Chinook Data from Azure SQL to Snowflake DW using ADF

End-to-end ETL pipeline using **Azure Data Factory** to migrate Chinook music store data from **Azure SQL Database** → **Azure Blob Storage** (Parquet) → **Snowflake Data Warehouse** with dimensional modeling.

Github Link: https://github.com/vedantmane12/chinook_movie_data_AzureSQL_to_Snowflake_Data_PL

Azure User: mane.v@northeastern.edu

Subscription: Azure for Students

Resource Group: dadabi

Region: US West 2

Key Vault: kvault-damg7370

Storage Account: storagedamg7370

Database: DAMG7370FALL2025 (dada-bi.database.windows.net)

User: damg7370

Data Factory (V2): adfdemo-damg7370

▼ Linked Services

Azure SQL: ls_azuresqldb

Azure Key Vault: ls_azurekeyvault

Azure BLOB Storage: ls_azureblobstorage

Snowflake: ls_snowflake

▼ Dataset

Azure SQL: chinook_ds

Azure BLOB Storage: chinook_ds_parquet

Snowflake: chinook_ds_snowflake

▼ Pipelines

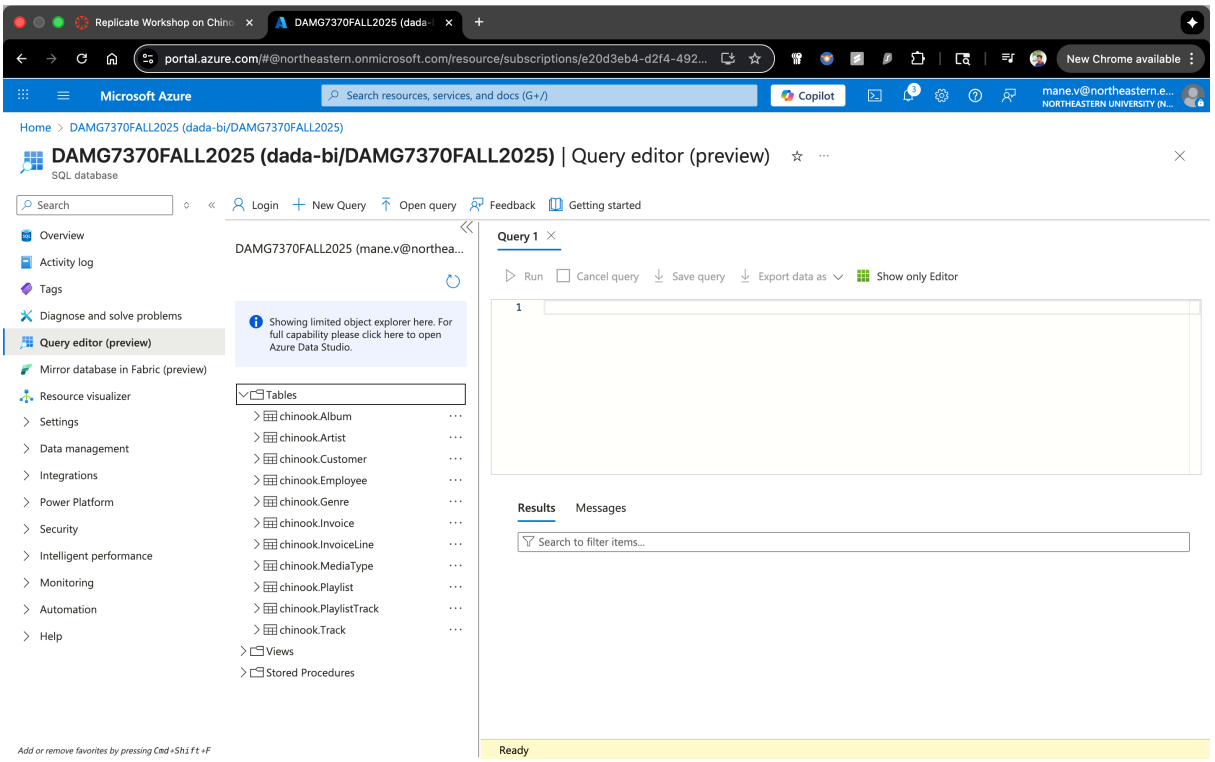
- `extract_SQLDB_PL` → Extract Azure SQL to Parquet files
- `stage_Partquet_PL` → Load Parquet to Snowflake STAGE
- `load_ALBUM_DIM_PL` → Load album dimension
- `load_ARTIST_DIM_PL` → Load artist dimension
- `load_CUSTOMER_DIM_PL` → Load customer dimension
- `load_GENRE_DIM_PL` → Load genre dimension
- `load_INVOICE_DIM_PL` → Load invoice dimension
- `load_INVOICELINE_DIM_PL` → Load invoice line dimension
- `load_SALES_FACT_PL` → Load sales fact table

▼ Dataflows

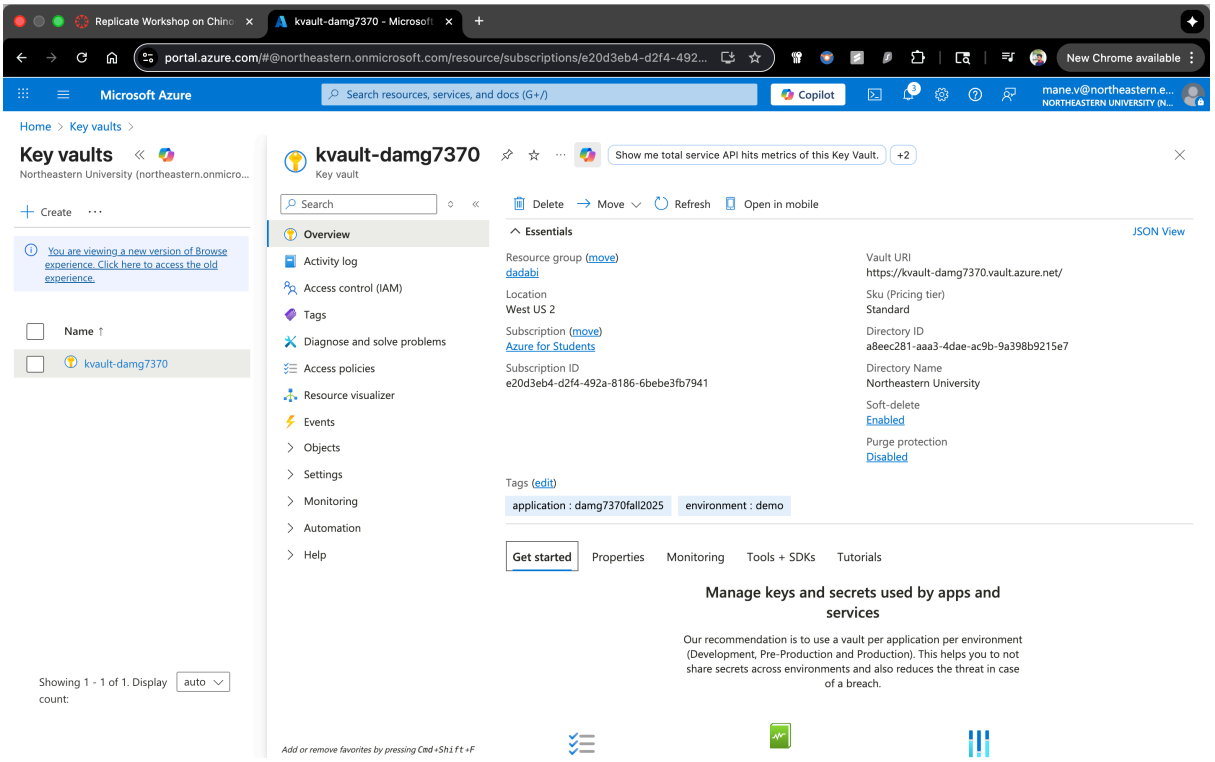
- `transform_source_parquet` → Transform Parquet files and add audit columns before loading to Snowflake STAGE
- `df_load_ALBUM_DIM` → Extract from STAGE.Album, generate surrogate keys, and load to DW.ALBUM_DIM
- `df_load_ARTIST_DIM` → Extract from STAGE.Artist, generate surrogate keys, and load to DW.ARTIST_DIM
- `df_load_CUSTOMER_DIM` → Extract from STAGE.Customer, generate surrogate keys, and load to DW.CUSTOMER_DIM

- `df_load_GENRE_DIM` → Extract from STAGE.Genre, generate surrogate keys, and load to DW.GENRE_DIM
- `df_load_INVOICE_DIM` → Extract from STAGE.Invoice, generate surrogate keys, and load to DW.INVOICE_DIM
- `df_load_INVOICELINE_DIM` → Extract from STAGE.InvoiceLine, generate surrogate keys, and load to DW.INVOICELINE_DIM

