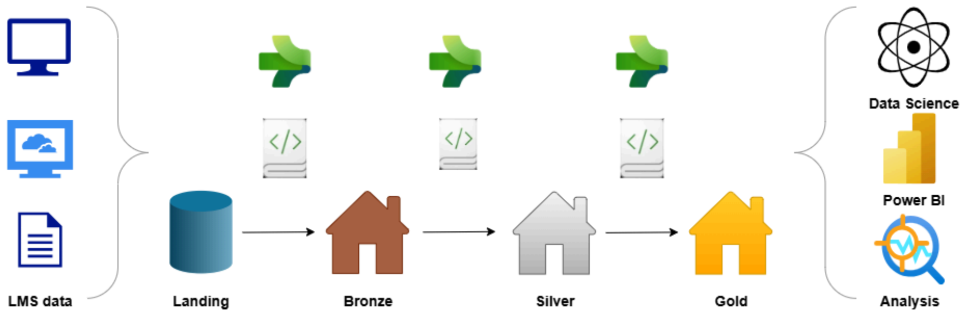
Fabric Project Architecture




# Microsoft Fabric LMS Lakehouse – Incremental Medallion Pipeline

| | |
|---|---|
| Tech Stack | Microsoft Fabric · Power BI Desktop & Service · Python · Spark/SQL |
| Brief Summary | **Implemented an incremental Medallion lakehouse in Microsoft Fabric** for daily LMS data: ADLS Gen2 **Lan** **Gold** facts/dimensions + semantic model published to **Power BI**→ **Result:** analytics-ready data powering in |
| Link | https://github.com/khanhmdinh/khanhmdinh.github.io/tree/main/02_Microsoft%20Fabric%20LMS%20Lake |

⭐ **Navigation bar**

# Table of Contents

# Summary


### Scope of Work

- Data Understanding & Profiling
- Data Cleaning & Transformation
- Data Modeling (star schema: **Fact** tables for Enrollments & Assessments; **Dimensions** for Student, Course, Date, Device/Access)
- KPI definition and validation
- Reporting & Visualization

### Deliverables

- Concise data dictionary & data standards
- Reproducible cleaning/transform pipeline (SQL/Python)
- Logical & physical data model diagrams
- Interactive dashboard (e.g., Power BI) with Overview, Course Performance, Engagement, Cohorts/Retention
- Insight report with prioritized actions to improve course design and learner success

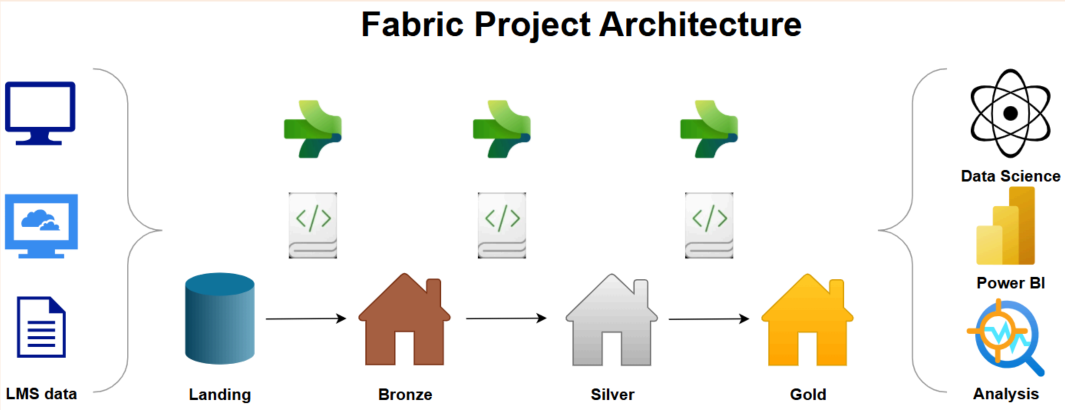
# Data Assessment

## ▼ Dataset Information

LMS Dataset of Students

| No | Column Name | Description |
|---|---|---|
| 1 | Student_ID | Unique identifier for each student. |
| 2 | Name | Student's full name. |
| 3 | Age | Student's age. |
| 4 | Gender | Student's gender (M/F). |
| 5 | Grade_Level | Student's current grade level. |
| 6 | Course_ID | Unique identifier for each course. |
| 7 | Course_Name | Name of the course. |
| 8 | Enrollment_Date | Date the student enrolled in the course. |
| 9 | Completion_Date | Date the student completed the course. |
| 10 | Status | Current status of the student in the course (In Progress, Completed). |
| 11 | Final_Grade | Final grade obtained by the student in the course. |
| 12 | Attendance_Rate | Percentage of classes attended by the student. |
| 13 | Time_Spent_on_Course (hrs) | Total hours spent by the student on the course. |
| 14 | Assignments_Completed | Number of assignments completed by the student. |
| 15 | Quizzes_Completed | Number of quizzes completed by the student. |
| 16 | Forum_Posts | Number of forum posts made by the student. |
| 17 | Messages_Sent | Number of messages sent by the student. |
| 18 | Quiz_Average_Score | Average score of all quizzes taken by the student. |
| 19 | Assignment_Scores | Assignment scores by students |
| 20 | Assignment_Average_Score | Average score of all assignments completed by the student. |
| 21 | Project_Score | Score of the final project completed by the student. |
| 22 | Extra_Credit | Extra credit points earned by the student. |
| 23 | Overall_Performance | Overall performance score considering all aspects of the course. |
| 24 | Feedback_Score | Average feedback score provided by the student for the course |
| 25 | Parent_Involvement | Level of parent involvement in the student's education (e.g., High, Medium, Low). |
| 26 | Demographic_Group | Demographic group the student belongs to (e.g., Urban, Suburban,<br>Rural). |
| 27 | Internet_Access | Whether the student has access to the internet at home (Yes/No). |
| 28 | Learning_Disabilities | Any learning disabilities the student may have. |
| 29 | Preferred_Learning_Style | Student's preferred learning style (e.g., Visual, Auditory, Kinesthetic). |
| 30 | Language_Proficiency | Proficiency level in the language of instruction (e.g., Beginner,<br>Intermediate, Advanced). |
| 31 | Participation_Rate | Percentage of active participation in class activities. |

