Delta Live Tables (DLT)

1. Introduction to Delta Live Tables

What is DLT?

Delta Live Tables is a **declarative ETL framework** that:

- Simplifies pipeline development with SQL/Python syntax
- Automates dependency management
- Provides built-in data quality monitoring
- Supports both batch and streaming pipelines

Key Benefits

✓ **Automatic Orchestration**: Handles task dependencies
✓ **Data Quality Enforcement**: Expectations and monitoring
✓ **Infrastructure Management**: Auto-scaling and recovery
✓ **Unified Batch/Streaming**: Same code for both paradigms

📌 **Diagram Idea:** DLT Architecture (Source → Transformations → Target with Quality Checks)

---

2. Creating Complete ETL with DLT

Python Example (Full Pipeline)

python

```
# /pipelines/ecommerce_dlt.py
import dlt
from pyspark.sql.functions import *

@dlt.table(
  comment="Raw ecommerce orders",
  table_properties={
    "quality": "bronze"
  }
)
def orders_bronze():
  return (
    spark.readStream
      .format("cloudFiles")
      .option("cloudFiles.format", "json")
      .load("/mnt/raw/orders/")
  )

@dlt.table(
  comment="Cleaned orders with valid amounts",
  spark_conf={
    "spark.sql.shuffle.partitions": 8
  }
)
@dlt.expect("valid_amount", "amount > 0")
def orders_silver():
  return (
    dlt.read_stream("orders_bronze")
      .filter("status = 'COMPLETED'")
      .withColumn("order_date", to_date("timestamp"))
  )
```

```
@dlt.table(
  comment="Daily customer spending",
  partition_cols=["date"]
)
def customer_gold():
  return (
    dlt.read("orders_silver")
      .groupBy("customer_id", "order_date")
      .agg(sum("amount").alias("daily_spend"))
  )
```

SQL Equivalent

sql

```
-- /pipelines/ecommerce_dlt.sql
CREATE OR REFRESH STREAMING LIVE TABLE orders_bronze
COMMENT "Raw ecommerce orders"
TBLPROPERTIES ("quality" = "bronze")
AS SELECT * FROM cloud_files("/mnt/raw/orders/", "json");

CREATE OR REFRESH STREAMING LIVE TABLE orders_silver
COMMENT "Cleaned orders"
CONSTRAINT valid_amount EXPECT (amount > 0)
AS
  SELECT
    order_id,
    customer_id,
    amount,
    to_date(timestamp) as order_date
  FROM STREAM(LIVE.orders_bronze)
  WHERE status = 'COMPLETED';

CREATE OR REFRESH LIVE TABLE customer_gold
PARTITIONED BY (order_date)
AS
  SELECT
    customer_id,
    order_date as date,
    SUM(amount) as daily_spend
  FROM LIVE.orders_silver
  GROUP BY customer_id, order_date;
```

3. Core DLT Operations

Read Operations

python

```
# Batch read
bronze_df = dlt.read("orders_bronze")

# Stream read
streaming_df = dlt.read_stream("clickstream_bronze")
```

Write Modes

python

```
@dlt.table(
  mode="append"  # or "complete", "merge"
)
def incremental_table():
  return df
```

Update/Merge Patterns

python

```
@dlt.table
def scd_type2():
  return (
    dlt.read("updates")
      .merge(
        target = dlt.read("customers"),
        condition = "customers.id = updates.id AND customers.current = true",
        source_alias = "updates",
        target_alias = "customers"
      )
      .whenMatchedUpdate(
        set = {"current": "false", "end_date": "updates.effective_date"}
      )
      .whenNotMatchedInsertAll()
      .execute()
  )
```

---

4. Data Quality with DLT

Expectation Levels

python

```
# Drop invalid rows (strict)
@dlt.expect_or_drop("valid_email", "email LIKE '%@%.%'")

# Record violations but process (soft)
@dlt.expect("positive_amount", "amount >= 0")

# Alert on failure
@dlt.expect_or_fail("no_duplicates", "COUNT(*) = COUNT(DISTINCT id)")
```

Quality Dashboard

python

```
@dlt.table(
  comment="Data quality metrics",
  table_properties={
    "pipelines.autoOptimize.managed": "true"
  }
)
def quality_metrics():
  return dlt.expectations_dfs("orders_silver")
```

DLT UI Shows: Pass/fail rates for all expectations

---

5. Delta Lake Operations in DLT

File Conversion

```
@dlt.table
def convert_parquet_to_delta():
  return (
    spark.read.parquet("/mnt/legacy/parquet_data/")
      .write.format("delta")
      .save("/mnt/delta/converted_data/")
  )
```

Incremental Load

python

```
@dlt.table(
  mode="merge",
  spark_conf={
    "delta.enableChangeDataFeed": "true"
  }
)
def incremental_sales():
  return (
    spark.read.format("delta")
      .option("readChangeFeed", "true")
      .option("startingVersion", 10)
      .load("/mnt/delta/sales/")
  )
```