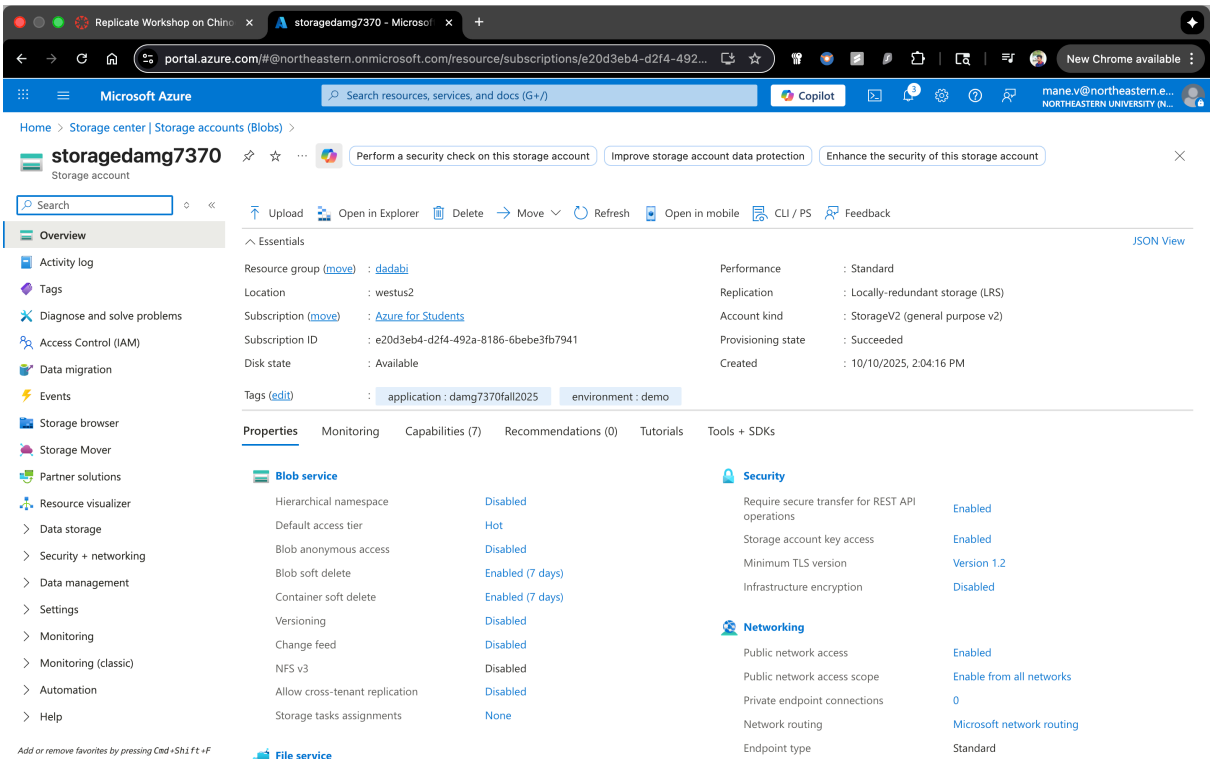
Azure SQL Database



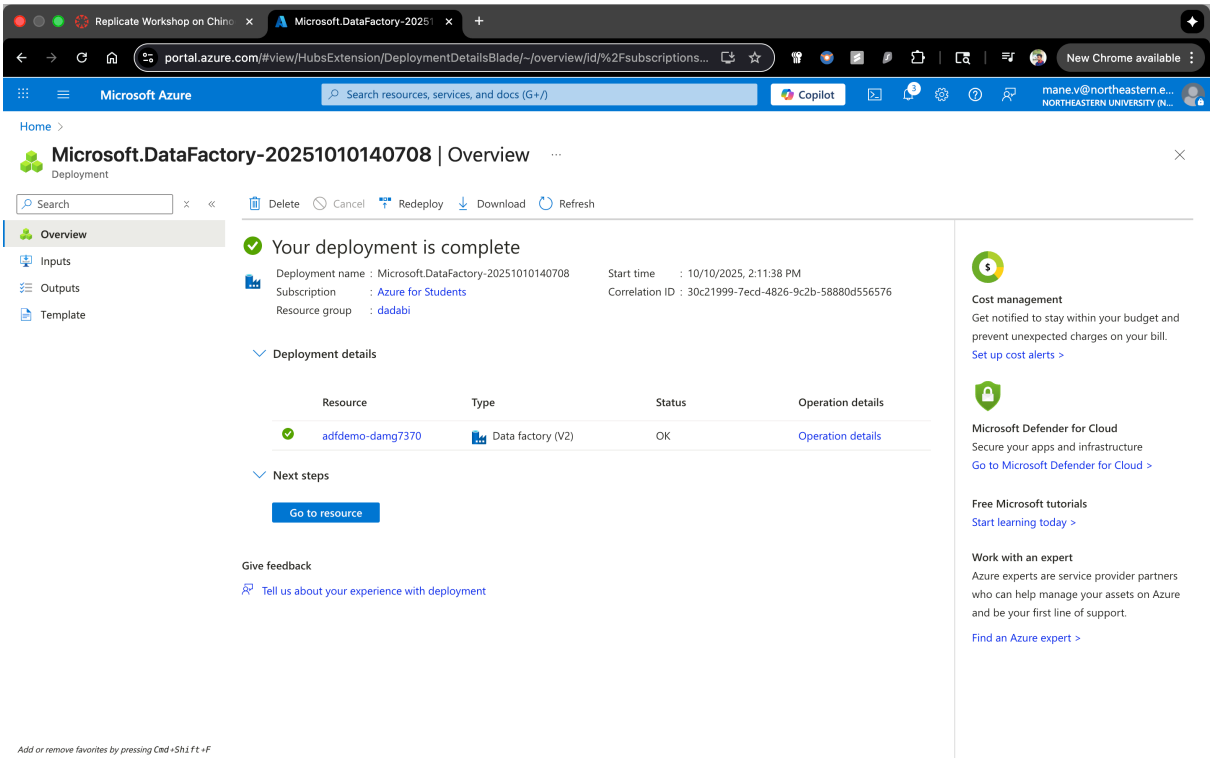
Key Vault



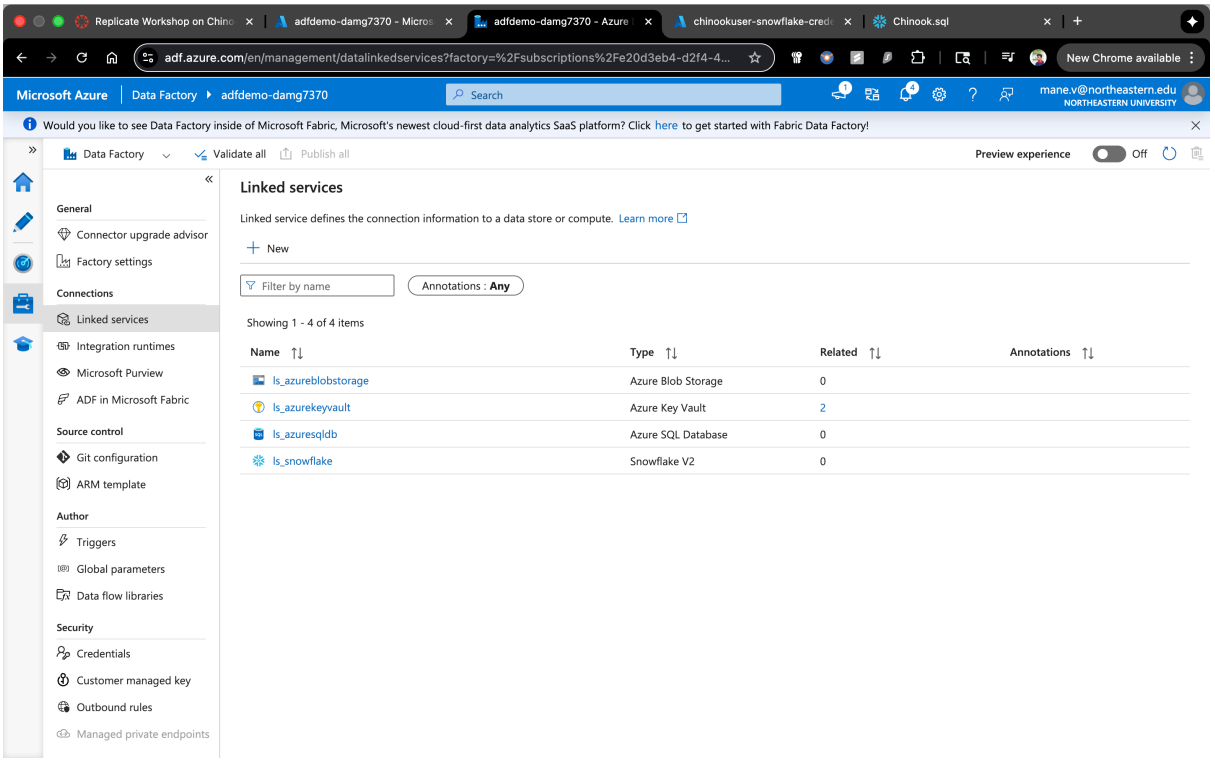
Azure BLOB Storage Account



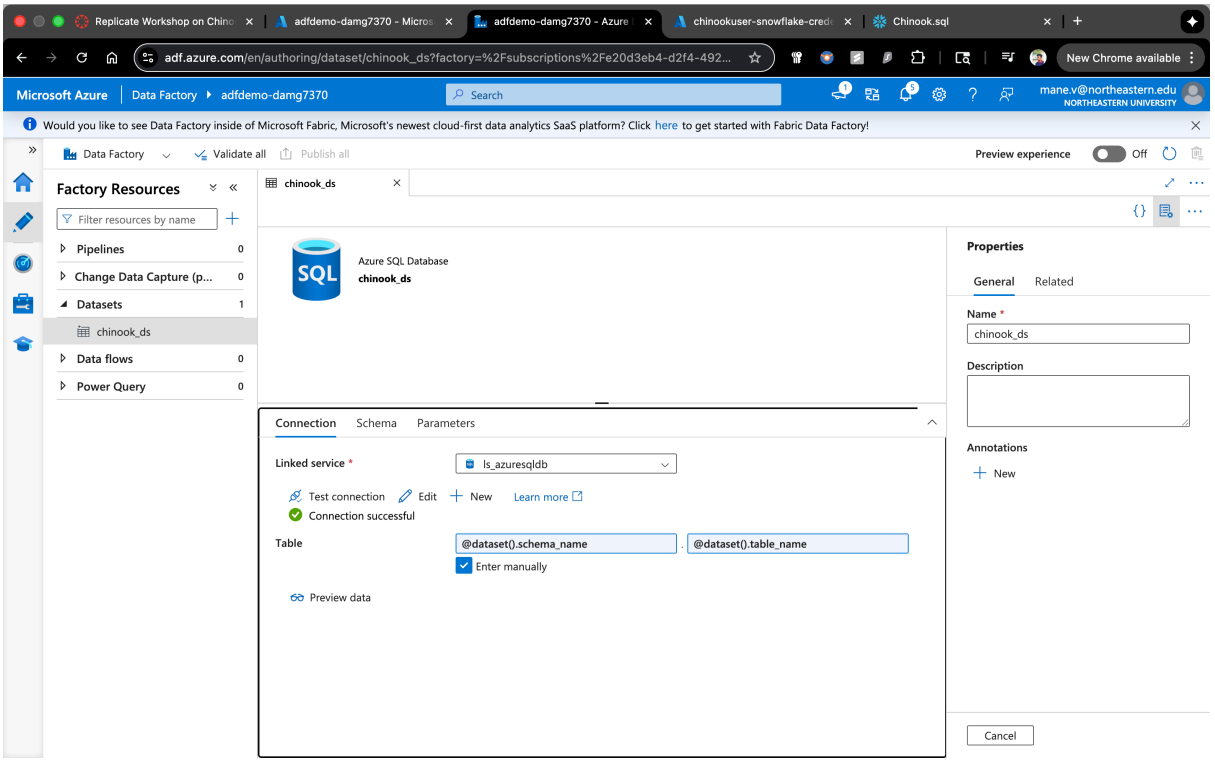
Azure Data Factory Setup



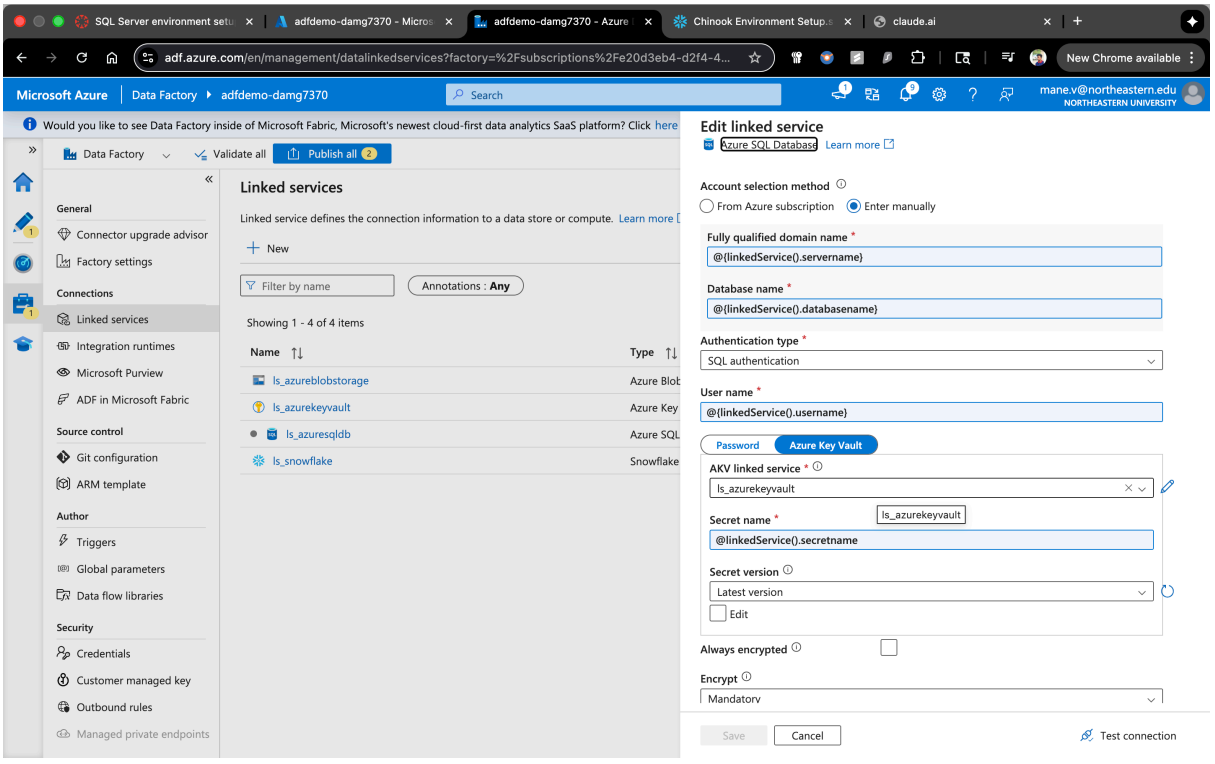
ADF Linked Services



Datasets in ADF

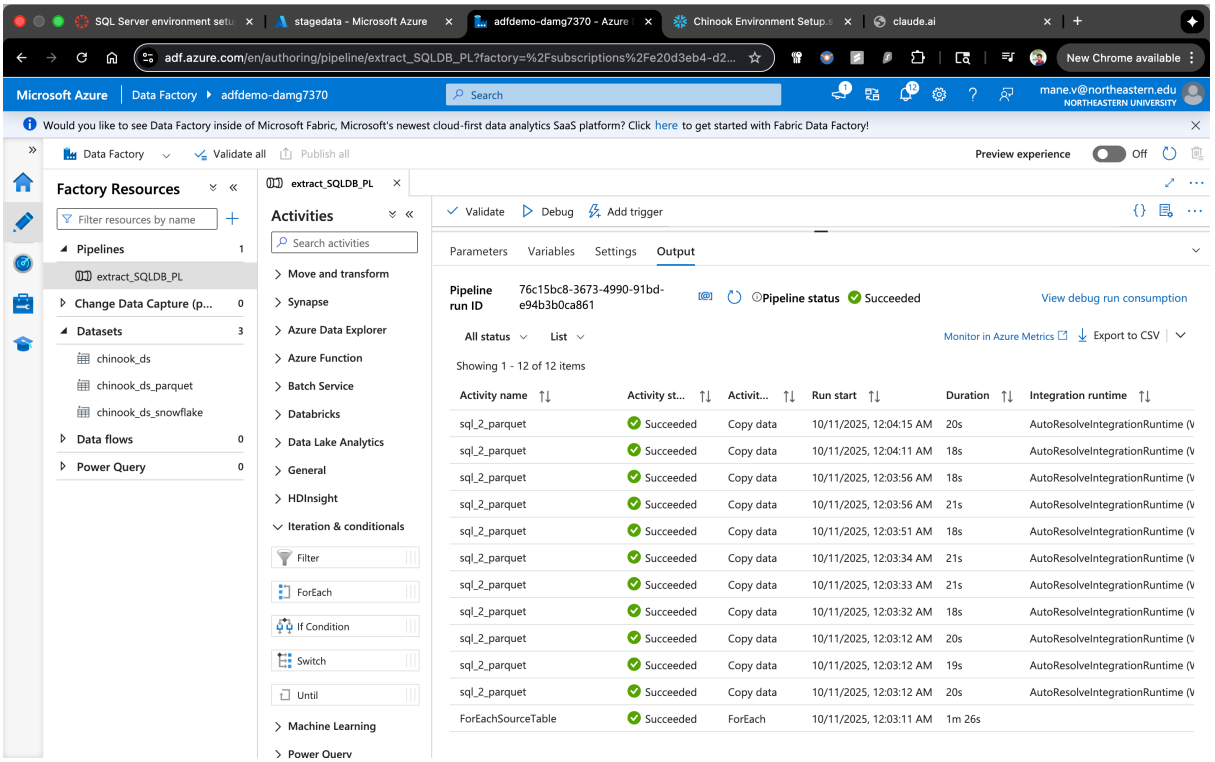


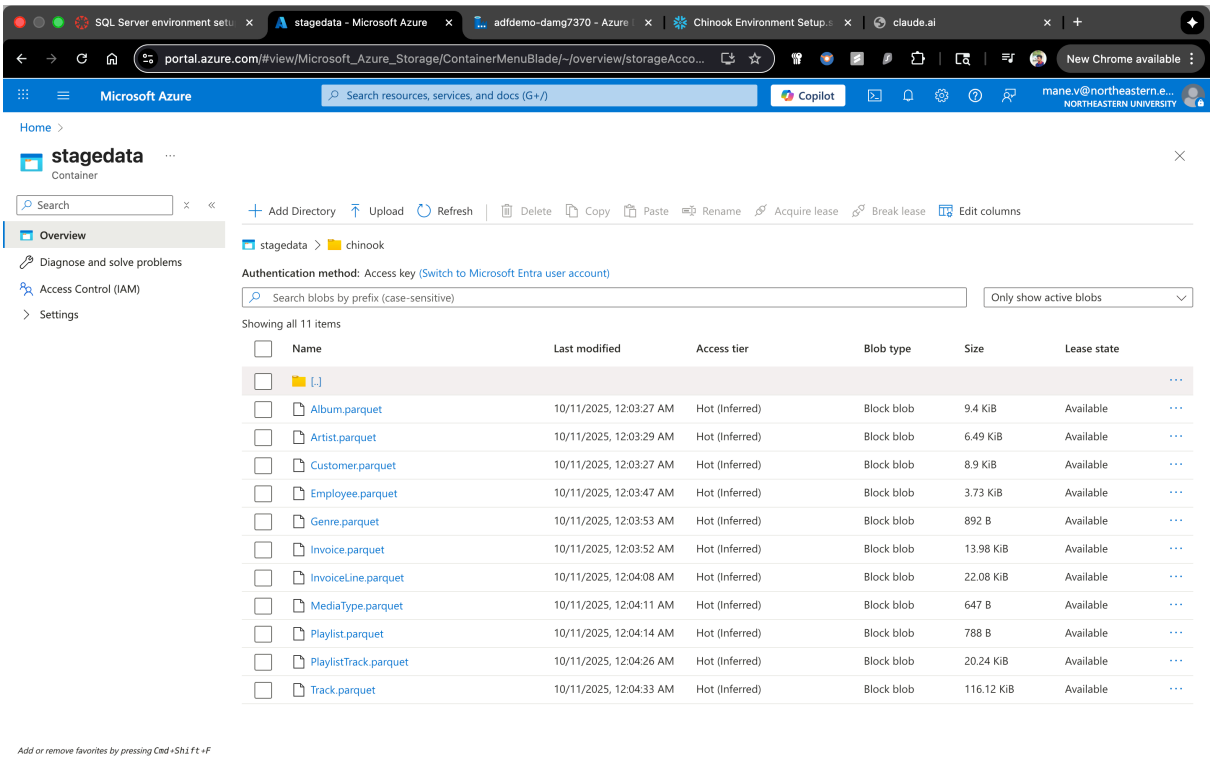
Azure Linked Service Parameters



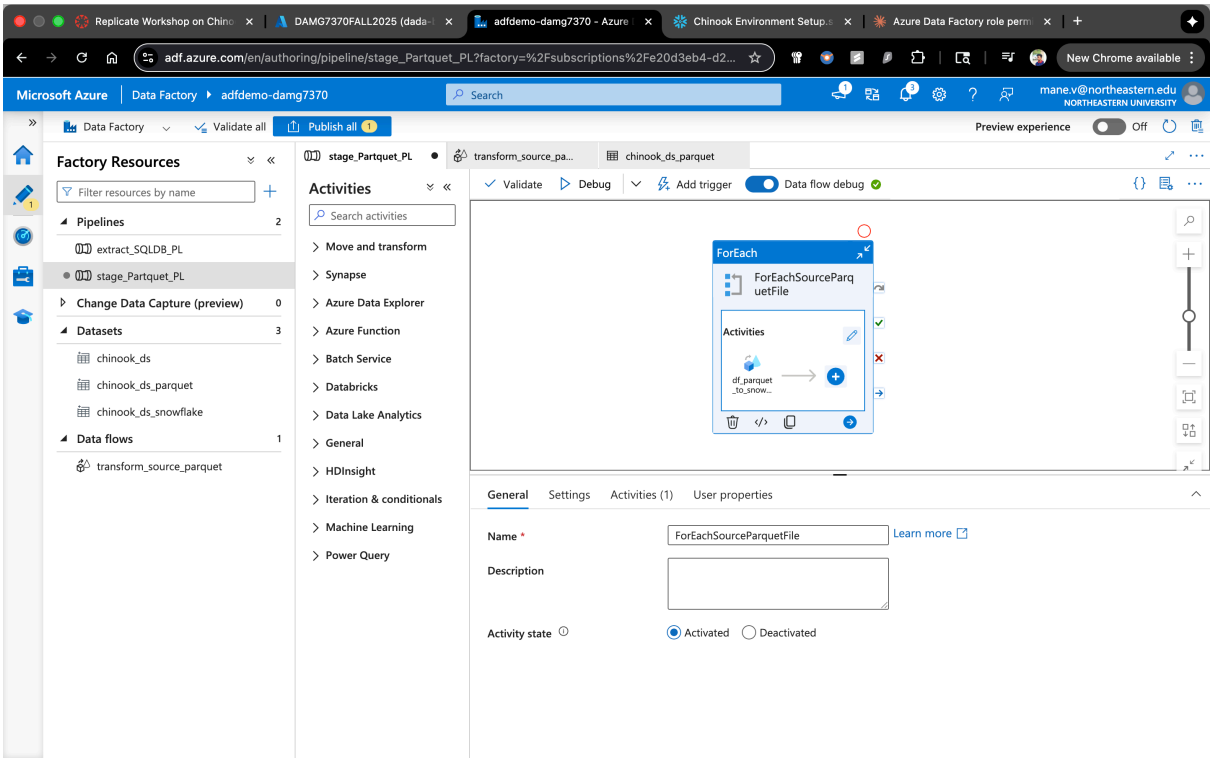
ADF Pipelines

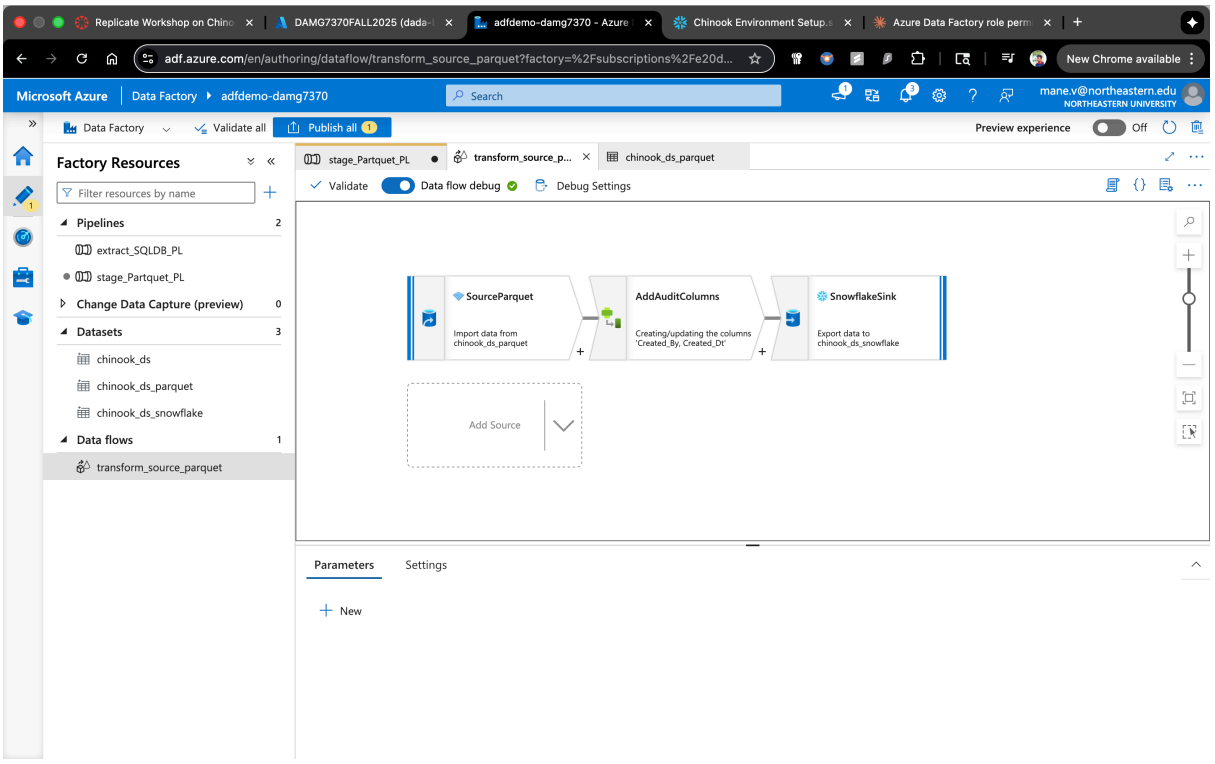
Copy Azure SQL to BLOB Storage

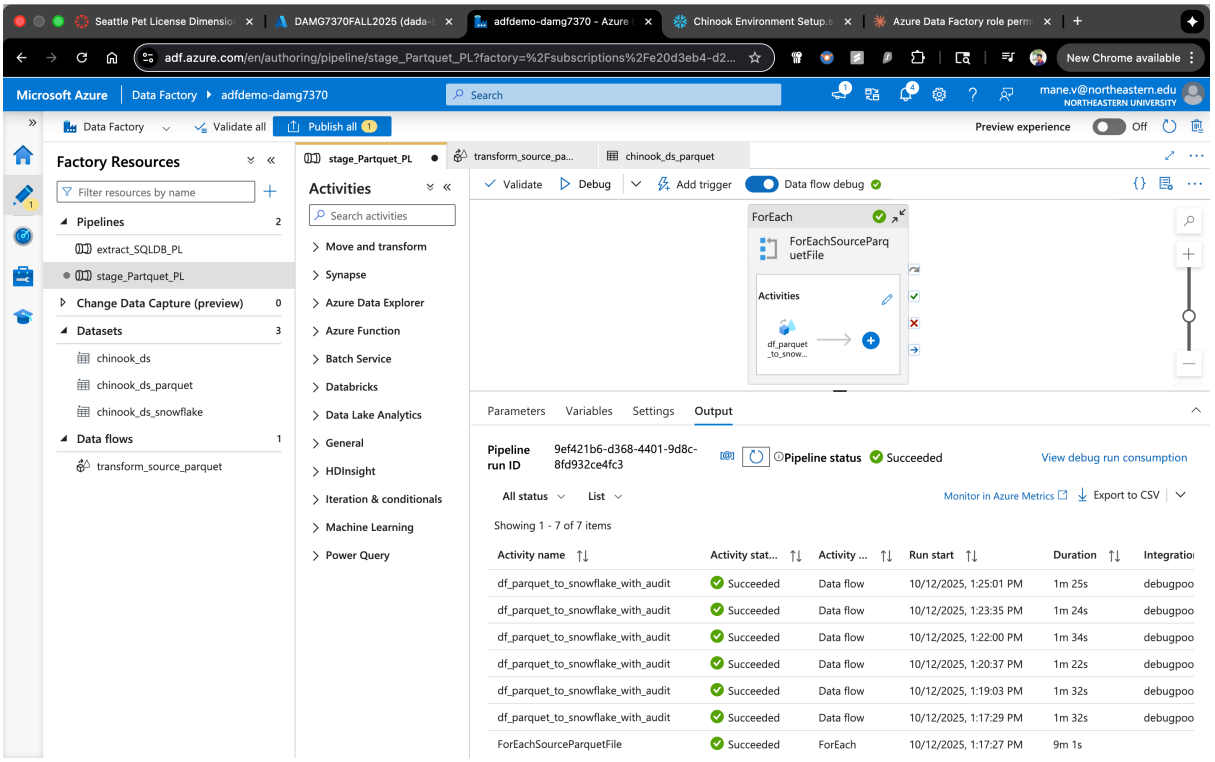




Azure BLOB Storage to Snowflake Staging with Data Column Transformations


Microsoft Azure
Data Factory
adfdemo-damg7370
Factory Resources
Pipelines
extract_SQLDB_PL
stage_Partquet_PL
Change Data Capture (preview)
Datasets
chinook_ds
chinook_ds_parquet
chinook_ds_snowflake
Data flows
transform_source_parquet
Activities
Move and transform
Synapse
Azure Data Explorer
Azure Function
Batch Service
Databricks
Data Lake Analytics
General
HDInsight
Iteration & conditionals
Machine Learning
Power Query
ForEach
ForEachSourceParquetFile
Activities
General
Settings
Activities (1)
User properties
Name *
ForEachSourceParquetFile
Learn more
Description
Activity state
Activated
Deactivated