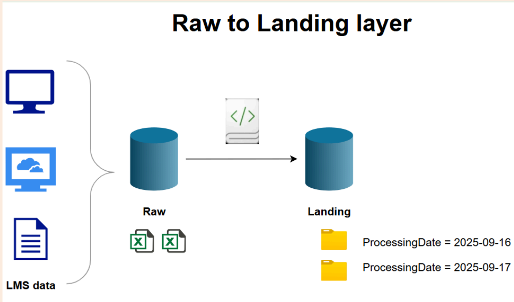
# I. The Fabric Project Architecture



# II. Getting the Data from Raw to Landing Zone

## 1. Context & Assumptions

- **Source**: In production this dataset would arrive from a website via API. In this project, the inbound feed is **simulated** by manually uploading one CSV per day into **Raw**.
- **Storage: Azure Data Lake Storage Gen2 (ADLS Gen2)** with a container (e.g., `fabric-project`) and two folders: `raw/` and `landing/`.
- **Daily file**: Exactly **one** file per day, containing all students enrolled on that day.
  - Naming convention (example): `LMS_YYYYMMDD.csv` (e.g., `LMS_20250917.csv`).
  - Files are uploaded to `raw/lms/`.




## 2. Overview

This mini-pipeline ingests daily CSV drops from **Raw** storage into a curated **Landing** zone in Azure Data Lake Storage Gen2 (ABFS). It performs a light validation, tags each record with a `processing_date`, and writes to Landing **as-is** using **append** semantics, partitioned by `processing_date`.

> **Why**: Landing acts as a faithful copy of incoming data for downstream bronze/raw consumption, auditing, and recovery. No de-duplication or business transformations occur here.

## 3. High-Level Flow

1. **Resolve the input path** (today: static path; future: dynamically detect the *latest* drop).
2. **Read CSV** with `header=true` and `inferSchema=true` into a Spark DataFrame.
3. **Validate non-empty data**: if `df.count() > 1` → proceed; else → log and skip.
4. **Tag with processing date**: add column `processing_date` (today: hard-coded for demo; future: supplied by the orchestrating pipeline/run date).
5. **Write to Landing** as CSV with `header=true`, **partitioned by** `processing_date`, `mode=append`.
6. **Emit log messages** for visibility (e.g., *"File has data"*, *"Data written to the landing zone successfully"*).

## 4. Storage Layout & Paths

### ABFS URI pattern

```
abfss://{container}@{storageAccount}.dfs.core.windows.net/{relativePath}
```

- **Raw example:** `abfss://<fabric-project>@<storage>.dfs.core.windows.net/raw/LMS/2023/09/01/`
- **Landing root:** `abfss://<fabric-project>@<storage>.dfs.core.windows.net/landing/`

### Landing partitioning example

```
landing/
└── processing_date=2024-09-17/
└── part-00000-...csv
```

## 5. Processing Steps

▼ **Step 1: Source resolution (Resolve account, container, and `raw/lms/` path; Build the `abfss://` URL)**

**Fig. X. Building the ADLS Gen2 source path for the Raw layer**

Variables: `account_name`, `container_name`, `relative_path='raw'` → `abfss://{container}@{account}.dfs.core.windows.net/raw`.

```
account_name = '<account>' #fill in your primary account name
container_name = 'fabricproject' #fill in your container name
relative_path = 'raw/' #fill in your relative folder path

adls_path = 'abfss://%s@%s.dfs.core.windows.net/%s' % (container_name, account_name, relative_path)

print('Source storage account path is', adls_path)
```

> **Output:** Source storage account path is abfss://{container}@{account}.dfs.core.windows.net/raw

### ▼ Step 2: Latest file detection (Preparation for reading CSV)

- **Setting `today_file` and `processed_date`**

```
today_file = 'LMS_09-01-2023.csv' #fill in the file name in folder LMS
processed_Date = 'YYYY-MM-DD' #fill in today
```

- **Reading CSV File of today**

```
latest_path = f"{adls_path}/{today_file}"
df = spark.read.csv(path= latest_path, header=True, inferSchema= True)
```

> **Output:** abfss://{container}@{account}.dfs.core.windows.net/raw/LMS_09-01-2023.csv

### ▼ Step 3: Read CSV & Write Landing

#### Validation

- **Header-only files:** If the file contains only the header (no data rows), the job **skips writing** and logs: *"The file only has the header row but no data."*
- **Row-count threshold:** Uses `df.count() > 1` to treat the input as valid.