Time Travel & Versioning

sql

```
-- In SQL DLT
CREATE OR REFRESH LIVE TABLE sales_restored
AS SELECT * FROM delta.time_travel(
  TABLE(LIVE.current_sales),
  TIMESTAMP AS OF '2023-01-01'
);
```

Vacuum Operations

python

```
dlt.vacuum(
  table = "sales",
  retention_hours = 168  # 7 days
)
```

6. Development vs Production

Development Mode

bash

```
# Run with development cluster
databricks pipelines create --name dev-pipeline \
  --development \
  --notebook /pipelines/ecommerce_dlt.py
```

Production Deployment

bash

```
# CI/CD pipeline example
databricks pipelines deploy --path /Pipelines/prod-config.json \
  --settings-json '{
    "target": "production",
    "storage": "dbfs:/pipelines/storage",
    "configuration": {
      "prod_db": "enterprise_data"
    }
  }'
```

Environment Configuration

python

```
# Access environment variables
storage_path = spark.conf.get("pipeline.storage")

# Target-specific logic
if dlt.runtime_env() == "production":
  dlt.set_target_schema("prod_analytics")
else:
  dlt.set_target_schema("dev_analytics")
```

---

7. Pipeline Execution

Run Modes

Mode	**Command**	**Use Case**
Full Refresh	dlt.run(full_refresh=True)	Rebuild all tables
Incremental	dlt.run()	Process new data only
Validate	dlt.run(validate_only=True)	Check syntax without execution

Triggering via API

python

```
import requests

response = requests.post(
  "https://<workspace>/api/2.0/pipelines/<pipeline-id>/updates",
  headers={"Authorization": "Bearer <token>"},
  json={"full_refresh": False}
)
```

Monitoring Runs

python

```
# Get active runs
runs = dlt.list_runs()

# Check latest status
latest = dlt.last_run_status()
print(f"State: {latest.state}, Duration: {latest.duration_seconds}s")
```

---

8. Advanced Patterns

Slowly Changing Dimensions (SCD)

python

```
@dlt.table
def scd_type2_dimension():
  current = dlt.read("current_dimension")
  updates = dlt.read("updates_stream")

  return (
    updates.join(current, "id", "left")
      .select(
        updates["*"],
        when(current["id"].isNull(), lit(True))
          .otherwise(lit(False)).alias("is_new"),
        current["version"].alias("old_version")
      )
      .write.format("delta")
      .option("mergeSchema", "true")
      .mode("overwrite")
      .saveAsTable("scd_type2_output")
  )
```

Change Data Capture (CDC)

python

```
@dlt.table(
  mode="merge",
  partition_cols=["date"]
)
def cdc_target():
  return (
    spark.readStream
      .format("delta")
      .option("readChangeFeed", "true")
      .load("/mnt/delta/cdc_source")
      .withColumn("date", to_date("change_timestamp"))
  )
```

Medallion Architecture

python

```
# Bronze (raw)
@dlt.table
def bronze_orders():
  return spark.read.json("/mnt/raw/orders/")

# Silver (validated)
@dlt.table
@dlt.expect("valid_amount", "amount > 0")
def silver_orders():
  return dlt.read("bronze_orders").filter("status = 'COMPLETED'")

# Gold (aggregated)
@dlt.table
def gold_customer_lifetime():
  return (
    dlt.read("silver_orders")
      .groupBy("customer_id")
      .agg(sum("amount").alias("lifetime_value"))
  )
```

9. Performance Optimization

Z-Ordering

python

```
@dlt.table(
  spark_conf={
    "delta.autoOptimize.optimizeWrite": "true",
    "delta.autoOptimize.zOrderCols": "customer_id,date"
  }
)
def optimized_table():
  return df
```

Auto Compaction

python

```
dlt.optimize(
  table = "large_table",
  z_order_by = ["id", "timestamp"]
)
```

Cluster Configuration

json

```
{
  "clusters": [
    {
      "label": "default",
      "num_workers": 8,
      "node_type_id": "Standard_DS3_v2",
      "spark_conf": {
        "spark.sql.shuffle.partitions": "200"
      }
    }
  ]
}
```

10. Troubleshooting Guide

Issue	Solution
Pipeline fails on startup	Check cluster permissions and dependencies
Data quality violations	Review expectation rules in DLT UI
Merge conflicts	Add OPTIMIZE step before merge operations
Streaming latency	Adjust trigger interval and worker count

11. Complete Cheat Sheet

Python DLT Decorators

python

```
@dlt.table                # Create table
@dlt.view                 # Create view
@dlt.expect               # Soft constraint
@dlt.expect_or_drop       # Strict constraint
@dlt.expect_or_fail       # Hard constraint
```

SQL DLT Keywords

sql

```
CREATE LIVE TABLE         # Batch table
CREATE STREAMING TABLE    # Streaming table
APPLY CHANGES INTO        # CDC operations
CONSTRAINT ... EXPECT     # Data quality
```

---

12. Learning Resources

- Official DLT Documentation
- DLT SQL Reference
- DLT Python API Docs