Awesome—let’s design a **portfolio-ready healthcare ETL project** that mirrors what real teams do. You’ll build a tiny, de-identified data mart from CSVs → SQL Server (star schema) → reports, using **SSIS** (and I’ll show the **ADF** cloud variant, too). Everything here is reproducible on your laptop with Docker.

**1) Project concept (what you’ll build)**

**Goal:** Ingest de-identified “encounters” (visits), normalize to a **star schema**, and publish 5 core KPIs often requested by hospital ops/finance.

**KPIs**

* Admissions (Encounters) trend by month
* Average Length of Stay (LOS)
* 30-day Readmission Rate
* Payer Mix (% of encounters by payer)
* Gross Charges (or proxy “Revenue”) by Service Line / Location

**2) Source data (CSV files you’ll control)**

Create a /data folder with four CSVs (fake/de-identified):

**encounters.csv** (fact-like raw)

encounter\_id,patient\_id,provider\_npi,location\_code,admit\_ts,discharge\_ts,primary\_dx,service\_line,payer,charges

10001,P0001,1234567890,ED-TAH,2024-01-02 08:41,2024-01-03 14:10,J18.9,Emergency,Medicare,3750.55

10002,P0002,1839475620,IP-TAH,2024-01-05 12:05,2024-01-10 10:32,I10,Inpatient,Blue Cross,15432.10

...

**patients.csv** (no PHI beyond a stable surrogate key + coarse attributes)

patient\_id,gender,birth\_year,zip3

P0001,F,1968,961

P0002,M,1979,894

...

**providers.csv**

provider\_npi,provider\_name,specialty

1234567890,Alex Carter,Emergency Medicine

1839475620,Jamie Lee,Hospitalist

...

**locations.csv**

location\_code,location\_name,city,state

ED-TAH,Tahoe ED,South Lake Tahoe,CA

IP-TAH,Inpatient Tower,South Lake Tahoe,CA

...

**3) Target model (star schema)**

Database: healthcare\_dm

**Dimensions**

CREATE TABLE dbo.DimDate (

DateKey int PRIMARY KEY, -- YYYYMMDD

Date date NOT NULL,

CalendarMonthName varchar(20),

CalendarYear int

);

CREATE TABLE dbo.DimPatient (

PatientKey int IDENTITY PRIMARY KEY,

PatientID varchar(20) UNIQUE, -- from CSV

Gender char(1),

BirthYear int,

Zip3 char(3),

EffectiveStart datetime2 NOT NULL DEFAULT sysutcdatetime(),

EffectiveEnd datetime2 NULL,

IsCurrent bit NOT NULL DEFAULT 1

);

CREATE TABLE dbo.DimProvider (

ProviderKey int IDENTITY PRIMARY KEY,

ProviderNPI varchar(20) UNIQUE,

ProviderName varchar(100),

Specialty varchar(100)

);

CREATE TABLE dbo.DimLocation (

LocationKey int IDENTITY PRIMARY KEY,

LocationCode varchar(20) UNIQUE,

LocationName varchar(100),

City varchar(100),

State char(2)

);

**Fact**

CREATE TABLE dbo.FactEncounter (

EncounterID bigint PRIMARY KEY,

AdmitDateKey int NOT NULL FOREIGN KEY REFERENCES dbo.DimDate(DateKey),

DischargeDateKey int NULL FOREIGN KEY REFERENCES dbo.DimDate(DateKey),

PatientKey int NOT NULL FOREIGN KEY REFERENCES dbo.DimPatient(PatientKey),

ProviderKey int NULL FOREIGN KEY REFERENCES dbo.DimProvider(ProviderKey),

LocationKey int NOT NULL FOREIGN KEY REFERENCES dbo.DimLocation(LocationKey),

PrimaryDx varchar(10),

ServiceLine varchar(50),

Payer varchar(50),

Charges decimal(12,2),

LengthOfStay decimal(6,2) -- days

);