#### Transformations

- Add `processing_date` using a literal value (demo):
  - **Today:** hard-coded literal (imported via Spark SQL `lit`), e.g., `F.lit("2024-09-17")`.
  - **Future:** parameter from the orchestration pipeline (run date), not hard-coded.
- No schema enforcement or business rules at Landing; data is preserved **as received**.

#### Write Strategy

- **Format:** CSV
- **Header:** `true`
- **Partitioning:** `partitionBy("processing_date")`
- **Mode:** `append` (allows duplicates by design at Landing)

> Landing is intentionally append-only and immutable. De-duplication, upserts, or slowly changing logic belong in later zones (e.g., Bronze/Silver).

#### PySpark

```
from pyspark.sql.functions import lit

# Read CSV
latest_path = f"{adls_path}/{today_file}"
df = spark.read.csv(path= latest_path, header=True, inferSchema= True)
```

```
# Validate data (exclude header-only files)
if df.count() >1:
    print("The file has data.")

# Tag with processing_date
    df_new = df.withColumn("Processing_Date", lit(processed_Date))
# Write to Landing
    df_new.write.format('csv').option('header','true').partitionBy('Processing_Date').mode('append').save('abfss://fabricproject@khanhmdinh.dfs.core.windows.net/landing/')
    print ('Data written to landing zone successfully !')
else:
    print('This file contains only header row and no data.')
```

# III. Automatically ingest from Raw to Landing Zone using Pipeline

## 1. Objective

Automate the **Raw → Landing** ingestion so it becomes **data-driven**: detect the current day's file from the Raw folder, pass parameters into a notebook, and write to Landing partitioned by `processing_date`. This guide reflects the exact steps from the transcript and adds production notes.

## 2. Project Approach

> Adopt a **hybrid of Partition-based and Batch Window Ingestion** to move data across layers (**Raw → Landing → Bronze/Silver → Gold**). Rationale:

- Partitions give natural slicing (e.g., daily drops).
- Windows align with the scheduled arrival pattern.
- Combination allows controlled re-runs and reduces duplicate risk.

💡 **Batch Window Ingestion**

**How it works**: Ingest all data within a **time window** (hourly/daily/weekly). Define `[window_start, window_end]` and fetch rows/files within this interval.

- **Supported sources: Relational, non-relational/NoSQL, and file systems.**
- **Control/metadata**: Track executed windows and re-runnable boundaries.
- **Pros**: Operationally straightforward; aligns with scheduled drops.
- **Cons**: Can fetch unchanged data; may need deduping across windows.

💡 **Partition-based Ingestion**

**How it works**: Ingest data based on **partitions** (e.g., by **date**, **region**) as the unit of increment. Useful when data is already **partitioned at source** or in the lake.

- **Supported sources**: Broadly applicable — **relational**, **NoSQL**, **file systems**, and **data warehouses**.
- **Control/metadata**: Track which partitions are **new/updated**; note that partitions can contain **repeated rows**.
- **Pros**: Scales well; maps cleanly to lake layouts.
- **Cons**: May include duplicates or unchanged records inside a partition; often needs a second technique to filter.

## 3. Fabric Notebook Preparation

Replace any hard-coded values with **parameters** (placeholders) that the pipeline will **overwrite at runtime**:

- **Parameters**:
  - `today_file` (string): file name to ingest from Raw (e.g., `LMS_09_01_2023.csv`).
  - `processing_date` (string): date stamp for the target partition (e.g., `2024-09-17` or `yyyyMMdd`).
- **Notebook logic**
  - Read CSV from **Raw** using `today_file`.
  - Validate **not header-only** (e.g., `df.count() > 1`).

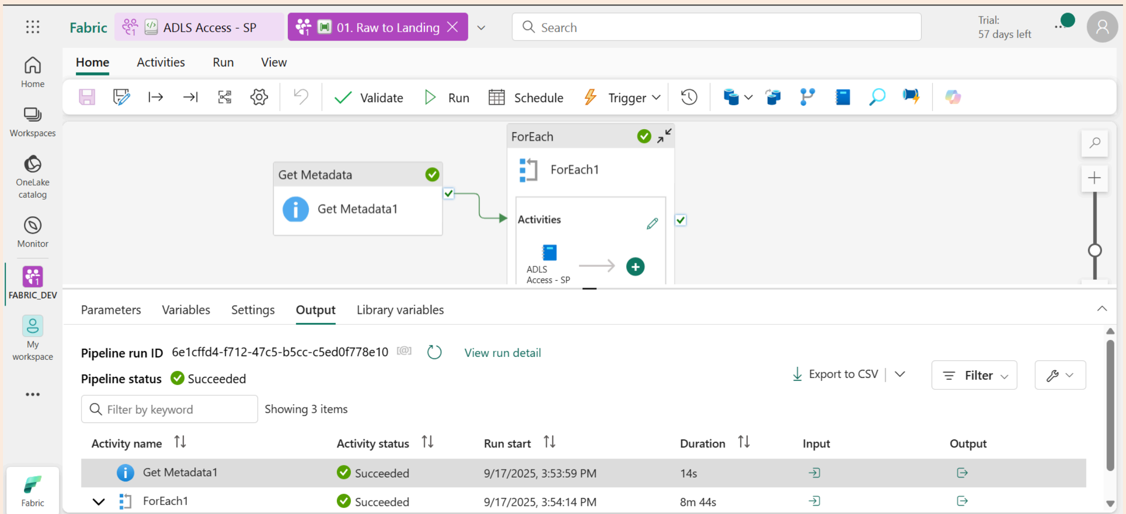
- Add `processing_date` column (literal from parameter).
- Write to **Landing** as CSV, `header=true`, `mode=append`, `partitionBy('processing_date')`.

