col

def get_rules(tag):
 df = spark.read.table("rules").filter(col("tag") == tag).collect()
 return {row['name']: row['constraint'] for row in df}

@dp.table
@dp.expect_all_or_drop(get_rules('validity'))
def raw_farmers_market():
 return (
 spark.read.format('csv').option("header", "true")
 .load('/databricks-datasets/data.gov/farmers_markets_geographic_data/')
 )

@dp.table
@dp.expect_all_or_drop(get_rules('maintained'))
def organic_farmers_market():
 return spark.read.table("raw_farmers_market").filter(expr("Organic = 'Y'"))
```

Ewolucja schematu

Ten wzorzec umożliwia migrację źródeł danych i zarządzanie wieloma wersjami, zapewniając kompatybilność wsteczną i jakość danych.

```
CREATE OR REFRESH MATERIALIZED VIEW evolving_table(
  CONSTRAINT valid_migrated_data EXPECT (
    (col1 IS NOT NULL AND col2 IS NOT NULL) AND
    (CASE WHEN col3 IS NOT NULL THEN col3 > 0 ELSE TRUE END)
  ) ON VIOLATION FAIL UPDATE
) AS
SELECT * FROM new_source
UNION
SELECT *, NULL as col3 FROM legacy_source;
```

`UNION` łączy nowe i starsze źródła. `CASE` obsługuje opcjonalne kolumny. `ON VIOLATION FAIL UPDATE` zatrzymuje pipeline przy krytycznych błędach.

Sprawdzenie liczby wierszy

Monitorowanie liczby wierszy w widokach zmaterializowanych za pomocą `COUNT(*)` pozwala na szybką weryfikację rekordów i identyfikację anomalii, pomagając w utrzymaniu jakości danych.

```
CREATE OR REFRESH MATERIALIZED VIEW
count_verification(
  CONSTRAINT no_rows_dropped
  EXPECT (a_count == b_count)
) AS
SELECT * FROM
  (SELECT COUNT(*) AS a_count FROM table_a),
  (SELECT COUNT(*) AS b_count FROM table_b)
```

Constraint `EXPECT (a_count == b_count)` automatycznie wykrywa rozbieżności w liczbie wierszy między tabelami źródłowymi.

Wykrywanie brakujących rekordów i unikalność kluczy

Detekcja brakujących rekordów

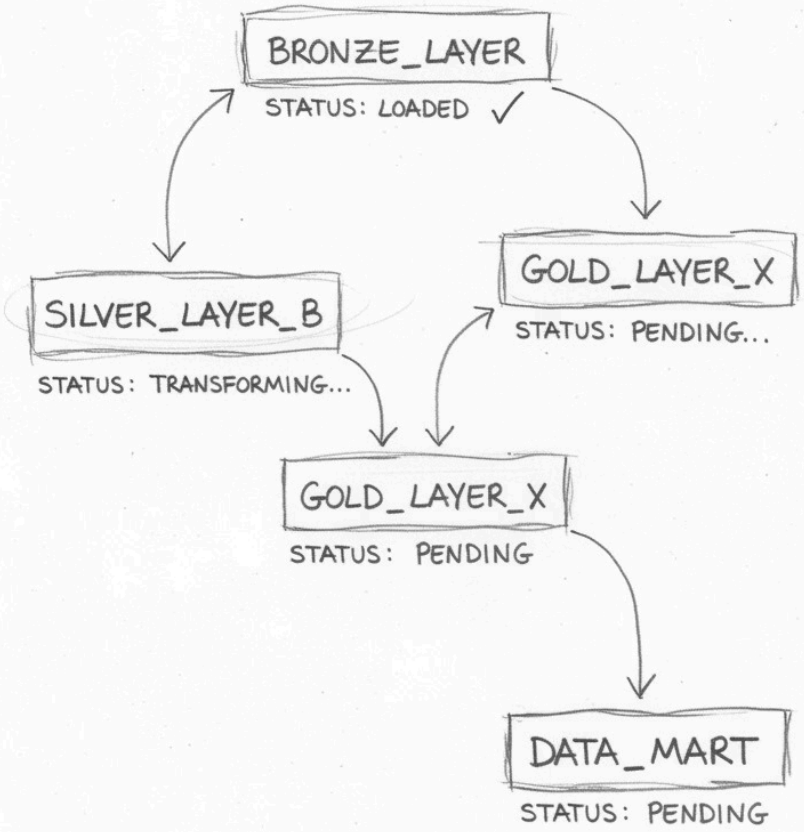
```
CREATE OR REFRESH MATERIALIZED VIEW
report_compare_tests(
  CONSTRAINT no_missing_records
  EXPECT (r_key IS NOT NULL)
) AS
SELECT v.*, r.key as r_key
FROM validation_copy v
LEFT OUTER JOIN report r
ON v.key = r.key
```

LEFT JOIN ujawnia rekordy obecne w walidacji, ale brakujące w raporcie.

