**Delta Live Tables (DLT) Analysis in ‘dbw-ddm-dev’ Workspace**

Delta Live Tables (DLT) is a framework in Databricks that allows for the declarative definition of data pipelines. It's built on top of Delta Lake and is designed to manage the complexity of data engineering pipelines. DLT emphasizes reliability, quality, and performance for ETL (extract, transform, load) processes. It enables batch and real-time data to be processed efficiently across your Medallion Architecture: Bronze, Silver and Gold layers. See explanation for each layer below.

**Bronze Layer (Raw Layer)**

* *Purpose*: This is the raw data ingestion layer. Data in this layer is often in the same format as it was sourced and has not been transformed or curated. It may include structured, semi-structured, or unstructured data.
* *Characteristics*:
  + Minimal Processing: Data is typically landed with little to no processing. It might be compressed or split into manageable files for storage efficiency.
  + Immutable: To preserve the original state, data is not modified or deleted.
  + High Fidelity: It serves as an exact replica of source systems, useful for audit purposes and as a fallback in case of issues with downstream processing.
  + Usage: For Data Engineers (DE), it's a starting point for ETL processes. Data Analysts and Scientists might access this layer for a very granular, raw view of the data when needed.

**Silver Layer (Curated Layer)**

* *Purpose*: The Silver layer is where data starts to become more usable for a wider audience. It includes cleansed, processed, and enriched data that is still quite granular but typically structured for query efficiency.
* *Characteristics*:
  + Cleaned: Data quality issues are resolved here, and erroneous or irrelevant data is filtered out.
  + Conformed: Data is transformed into consistent formats and structures, often conforming to a data model.
  + Enriched: Additional value is added through the combination of different data sources or the calculation of new metrics.
  + Usage: This layer is more suitable for analysts who need reliable data but don’t require highly aggregated or business-specific views. It's also the foundation for more refined data products.

**Gold Layer (Business Layer):**

* *Purpose*: The Gold layer contains business-level aggregates and metrics that are ready for consumption by business users, stakeholders, and Teams. This data is often the source of truth for Business Intelligence (BI) reporting, dashboards, and analytics.
* *Characteristics*:
  + Refined: Data is highly curated and often aggregated to align closely with business concepts and KPIs.
  + Optimized: It is optimized for performance to facilitate quick queries, often supporting BI tools and data visualization platforms.
  + Quality Assured: The data is of the highest quality, having gone through rigorous testing and validation.
  + Usage: This is the go-to layer for Business Analysts (BAs), decision-makers, and any reporting that will be consumed across the business. It’s intended for those who need clear, accurate, and actionable insights without requiring in-depth technical knowledge of the data's origins or underlying structure.

**Some of the use cases for DLT are the following:**

* DLT is well-suited for scenarios that involve streaming and batch where data is continuously or incrementally ingested into a Delta Lake. You can use DLT for the following:
  + Ingest streaming data from Kafka, Event Hubs, or other streaming sources.
  + Perform real-time data transformations and aggregations.
  + Merge streaming data with existing data sets in a Delta Lake.
  + Simplify complex ETL pipelines by reducing boilerplate code needed for incremental data processing.
  + Implementing slowly changing dimensions (SCD) type patterns to track historical data over time.
  + Full lineage and logging of data transformations and quality checks.

**Is Mckesson’s Domain Environments Setup for DLT?**

Yes, when the Databricks Resident Solution Architect (RSA) analyzed DLT pipelines in ‘**dbw-ddm-dev’** environment there are full working DLT pipelines that move incremental batch data from Kafka through Medallion architecture layers as described above (see image below).

A screenshot of a computer

Description automatically generated

A DLT job cluster which uses a DLT runtime must be created and used to be able to create DLT pipelines, and the ‘dlt’ python library will need to be installed on the DLT cluster as a prerequisite to using it. Here is [custom script](https://dev.azure.com/mckessontech/mtdatamesh/_git/mtdatamesh_databricks?path=/scripts/copy_jobs_create_uc_cluster_attach_to_jobs/create_dlt_cluster_single_user_isolation_mode.py) which can be used to set up a DLT cluster for creating DLT pipelines. Here is an [example notebook](https://adb-3809014642669148.8.azuredatabricks.net/?o=3809014642669148#notebook/3157082625198981/command/3438857241228258) in ‘**dbw-ddm-dev**’ which creates the ‘dlt\_esg\_mobile\_combustion01\_pipeline’ in the screenshot above.

