

Automated ETL Pipeline for Data Ingestion and SCD2 Implementation Using AWS and Snowflake and PowerBI

Name: **HIMA SREE CHALASANI**

Role: Data Analyst Trainee

This is the S3 bucket named “ssttrainingdata” that I am accessing for my data needs. This bucket contains the data files that I use in my Snowflake environment.

The screenshot shows the AWS S3 console interface. On the left, a sidebar menu includes options like Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens, Dashboards, Storage Lens groups, and AWS Organizations settings. The main panel displays two tabs: "General purpose buckets (17)" and "Objects (5)".

General purpose buckets (17) Info

Name	AWS Region	IAM Access Analyzer	Creation date
383493188589-8uh8gsffw83	US East (N. Virginia) us-east-1	View analyzer for us-east-1	(UTC-04:00)
sagemaker-studio-383493188589-f04qnipn1fj	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 21, 2024, 12:23:20 (UTC-04:00)
sagemaker-us-east-1-383493188589	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 21, 2024, 12:23:22 (UTC-04:00)
sagemaker-us-east-2-383493188589	US East (Ohio) us-east-2	View analyzer for us-east-2	May 21, 2024, 14:35:01 (UTC-04:00)
ssttrainingdata	US East (Ohio) us-east-2	View analyzer for us-east-2	February 22, 2024, 23:18:11 (UTC-05:00)
workshop-bucket-383493188589	US East (Ohio) us-east-2	View analyzer for us-east-2	April 2, 2024, 15:28:28 (UTC-04:00)

Objects (5) Info

Name	Type	Last modified	Size	Storage class
customers.csv	csv	August 13, 2024, 06:00:17 (UTC-04:00)	1.5 KB	Standard
dimdates.csv	csv	February 22, 2024, 23:54:04 (UTC-05:00)	5.9 MB	Standard
products.csv	csv	August 13, 2024, 06:00:17 (UTC-04:00)	2.3 KB	Standard
sales.csv	csv	August 13, 2024, 06:00:17 (UTC-04:00)	364.1 KB	Standard
stores.csv	csv	February 22, 2024, 23:54:07 (UTC-05:00)	163.0 B	Standard

I created a policy that allows Snowflake to connect to the bucket securely. This policy is associated with an AWS IAM role, which grants the necessary permissions for Snowflake to read the files in the bucket.

Policy name : himasree_policy

The screenshot shows the AWS IAM Policies page. The left sidebar includes options for Identity providers, Account settings, Access reports, Access Analyzer, External access, Unused access, and Analyzer settings. The main panel shows the "himasree_policy" details.

himasree_policy Info

Policy details

Type	Creation time	Edited time	ARN
Customer managed	August 07, 2024, 17:01 (UTC-04:00)	August 07, 2024, 17:01 (UTC-04:00)	arn:aws:iam::383493188589:policy/himasree_policy

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose Show remaining. [Learn more](#)

Summary | JSON

Create an IAM Role: I created an AWS Identity and Access Management (IAM) role specifically for Snowflake to access the S3 bucket. This role defines the permissions needed to interact with the bucket.

Attach Policy to the Role: I attached a policy to this IAM role that grants read access to the S3 bucket ssttrainingdata. The policy specifies which actions Snowflake can perform, such as listing and reading files.

Role: himasree_role

The screenshot shows the AWS IAM Roles page. On the left, the navigation menu includes 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'Roles' selected), 'Policies', 'Identity providers', 'Account settings', 'Access reports' (with 'Analyzer settings' selected), and 'CloudShell Feedback'. The main content area shows the 'himasree_role' details. The 'Summary' tab displays creation date (August 07, 2024, 17:12 UTC-04:00), ARN (arn:aws:iam::383493188589:role/himasree_role), last activity (4 days ago), and maximum session duration (1 hour). The 'Trust relationships' tab shows a JSON policy document:

```

1  [{}]
2  {
3    "Version": "2012-10-17",
4    "Statement": [
5      {
6        "Sid": "",
7        "Effect": "Allow",
8        "Principal": {
9          "AWS": "arn:aws:iam::008971645028:user/julp0000-s"
10         },
11        "Action": "sts:AssumeRole",
12        "Condition": {
13          "StringEquals": {
14            "sts:ExternalId": "NKB10110_SFCRole=2_bEKfz1AHR8q1CkhHzx3bQhT32iw="
15          }
16        }
17      }
18    ]
}

```

This screenshot is identical to the one above, showing the 'himasree_role' summary and trust relationships. The JSON policy document is shown in its entirety:

```

1  [{}]
2  {
3    "Version": "2012-10-17",
4    "Statement": [
5      {
6        "Sid": "",
7        "Effect": "Allow",
8        "Principal": {
9          "AWS": "arn:aws:iam::008971645028:user/julp0000-s"
10         },
11        "Action": "sts:AssumeRole",
12        "Condition": {
13          "StringEquals": {
14            "sts:ExternalId": "NKB10110_SFCRole=2_bEKfz1AHR8q1CkhHzx3bQhT32iw="
15          }
16        }
17      }
18    ]
}