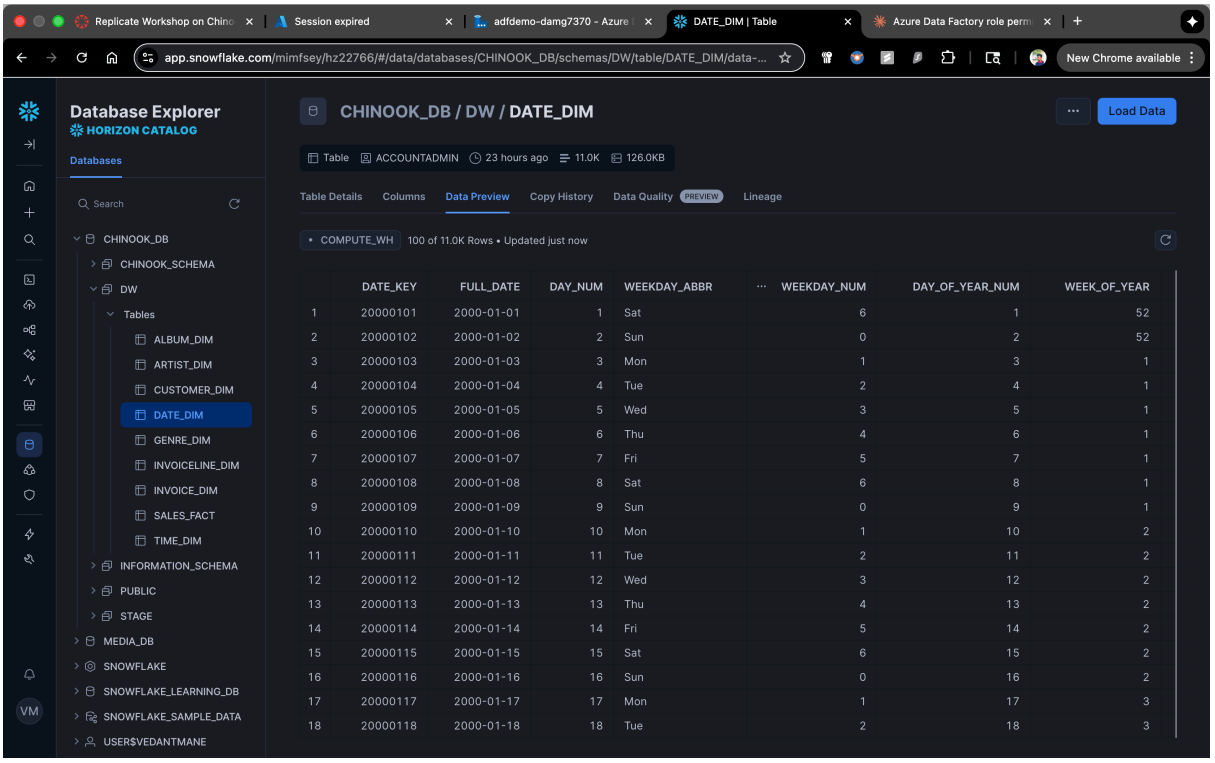

Microsoft Azure
Data Factory
adfdemo-damg7370
Factory Resources
Pipelines
extract_SQLDB_PL
stage_Partquet_PL
Change Data Capture (preview)
Datasets
chinook_ds
chinook_ds_parquet
chinook_ds_snowflake
Data flows
transform_source_parquet
SourceParquet
Import data from chinook_ds_parquet
AddAuditColumns
Creating/updating the columns 'Created_By, Created_Dt'
SnowflakeSink
Export data to chinook_ds_snowflake
Add Source
Parameters
Settings
New



Load DATE_DIM into SNOWFLAKE WH

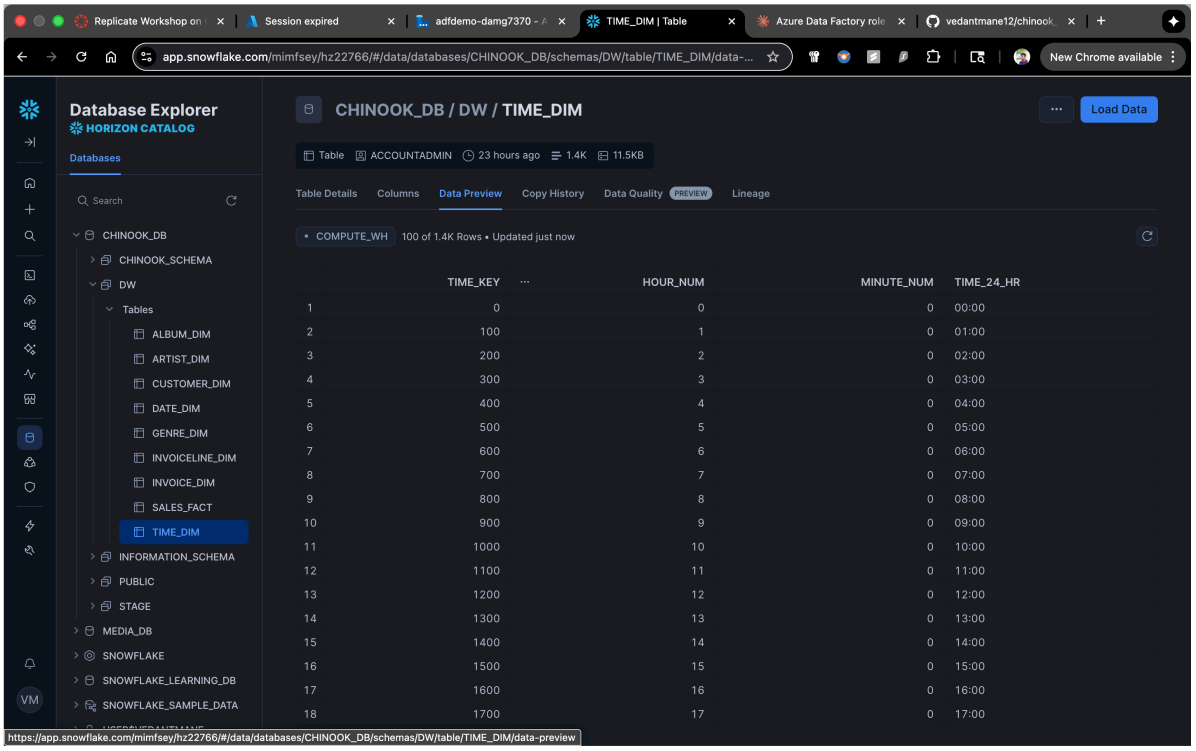
```
INSERT INTO DW.DATE_DIM
WITH DATES AS (
   SELECT
      DATEADD(DAY, SEQ4(), DATE '2000-01-01') AS DATE_KEY
   FROM TABLE(GENERATOR(ROWCOUNT ⇒ 365*30))
)
SELECT
   TO_CHAR(DATE_KEY, 'YYYYMMDD') DATE_KEY,
   DATE_KEY FULL_DATE,
   DAY(DATE_KEY) DAY_NUM,
   TO_CHAR(DATE_KEY, 'DY') WEEKDAY_ABBR,
   DAYOFWEEK(DATE_KEY) WEEKDAY_NUM,
   DAYOFYEAR(DATE_KEY) DAY_OF_YEAR_NUM,
   WEEK(DATE_KEY) WEEK_OF_YEAR,
   MONTH(DATE_KEY) MONTH_NUM,
   TO_CHAR(DATE_KEY, 'Mon') MONTH_ABBR,
   QUARTER(DATE_KEY) QUARTER_NUM,
   CASE QUARTER(DATE_KEY)
      WHEN 1 THEN 'Q1'
      WHEN 2 THEN 'Q2'
      WHEN 3 THEN 'Q3'
      WHEN 4 THEN 'Q4'
   END QUARTER_NAME,
   YEAR(DATE_KEY) YEAR_NUM,
   DATE_TRUNC('MONTH', DATE_KEY) FIRST_DAY_OF_MONTH,
   LAST_DAY(DATE_KEY) LAST_DAY_OF_MONTH,
   CASE WHEN DAYOFWEEK(DATE_KEY) IN (0, 6)
      THEN 'Y'
      ELSE 'N'
```

```
    END IS_WEEKEND
FROM DATES;
```

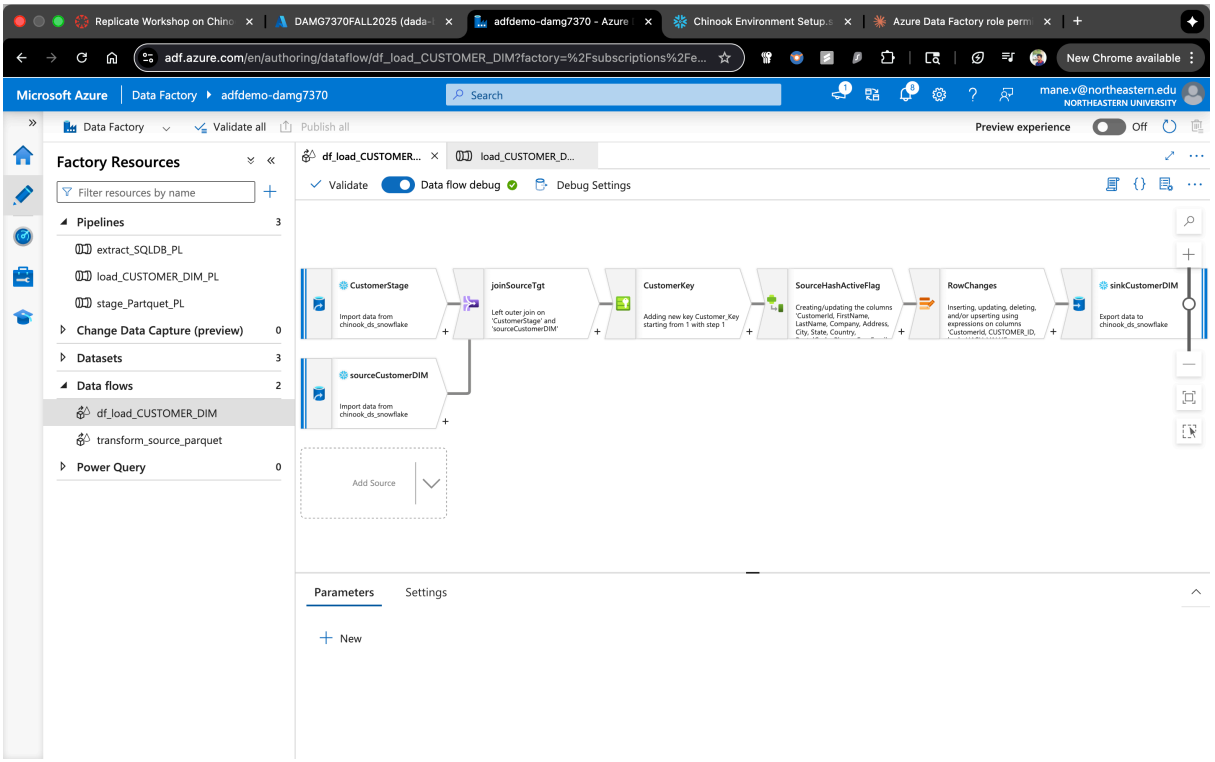


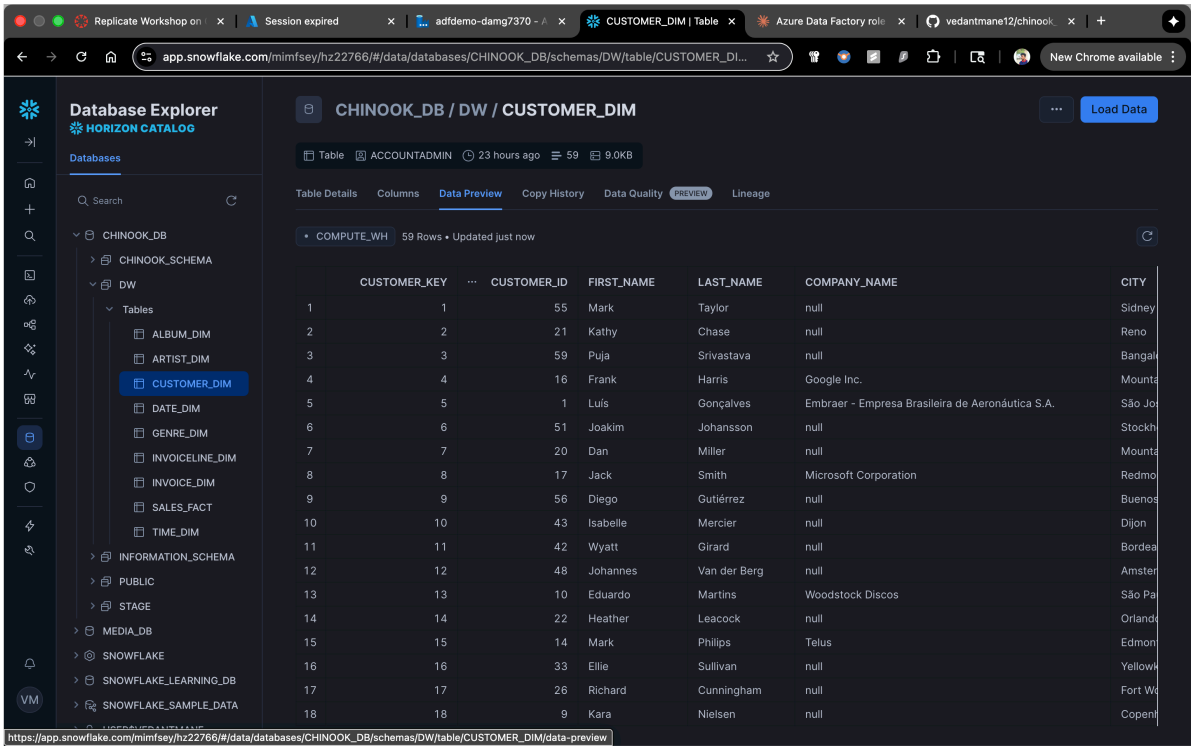
Load TIME_DIM into Snowflake WH

```
INSERT INTO DW.TIME_DIM
WITH H AS (
    SELECT SEQ4() AS Hr FROM TABLE(GENERATOR(ROWCOUNT => 24))
),
M AS (
    SELECT SEQ4() AS Mi FROM TABLE(GENERATOR(ROWCOUNT => 60))
)
SELECT
    LPAD(H.Hr, 2, '0') || LPAD(M.Mi, 2, '0') AS TIME_KEY,
    H.Hr,
    M.Mi,
    LPAD(H.Hr, 2, '0') || ':' || LPAD(M.Mi, 2, '0') AS TIME_24_HR
FROM H, M;
```