Other considerations in evaluating DLT readiness if all the sources of streaming and batch data are defined, documented, and known. For example, understanding where all the Kafka topics are landing data in different Azure Storage Accounts across Domain Environments is important to know so DLT pipelines know how to source the streaming or batch data from the start.

We also need to know if data quality rules are defined and stored somewhere or if they need to be created. These data quality rules are the input into a DLT pipeline when curated and enriched datasets are created. The data quality (DQ) rules are applied to data as it streams through the DLT pipeline process. DQ quality metrics are captured for each DLT pipeline for later use with audit, performance checks, and business process improvements and validation.

**Delta Live Table Limitations With Unity Catalog (UC)**

If Mckesson plans on using DLT pipelines with Unity Catalog integration it is important to understand the limitations and plan accordingly:

* Existing pipelines that use the Hive metastore cannot be upgraded to use Unity Catalog. To migrate an existing pipeline that writes to Hive metastore, you must create a new pipeline and re-ingest data from the data source(s).
* Init scripts, third-party libraries and JARs are not supported.
* Data manipulation language (DML) queries that modify the schema of a streaming table are not supported.
* A materialized view created in a Delta Live Tables pipeline cannot be used as a streaming source outside of that pipeline, for example, in another pipeline or in a downstream notebook.
* You cannot change the owner of a pipeline that uses Unity Catalog.
* Publishing to schemas that specify a managed storage location is not supported. All tables are stored in the catalog storage location if the target catalog specifies one, otherwise, they are stored in the metastore root storage location.
* The **History** tab in Catalog Explorer does not show history for streaming tables and mateialized views.
* **\*\*You cannot use**[**Delta Sharing**](https://learn.microsoft.com/en-us/azure/databricks/data-sharing/)**with a Delta Live Tables materialized view or streaming table published to Unity Catalog.\*\***
* To query tables created by a Delta Live Tables pipeline, you must use a shared cluster using Databricks Runtime 13.1 and above or a SQL warehouse.
* Delta Live Tables uses a shared cluster to run a Unity Catalog-enabled pipeline.
* **\*\*You cannot use**[**row filters or column masks**](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/row-and-column-filters)**with materialized views or streaming tables published to Unity Catalog.\*\***

**DLT Pipelines Source Data Analysis in dbw-ddm-dev**

**DLT Pipeline Source Data:**

* Landing Zone Source Data:
  + *comdata\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/comdata/
  + *penske\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/penske/
  + *ryder\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/ryder/
  + *fuel\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/sales\_fleet/element\_fuel/
  + *miles\_driven\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/sales\_fleet/element\_miles\_driven/
  + *ontario\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/canada\_ontario/
  + *quebec\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/canada\_quebec/
  + *jetfuel\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/mobile\_combustion/mms\_fleet/jetfuel/
  + rail\_source\_folder: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/business\_travel/rail/
  + air\_source\_folder: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/business\_travel/air/
  + sap\_personal\_vehicle\_source\_folder: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/business\_travel/sap\_personal\_vehicle/
  + *sap\_rental\_car\_source\_folder*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/business\_travel/sap\_rental\_car/
  + *source\_dataset*: /mnt/landingzone\_ddm\_stddmlanding/esg\_data/managed-path/
  + *cfo\_dashboard:* abfss://landingzone@stddmlanding.dfs.core.windows.net/Dev/cfo\_dashboard
* Silver Source Data:
  + silver\_adls\_location\_1: /mnt/silver\_ddm\_stddmlanding/ESG/mobile\_combustion/mms\_sales\_fleet/
  + silver\_adls\_location\_1: /mnt/silver\_ddm\_stddmlanding/ESG/business\_travel/
  + silver\_adls\_location\_1: /mnt/silver\_ddm\_stddmlanding/ESG/capturis/stationary\_combustion
  + silver\_adls\_location\_2: /mnt/silver\_ddm\_stddmlanding/ESG/capturis/purchased\_electricity
  + silver\_adls\_location\_3: /mnt/silver\_ddm\_stddmlanding/ESG/capturis/waste\_generated
  + silver\_adls\_location\_1: /mnt/silver\_ddm\_stddmlanding/ESG/fugitive\_emissions/
  + silver\_adls\_location\_1: /mnt/silver\_ddm\_stddmlanding/ESG/managepath/facility/
  + silver\_adls\_location\_2: /mnt/silver\_ddm\_stddmlanding/ESG/managepath/facility\_usage\_detail/

