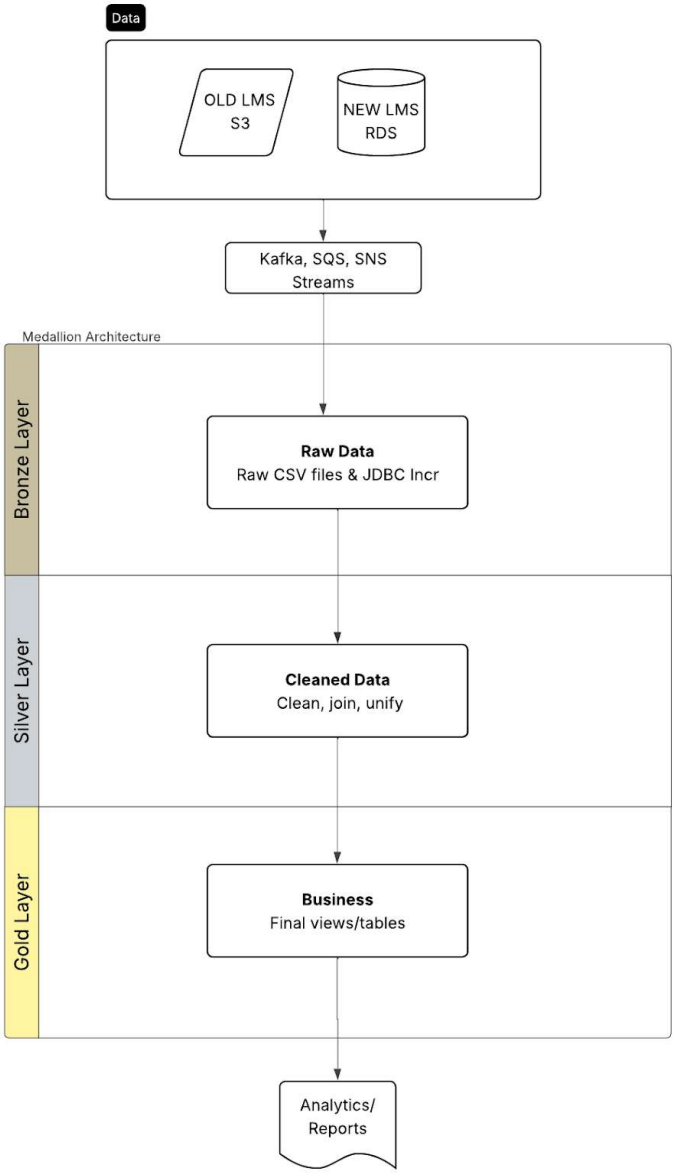
Q1. Design an architecture and schema for a data warehouse for unification of the provided data sets

The medallion architecture design follows a three-layer approach, Bronze, Silver, and Gold is built using Databricks on top of the AWS. Using this architecture, it's easy to maintain data, spot and fix issues and best for real-time and historical data. As mentioned, the Old LMS data is stored in the static CSV files in S3 bucket and the New LMS data is in the MySQL database that changes frequently. In the Bronze Layer, raw data is being moved exactly as it is, from files and the database, so we have an unmodified backup. In the Silver Layer, data is cleaned, standardized, and merged from both systems into a consistent format. Finally, in the Gold Layer, we have business-ready tables created that analysts and businesses can use.



Schema

I went with the star schema because it keeps things simple, clean, and easy to work with, especially when doing analysis. It puts the important stuff like loans and transactions at the center (fact tables) and connects that to extra details like customer info (dimension tables). By organizing everything this way, it's much easier to combine data from both the old and new LMS systems into one clear structure. This avoids confusion and makes it feel like all the data is coming from a single source. It also helps answer questions quickly, like "How much did we loan out last month?" or "Which customers are behind on payments?", even when that information comes from two different systems.