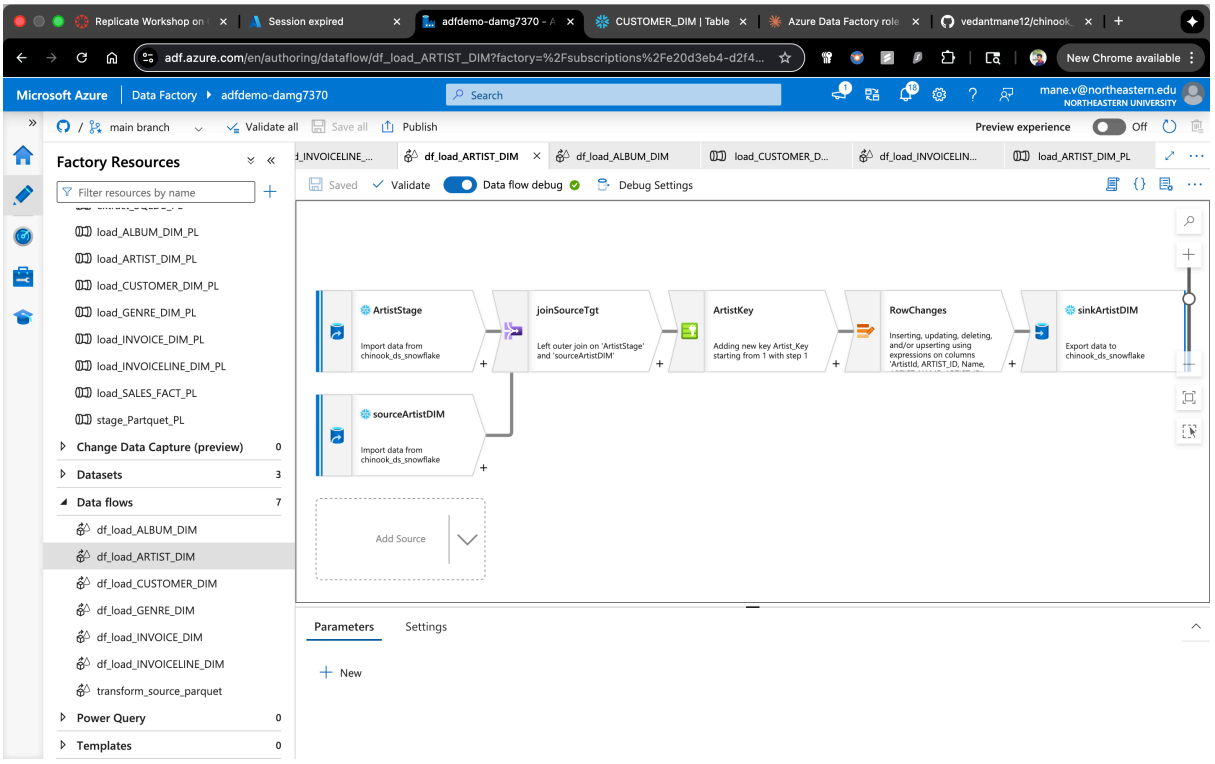


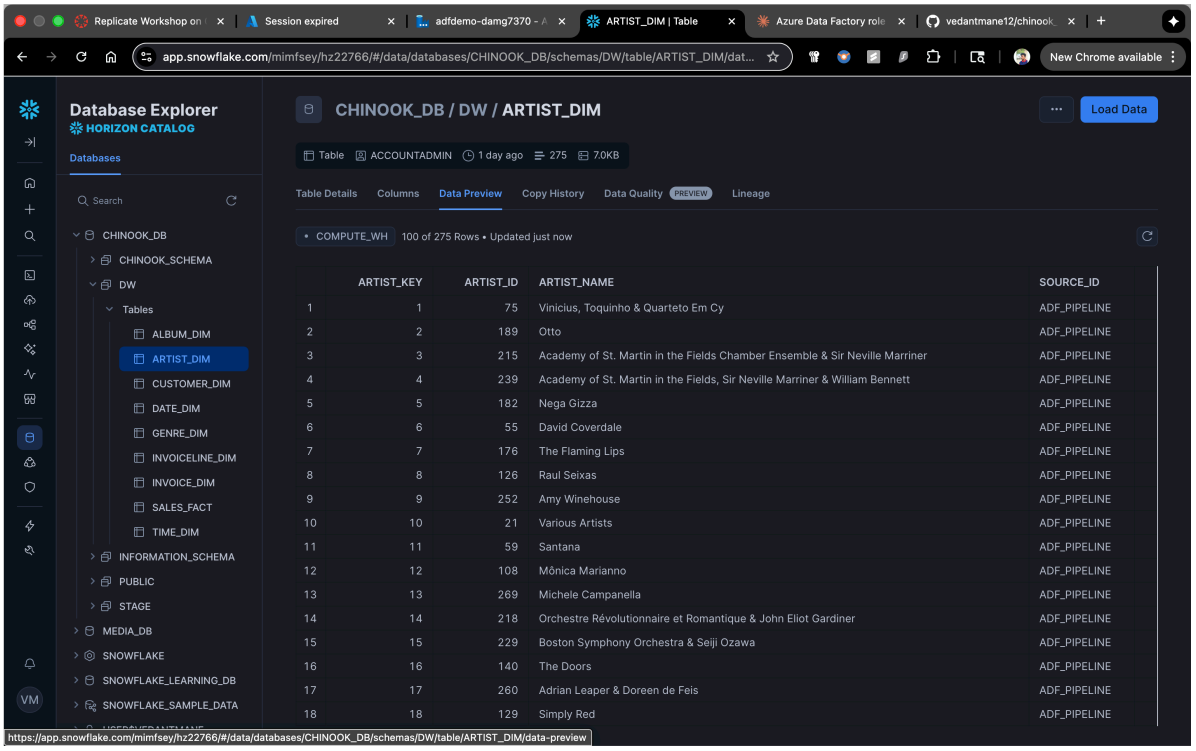
Load CUSTOMER_DIM from Snowflake STAGE



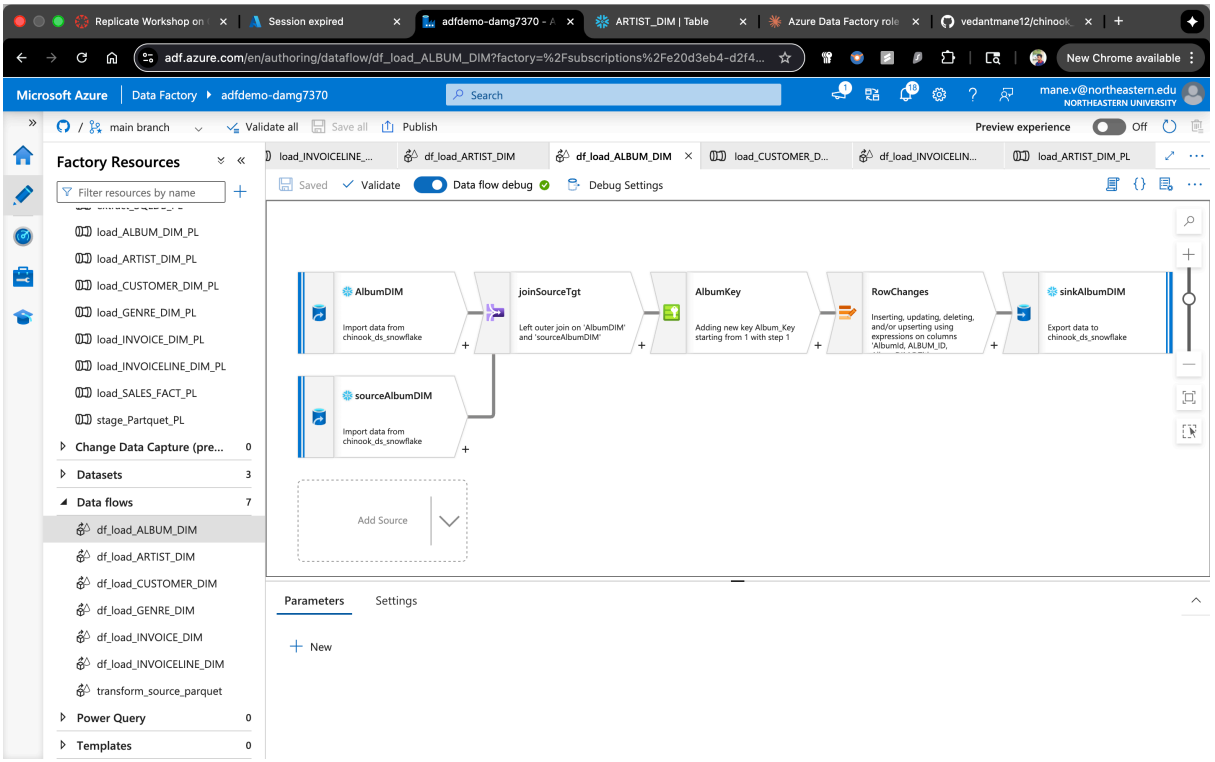


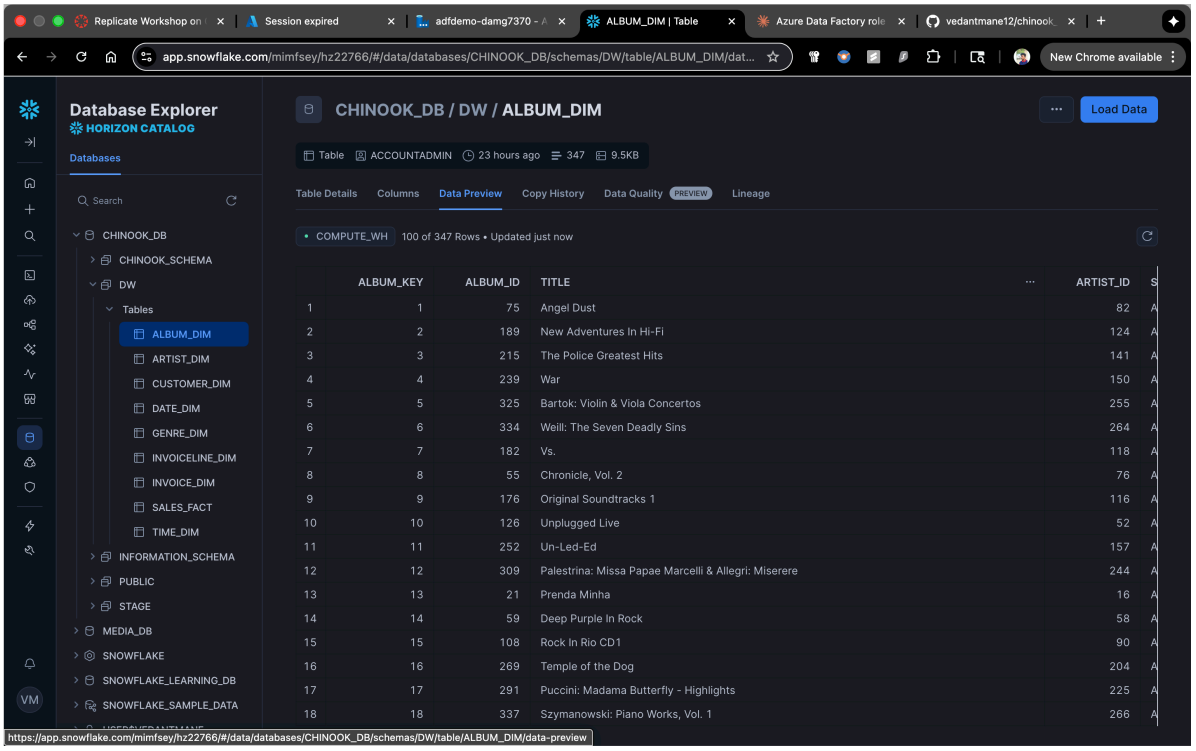
Load ARTIST_DIM from Snowflake STAGE



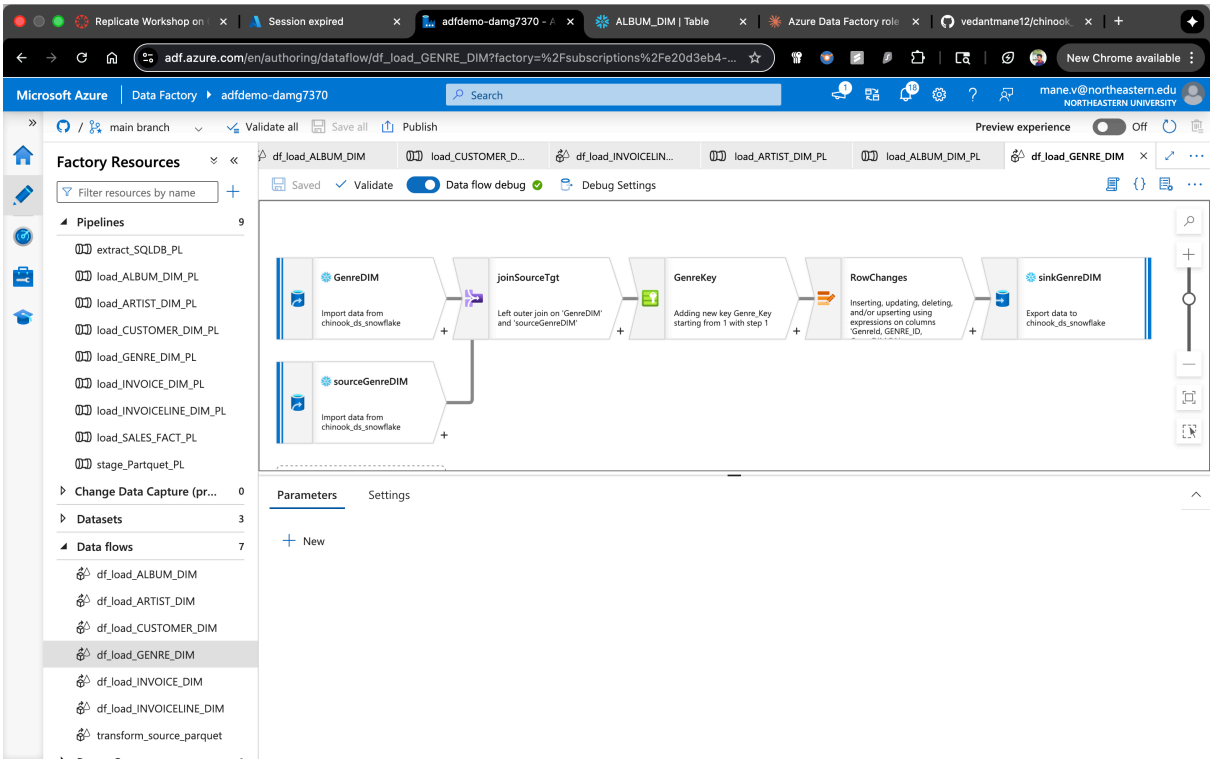


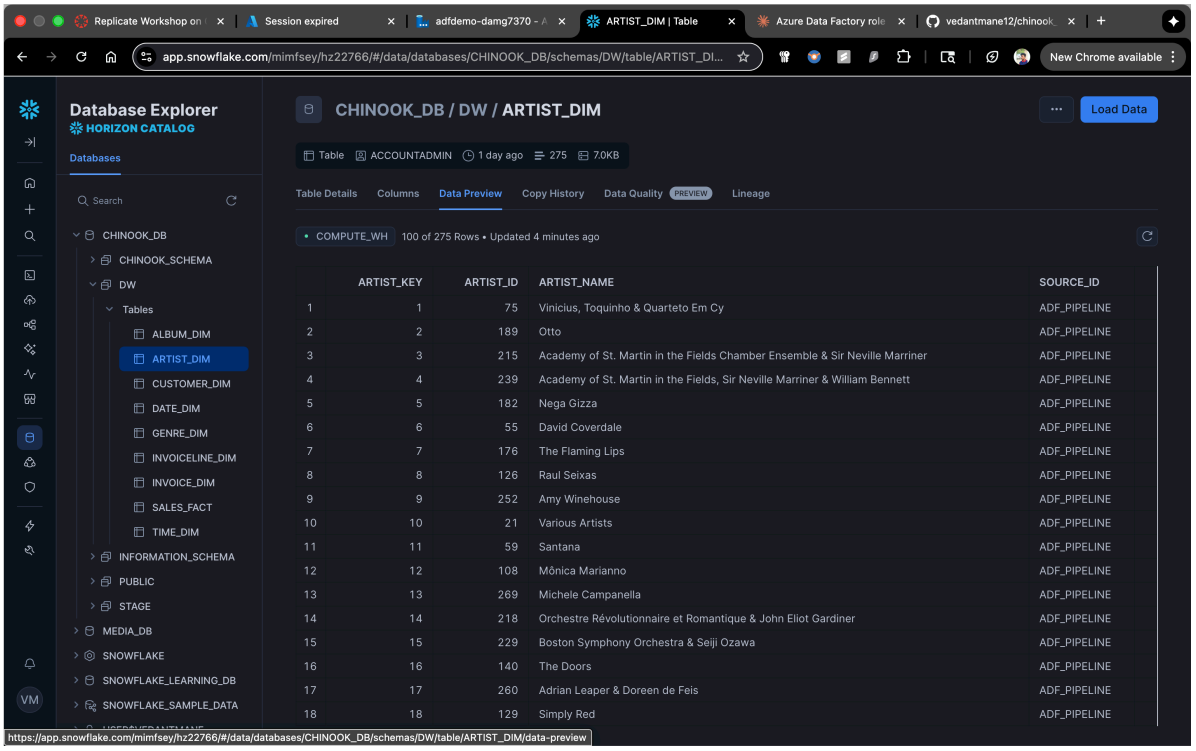
Load ALBUM_DIM from Snowflake STAGE



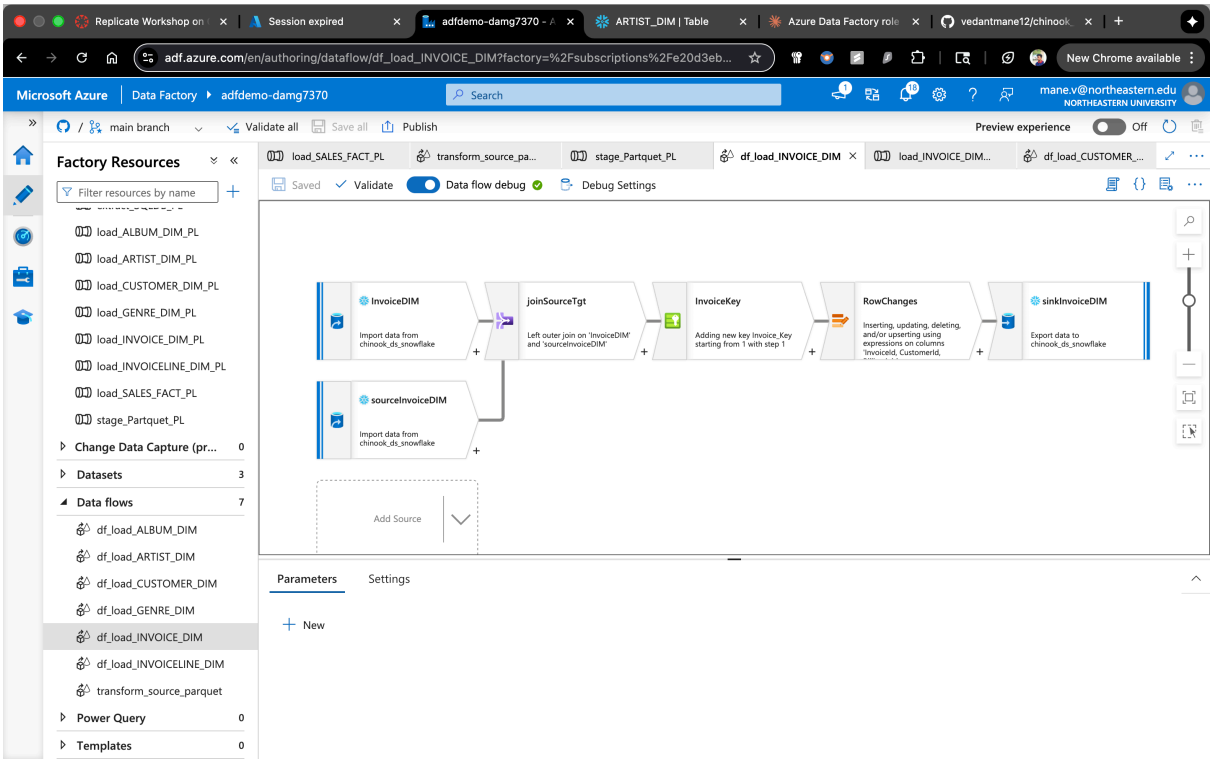


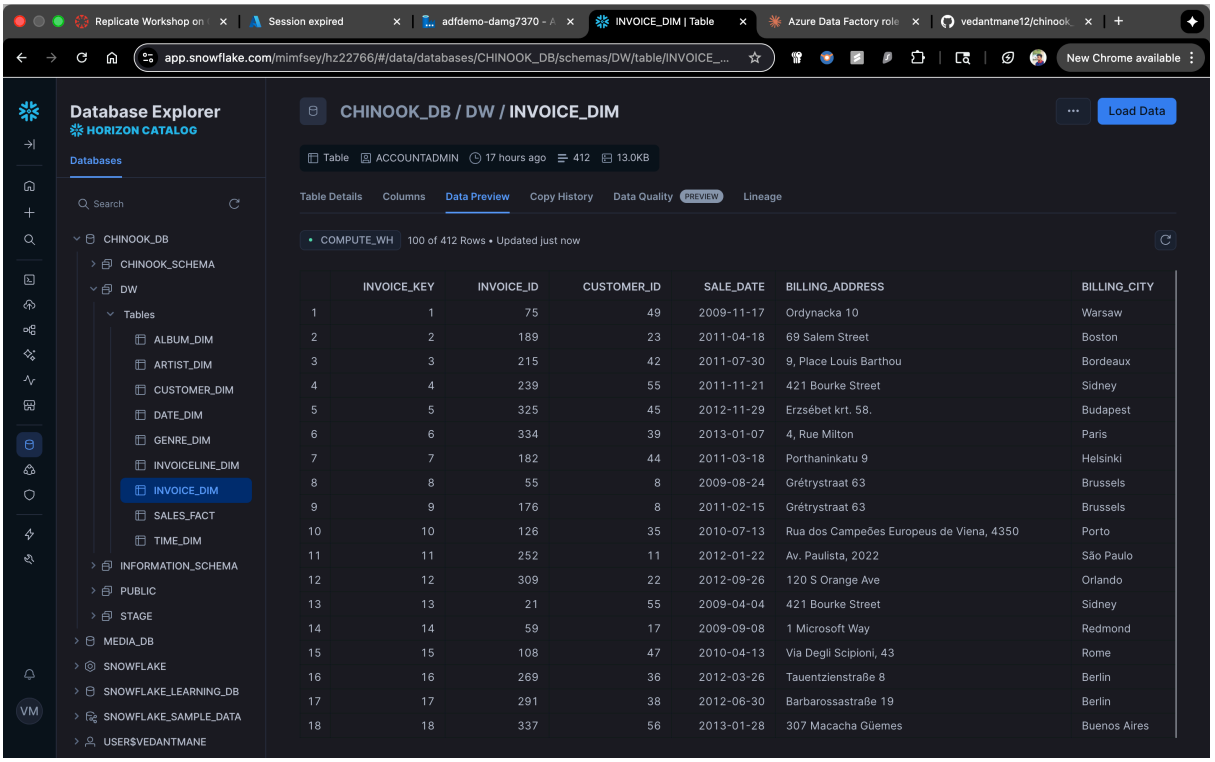
Load GENRE_DIM from Snowflake STAGE





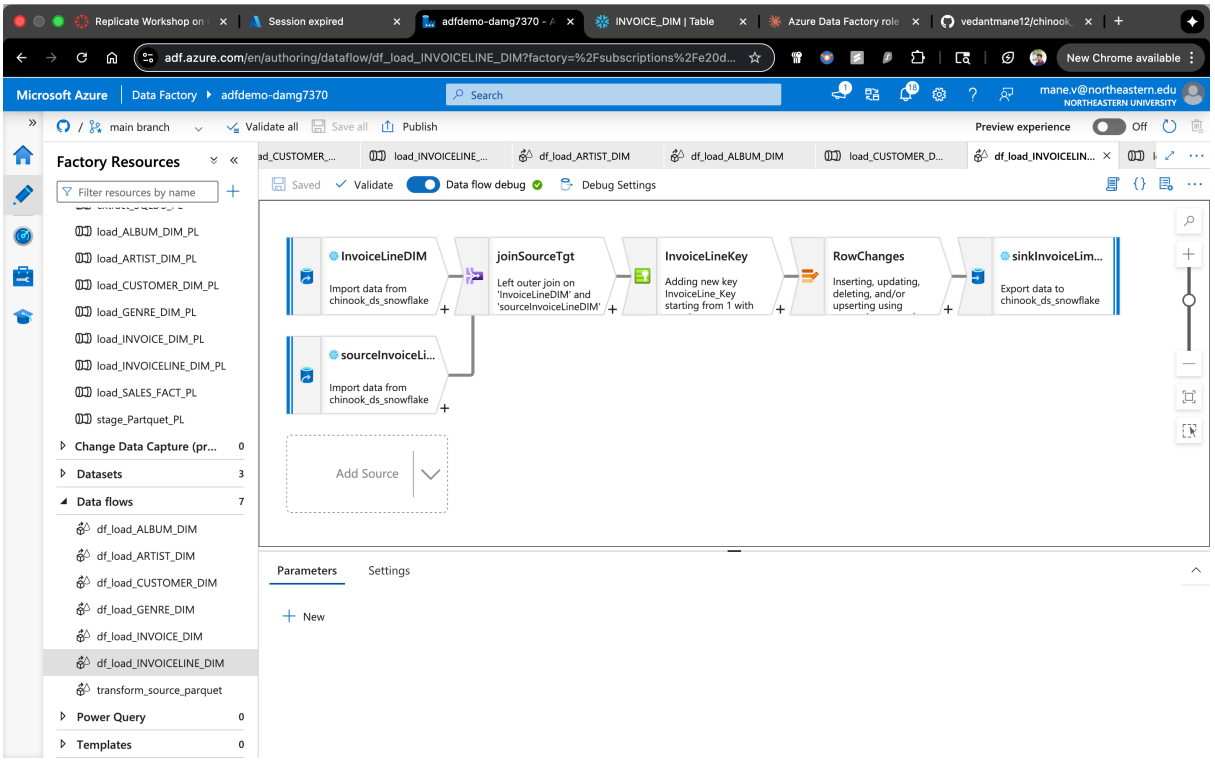