**DLT Pipeline Owners in dbw-ddm-dev:**

[snyqes3@mckesson.com](mailto:snyqes3@mckesson.com) – Abhishek Sharm (Director of Systems Engineering)  
[smiitvj@mckesson.com](mailto:smiitvj@mckesson.com) – Harikrishna Ongolun (Outside Services Worker)  
[s9h7553@mckesson.com](mailto:s9h7553@mckesson.com) – Jain Praful (No Title)

[sxznwnu@mckesson.com](mailto:sxznwnu@mckesson.com) – Abhishek Shukla (Outside Services Worker)

[s4q1e08@mckesson.com](mailto:s4q1e08@mckesson.com) – Sunil Patidar (No Title)   
[ssgzter@mckesson.com](mailto:ssgzter@mckesson.com) – Raman Parik (Outside Services Worker)

[sh0ysne@mckesson.com](mailto:sh0ysne@mckesson.com) - Kisanu Bhattacharya (Outside Services Worker)  
[smhk8em@mckesson.com](mailto:smhk8em@mckesson.com) - Ahsaanallah Ustad (Outside Services Worker)

[sfnh5dj@mckesson.com](mailto:sfnh5dj@mckesson.com) – Adam Rosa (Outside Services Worker)

**Working DLT Pipeline (All Green):**

Owner: Harikrishna Ongolun (Outside Services Worker)

dlt\_esg\_mobile\_combustion01\_pipeline: <https://adb-3809014642669148.8.azuredatabricks.net/?o=3809014642669148#joblist/pipelines/457edc45-6c55-4f39-9163-cf3bdb6465b5/updates/39f26d80-5ab0-4bca-afbb-39f09fb71b41>

A screenshot of a computer

Description automatically generated

Associated Notebook For Creating DLT Pipeline: <https://adb-3809014642669148.8.azuredatabricks.net/?o=3809014642669148#notebook/3157082625198981/command/3438857241228258>

**Reusable DLT Data Quality Components / Functions:**

- [**dlt\_dq\_generic\_utility.py**](https://adb-3809014642669148.8.azuredatabricks.net/?o=3809014642669148#files/1591895553370061): It appears that this utility is creating a JDBC connection to an Azure SQL DB and reading data quality rules which are then applied to the DLT transformation steps.

**Create a Streaming Bronze table Using Delta Live Tables in ‘dbw-ddm-dev’:**

A screenshot of a computer program

Description automatically generated

In the code above a DLT table is defined with the **@dlt.table** decorator. This table is tagged as a "bronze" quality table, which means it's a raw or slightly processed data storage level in a multi-tiered data lake architecture. The **create\_steaming\_bronze\_table** function uses Spark's structured streaming with the Auto Loader to read incoming CSV files from a specified cloud storage folder (comdata\_source\_folder). The **.select("\*", "\_metadata.file\_path")** statement selects all the columns from the ingested data and adds an additional column that captures the file path from where each record is sourced. This is often used for:

* Data governance to track where the data is coming from.
* Debugging and monitoring to quickly identify and address any issues with specific source files.
* Incremental processing to understand which files have been processed, especially if you need to reprocess data or handle late-arriving files.

The **.withColumn("load\_timestamp", current\_timestamp())** statement adds another column that captures the timestamp of when the data was loaded. This is useful for time-based data analysis and can also be valuable for tracking and monitoring purposes.

Here is a sample out-of-the-box Databricks DLT pipeline using customer and sales data: <https://adb-3809014642669148.8.azuredatabricks.net/?o=3809014642669148#joblist/pipelines/9ee35786-dc71-4e69-80ce-1d829f4c4144/updates/5f5dd29e-0e13-407c-aa35-f59354e4d15c>

**DLT Reusable Framework**

The [DLT reusable framework](https://dev.azure.com/mckessontech/mtdatamesh/_git/mtdatamesh_databricks?path=/scripts/create_dlt_pipelines) introduces the concept of parameterization and reusability into DLT pipeline creation. The **create\_dlt\_pipelines** notebook uses multiple python tuples to create bronze, silver, and gold layer streaming DLT tables.