**Helper** (DimDate seed—generate 3–5 years)

-- Seed a calendar for 2023-2026

WITH d AS (

SELECT CAST('2023-01-01' as date) as dt

UNION ALL

SELECT DATEADD(day,1,dt) FROM d WHERE dt < '2026-12-31'

)

INSERT dbo.DimDate (DateKey, Date, CalendarMonthName, CalendarYear)

SELECT CONVERT(int, FORMAT(dt,'yyyyMMdd')),

dt,

DATENAME(month, dt),

YEAR(dt)

FROM d

OPTION (MAXRECURSION 0);

**4) Spin up SQL Server locally (Docker)**

Create docker-compose.yml:

services:

sqlserver:

image: mcr.microsoft.com/mssql/server:2022-latest

container\_name: sqlserver-2022

environment:

- ACCEPT\_EULA=Y

- SA\_PASSWORD=YourStrong!Passw0rd

ports:

- "1433:1433"

volumes:

- ./mssql:/var/opt/mssql

Then:

docker compose up -d

Connect with **DBeaver** or **SSMS** and create the healthcare\_dm DB:

CREATE DATABASE healthcare\_dm; GO

Run the DDL above to create dimensions/fact + seed DimDate.

**5) Build the ETL with SSIS (on-prem Microsoft pattern)**

**Package: ETL\_Healthcare\_Demo.dtsx**

**For each dimension (typical flow):**

* **Flat File Source** (patients.csv/providers.csv/locations.csv)
* **Derived Column** (trim, uppercase codes, normalize nulls)
* **Lookup** to target dimension on business key (PatientID, ProviderNPI, LocationCode)
* **Conditional Split**: new vs existing
* **Slowly Changing Dimension** (SCD Type 2) for **DimPatient** (gender/Zip3/birth\_year changes)
* **OLE DB Destination** to upsert rows

**For the fact:**

* Flat File Source → Data Conversion (timestamps) → **Script/Derived Column**:
  + AdmitDateKey = (int)YYYYMMDD(admit\_ts)
  + DischargeDateKey = null or YYYYMMDD(discharge\_ts)
  + LengthOfStay = DATEDIFF(minute, admit\_ts, discharge\_ts)/1440.0
* **Lookups**:
  + PatientKey via PatientID
  + ProviderKey via NPI (redirect unmatched to “Unknown Provider” row or error table)
  + LocationKey via LocationCode
  + DateKeys via DimDate
* OLE DB Destination → FactEncounter

**Error handling:**

* Add **Redirect Rows** to an etl.ErrorRows table with file\_name, row\_num, error\_code, error\_column, raw\_row.

**Schedule:**

* Develop one **Master package** that calls:
  1. Load\_DimLocation
  2. Load\_DimProvider
  3. Load\_DimPatient (SCD2)
  4. Load\_FactEncounter

**6) Cloud variant (Azure Data Factory) — optional but great on a résumé**

**Linked Services:** Azure Storage (CSV), Azure SQL DB (target)  
**Datasets:** Patients, Providers, Locations, Encounters CSV; Dim/Fact tables  
**Pipeline activities:**

* **Copy Activity** for each dimension → **Mapping Data Flow** (trim/clean, SCD2 on DimPatient)
* **Data Flow** for FactEncounter (derive DateKeys, LOS, lookups)
* **Pipeline** orchestrates dims → fact; add **Validation** and **Until** activity for dependency checks
* **Triggers**: schedule daily/weekly; store config in Key Vault

**7) Sample transformations you’ll implement**

**LOS and date keys (SQL check after load):**

SELECT TOP 20

EncounterID,

AdmitDateKey, DischargeDateKey,

LengthOfStay,

PrimaryDx, ServiceLine, Payer, Charges

FROM dbo.FactEncounter

ORDER BY EncounterID;

**Create a safe “Unknown” row for late-arriving dimensions:**

INSERT INTO dbo.DimProvider (ProviderNPI, ProviderName, Specialty)