## 4. Fabric Connection (ADLS Gen2)

Create a Linked Service–style **Connection** the pipeline can reuse:

1. Open **Manage connections & gateways** (gear icon) → **New connection** → **Cloud**.
2. **Name**: `raw`. **Type**: **Azure Data Lake Storage Gen2**.
3. **Server name (DFS endpoint)**: `https://<storage-account>.dfs.core.windows.net`
4. **Full path**: the **container** name (e.g., `fabric-project`).
5. **Auth method**: For the demo, **OAuth 2.0**. For production, prefer **Service Principal** with least-privilege RBAC.
6. **Edit credential** → authenticate (demo user had rights) → **Create**.

## 5. Build the Data Pipeline

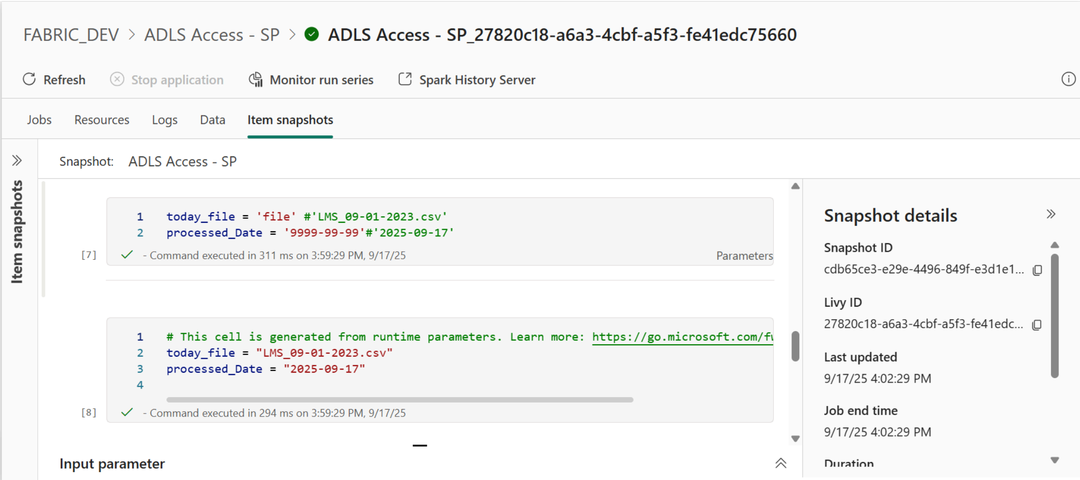


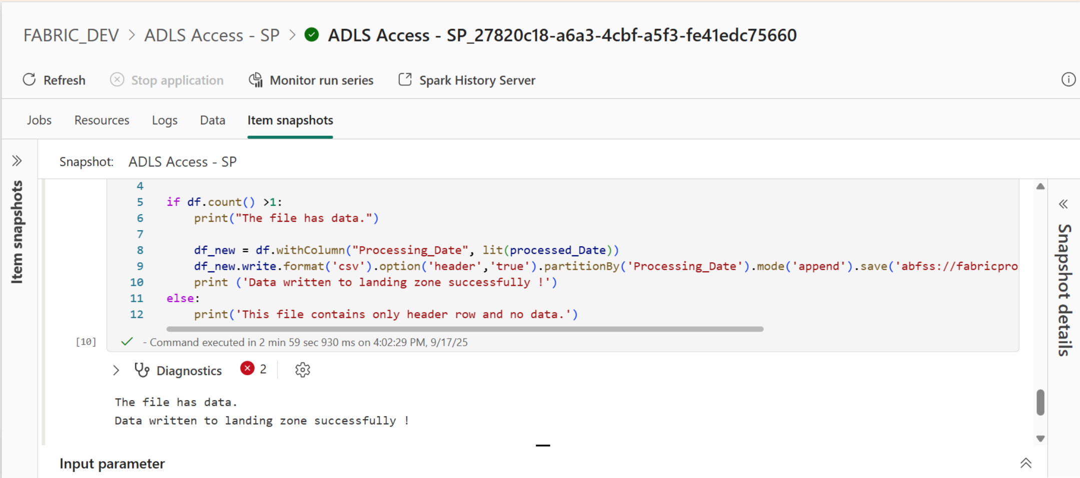

## 6. Results

- **Get Metadata** output contains today's file (e.g., `LMS_09_01_2023.csv`).