```

TASK : Creating Tables and Loading Data from Amazon AWS S3 Bucket to Snowflake Data Warehouse Staging Tables

- Created a Snowflake Warehouse, Database, Schema

```
create warehouse HIMA_WH;
create database HIMA_DB;
create schema HIMA_DB.STAGING;
```

- Created Storage Integration

Storage integration allows Snowflake to securely access your S3 bucket without directly exposing AWS credentials.

```
create or replace storage integration Hima_OBJ
type = external_stage
storage_provider = s3
enabled = true
storage_aws_role_arn = 'arn:aws:iam::383493188589:role/himasree_role'
storage_allowed_locations = ('s3://ssttrainingdata/');
```

```
desc integration Hima_OBJ;
```

- Create File Format

File formats define how Snowflake reads data files from S3. This setup will handle CSV files.

```
create or replace file format csv_format type = csv field_optionally_enclosed_by = ""
field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null') empty_field_as_null = true;
```

- Set Up the External Stage in Snowflake

Created an External Stage: An external stage points to your S3 bucket and tells Snowflake where to find the data. To do this, configure the external stage with the S3 bucket URL, file format, and the necessary credentials.

```
create or replace stage hima_stage_2024
storage_integration = Hima_OBJ
url = 's3://ssttrainingdata/'
file_format = csv_format;
```

```

1 -- TASK 1 Creating Tables, Load Data from Amazon AWS S3 Bucket to Snowflake Data Warehouse
2
3 create warehouse HIMA_WH;
4 create database HIMA_DB;
5 create schema HIMA_DB.STAGING;
6
7 create or replace storage integration Hima_OBJ
8 type = external_stage
9 storage_provider = s3
10 enabled = true
11 storage_aws_role_arn = 'arn:aws:iam::383493188589:role/himasree_role'
12 storage_allowed_locations = ('s3://ssstrainingdata/');
13
14 desc integration Hima_OBJ;
15
16 create or replace file format csv_format type = csv field Optionally_enclosed_by = ''' field_delimiter =
17 |
18 create or replace stage hima_stage_2024
19 storage_integration = Hima_OBJ
20 url = 's3://ssstrainingdata/'
21 file_format = csv_format;
22

```

Ask Copilot

STAGING LAYER

- Created Staging Tables in Snowflake

CREATE OR REPLACE TABLE staging_customers(

CUSTOMER_ID text NOT NULL, CUSTOMER_NAME text, DATE_OF_BIRTH date, LOAD_TIME TIMESTAMP,
 PRIMARY KEY (CUSTOMER_ID)

);

CREATE OR REPLACE TABLE staging_products(

PRODUCT_ID text NOT NULL, PRODUCT_NAME text, PRICE float,
 LOAD_TIME TIMESTAMP,
 PRIMARY KEY (PRODUCT_ID)

);

CREATE OR REPLACE TABLE staging_stores(

STORE_ID text NOT NULL, STORE_NAME text, LOAD_TIME TIMESTAMP,
 PRIMARY KEY (STORE_ID)

);

CREATE OR REPLACE TABLE staging_sales (

TRANSACTION_ID INTEGER, PRODUCT_ID TEXT NOT NULL,
 CUSTOMER_ID TEXT NOT NULL,
 STORE_ID TEXT NOT NULL, T_DATE DATE, QUANTITY INTEGER,
 LOAD_TIME TIMESTAMP,
 PRIMARY KEY (TRANSACTION_ID),
 FOREIGN KEY (PRODUCT_ID) REFERENCES staging_products(PRODUCT_ID),

```

FOREIGN KEY (CUSTOMER_ID) REFERENCES
staging_customers(CUSTOMER_ID),
    FOREIGN KEY (STORE_ID) REFERENCES staging_stores(STORE_ID)
);

```

```

CREATE OR REPLACE TABLE staging_dimdates (
    Id INT, Date DATE, DateLongDescription VARCHAR, DateShortDescription VARCHAR,
    DayLongName VARCHAR,
    DayShortName VARCHAR, MonthLongName VARCHAR, MonthShortName VARCHAR,
    CalendarDay INT, CalendarWeek INT,
    CalendarWeekStartDateId INT, CalendarWeekEndDateId INT, CalendarDayInWeek INT,
    CalendarMonth INT,
    CalendarMonthStartDateId INT, CalendarMonthEndDateId INT,
    CalendarNumberOfDaysInMonth INT,
    CalendarDayInMonth INT, CalendarQuarter INT, CalendarQuarterStartDateId INT,
    CalendarQuarterEndDateId INT,
    CalendarNumberOfDaysInQuarter INT, CalendarDayInQuarter INT, CalendarYear INT,
    CalendarYearStartDateId INT,
    CalendarYearEndDateId INT, CalendarNumberOfDaysInYear INT, FiscalDay INT,
    FiscalWeek INT,
    FiscalWeekStartDateId INT, FiscalWeekEndDateId INT, FiscalDayInWeek INT, FiscalMonth
    INT,
    FiscalMonthStartDateId INT, FiscalMonthEndDateId INT, FiscalNumberOfDaysInMonth
    INT, FiscalDayInMonth INT,
    FiscalQuarter INT, FiscalQuarterStartDateId INT, FiscalQuarterEndDateId INT,
    FiscalNumberOfDaysInQuarter INT,
    FiscalDayInQuarter INT, FiscalYear INT, FiscalYearStartDateId INT, FiscalYearEndDateId
    INT,
    FiscalNumberOfDaysInYear INT, LOAD_TIME TIMESTAMP
);

