Here is the complete documentation for **Project 4: The Modern Data Warehouse**.

This is your entry into **Phase 2 (The Builder)**. The difficulty jumps significantly here. We are no longer just "copying" data; we are managing **History** and **State** using the industry-standard **Slowly Changing Dimension (SCD Type 2)** pattern.

0. 📚 Topic Brush-Up (Read Before You Start)

To succeed in this project, you need to understand these core Data Warehousing concepts. If any term below is unfamiliar, take 10 minutes to Google it or ask me for a specific explanation.

A. Dimensional Modeling Concepts

- **SCD Type 1 vs. Type 2:**
  - *Type 1 (Overwrite):* No history. If Alice moves to "Sales", the "IT" record is gone forever.
  - *Type 2 (Add Row):* History preserved. Alice has two rows: one for "IT" (closed) and one for "Sales" (open).
- **Surrogate Keys:** Why we use an artificial key (e.g., `EmployeeKey` = 1, 2, 3...) instead of the business key (`EmpID` = A101).
  - *Why:* Because in SCD Type 2, `EmpID` A101 will appear multiple times (once for each historical version).
- **Fact vs. Dimension Tables:**
  - *Dimension:* Descriptive data (Who, What, Where) - e.g., `DimEmployee`, `DimProduct`.
  - *Fact:* Numerical metrics (How much) - e.g., `FactSales`.

B. Azure Synapse Specifics

- **Dedicated SQL Pool (formerly SQL DW):** Understanding that this is a distributed system (MPP), not a standard SQL Server.
- **Distribution Strategy (Hash vs. Round Robin):** How data is spread across nodes. (For this project, `ROUND_ROBIN` is fine for small dimensions).
- **Identity Columns:** How `IDENTITY(1,1)` works in Synapse to auto-generate keys.

C. ADF Mapping Data Flows

- **Lookup Transformation:** How to join your incoming CSV with the existing SQL table to check for changes.
- **Conditional Split:** How to route data into different paths (e.g., "Path A: New Hires", "Path B: Promotions").

- **Alter Row Strategy:** The specific command needed to tell Synapse "UPDATE this specific row" (since Data Lakes usually only append).
- **Hashing (SHA256):** Using a "Fingerprint" column to detect changes in wide tables.

PROJECT 4: THE MODERN DATA WAREHOUSE (SCD Type 2)

Role: Data Warehouse Architect

Difficulty: Medium (Builder)

Est. Time: 6-8 Hours

---

1. Executive Summary & Scenario

The Business Problem:

"GlobalHealth," a large hospital network, needs to track the history of their staff.

- **Scenario:** Dr. Smith starts as a "Resident" in 2020. In 2022, she becomes a "Surgeon." In 2023, she moves to the "Oncology" department.
- **The Issue:** If we just update her record in the database, we overwrite the fact that she was a "Resident" in 2020. Historical reports will be wrong ("Dr. Smith performed surgery in 2020" - No, she was a resident then!).

The Goal:

Implement a **Slowly Changing Dimension (SCD) Type 2** architecture.

- We must keep **ALL** versions of a record.
- New records get inserted.
- Changed records cause the *old* version to expire (close date) and a *new* version to be created (open date).

The Senior-Level Requirement:

- **Surrogate Keys:** You cannot use the EmployeeID as the Primary Key anymore (because Dr. Smith has 3 rows). You must generate a new, unique key for the Data Warehouse.
- **Hash Comparison:** You must efficiently detect changes without comparing every single

column one by one.

---

2. Architecture Overview

We are moving to **Azure Synapse Analytics (Dedicated SQL Pool)**, the heavy-lifter for Data Warehousing (MPP - Massively Parallel Processing).

The Stack:

- **Source:** Daily HR CSV Export (ADLS Gen2).
- **Transformation Engine:** ADF **Mapping Data Flows** (Visual ETL).
- **Destination:** Synapse Dedicated SQL Pool (Dimension Table).