SELECT 'UNKNOWN','Unknown Provider','Unknown'

WHERE NOT EXISTS (SELECT 1 FROM dbo.DimProvider WHERE ProviderNPI='UNKNOWN');

**8) Reporting (Power BI or Tableau)**

Since healthcare shops commonly use **Power BI**, include both options:

**A) Power BI model**

* Import tables; set relationships:
  + FactEncounter → DimPatient (PatientKey)
  + FactEncounter → DimProvider (ProviderKey)
  + FactEncounter → DimLocation (LocationKey)
  + FactEncounter → DimDate on AdmitDateKey (mark DimDate as **Date table**)
* **Measures (DAX):**

Encounters := COUNTROWS(FactEncounter)

Avg LOS := AVERAGE(FactEncounter[LengthOfStay])

Readmissions 30d :=

VAR ReadmitPairs =

SUMX(

VALUES(DimPatient[PatientKey]),

VAR Enc = CALCULATETABLE(

FactEncounter,

ALLEXCEPT(DimPatient, DimPatient[PatientKey])

)

VAR CurrentAdmits =

ADDCOLUMNS(

Enc,

"IsReadmit",

CALCULATE(

COUNTROWS(Enc),

FILTER(Enc,

FactEncounter[EncounterID] <> EARLIER(FactEncounter[EncounterID]) &&

DATEDIFF(

MINX(ROW("d",FactEncounter[DischargeDateKey]), DimDate[Date]),

MINX(ROW("a",FactEncounter[AdmitDateKey]), DimDate[Date]),

DAY

) <= 30

)

)

)

RETURN SUMX(CurrentAdmits, [IsReadmit])

)

RETURN ReadmitPairs

Readmission Rate :=

DIVIDE([Readmissions 30d],[Encounters],0)

Total Charges := SUM(FactEncounter[Charges])

Payer Mix % :=

DIVIDE([Encounters], CALCULATE([Encounters], ALL(FactEncounter[Payer])), 0)

*(For a demo, you can simplify readmission logic to “same patient has another admit within 30 days”—good enough for a portfolio.)*

**Pages to build:**

* **Executive Overview:** Cards (Encounters, Avg LOS, Readmission Rate, Total Charges), monthly line chart, slicers (Service Line, Location, Payer).
* **Operations:** Heatmap by Location × Service Line, bar by Provider.
* **Finance:** Charges by Payer, trend by month.
* **Quality:** Readmission Rate by Month & Service Line.

**B) Tableau (if you prefer)**

* Connect to SQL Server → healthcare\_dm
* Build:
  + Sheet 1: Monthly Encounters
  + Sheet 2: Avg LOS by Service Line
  + Sheet 3: Payer Mix (pie or stacked bar)
  + Dashboard 1: Executive Overview with filters (Location, Service Line, Payer)

**9) What to put on GitHub (recruiter-friendly)**

healthcare-etl-demo/

├─ data/ # sample CSVs (de-identified)

├─ sql/

│ ├─ 01\_create\_schema.sql

│ ├─ 02\_seed\_dimdate.sql

│ ├─ 03\_indexes.sql

│ └─ 99\_quality\_checks.sql

├─ ssis/

│ ├─ ETL\_Healthcare\_Demo.sln

│ └─ packages/\*.dtsx

├─ adf/ # optional: ARM/Bicep & JSON for datasets/pipelines

├─ docker/

│ └─ docker-compose.yml

├─ reports/

│ ├─ PowerBI\_HE\_DM.pbix (or)

│ └─ Tableau\_HE\_DM.twbx

└─ README.md

**README highlights**

* Architecture diagram (CSV → SSIS/ADF → SQL DW → BI)
* ERD/star schema image
* How to run locally (Docker commands + simple steps)
* Screenshots/GIFs of the dashboard
* Notes on **PHI handling** and why data is de-identified
* “What I’d do next in production” section (see below)

**10) Data quality + validation (quick win for interviews)**

Create a 99\_quality\_checks.sql:

-- 1) No orphaned fact rows

SELECT COUNT(\*) Orphans

FROM dbo.FactEncounter f

LEFT JOIN dbo.DimPatient p ON f.PatientKey = p.PatientKey

WHERE p.PatientKey IS NULL;

-- 2) LOS sanity: negative or > 120 days?