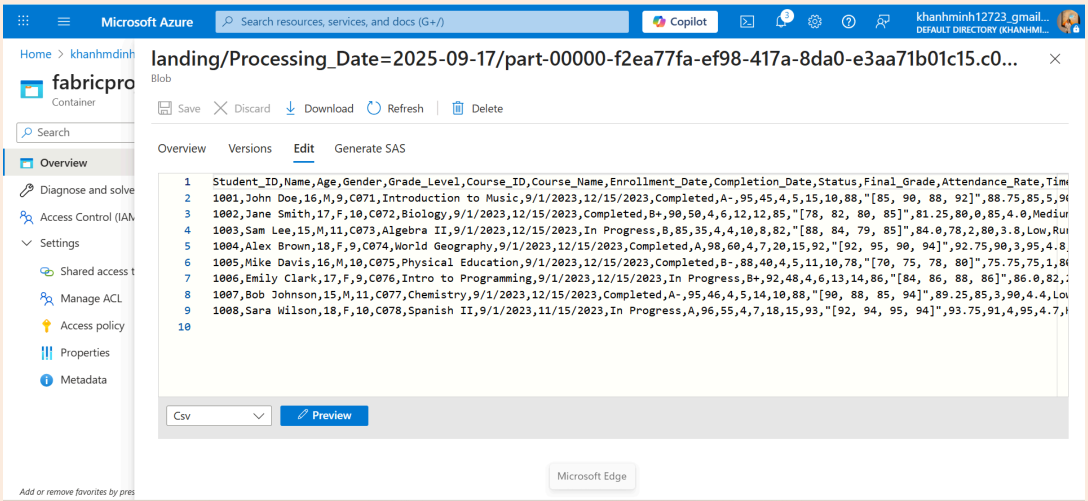
```
{
  "childItems": [
    {
      "name": "LMS_09-01-2023.csv",
      "type": "File"
    }
  ],
  "executionDuration": 1
}
```

- **ForEach** → **Notebook** succeeds; **snapshot** shows `today_file` and `processing_date` **injected at runtime**.



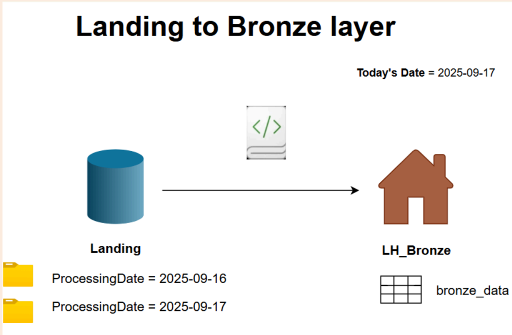

- **Landing** has a new partition path



# IV. Landing to Bronze Layer Incremental Load

## 1. Objective

Move daily data from **Landing** (ADLS Gen2) into a **Bronze** table in a Fabric **Lakehouse** (e.g., `LH_bronze`), **incrementally** and **safely** (no duplicate business keys). The process ingests only the **current day's partition** from Landing and **upserts** into Bronze.




## 2. Source & Target

- **Source**: ADLS Gen2 `landing/processing_date=YYYY-MM-DD/` (daily partitioned CSVs written by Raw→Landing pipeline).
- **Target**: Fabric Lakehouse `LH_bronze`, table `bronze_<entity>` (Delta-backed table).

## 3. Incremental Approach

1. **Select today's partition**: Compare `processing_date` to **today** (e.g., `2024-09-17`). Ingest only files under `landing/processing_date=<today>/`.
2. **Load to staging**: Read CSV(s) into a **staging DataFrame** (`df_stage`). Apply light validation (non-empty, basic schema checks).
3. **Upsert to Bronze**: Use a **business key** (e.g., `student_id + course_id`) to perform **MERGE** (SCD-Type-1 semantics):
   - **Insert** rows that are **new**.
   - **Update** rows that **exist** but have changed attributes (e.g., last name, address).
4. **Idempotency**: Re-running the same date will not create duplicates; existing rows will be overwritten consistently.

## 4. Why Upsert?

Landing may contain legit repeats of earlier keys across days (e.g., a student updates profile on `2024-09-17` after enrolling on `2024-09-16`). A blind append would duplicate `(student_id, course_id)`. **Upsert** keeps one canonical row per key while allowing attributes to change over time (Type-1 overwrite).

## 5. Table Design (Bronze)

- **Storage**: Delta table in `LH_bronze`.
- **Schema**: Mirrors Landing columns + **ingestion metadata**.
- **Business key**: Composite, e.g., `student_id`, `course_id` (adjust per entity).
- **Metadata columns**:
  - `processing_date` (from Landing partition)
  - `ingest_ts` (current timestamp)
  - optional `source_file`, `run_id`

## 6. PySpark

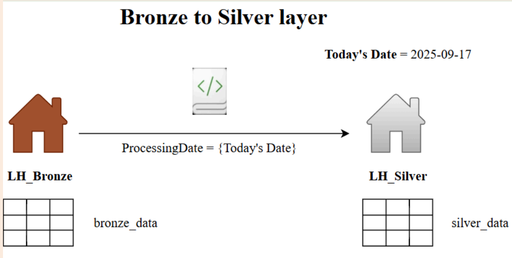
```
# 1) Read today's partition
account_name = '{storageAccount}' # fill in your primary account name
container_name = '{container}' # fill in your container name
relative_path = '{landing}' # fill in your relative folder path
```

```
adls_path = 'abfss://%s@%s.dfs.core.windows.net/%s' % (container_name, account_name, relative_
path)
partition_path = f"/Processing_Date={today_date}/"

complete_path = adls_path + partition_path
print('Source storage account complete path is ', complete_path)

# 2) Materialize staging view
df = spark.read.format('csv').option('header','true').schema(schema).load(complete_path)
print("Reading data of : ", partition_path)

df_stage.createOrReplaceTempView('new_data') #this will be holding all the data that is coming from
dataframe

# 3) Create Empty Bronze Table if missing (Delta)
fabric_bronze_path = f"abfss://{workspace}@onelake.dfs.fabric.microsoft.com/LH_Bronze.Lakehous
e/Tables/bronze_data"

try:
  spark.read.format('delta').load(fabric_bronze_path).createOrReplaceTempView('bronze_data')
except:
  create_table = f"""CREATE TABLE IF NOT EXISTS bronze_data ({data})""" # including all the colum
n data in the processing_date
spark.sql(create_table)
spark.read.format('delta').load(fabric_bronze_path).createOrReplaceTempView('bronze_data')

# 4) UPSERT logic for inserting / updating data into bronze table
sql_statement = f""" MERGE INTO bronze_data AS target
          USING new_data AS source
          ON target.Student_ID = source.Student_ID  AND target.Course_ID = source.Course_ID

          WHEN MATCHED THEN
            UPDATE SET
            {data} # update all the columns in the processing_date
          WHEN NOT MATCHED THEN
          INSERT({data}) # including all the columns in the processing_date
          """
spark.sql(sql_statement).show()
```