Load INVOICE_DIM from Snowflake STAGE

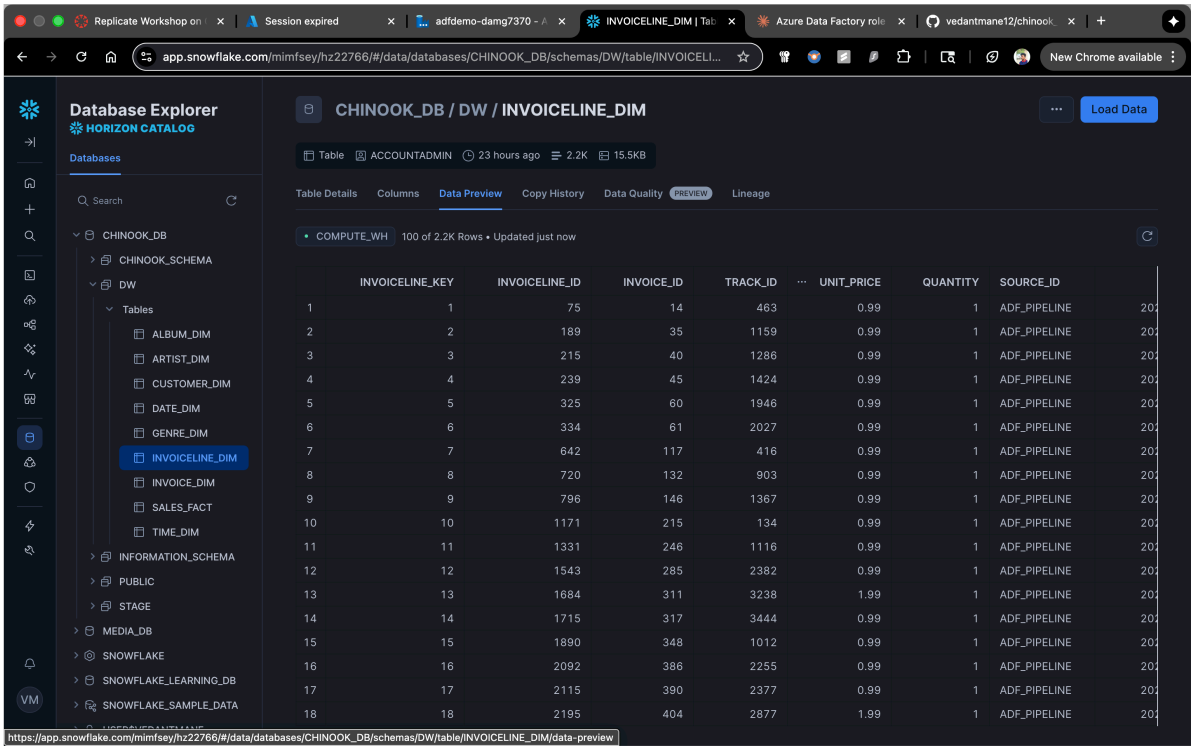




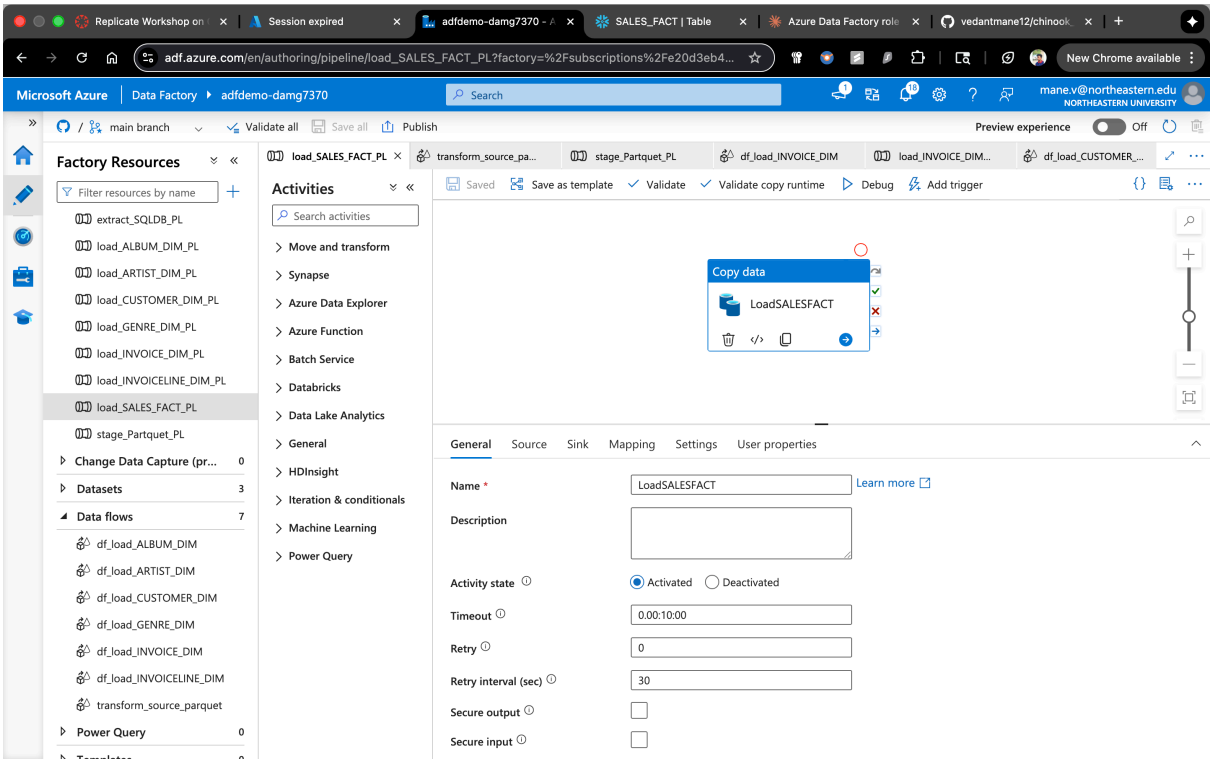
	INVOICE_KEY	INVOICE_ID	CUSTOMER_ID	SALE_DATE	BILLING_ADDRESS	BILLING_CITY
1	1	75	49	2009-11-17	Ordynacka 10	Warsaw
2	2	189	23	2011-04-18	69 Salem Street	Boston
3	3	215	42	2011-07-30	9, Place Louis Barthou	Bordeaux
4	4	239	55	2011-11-21	421 Bourke Street	Sidney
5	5	325	45	2012-11-29	Erzsébet krt. 58.	Budapest
6	6	334	39	2013-01-07	4, Rue Milton	Paris
7	7	182	44	2011-03-18	Porthaninkatu 9	Helsinki
8	8	55	8	2009-08-24	Grétrystraat 63	Brussels
9	9	176	8	2011-02-15	Grétrystraat 63	Brussels
10	10	126	35	2010-07-13	Rua dos Campeões Europeus de Viena, 4350	Porto
11	11	252	11	2012-01-22	Av. Paulista, 2022	São Paulo
12	12	309	22	2012-09-26	120 S Orange Ave	Orlando
13	13	21	55	2009-04-04	421 Bourke Street	Sidney
14	14	59	17	2009-09-08	1 Microsoft Way	Redmond
15	15	108	47	2010-04-13	Via Degli Scipioni, 43	Rome
16	16	269	36	2012-03-26	Tauentzienstraße 8	Berlin
17	17	291	38	2012-06-30	Barbarossastraße 19	Berlin
18	18	337	56	2013-01-28	307 Macacha Güemes	Buenos Aires