Walidacja klucza głównego

```
CREATE OR REFRESH MATERIALIZED VIEW
report_pk_tests(
  CONSTRAINT unique_pk
  EXPECT (num_entries = 1)
) AS
SELECT pk, count(*) as num_entries
FROM report
GROUP BY pk
```

GROUP BY wykrywa duplikaty klucza głównego poprzez zliczanie wystąpień.



Widok DAG w interfejsie Lakeflow

Interfejs Databricks UI automatycznie generuje wizualny DAG (skierowany graf acykliczny) wszystkich tabel w potoku. Każdy węzeł pokazuje status (działa/sukces/niepowodzenie), metryki przetwarzania i naruszenia jakości danych. Możesz kliknąć węzeł, aby zobaczyć kod definicji, podgląd danych i szczegółowe logi.

Co pokazuje DAG

- Zależności między tabelami (upstream → downstream)
- Status każdej tabeli w czasie rzeczywistym
- Liczba wierszy i objętość danych
- Czas trwania przetwarzania
- Podsumowanie naruszeń oczekiwań

Interakcja

- Kliknięcie węzła → szczegóły tabeli
- Najazd myszką → szybki podgląd metryk
- Widok osi czasu → historyczne uruchomienia
- Szczegółowa analiza błędów
- Śledzenie ewolucji schematu

Standardowy Spark vs Lakeflow Pipelines

Cecha	Standardowy Spark SQL/Python	Lakeflow Pipelines (DLT)
Model programowania	SQL/DataFrame API, więcej kodu boilerplate.	Rozszerzony SQL/Python z DLT (CREATE OR REFRESH, EXPECT).
Automatyzacja infrastruktury	Ręczna konfiguracja klastrów, zarządzanie punktami kontrolnymi.	Automatyczne zarządzanie klastrami, punktami kontrolnymi, odświeżaniem.
Jakość danych	Własna implementacja reguł jakości danych.	Wbudowane oczekiwania (EXPECT) z konfigurowalnymi reakcjami.
Monitoring i obserwowalność	Integracja z zew. narzędziami lub własna implementacja.	Automatyczny DAG, metryki, logi i statusy w UI.
Zarządzanie zależnościami	Zewnętrzny orkiestrator (np. Airflow).	Automatyczne wykrywanie i zarządzanie zależnościami tabel.
Optymalizacja wydajności	Ręczna optymalizacja (np. partycjonowanie, indeksowanie).	Automatyczne optymalizacje Photon, buforowanie i skalowanie.

Automatyzacja – co dostajemy za darmo?

Pochodzenie danych

Automatyczny wykres zależności tabel wskazuje źródła i konsumentów, propagując zmiany w górę strumienia.

Zarządzanie stanem

Automatyczne zarządzanie punktami kontrolnymi i przetwarzaniem stanu, eliminujące ręczną konfigurację i śledzenie plików.

Odświeżanie przyrostowe

Potok przetwarza tylko zmienione dane (delty), oszczędzając zasoby i skracając czas wykonania 10-100 razy.

Obsługa błędów

Automatyczne ponawianie prób i łagodna degradacja w przypadku błędów, zapobiegające zawieszaniu się potoku przez wadliwe rekordy.