# V. Bronze to Silver Layer: Incremental Data Quality & Business Transformations

## 1. Objective

Promote data from **Bronze (landing/raw)** to **Silver (cleaned/conformed)** using an **incremental filter** on `processing_date` (today's run), apply **data cleaning** to ensure quality, perform **minimal business transformations** for analytics readiness, and **upsert** into Silver with an idempotent MERGE keyed by `(student_id, course_id)`.




## 2. Incremental Read

- Read only rows with `processing_date = run_date` (today).
- Avoid reprocessing the full Bronze table; handle only new/changed rows.

## 3. Data Cleaning

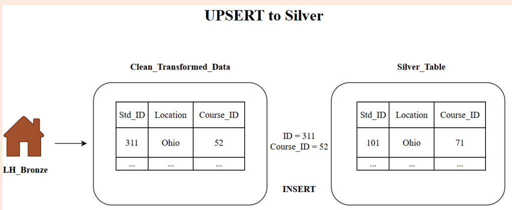
- **Duplicates:** De-duplicate per business key (latest record wins). Even though upstream Bronze uses upsert, re-check at Silver for defense-in-depth.
- **Missing/Nulls:**
  - **Critical columns** (e.g., `student_id`, `enrollment_date`, `course_id`): **drop** rows if null.
  - **Non-critical columns: impute defaults** (e.g., pending course status → `InProgress`).
- **Date standardization:** Normalize all date columns to a consistent format (e.g., `YYYY-MM-DD`) to prevent downstream ambiguity.
- **Logical consistency:** Enforce domain rules (e.g., `completion_date > enrollment_date` if a course is completed).

## 4. Business Transformations

| Column Name | Logic |
| --- | --- |
| `completion_time_days` | `completion_date - enrollment_date` |
| `performance_score` | **(Quiz_Average_Score * 0.2 + (Assignment_Average_Score) * 0.2 + (Project_Score) * 0.1** |
| `course_completion_rate` | `OnTime` **if complete_time_days ≤ 90 days, else** `Delayed` |

## 5. UPSERT to Silver

- **Target:** `silver.lms_enrollments` (Delta).
- **Match keys:** `(student_id, course_id)`.
- **Behavior:**
  - **INSERT** new pairs not present in Silver.
  - **UPDATE** existing pairs when incoming cleaned data differs (optional row-hash or column-level compare).




- **Illustrative flow:** If `student_id=311, course_id=52` moves from **Ohio** to **London**, Bronze already reflects the change and updates `processing_date=today`. The incremental read picks it up, cleaning/transforms run, and MERGE **updates** the existing Silver row.

## 6. Data Cleaning (Silver Layer)

1. **De-duplication**

- Applied `dropDuplicates(...)` on business keys to guarantee unique records even if upstream already upserts.

2. **Missing / null handling**
   - **Critical columns:** `student_id`, `course_id`, `enrollment_date` → **drop rows** if null.
   - **Non-critical columns:** sensible defaults via `fillna({ age: 0, gender: 'Unknown', status: 'InProgress', final_grade: 'NA' })`.
3. **Date standardization**
   - Converted string dates (e.g., `M d yyyy`) to proper dates using `to_date(...)` and stored in `yyyy-MM-dd`.
   - Columns standardized: `enrollment_date`, `completion_date`.
4. **Logical consistency**
   - Enforced domain rule: if a course is completed, **`completion_date > enrollment_date`**.
5. **Quality outputs**
   - Produced a single, trusted DataFrame **`df_consistent`** (today's slice, clean and consistent) for downstream business transforms and upsert.

```
from pyspark.sql.functions import to_date, col
from pyspark.sql import functions as F

# Parameters (passed by pipeline in prod)
run_date   = spark.conf.get("p_run_date", "2025-09-17")
bronze_tbl = "bronze.lms_enrollments"   # example logical name

# 1) Incremental read (today only)
df_today = (spark.table(bronze_tbl)
            .where(col("processing_date") == lit(run_date)))

# 2) Deduplicate (defense-in-depth)
df_no_dups = df_today.dropDuplicates(["student_id","course_id","enrollment_date"])

# 3) Critical nulls → drop
df_critical = df_no_dups.dropna(subset=["student_id","course_id","enrollment_date"])

# 4) Non-critical nulls → defaults
df_filled = df_critical.fillna({
    "age": 0, "gender": "Unknown", "status": "InProgress", "final_grade": "NA"
})

# 5) Date standardization
df_dates = (df_filled
    .withColumn("enrollment_date",  to_date(col("enrollment_date"),  "M d yyyy"))
    .withColumn("completion_date",  to_date(col("completion_date"),  "M d yyyy")))

# 6) Logical consistency: completion > enrollment (or still in progress)
df_consistent = df_dates.filter(
    col("completion_date").isNull() | (col("completion_date") > col("enrollment_date"))
)

# Ready for business transforms + upsert to Silver
df_consistent.createOrReplaceTempView("v_cleaned_silver")
```