Fact Tables:

Fact_Loans
loan_id (PK)
customer_id (FK)
loan_amount
loan_date
loan_status
loan_rate
balance_interest
balance_principal
source_system
last_updated

Fact_transactions
transaction_id (PK)
loan_id (FK)
transaction_type
principal_amount
interest_amount
post_date
effective_date
created_at
updated_at
source_system

Dimension Tables:

Dim_customers
customer_id (PK)
first_name
last_name
address_line1
city
state
zip_code

Q2. Design ingestion processes for each of the datasets

1. **Old LMS Data - Static Files from S3:**
   The data from the old LMS system is CSV files stored in the cloud (like in an S3 bucket), and we know this data won't be changing anymore. Since it's fixed and won't update, we only need to load it once. To do this, I'd use Databricks Auto Loader, which makes the process easy. It automatically reads all the files, figures out the schema (columns and data types), and loads everything into our Bronze layer. I picked Auto Loader because it handles tracking and avoids manual effort. It's simple, reliable and perfect for one-time ingestion like this.

2. **New LMS Data - Live MySQL Database:**
   The new LMS is a live system that keeps fetching new loan records and updates over time. This means we need a way to keep pulling in the latest data without reloading everything from scratch.

   For this, I'd use Delta Live Tables (DLT) with a JDBC connection to MySQL. I'd set it up to only pull new or changed records using a column like "lastUpdated". That way, we are not wasting time or resources reprocessing old data. This can run every few minutes so our data stays fresh and almost in real-time.

   In the future, we could even upgrade this setup to use tools like Kafka or AWS DMS + SQS to stream

Using this approach, both the static data from the old system and the dynamic data from the new system are handled properly, and everything flows into the Bronze layer as the first step in the ETL process.

Q3. Design ETL processes for any data conversions needed

I opted for a layered ETL process using the Bronze, Silver, and Gold table structure using Databricks. This supports modular pipelines with DLT expectations and data quality rules.

Bronze Layer - Raw Ingestion:
I load the raw data just as it comes from the sources in the bronze layer. Here, we have CSV files from the old LMS and tabular data pulled from MySQL for the new LMS. There are no cleaning and transformations performed because if something goes wrong later, this layer acts like a backup of the source; we can always go back and check what the data looked like when we got it.

Silver Layer - Cleaned and Unified:
Once the raw data is in, we clean the data in the Silver layer. This step ensures we are comparing apples to apples when merging or analyzing the data.

These are some of the clean ups I would perform:

- Rename columns to follow the same naming style across both systems (like turning txtFname into first_name)
  Example: txtLname = "Williams" → becomes last_name = "Williams"
- Fix formats and make sure everything lines up properly.
  Example: dtPost = "2024-03-14 00:00:00" → formatted as 2024-03-14
  zipCode = 41569.0 (float) → converted to string "41569"
- Remove things like ID prefixes (B-, L-) so it's easier to join tables.
  Example: In the old LMS, the customer ID is stored as idCust = "0000012160" and in the new LMS, it shows up as borrower_tag = "B-0000012160". In the Silver layer, I strip the "B-" prefix so both become "0000012160"—now I can join the customer and loan tables across both systems easily.
- Loan statuses like "Paid off" and "Paid Off - Paid In Full" mean the same thing but are written differently in the two systems. I standardize these into something simple like "paid_off" so analysts don't have to guess what they mean.
- Add a new column source_system = "new_lms" or "old_lms" to track where each row came from.

Gold Layer – Final Business Tables:
After cleaning and organizing everything in the Silver layer, the Gold layer is where the data becomes truly useful for the business. This is the final and most polished step in the process, where I pull everything together into one clear view that's easy to use for reporting and dashboards.
In this step, I'm not just tidying things up, I'm turning the data into something meaningful. For example, combining loan and transaction data lets us calculate total payments or outstanding balances. And by joining with customer data, we can answer questions like "How are loans performing in different states?" or "Which customer types tend to repay late?"