---

3. Pre-Requisites (Data Generator)

To learn SCD Type 2, you need data that *changes* over time. I have written a Python script that generates "Day 1" data and then "Day 2" data (with some employees promoted and some new hires).

Action:

1. Save this as generate_hr_data.py.
2. Run it to get employees_day1.csv and employees_day2.csv.
3. Upload employees_day1.csv to ADLS raw-bronze/hr/day1/.
4. Upload employees_day2.csv to ADLS raw-bronze/hr/day2/.

Python

```
import csv

# Day 1: Initial Load
data_day1 = [
    {"EmpID": 101, "Name": "Alice Smith", "Role": "Intern", "Dept": "IT", "Salary": 30000},
    {"EmpID": 102, "Name": "Bob Jones", "Role": "Manager", "Dept": "Sales", "Salary": 80000},
    {"EmpID": 103, "Name": "Charlie Brown", "Role": "Analyst", "Dept": "Finance", "Salary": 60000}
]

# Day 2: Changes happen!
# Alice gets promoted (Update). Bob stays same. Charlie quits (Not in file). David joins (New Insert).
```

```
data_day2 = [
    {"EmpID": 101, "Name": "Alice Smith", "Role": "Developer", "Dept": "IT", "Salary": 50000}, # CHANGED
    {"EmpID": 102, "Name": "Bob Jones", "Role": "Manager", "Dept": "Sales", "Salary": 80000}, # NO CHANGE
    {"EmpID": 104, "Name": "David White", "Role": "Director", "Dept": "HR", "Salary": 120000}  # NEW
]

def write_csv(filename, data):
    with open(filename, 'w', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["EmpID", "Name", "Role", "Dept", "Salary"])
        writer.writeheader()
        writer.writerows(data)
    print(f"Generated {filename}")

write_csv("employees_day1.csv", data_day1)
write_csv("employees_day2.csv", data_day2)
```

---

4. The Design Phase (Interactive)

As the Warehouse Architect, you must design the **Dimension Table**.

Challenge A: The Surrogate Key

Scenario: Dr. Smith (EmpID 101) has 2 rows in the warehouse.

1. Role: Intern (2020-2022)
2. Role: Developer (2022-Present)
   Task: If I query WHERE EmpID = 101, I get both.

- *Question:* How do I uniquely identify the *specific version* of Dr. Smith? We need a new Primary Key. What do we call it?
- *Design:* DimEmployeeKey (Identity Column).

Challenge B: The "Active" Flag

Scenario: An analyst wants to know "How many Developers do we have *right now*?"

Task: They don't want to write complex date logic (WHERE EndDate IS NULL).

- *Question:* What column can we add to make this query super simple?

Challenge C: Detecting Change

Scenario: You have 50 columns (Address, Phone, Role, Salary, Manager...).

Task: To check if an employee changed, do you write:

IF Source.Address != Dest.Address OR Source.Phone != Dest.Phone OR ...?

- *Problem:* This is slow and ugly.
- *Solution:* What cryptographic function can we use to compare the *entire row* in one go? (Hint: SHA256).

🛑 STOP! Attempt the design above before scrolling down.

✅ Architect's Solution (Reference)

Answer A: The Schema

We need a **Surrogate Key** (EmployeeKey) and **Audit Columns**.

SQL

```
CREATE TABLE DimEmployee (
    EmployeeKey INT IDENTITY(1,1) PRIMARY KEY, -- Surrogate Key
    EmpID INT,                          -- Natural/Business Key
    Name VARCHAR(100),
    Role VARCHAR(50),
    Dept VARCHAR(50),
    Salary INT,
    ValidFrom DATETIME,                 -- SCD Start
    ValidTo DATETIME,                   -- SCD End
    IsActive BIT                        -- 1 = Current, 0 = History
);
```

Answer B: The Active Flag

We add IsActive (BIT/Boolean).

- Query: SELECT * FROM DimEmployee WHERE IsActive = 1.

Answer C: Fingerprinting

We create a RowHash column.