SELECT COUNT(\*) BadLOS

FROM dbo.FactEncounter

WHERE LengthOfStay < 0 OR LengthOfStay > 120;

-- 3) Date keys exist

SELECT COUNT(\*) MissingDates

FROM dbo.FactEncounter f

LEFT JOIN dbo.DimDate d1 ON f.AdmitDateKey = d1.DateKey

LEFT JOIN dbo.DimDate d2 ON f.DischargeDateKey = d2.DateKey OR f.DischargeDateKey IS NULL

WHERE d1.DateKey IS NULL OR (f.DischargeDateKey IS NOT NULL AND d2.DateKey IS NULL);

**11) Stretch goals (talking points for hiring managers)**

* **Incremental loads** with CDC or file-name watermark table
* **Parameterize environments** (Dev/Test/Prod)
* **Metadata-driven ETL** (config table drives column maps)
* **Orchestration & alerts** (SQL Agent / ADF triggers + email on fail)
* **HIPAA posture**: no PHI; access control via AD roles; encrypt at rest/in transit; audit logs

**12) How to demo in 5 minutes**

1. Show the **Docker** container running SQL Server.
2. Open the **CSV** files; show the **SSIS package**; run it.
3. Query the **FactEncounter** table (show LOS calculated).
4. Open **Power BI/Tableau**; refresh; click through the 5 KPIs.
5. Point to **README** + screenshots and “next steps.”

If you want, I can generate:

* The **four sample CSVs** (a few hundred rows),
* The **SQL scripts** (schema + checks),
* A **starter SSIS solution** layout (package names + connection managers),
* And a **Power BI** template with relationships pre-wired.

Tell me which pieces you want first, and I’ll drop them in ready to use.

SSIS DETAILS

Absolutely—here’s a **click-by-click guide** to build the SSIS part (Step 5) from your CSVs into the SQL Server star schema. Assume you already have SQL Server running in Docker and the database/tables created (healthcare\_dm with Dim\* and FactEncounter).

**0) Prereqs (quick)**

* **Windows + Visual Studio 2019/2022** with **SQL Server Data Tools** and the **SSIS Projects** extension.
* CSVs saved locally (e.g., C:\healthcare-etl\data\encounters.csv, patients.csv, providers.csv, locations.csv).
* Your SQL Server connection works (e.g., Server=localhost,1433; User=sa; Password=YourStrong!Passw0rd).

**1) Create the SSIS project**

1. Open **Visual Studio** → **Create a new project** → **Integration Services Project**.
2. Name it Healthcare\_ETL\_Demo.
3. In **Solution Explorer**, you’ll see Package.dtsx. Rename it to Load\_DimLocation.dtsx (Right-click → Rename).

We’ll create 4 packages:

* Load\_DimLocation.dtsx
* Load\_DimProvider.dtsx
* Load\_DimPatient\_SCD2.dtsx
* Load\_FactEncounter.dtsx  
  …and later a master Run\_All.dtsx.

**2) Create connection managers (once, reuse everywhere)**

At the bottom of the SSIS designer (Connection Managers pane):

**A. OLE DB to SQL Server**

1. Right-click **Connection Managers** → **New OLE DB Connection**.
2. **New…** → Server name: localhost,1433; Authentication: **SQL Server Authentication**; User: sa; Password: YourStrong!Passw0rd.
3. Select database: healthcare\_dm.
4. Test Connection → OK. Rename it to CM\_OLEDB\_SQL.