## 7. Data Transformation (Silver Layer)

```
# logic = completion_time_days = completion_date - enrollment_date
# we are subtracting completion_time_days
```

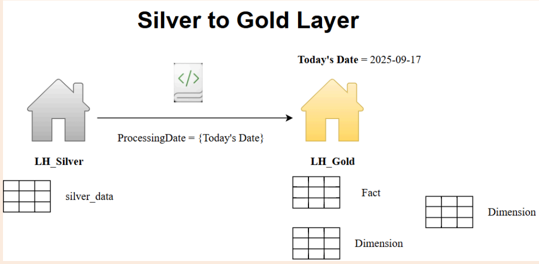
```
# we are converting that to integer
from pyspark.sql.functions import col

df_completion = (df_consistent
    .withColumn("completion_time_days", (col("completion_date") - col("enrollment_date")).cast("int"))
    .withColumn("performance_score", 0.30*col("quiz_avg") + 0.40*col("assignment_avg") + 0.30*col
("project_score"))
    .withColumn("course_completion_rate", when(col("completion_time_days") <= 90, F.lit("OnTime")).
otherwise(F.lit("Delayed")))
)

df_completion.createOrReplaceTempView("new_data")
```

```
-- Create the Silver table once (empty schema that matches our view)
CREATE TABLE IF NOT EXISTS silver.lms_enrollments
USING DELTA
AS SELECT * FROM new_data WHERE 1=0;

-- Convenience view on the target
CREATE OR REPLACE TEMP VIEW silver_data AS
SELECT * FROM silver.lms_enrollments;

-- Idempotent upsert (keys: student_id + course_id)
MERGE INTO silver.lms_enrollments AS tgt
USING new_data AS src
ON  tgt.student_id = src.student_id
AND tgt.course_id  = src.course_id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *;
```

# VI. Gold Layer: Star Schema (Facts & Dimensions) and Incremental Loads

## 1. Objective

Model a **star schema** in the **Gold Lakehouse** and load it incrementally from Silver. Implement dimension tables (`dim_student`, `dim_course`) and a fact table (`fact_enrollments`) with upsert logic so the model is analytics-ready for semantic modeling/Power BI.




## 2. Dimensional Design

- `dim_student` (business/natural key: `student_id`): `student_id`, `student_name`, `age`, `gender`, `demographic_group`, `preferred_language`, `learning_style`, `language_proficiency`, `parent_involvement`, …
- `dim_course` (business/natural key: `course_id`): `course_id`, `course_name`, `grade_level`, …
- `fact_enrollments` (FKs: `student_id`, `course_id`): `enrollment_date`, `completion_date`, `completion_time_days`, `performance_score`, `course_completion_rate`, `status`, `final_grade`, `processing_date` (lineage), …

## 3. Incremental Load & Upsert Strategy

- **Read scope:** From Silver, **filter by** `processing_date = run_date` to avoid full scans.
- **De-duplication in dimensions:** Apply `dropDuplicates(['student_id'])` and `dropDuplicates(['course_id'])` before merging to preserve **many-to-one** cardinality (Fact→Dim).
- **MERGE contracts:**
  - `dim_student`: match on `student_id` → **UPDATE** non-key attributes; **INSERT** if not matched.
  - `dim_course`: match on `course_id` → **UPDATE/INSERT** as above.
  - `fact_enrollments`: match on composite `(student_id, course_id, enrollment_date)` (or your chosen business key) → **UPDATE** changed rows; **INSERT** new rows.
- **Metrics:** After each MERGE, retrieve **Delta operation metrics** (inserted/updated/deleted) via table history for observability.

## 4. PySpark

```
# Read today's slice from Silver
from pyspark.sql.functions import col, lit, datediff, when
run_date = spark.conf.get("p_run_date")  # e.g., '2024-09-17'
silver_tbl = "silver.lms_enrollments"

df_today = (spark.table(silver_tbl)
          .where(col("processing_date") == lit(run_date)))

# Minimal transforms for fact (already created in Silver step)
df_fact = (df_today
  .withColumn("completion_time_days",
           datediff(col("completion_date"), col("enrollment_date")))
  .withColumn("course_completion_rate",
           when(col("completion_time_days") <= 90, "OnTime").otherwise("Delayed")))

# Dimension extracts (defense-in-depth de-dup)
df_dim_student = (df_today
```

```
 .select("student_id","student_name","age","gender","demographic_group",
         "preferred_language","learning_style","language_proficiency","parent_involvement")
 .dropDuplicates(["student_id"]))

df_dim_course  = (df_today
 .select("course_id","course_name","grade_level")
 .dropDuplicates(["course_id"]))
```

```
# Upsert dims
merge_upsert(
 "gold.dim_student",
 "t.student_id = s.student_id",
 df_dim_student,
 update_map={c: f"s.{c}" for c in df_dim_student.columns if c != "student_id"},
 insert_cols=df_dim_student.columns
)

merge_upsert(
 "gold.dim_course",
 "t.course_id = s.course_id",
 df_dim_course,
 update_map={c: f"s.{c}" for c in df_dim_course.columns if c != "course_id"},
 insert_cols=df_dim_course.columns
)

# Upsert fact (choose appropriate business key)
fact_key = "t.student_id = s.student_id AND t.course_id = s.course_id AND t.enrollment_date = s.enrollmen
t_date"
merge_upsert(
 "gold.fact_enrollments",
 fact_key,
 df_fact,
 update_map={c: f"s.{c}" for c in df_fact.columns if c not in ["student_id","course_id","enrollment_date"]},
 insert_cols=df_fact.columns
)
```