Obserwowalność

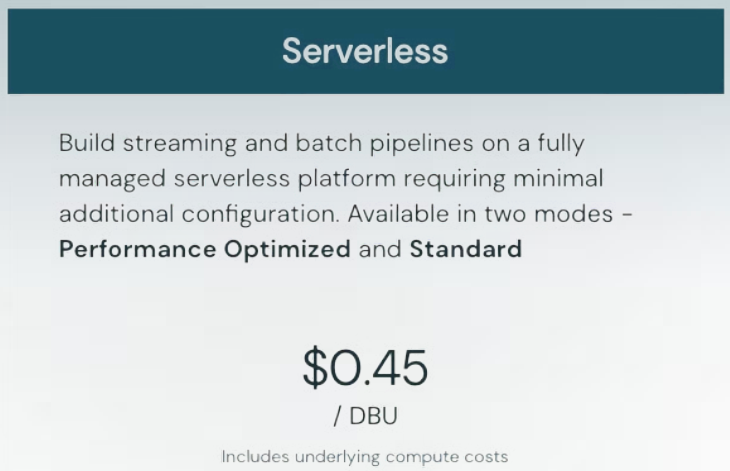
Wbudowane metryki (liczba wierszy, czas przetwarzania, jakość danych) dostępne w UI, bez dodatkowej instrumentacji kodu.

Optymalizacja

Automatyczna optymalizacja planu wykonania i buforowanie wyników, zapewniające wydajność bez ręcznej interwencji.

Compute

Lakeflow Pipelines oferuje dwie opcje compute, dostosowane do różnych obciążeń danych.



Serverless

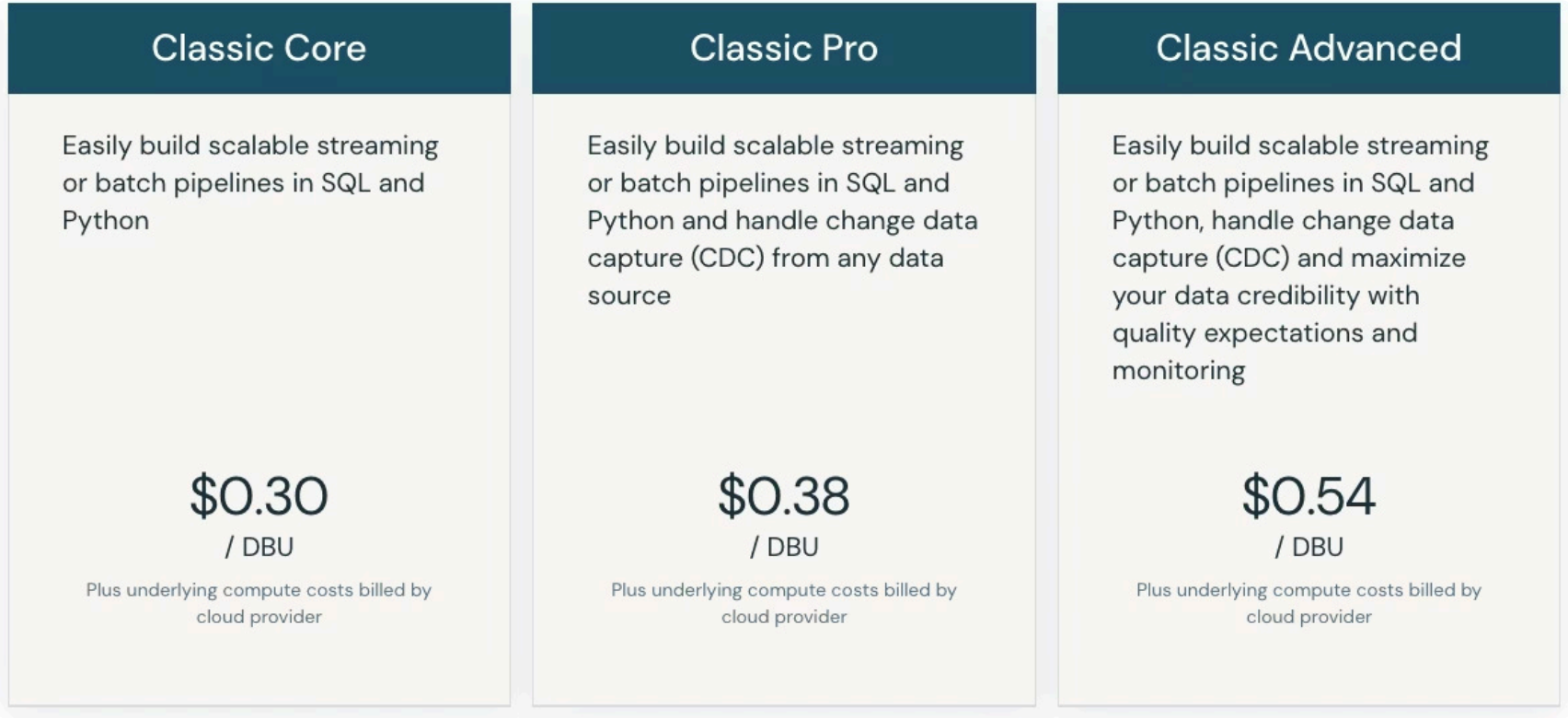
Serverless dla Lakeflow Pipelines to w pełni zarządzana opcja. Eliminacja konfiguracji klastrów i cena wliczona w koszt DBU, bez dodatkowych opłat za infrastrukturę chmury. Dostępne są dwa warianty:

- **Standard:** do 70% tańszy, czas startu 4-6 minut.