**Bronze DLT Tables Reusable Components / Functions:**

A ‘bronze\_table\_source\_data’ Python tuple contains ‘bronze\_table\_name’, ‘source\_data\_path’, and ‘source\_data\_type’. These three parameters are all that is needed to create the first group of bronze tables in your DLT pipeline.

**A screen shot of a computer code

Description automatically generated**

A create\_stream() generic function has been created to take these tuple inputs and create the appropriate read\_stream() method based on ‘source\_data\_type’ in the python tuple.

A computer screen shot of a program code

Description automatically generated

A create\_bronze\_dlt\_streaming() generic function has been created to take these tuple inputs and create multiple bronze DLT streaming table with duplicate columns removed, and column names cleaned.

A screenshot of a computer program

Description automatically generated

We iterate through the python tuple to create each DLT bronze table. We do NOT infer column types so each of the bronze table columns is of StringType().

A screen shot of a computer code

Description automatically generated

**DLT SCD Type 1 and SCD Type 2**

**Slowly Changing Dimension (SCD) Type 1** involves **overwriting existing records with new data, thereby NOT maintaining any historical changes**. DLT supports SCD Type 1 by allowing for the easy updating of existing records in a Delta table. When new data arrives, DLT can be configured to overwrite the existing records with the new information, reflecting the most current state of the data. DLT also supports the ‘merge’ operation can be used to insert new records or update existing records based on a match condition. For SCD Type 1, the match condition typically checks for existing records, and if found, the existing records are updated with new values. This approach is easy to implement because it always reflects the most recent or current values.

**Slowly Changing Dimension (SCD) Type** **2** involves **preserving historical data and does NOT overwrite existing data**. SCD Type 2 is not recommended for tables which are subject to new schema / structure changes since it’s an expensive database operation to modify all rows. DLT supports SCD Type 2 by allowing for easy inserts of all new data. DLT also supports the ‘merge’ operation, and when new data arrives DLT can be configured to append all records with the new information without updating any existing data and mark old records at outdated using a **version** / **effective date** / **expiry date** columns to track the data record validity period. DLT also supports the use of surrogate keys to uniquely identify each version of a record. This helps in distinguishing between different historical versions of the same entity.

**A screenshot of a computer

Description automatically generated**

**DLT Fact & Dimension Tables**

**Fact** **tables** store quantitative or measurement data. They contain numerical values that are the result of transactions or events based on a business workflow processes. They typically contain foreign keys that relate to primary keys in the **dimension** tables. The numeric data in **fact** tables are used for custom calculations and aggregations. The granularity of numerical data directly affects the size and scope of the **fact** table.

**Dimension** **tables** store descriptive / metadata attributes or information about the dimensions of the business.

They provide context to the data captured in **fact** tables. **Dimension** tables typically have fewer rows than **fact** tables, have many textual or descriptive attributes columns and contain primary keys that are referenced by **fact** tables.

From a **DLT perspective** your data (e.g. fact / dimension) needs to have a well-defined schemas, data-types and primary and foreign key relationships. Fact and dimension tables should be designed based on common query patterns such as denormalization and pre-aggregation for faster query performance. This might include de-normalizing some dimension tables or pre-aggregating data in fact tables for faster query performance. Consider the use of time-based *partitions* for large DLT datasets to reduce cost by scanning relevant query partitions only. Use **Autoloader** for incremental loading strategies for fact tables in DLT pipelines to reduce pipeline run-time and resource cost. Implement data quality and business rule logic for data completeness, accuracy, consistency, and completeness across your fact / dimension DLT tables. DLT supports strategies for slowly changing dimensions (SCDs). Utilize DLT-specific features like Auto Optimize for automatic file management and Z-Ordering for optimizing data layout to enhance query performance on your fact and dimension tables.

**Important Links**

* [Getting Started With Delta Live Table (DLT) in Python](https://learn.microsoft.com/en-us/azure/databricks/_extras/notebooks/source/dlt-wikipedia-python.html)
* [Getting Started with Delta Live Tables (DLT) Using Databricks SQL](https://learn.microsoft.com/en-us/azure/databricks/_extras/notebooks/source/dlt-wikipedia-sql.html)
* [What is Delta Live Tables in Azure Databricks](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/)
* [Run a Delta Live Table Pipeline with Apache Airflow or Azure Data Factory](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/workflows)
* [\*\*Limitations When Using Unity Catalog with Delta Live Tables](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/unity-catalog#--limitations)