**B. Flat File for each CSV**  
Repeat for each file (locations.csv, providers.csv, patients.csv, encounters.csv):

1. Right-click **Connection Managers** → **New Flat File Connection**.
2. Browse to the CSV file.
3. Set **Format** = Delimited, **Header row delimiter** = {CR}{LF}, **Text qualifier** = " if your CSV uses quotes.
4. On **Columns**, confirm delimiter is comma.
5. On **Advanced**, set data types carefully:
   * Codes/IDs/Names = string [DT\_STR] (set reasonable lengths).
   * Timestamps in encounters = string [DT\_STR] (we’ll convert later).
6. Name them CM\_FF\_Locations, CM\_FF\_Providers, CM\_FF\_Patients, CM\_FF\_Encounters.

Tip: If you’ll move files later, you can parameterize file paths after you have a first run working.

**3) Load DimLocation (Type 1 upsert)**

Open **Load\_DimLocation.dtsx** (Control Flow):

1. Drag a **Data Flow Task**. Name: DFT Load DimLocation. Double-click to enter the **Data Flow**.
2. **Flat File Source** → Connection: CM\_FF\_Locations.
3. **Derived Column** (cleanup):
   * LocationCode\_clean = LTRIM(RTRIM([location\_code]))
   * LocationName\_clean = LTRIM(RTRIM([location\_name]))
   * City\_clean = LTRIM(RTRIM([city]))
   * State\_clean = UPPER(LTRIM(RTRIM([state])))
4. **Lookup** (on existing dim):
   * Drag **Lookup**; connect from Derived Column.
   * Connection: CM\_OLEDB\_SQL.
   * Use table: dbo.DimLocation.
   * **Columns**: Map LocationCode\_clean → DimLocation.LocationCode.
   * Output: select LocationKey, LocationName, City, State (rename outputs to e.g., lk\_LocationKey).
   * **General → No matching entries**: choose **Redirect rows to no match output** (so new rows are separated).
5. **Conditional Split** (optional; or just use the two outputs of Lookup):
   * Not needed if you use Redirects; you already have **Match** vs **No Match** outputs.
6. **Insert path (No Match)**:
   * **OLE DB Destination** → table dbo.DimLocation
   * Map clean columns → LocationCode, LocationName, City, State.
7. **Update path (Match)**:
   * **OLE DB Command** (row-by-row update; fine for small dims):
     + Connection: CM\_OLEDB\_SQL
     + SQLCommand:
     + UPDATE dbo.DimLocation
     + SET LocationName=?, City=?, State=?
     + WHERE LocationKey=?
     + **Column Mappings**: parameters in this order:
       1. LocationName\_clean
       2. City\_clean
       3. State\_clean
       4. lk\_LocationKey

**Run the package** (F5). Check SELECT COUNT(\*) FROM dbo.DimLocation;

Repeat the same pattern to build **Load\_DimProvider.dtsx** (keys on ProviderNPI). Use clean fields and upsert.

**4) Prepare “Unknown” rows (for late arriving keys)**

In **SQL Server**, run once:

IF NOT EXISTS (SELECT 1 FROM dbo.DimProvider WHERE ProviderNPI='UNKNOWN')

INSERT dbo.DimProvider(ProviderNPI,ProviderName,Specialty) VALUES ('UNKNOWN','Unknown Provider','Unknown');

IF NOT EXISTS (SELECT 1 FROM dbo.DimPatient WHERE PatientID='UNKNOWN')

INSERT dbo.DimPatient(PatientID,Gender,BirthYear,Zip3) VALUES ('UNKNOWN','U',1900,'000');

IF NOT EXISTS (SELECT 1 FROM dbo.DimLocation WHERE LocationCode='UNKNOWN')

INSERT dbo.DimLocation(LocationCode,LocationName,City,State) VALUES ('UNKNOWN','Unknown','Unknown','NA');

Note the **surrogate keys** assigned to these rows (they’ll likely be 1 if inserted first). We’ll use them as fallbacks in the fact load.