- SHA2_256(Name + Role + Dept + ...)
- If Source.RowHash != Destination.RowHash, then *something* changed. We don't care what; we just know we need to trigger an update.

---

5. Implementation Guide (Step-by-Step)

Phase 1: Synapse Setup

1. **Create Resource: Azure Synapse Analytics** (Workspace).
2. **Dedicated SQL Pool:** Inside Synapse, create a **Dedicated SQL Pool** (formerly SQL DW).
   - *Name:* sqlpool01.
   - *Performance:* **DW100c** (Lowest tier, ~$1.50/hour). **PAUSE IT** when not using!
3. **Create Table:** Run the SQL from *Answer A* in Synapse Studio.

Phase 2: ADF Data Flow (The Logic)

This is the hardest part of Phase 2. We use **Mapping Data Flows** (Visual Interface).

Step 2.1: The Source

1. Create Data Flow: DF_SCD_Type2.
2. **Source 1 (CSV):** employees_day1.csv (simulating the daily feed).
3. **Source 2 (Dim):** DimEmployee table (from Synapse). *Alias: ExistingDim*.

Step 2.2: The Lookup (Matching)

1. Add **Lookup Transformation**.
2. Match Source.EmpID == ExistingDim.EmpID.
3. Why? To see if this employee already exists in our warehouse.

Step 2.3: The Split (New vs. Changed)

1. Add **Conditional Split**.
2. **Branch 1 (NewRow):** isNull(ExistingDim.EmpID) (It's a new hire).
3. **Branch 2 (ChangedRow):** !isNull(ExistingDim.EmpID) && Source.Role != ExistingDim.Role (It's a promotion).
   - *Note:* In a real project, use the SHA256 Hash here.

Step 2.4: Handling "New Rows" (Insert)

1. From "NewRow" branch -> **Derived Column**.
   - ValidFrom = currentTimestamp()
   - ValidTo = NULL
   - IsActive = 1
2. **Sink:** Write to DimEmployee. Allow **Insert**.

Step 2.5: Handling "Changed Rows" (The Update Strategy)

This is tricky. We need to do TWO things:

1. **Close** the old row (Set ValidTo = Now, IsActive = 0).
2. **Insert** the new row (Set ValidFrom = Now, IsActive = 1).

- **Action A (Update Old):**
  - Use **Alter Row** transformation -> Update if: true().
  - **Sink:** Synapse Table. Key columns: EmployeeKey. Mappings: Set IsActive to 0.
- **Action B (Insert New):**
  - Same as Step 2.4.

---

6. Execution & Verification

1. **Run 1 (Day 1):** Point the pipeline to employees_day1.csv.
   - *Result:* Synapse table should have 3 rows. All IsActive = 1.
2. **Run 2 (Day 2):** Point the pipeline to employees_day2.csv.
   - *Result:*
     - **Alice (101):** Two rows. One Intern (Active=0), One Developer (Active=1).
     - **Bob (102):** One row (No change).
     - **David (104):** One row (New insert).

---

7. Project Defense (Thesis Questions)

Q1. The "Race Condition"

"You are running the Update and Insert in parallel in the Data Flow. What happens if the 'Insert' finishes before the 'Update'? Is there a brief millisecond where Alice has TWO active rows? How does Synapse Transaction safety handle this?"

Q2. Hash Collisions

"You suggested using SHA256 to compare rows. Is it mathematically possible for two different names to produce the same Hash? If so, is this a risk we should worry about in a hospital?"

Q3. Dedicated vs. Serverless

"Why did we pay for a Dedicated Pool (DW100c) for this? Could we have done SCD Type 2 using Synapse Serverless SQL? Why or why not?" (Hint: Can Serverless SQL *update* rows?)

Q4. The "Late Arriving" Fact

"The data for Alice's promotion arrived 3 days late. We already processed data for yesterday. If we insert her promotion now with ValidFrom = Today, we create a gap in history. How do 'Bi-Temporal' models solve this?"

Reply "Proceed" to start Project 5 (The Medallion Lakehouse) or ask for clarification on the SCD Logic.