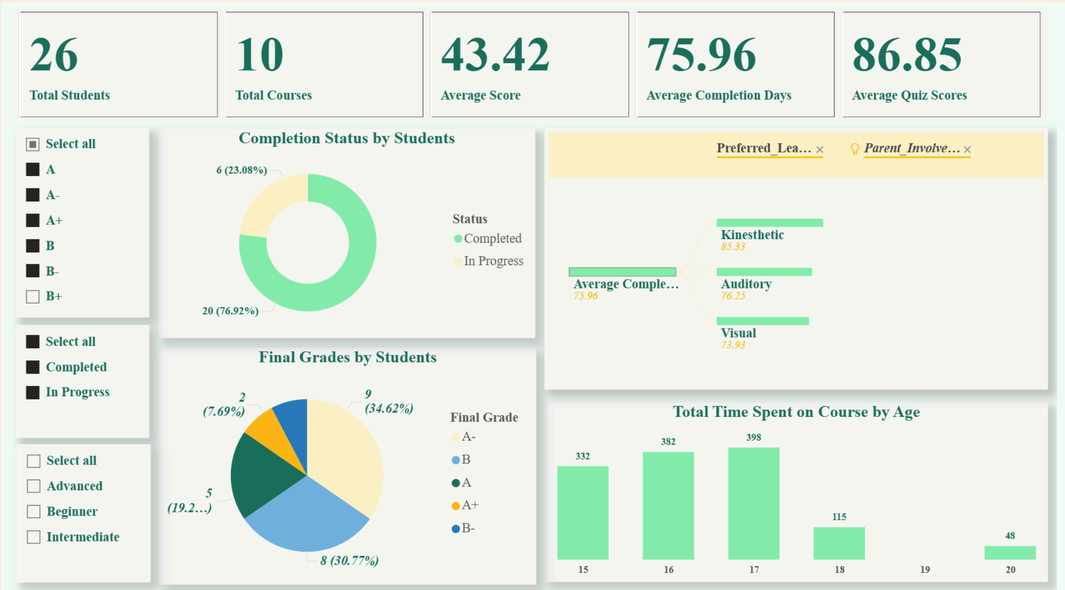
### 5. Results

The **Gold** layer now holds conformed **dimensions** and an **activity fact**, loaded **incrementally** and safe to re-run. It's ready for:

- **Semantic model** (relationships: `fact.student_id → dim_student.student_id`, `fact.course_id → dim_course.course_id`),
- **Power BI** measures and visuals built on top of trustworthy, deduplicated keys.

## Insights & Actions

**View the Live Dashboard:** https://app.powerbi.com/reportEmbed?reportId=3fd41b9f-72c4-4eb4-8f87-81ce787aa1ce&autoAuth=true&ctid=216e5950-5a9c-4dc3-96cf-437406f9c7a3




> **Completion: solid baseline, clear upside.** Current **Completion Rate ≈ 76% (19/25)** with **24% in progress**. Prioritize "near-done" learners with targeted nudges and office hours to capture quick wins.

- **Action.** Early-warning tile (in-progress + low activity) and counselor/mentor outreach cadences.
- **Target (1–2 course cycles). Completion +5pp → ~81%.**

> **Pacing: long cycle time is fixable. Avg completion time ≈ 75 days** suggests pacing or friction.

- **Action.** Publish weekly milestones in LMS, require plan-on-a-page per learner, enable reminder rules (Fabric refresh → Power BI alerts).
- **Target. −10 to −15 days** (→ **60–65 days**).

> **Assessment alignment gap: Average Score ≈ 43.16** vs **Average Quiz Score ≈ 86.36** indicates misalignment between formative and summative grading.

- **Action.** Re-weight rubric, add capstone scaffolds and practice reviews, standardize final-grade scale; monitor "quiz→final" conversion.
- **Target.** Narrow the **Final vs Quiz** gap materially; lift final-grade distribution center.

> **Segment plays: engage where it matters. Age 17** shows the **highest total hours**; older cohorts under-invest.

- **Action.** (a) "Advanced track" challenges for high-engagement cohort; (b) micro-learning & flexible deadlines for older cohorts.
- **Target.** Raise low-engagement cohort time-on-task; protect high-engagement cohort outcomes.

> **What works: hands-on + support. Kinesthetic** learning preference leads; **high parent involvement** associates with better outcomes.

- **Action.** Increase labs/simulations; for adult programs, replicate "parent involvement" via nudges, peer mentors, and check-ins.

- **Target.** Lift **performance_score** and module completion in hands-on content; sustain engagement slope.

⭐ **Navigation bar**