In the Gold Layer, I do things like:

- Build final fact tables like fact_loans and fact_transactions and dimension tables like dim_customers.
- Join customer, loan, and transaction data into a unified structure.
- Remove duplicates using the lastUpdated field to keep the most recent records.
- Add calculated fields, like total payments or days overdue.
- Filter out unnecessary fields to keep the tables clean and focused.

- Add performance optimizations (like ZORDER or partitioning) so queries run faster.
- Rename columns for readability if needed, such as changing amtLoan to loan_amount.

In this layer, business teams will connect to dashboards or use it for ad hoc analysis. It gives them a clear, trusted view of the loans, customers, and transactions, whether they came from the old system, the new one, or both.

Q4. Merging Old and New LMS Data into a Unified Dataset for the loans, transactions, customers, etc.

Since the company is moving from an old loan system to the new one, some loan records exist in both places, while others are only in one. The goal here is to combine all of this into one clean, unified dataset that gives the full picture without duplicates.
To do this, I made sure that data from both systems could be aligned properly and merged smoothly. Here's how I approached it:

Steps to merge the data:

- **Match customer IDs**: In the old system, customer IDs look like "0000012160", and in the new system, they appear as "B-0000012160". I removed the "B-" so they match.
- **Align column names**: Renamed fields across both systems to follow the same naming format (e.g., amtLoan and loan_amount both become loan_amount).
- **Standardized data formats**: Make sure all dates, amounts, and status values follow the same format.
- **Add a source_system column**: This shows whether the data came from the old or new LMS, which helps with tracking and debugging later.
- **Remove duplicates**: If a loan exists in both systems, I keep the most recent one using the lastUpdated field.
- **Combine records**: Merge customers, loans, and transactions into shared tables so analysts can access one complete version of the truth.

This approach helps avoid confusion, ensures accuracy, and gives the business team one reliable dataset to work with, no matter where the loan started.

Unified Dataset:

loan_id	customer_id	loan_amount	loan_date	loan_status	source_system	first_name	last_name	state	last_updated
0012160-01	0000012160	300.00	2020-08-12	paid_off	old_lms	Monica	Williams	Arkansas	2023-02-21 08:08:38
6cdaed69-bcbd-56be-b2ba-3d042beb3f3f	00005D3F96	400.00	2024-03-18	paid_off	new_lms	Marcus	Taylor	Kentucky	2024-11-15 06:09:17
R-75839	0000714F14	500.00	2022-06-10	recovered	recovery	Marisa	Grant	Kentucky	2023-07-21 10:00:00

Q5. How would you support a 3rd source of loan information from, for example, a system for recovering defaulted loans?

Let's say the company starts using a third system that handles recovery or collections for the defaulted loans. The good news is that we don't need to rebuild anything from scratch. The pipeline we designed with Bronze , Silver , and Gold layers is very flexible and built for this kind of growth.

Since we already have a process for handling multiple sources (Old LMS and New LMS), we can plug in the third source the same way, just with a few additions.

Here's how we'd bring in recovery loan data:

Bronze Layer: First, we bring in the raw recovery data. Whether it's coming from a file on S3, a database, or through Kafka, we just ingest it into the Bronze layer as is, the same way we did for the other systems.

Silver Layer: Now we clean and standardize the data. We rename columns to match the structure of the existing tables, format dates and amounts properly, and make sure it lines up with the loans and customers we already have. We also add a source_system column with the value "recovery" so we know where it came from.

Gold Layer: Finally, in the Gold layer, we can choose whether or not to include the recovery loans in the final dashboards or views. For example we might want to show recovery data in a special report or keep it separate from active loans, depending on the business need.

Why this works well:

Because the pipeline is modular and layered, we can add this new source without breaking anything. Each system's data is handled the same way, and by tagging the source of each record we can keep everything organized. This makes the data pipeline flexible, future-proof and easy to scale as new systems are added later.