- **Performance Optimized:** szybszy start (<1 minuta), z Enzyme (przyrostowe odświeżanie), Stream Pipelining i automatycznym skalowaniem zapobiegającym problemom OOM.

Oba warianty posiadają automatycznie włączony Photon engine, zapewniają automatyczny wybór instancji i skalowanie. Obsługują także zaawansowane funkcje: CDC, SCD Typ 2 oraz reguły jakości danych.

Classic Clusters

Tryb Classic dla Lakeflow Pipelines wykorzystuje zaawansowane klastry, które wymagają ręcznej konfiguracji. W tym wariancie obowiązuje oddzielna płatność za instancje chmurowe oraz koszt DBU Lakeflow Pipelines. Czas uruchomienia klastrów wynosi od 1 do 6 minut, a silnik Photon jest opcjonalny. Należy pamiętać, że zaawansowane funkcje takie jak CDC, SCD Typ 2 oraz reguły jakości danych są niedostępne w Classic Core.



Porównanie: Classic vs Serverless

	Classic	Serverless Standard	Serverless Performance Optimized
Typowy przypadek użycia	Tradycyjne obciążenia, pełna kontrola	Ogólne potoki danych, mniejsze koszty	Potoki o wysokiej wydajności, szybki start
Przechwytywanie Zmian Danych (CDC)	Niedostępne	Tak	Tak
Powoli Zmieniające się Wymiary (SCD) Typ 2	Niedostępne	Tak	Tak
Reguły Oczekiwań Jakości Danych	Niedostępne	Tak	Tak
Silnik Photon	Opcjonalny (2.5X DBU)	Automatycznie	Automatycznie
Automatyczny wybór instancji	Ręczny	Automatyczny	Automatyczny
Auto-skalowanie (OOM)	Nie	Nie	Tak
Enzym (odświeżanie przyrostowe)	Nie	Nie	Tak
Potoki Strumieniowe	Nie	Nie	Tak
Czas startu	4-6 min	4-6 min	<1 min
Całkowity koszt	Instancje płatne oddzielnie	Wliczone w cenę (do 70% taniej)	Wliczone w cenę

Techniki Odświeżania w Lakeflow Pipelines