**5) Load DimPatient (Type 2 with the SCD wizard)**

Create package **Load\_DimPatient\_SCD2.dtsx**:

1. **Data Flow Task** → name DFT Load DimPatient SCD2.
2. **Flat File Source** → CM\_FF\_Patients.
3. **Derived Column**:
   * PatientID\_clean = LTRIM(RTRIM([patient\_id]))
   * Gender\_clean = UPPER(LTRIM(RTRIM([gender])))
   * BirthYear\_int = (cast if needed) or keep as string if your CSV is safe
   * Zip3\_clean = LTRIM(RTRIM([zip3]))
4. **Slowly Changing Dimension** transformation:
   * Drag **Slowly Changing Dimension**; connect Derived Column.
   * **OLE DB connection**: CM\_OLEDB\_SQL
   * **Table**: dbo.DimPatient
   * **Business Key**: PatientID (map to PatientID\_clean)
   * **Columns**:
     + Gender, Zip3 as **Changing** or **Historical** (choose **Historical** to create new versioned rows)
     + BirthYear typically **Fixed** (rarely changes)
   * Check **Enable inferred members** only if you expect facts arriving without matched patients (optional here).
   * Set **Start/End** columns to your SCD fields if you created them (EffectiveStart, EffectiveEnd, IsCurrent).
   * Finish the wizard; it auto-creates the necessary components (Lookups, Inserts, Updates).

**Run it**. Check:

SELECT TOP 10 \* FROM dbo.DimPatient ORDER BY PatientKey;

If you re-run with modified CSVs, you should see historical rows as designed.

**6) Load the FactEncounter**

Create **Load\_FactEncounter.dtsx**:

1. **Data Flow Task** → DFT Load FactEncounter.
2. **Flat File Source** → CM\_FF\_Encounters.
   * Columns like: encounter\_id, patient\_id, provider\_npi, location\_code, admit\_ts, discharge\_ts, primary\_dx, service\_line, payer, charges
3. **Data Conversion** (for timestamps / numeric):
   * Convert admit\_ts → admit\_ts\_ts (type DT\_DBTIMESTAMP)
   * Convert discharge\_ts → discharge\_ts\_ts (type DT\_DBTIMESTAMP, allow nulls if present)
   * Convert charges → decimal: often DT\_NUMERIC (e.g., precision 12, scale 2)
4. **Derived Column** (keys + LOS):
   * AdmitDateKey =
   * (DT\_I4)(
   * DATEPART("yyyy",[admit\_ts\_ts]) \* 10000
   * + DATEPART("mm",[admit\_ts\_ts]) \* 100
   * + DATEPART("dd",[admit\_ts\_ts])
   * )
   * DischargeDateKey =
   * ISNULL([discharge\_ts\_ts]) ?
   * NULL(DT\_I4) :
   * (DT\_I4)(
   * DATEPART("yyyy",[discharge\_ts\_ts]) \* 10000
   * + DATEPART("mm",[discharge\_ts\_ts]) \* 100
   * + DATEPART("dd",[discharge\_ts\_ts])
   * )
   * LengthOfStay =
   * ISNULL([discharge\_ts\_ts]) ?
   * (DT\_NUMERIC,6,2)0
   * :
   * (DT\_NUMERIC,6,2)(
   * (DATEDIFF("minute",[admit\_ts\_ts],[discharge\_ts\_ts]) / 1440.0)
   * )
5. **Lookups** (get surrogate keys):
   * **LKP Patient**: on DimPatient matching PatientID = cleaned patient\_id
     + Select PatientKey as output (call it lk\_PatientKey)
     + **No match**: choose **Ignore failure** (so unmatched become NULL), we’ll default to Unknown.
   * **LKP Provider**: match ProviderNPI → output ProviderKey (ignore failure).
   * **LKP Location**: match LocationCode → output LocationKey (ignore failure).
   * **(Optional) LKP DimDate** for Admit/Discharge to validate keys. For a demo, you can skip the lookup and trust the computed keys since you seeded DimDate.
6. **Derived Column** (fallback to Unknown for NULL lookups):
   * PatientKey\_final = ISNULL([lk\_PatientKey]) ? (DT\_I4)1 : [lk\_PatientKey]
   * ProviderKey\_final = ISNULL([lk\_ProviderKey]) ? (DT\_I4)1 : [lk\_ProviderKey]
   * LocationKey\_final = ISNULL([lk\_LocationKey]) ? (DT\_I4)1 : [lk\_LocationKey]  
     *(Assumes your Unknown rows were first and got key = 1. If different, use that key.)*
7. **OLE DB Destination** → table dbo.FactEncounter
   * Map:
     + EncounterID → EncounterID
     + AdmitDateKey → AdmitDateKey
     + DischargeDateKey → DischargeDateKey
     + PatientKey\_final → PatientKey
     + ProviderKey\_final → ProviderKey
     + LocationKey\_final → LocationKey
     + plus direct columns: PrimaryDx, ServiceLine, Payer, Charges, LengthOfStay