Load INVOICELINE_DIM from Snowflake STAGE



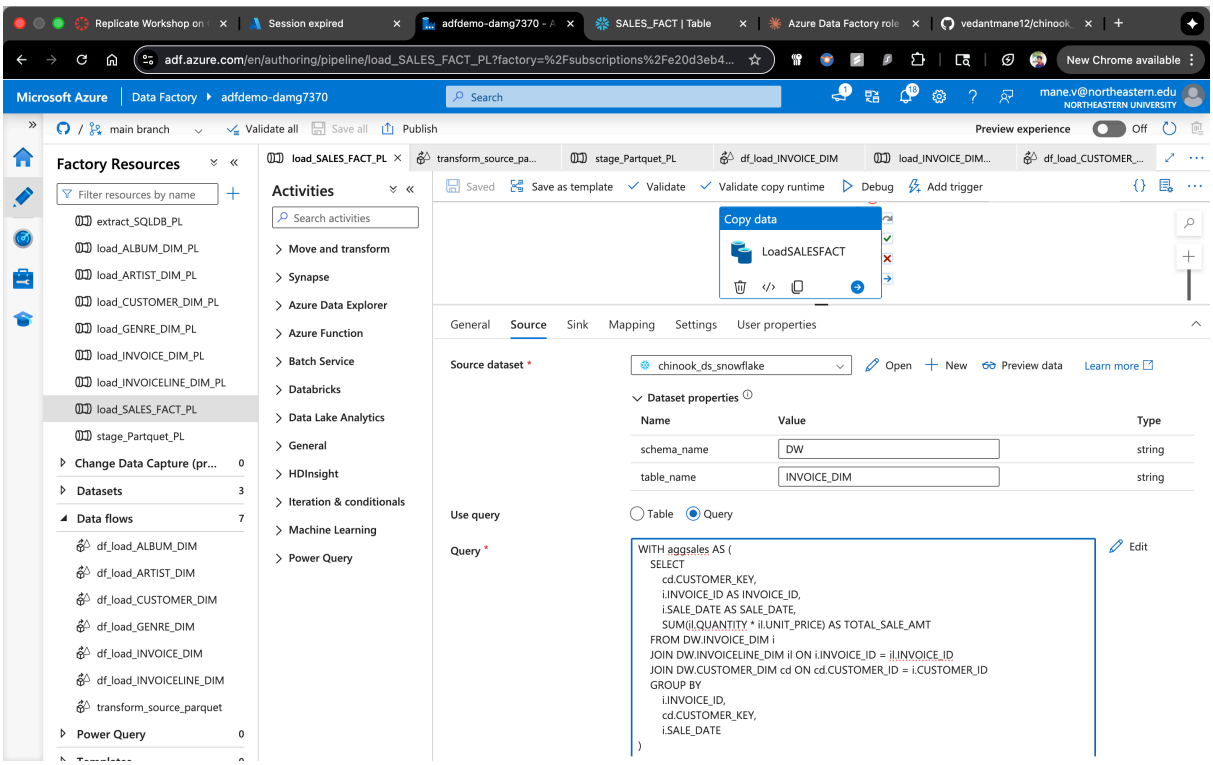


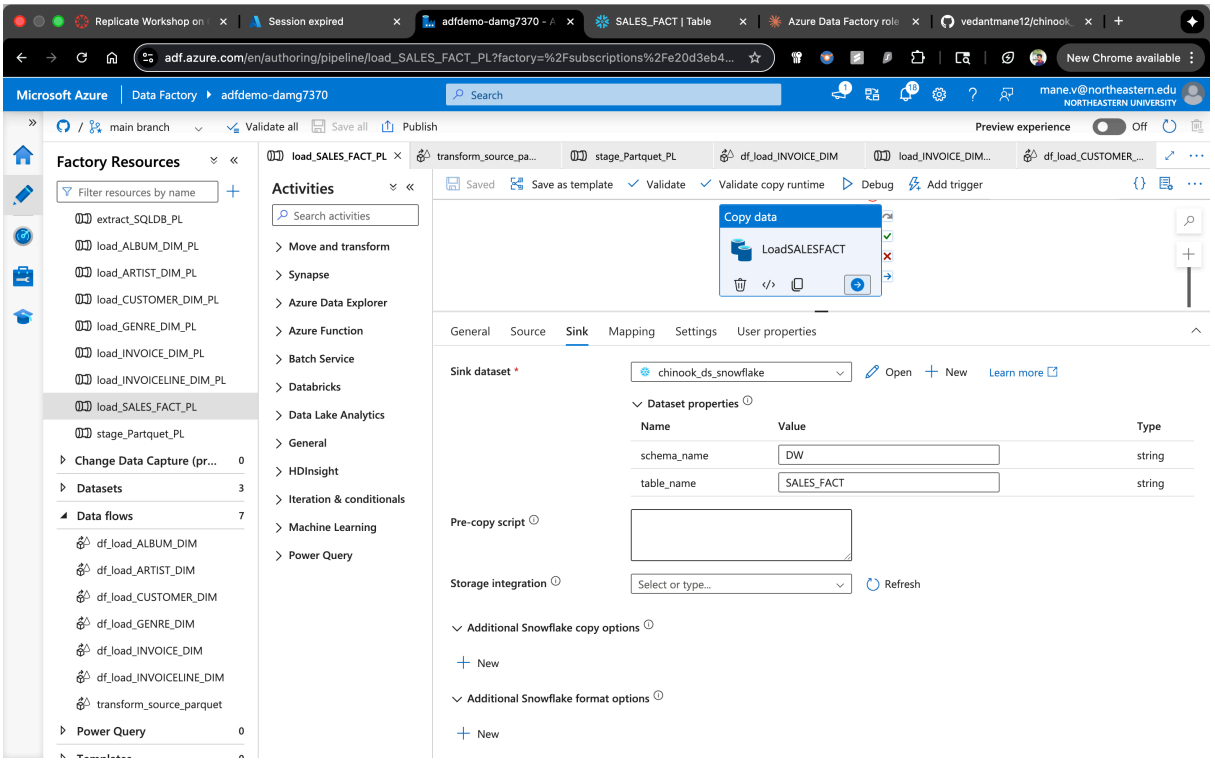
Load SALES_FACT from Snowflake WH

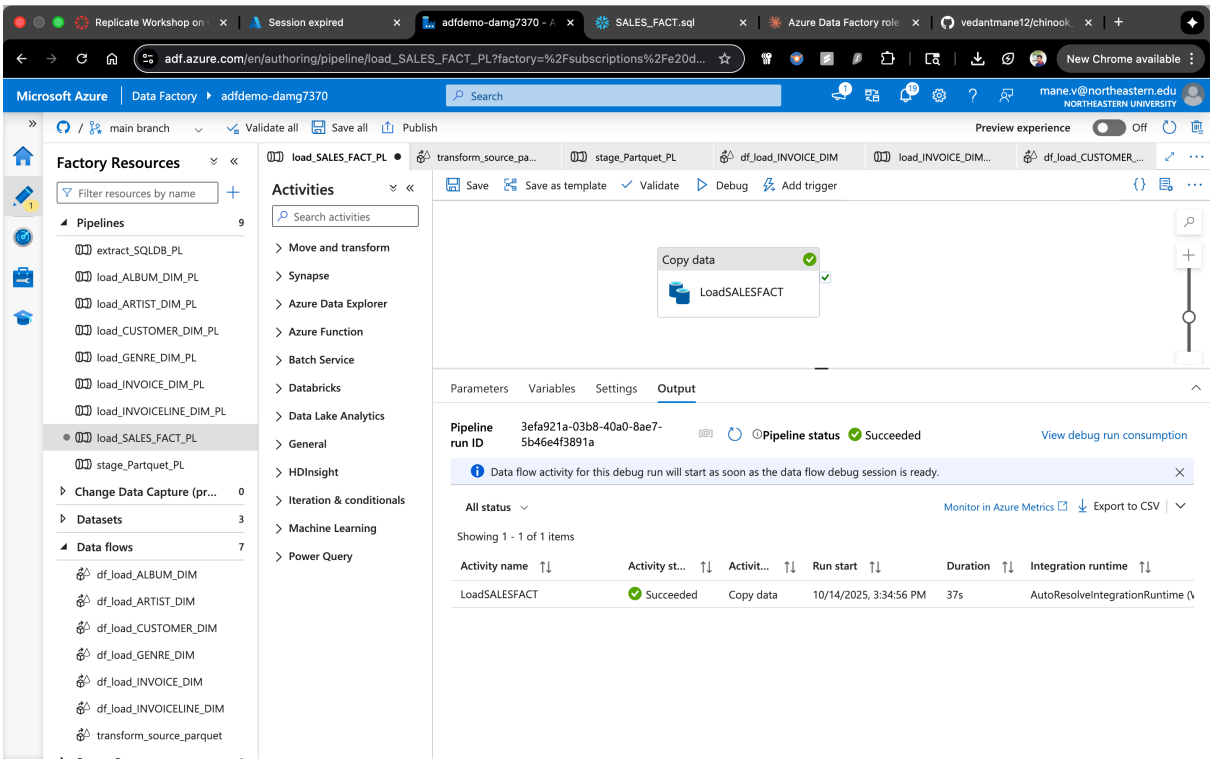


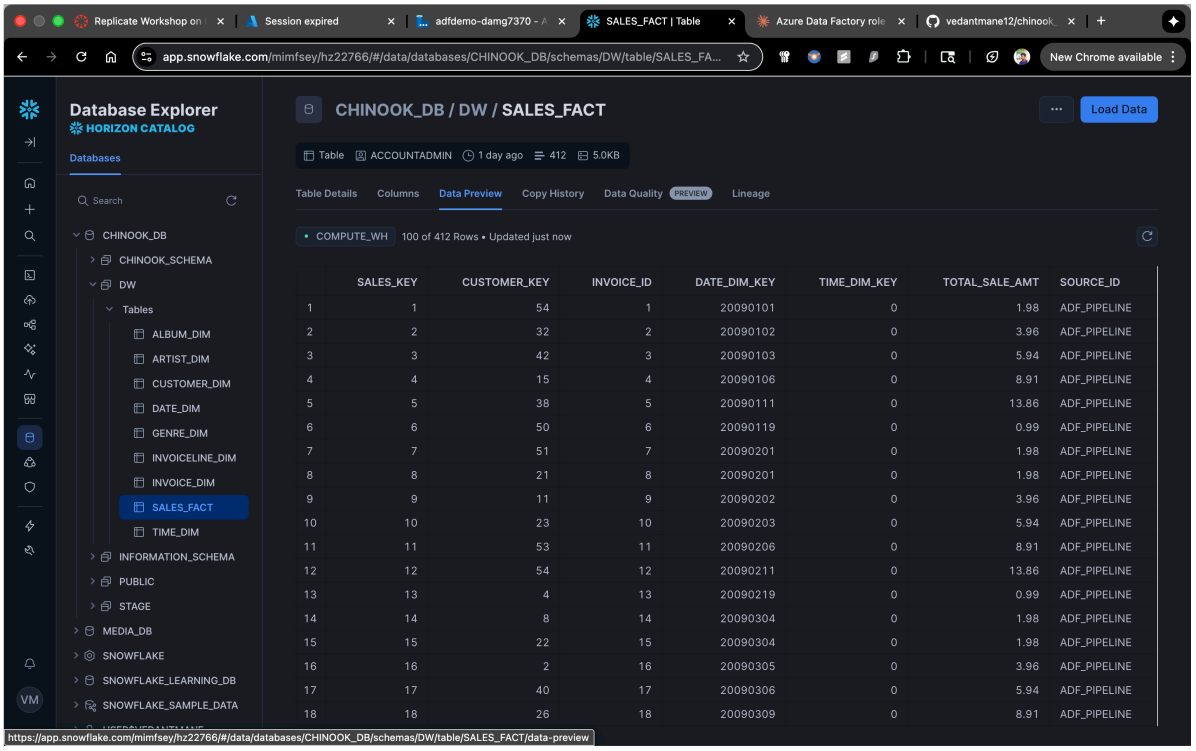

```
WITH aggsales AS (
  SELECT
    cd.CUSTOMER_KEY,
    i.INVOICE_ID AS INVOICE_ID,
    i.SALE_DATE AS SALE_DATE,
```

```
    SUM(il.QUANTITY * il.UNIT_PRICE) AS TOTAL_SALE_AMT
  FROM DW.INVOICE_DIM i
  JOIN DW.INVOICELINE_DIM il ON i.INVOICE_ID = il.INVOICE_ID
  JOIN DW.CUSTOMER_DIM cd ON cd.CUSTOMER_ID = i.CUSTOMER_ID
  GROUP BY
    i.INVOICE_ID,
    cd.CUSTOMER_KEY,
    i.SALE_DATE
)
SELECT
  STAGE.SALES_FACT_SEQ.nextval AS SALES_KEY,
  CUSTOMER_KEY,
  INVOICE_ID,
  -- SALE_DATE,
  dd.DATE_KEY AS DATE_DIM_KEY,
  td.TIME_KEY AS TIME_DIM_KEY,
  TOTAL_SALE_AMT,
  'ADF_PIPELINE' AS SOURCE_ID,
  CURRENT_TIMESTAMP() AS DATE_TO_WAREHOUSE
FROM aggsales a
JOIN DW.DATE_DIM dd ON dd.FULL_DATE = DATE(a.SALE_DATE)
JOIN DW.TIME_DIM td ON td.TIME_24_HR = TO_CHAR(a.SALE_DATE, 'HH24:MI')
WHERE NOT EXISTS (
  SELECT 1
  FROM DW.SALES_FACT sf
  WHERE sf.INVOICE_ID = a.INVOICE_ID
);
```




Factory Resources
load_SALES_FACT_PL
Activities
Copy data
LoadSALESFACT
General
Source
Sink
Mapping
Settings
User properties
Sink dataset *
chinook_ds_snowflake
Dataset properties
schema_name
DW
table_name
SALES_FACT
Pre-copy script
Storage integration
Additional Snowflake copy options
Additional Snowflake format options


Factory Resources
load_SALES_FACT_PL
Activities
Copy data
LoadSALESFACT
Parameters
Variables
Settings
Output
Pipeline run ID
3efa921a-03b8-40a0-8ae7-5b46e4f3891a
Pipeline status
Succeeded
Data flow activity for this debug run will start as soon as the data flow debug session is ready.
Showing 1 - 1 of 1 items
Activity name
Activity st...
Activit...
Run start
Duration
Integration runtime
LoadSALESFACT
Succeeded
Copy data
10/14/2025, 3:34:56 PM
37s



Why Use JOIN for loading SALES_FACT?

Overview

When loading the `SALES_FACT` table, we use **INNER JOIN** for dimension lookups instead of LEFT JOIN. This design choice ensures data integrity, enforces business rules, and maintains referential integrity in our star schema.

Joins Used in SALES_FACT Load

INNER JOINs (Dimension Lookups)

```
-- Invoice to Invoice Lines
FROM DW.INVOICE_DIM i
JOIN DW.INVOICELINE_DIM il ON i.INVOICE_ID = il.INVOICE_ID

-- Invoice to Customer
JOIN DW.CUSTOMER_DIM cd ON cd.CUSTOMER_ID = i.CUSTOMER_ID

-- Sales to Date Dimension
JOIN DW.DATE_DIM dd ON dd.FULL_DATE = DATE(a.SALE_DATE)

-- Sales to Time Dimension
JOIN DW.TIME_DIM td ON td.TIME_24_HR = TO_CHAR(a.SALE_DATE, 'HH24:MI')
```

LEFT JOIN (Incremental Load Only)

```
-- Anti-join pattern to find new records
LEFT JOIN DW.SALES_FACT sf ON sf.INVOICE_ID = a.INVOICE_ID
WHERE sf.INVOICE_ID IS NULL
```

Why INNER JOIN?

1. Enforces Referential Integrity

Our `SALES_FACT` table has foreign key constraints:

```
CONSTRAINT FK_SALES_DATE FOREIGN KEY (DATE_DIM_KEY) REFERENCES DW.DATE_DIM(DATE_KEY)
CONSTRAINT FK_SALES_TIME FOREIGN KEY (TIME_DIM_KEY) REFERENCES DW.TIME_DIM(TIME_KEY)
CONSTRAINT FK_SALES_CUSTOMER FOREIGN KEY (CUSTOMER_KEY) REFERENCES DW.CUSTOMER_DIM(C
USTOMER_KEY)
```

INNER JOIN ensures:

- Every fact record has valid dimension keys
- No NULL foreign keys that would violate constraints
- Insert operations succeed without constraint violations

What happens with LEFT JOIN:

- Missing dimensions would create NULL foreign keys
- Foreign key constraint violations would cause pipeline failures
- Less clear error messages

---

2. Acts as Data Quality Gate

INNER JOIN automatically **excludes incomplete or invalid records**:

Issue	INNER JOIN Behavior	LEFT JOIN Behavior
Missing customer	Record excluded ✅	NULL customer_key → Fails insert ❌
Date outside DATE_DIM range	Record excluded ✅	NULL date_key → Fails insert ❌
Invoice without line items	Record excluded ✅	NULL amounts → Bad data ❌

Result: Only complete, valid records enter the fact table.

---

3. Implements Business Rules

Certain relationships are **mandatory** from a business perspective:

- **Every sale MUST have a customer** - Cannot have anonymous sales
- **Every invoice MUST have line items** - No invoice without products
- **Every sale MUST have a date/time** - Required for time-series analysis

INNER JOIN enforces these rules at the data level, preventing logically invalid records.

---

4. Maintains Star Schema Integrity

In dimensional modeling (Kimball methodology):

- Fact tables should always reference valid dimension keys
- NULL foreign keys break analytical queries
- All dimension lookups must succeed for reliable reporting

INNER JOIN guarantees:

- All aggregations work correctly
- No NULL handling needed in BI queries
- Query performance remains optimal

When We Use LEFT JOIN

Only for incremental loading (anti-join pattern):

```
LEFT JOIN DW.SALES_FACT sf ON sf.INVOICE_ID = a.INVOICE_ID
WHERE sf.INVOICE_ID IS NULL
```

Purpose:

- Find invoices that DON'T exist in SALES_FACT yet
- Prevent duplicate loads
- This is NOT a dimension lookup - it's a duplicate-check

What Would Happen with LEFT JOIN for Dimensions?

Example: Missing Customer

```
-- Using LEFT JOIN (wrong approach)
LEFT JOIN DW.CUSTOMER_DIM cd ON cd.CUSTOMER_ID = i.CUSTOMER_ID
```

Result:

- Invoice with missing customer has `CUSTOMER_KEY = NULL`
- Foreign key constraint violation during insert
- Pipeline fails with cryptic error
- Harder to identify which records caused the issue

Example: Missing Date

```
-- Using LEFT JOIN (wrong approach)
LEFT JOIN DW.DATE_DIM dd ON dd.FULL_DATE = DATE(a.SALE_DATE)
```

Result:

- Sales outside DATE_DIM range get `DATE_DIM_KEY = NULL`
- Cannot analyze sales by time period
- Foreign key constraint fails
- Less obvious what the root cause is

Summary

Aspect	Why INNER JOIN
Data Integrity	Prevents NULL foreign keys and constraint violations
Data Quality	Excludes incomplete records automatically
Business Rules	Enforces mandatory relationships
Star Schema	Maintains dimensional model integrity
Error Detection	Clear failure points when dimensions are missing

The only LEFT JOIN in our query is for the incremental load check (anti-join pattern), which serves a different purpose: identifying new records that haven't been loaded yet.

This strategy ensures `SALES_FACT` contains only **complete, valid, and analytically useful** records for reliable reporting and analysis.

Key Takeaway

> INNER JOIN = Data Quality Gate

> If a record can't join to all required dimensions, it shouldn't be in the fact table. This prevents bad data from entering the warehouse and ensures downstream analytics remain reliable.