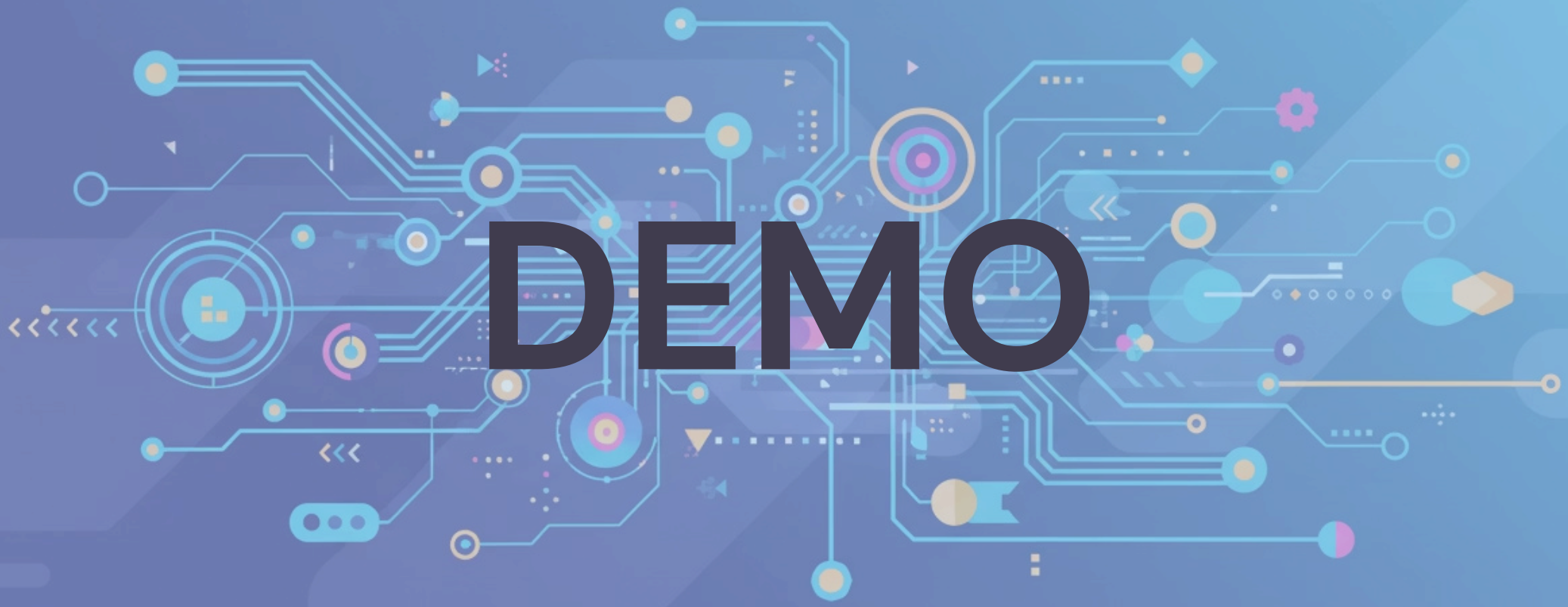
Lakeflow Pipelines automatycznie zarządza odświeżaniem widoków zmaterializowanych. Poniżej przedstawiono dostępne metody odświeżania i ich charakterystykę:

Technika	Przyrostowe?	Opis
FULL_RECOMPUTE	Nie	Widok w pełni przeliczony od nowa.
NO_OP	N/D	Widok nie zaktualizowany (brak zmian w tabeli bazowej).
ROW_BASED, PARTITION_OVERWRITE, WINDOW_FUNCTION, APPEND_ONLY, GROUP_AGGREGATE, GENERIC_AGGREGATE	Tak	Widok odświeżony przyrostowo, minimalizując zużycie zasobów i czas.

Lakeflow Pipelines vs MLV (Fabric)

W kontekście nowoczesnych architektur lakehouse, Materialized Lake Views (MLV) oraz Lakeflow służą do automatyzacji potoków danych, ale różnią się zakresem i platformą.

Cecha	Lakeflow Pipelines (Databricks)	Microsoft Fabric MLV
Ekosystem	Databricks (Unified Platform)	Microsoft Fabric (SaaS)
Zakres	Cały cykl życia danych (ingest, ETL, orkiestracja)	Pojedynczy obiekt/transformacja
Język	SQL i Python	SQL (głównie)
Główny cel	Skalowalne, profesjonalne potoki danych	Szybka dostawa danych, uproszczenie architektury Medallion
Mechanizm	Automatyczna orkiestracja, przetwarzanie przyrostowe, autoskalowanie, real-time	Deklaratywne podejście – system zarządza odświeżaniem
Typy obiektów	Streaming Tables i Materialized Views	Materialized Lake Views (format Delta w OneLake)
Jakość danych	Wbudowane EXPECT (konfigurowalne reakcje)	Wbudowane monitorowanie (np. CONSTRAINT ON MISMATCH)
CDC / SCD Typ 2	Wbudowana składnia (APPLY CHANGES INTO)	Wymaga ręcznej implementacji
Monitoring	Automatyczny DAG, metryki, logi	Automatyczna wizualizacja lineage, monitoring w Fabric


DEMO