**Run it**. Verify:

SELECT TOP 20 EncounterID, AdmitDateKey, DischargeDateKey, PatientKey, ProviderKey, LocationKey, LengthOfStay

FROM dbo.FactEncounter

ORDER BY EncounterID;

**7) Build a Master package to orchestrate**

Create **Run\_All.dtsx** (Control Flow):

1. Add an **Execute SQL Task** named Ensure Unknown Rows with a batch containing the Unknown INSERT statements (from step 4).
2. Add **Execute Package Task** (four of them) in this order with **success** precedence constraints:
   * Load\_DimLocation.dtsx
   * Load\_DimProvider.dtsx
   * Load\_DimPatient\_SCD2.dtsx
   * Load\_FactEncounter.dtsx
3. Press **F5** to run the whole pipeline end-to-end.

**8) Basic error handling (nice to have)**

* In each **Lookup**, set **No match** to **Redirect rows** and send those to a separate **Flat File Destination** or an **OLE DB Destination** table like etl.ErrorRows with extra columns: file\_name, row\_number, error\_reason.
* In **OLE DB Destination**, set **Error Output** to **Redirect row** and land bad rows in the same error table.

Minimal error table DDL:

CREATE TABLE etl.ErrorRows (

ErrorDT datetime2 DEFAULT sysutcdatetime(),

SourceFile varchar(100),

RowText nvarchar(max),

ErrorReason varchar(200)

);

**9) Quick checks & indexes**

Run your quality checks:

-- Orphans

SELECT COUNT(\*) AS OrphanPatients

FROM dbo.FactEncounter f

LEFT JOIN dbo.DimPatient p ON f.PatientKey=p.PatientKey

WHERE p.PatientKey IS NULL;

-- LOS sanity

SELECT COUNT(\*) AS BadLOS

FROM dbo.FactEncounter

WHERE LengthOfStay < 0 OR LengthOfStay > 120;

Add simple indexes for performance:

CREATE INDEX IX\_FactEncounter\_AdmitDateKey ON dbo.FactEncounter(AdmitDateKey);

CREATE INDEX IX\_FactEncounter\_PatientKey ON dbo.FactEncounter(PatientKey);

**10) Tips that save time**

* If a Flat File column keeps truncating, increase **OutputColumnWidth** in the Flat File connection **Advanced** tab.
* If lookups are slow, set **Cache mode** to **Full cache** (good for small dims).
* When testing, you can **Preview** each component (right-click → **Data Viewers** on paths) to see rows flowing.
* Keep **CSV column names** exactly matching your mapping plan; fewer surprises.

If you want, I can also give you:

* A tiny set of **CSV samples**,
* A ready-to-run **SQL script** to create all tables + unknown rows,
* And a screenshot-style checklist you can print while building the packages.