```

```

1. Create WH,DB,Stage+ St...
2. Bronze Schema Tables
3. Silver Schema Tables
4. Create Pipeline schema ...
+ ▾

Databases Worksheets
+ Search objects HIMA_DB.STAGING Settings Code Versions
HIMA_DB
SNOWFLAKE
SNOWFLAKE_SAMPLE_DATA

22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

CREATE OR REPLACE TABLE staging_customers(
    CUSTOMER_ID text NOT NULL, CUSTOMER_NAME text, DATE_OF_BIRTH date, LOAD_TIME TIMESTAMP,
    PRIMARY KEY (CUSTOMER_ID)
);

CREATE OR REPLACE TABLE staging_products(
    PRODUCT_ID text NOT NULL, PRODUCT_NAME text, PRICE float, LOAD_TIME TIMESTAMP,
    PRIMARY KEY (PRODUCT_ID)
);

CREATE OR REPLACE TABLE staging_stores(
    STORE_ID text NOT NULL, STORE_NAME text, LOAD_TIME TIMESTAMP,
    PRIMARY KEY (STORE_ID)
);

CREATE OR REPLACE TABLE staging_sales (
    TRANSACTION_ID INTEGER, PRODUCT_ID TEXT NOT NULL, CUSTOMER_ID TEXT NOT NULL,
    STORE_ID TEXT NOT NULL, DATE_DATE DATE, QUANTITY INTEGER, LOAD_TIME TIMESTAMP
);

Ask Copilot
Type here to search
66°F 10:59 AM ENG 08-09-2024

```

The screenshot shows a Snowflake database interface. On the left, there's a sidebar with icons for databases, worksheets, and other tools. The main area is a code editor with tabs for different parts of the pipeline: 1. Create WHI_DB.Stage+ St..., 2. Bronze Schema Tables, 3. Silver Schema Tables, and 4. Create Pipeline schema The current tab is '1. Create WHI_DB.Stage+ St...'. The code in the editor is as follows:

```

CREATE OR REPLACE TABLE staging_sales (
    TRANSACTION_ID INTEGER, PRODUCT_ID TEXT NOT NULL, CUSTOMER_ID TEXT NOT NULL,
    STORE_ID TEXT NOT NULL, T_DATE DATE, QUANTITY INTEGER, LOAD_TIME TIMESTAMP,
    PRIMARY KEY (TRANSACTION_ID),
    FOREIGN KEY (PRODUCT_ID) REFERENCES staging_products(PRODUCT_ID),
    FOREIGN KEY (CUSTOMER_ID) REFERENCES staging_customers(CUSTOMER_ID),
    FOREIGN KEY (STORE_ID) REFERENCES staging_stores(STORE_ID)
);

CREATE OR REPLACE TABLE staging_dimdates (
    Id INT, Date DATE, DateLongDescription VARCHAR, DateShortDescription VARCHAR, DayLongName VARCHAR,
    DayShortName VARCHAR, MonthLongName VARCHAR, MonthShortName VARCHAR, CalendarDay INT, CalendarWeek INT,
    CalendarWeekStartDate INT, CalendarWeekEndDate INT, CalendarDayInWeek INT, CalendarMonth INT,
    CalendarMonthStartDate INT, CalendarMonthEndDate INT, CalendarNumberOfDaysInMonth INT,
    CalendarDayInMonth INT, CalendarQuarter INT, CalendarQuarterStartDate INT, CalendarQuarterEndDate INT,
    CalendarNumberOfDaysInQuarter INT, CalendarDayInQuarter INT, CalendarYear INT, CalendarYearStartDate INT,
    CalendarYearEndDate INT, CalendarNumberOfDaysInYear INT, FiscalDay INT, FiscalWeek INT,
    FiscalWeekStartDate INT, FiscalWeekEndDate INT, FiscalDayInWeek INT, FiscalMonth INT,
    FiscalMonthStartDate INT, FiscalMonthEndDate INT, FiscalNumberOfDaysInMonth INT, FiscalDayInMonth INT,
    FiscalQuarter INT, FiscalQuarterStartDate INT, FiscalQuarterEndDate INT, FiscalNumberOfDaysInQuarter INT,
    FiscalDayInQuarter INT, FiscalYear INT, FiscalYearStartDate INT, FiscalYearEndDate INT,
    FiscalNumberOfDaysInYear INT, LOAD_TIME TIMESTAMP
);

```

The code editor has line numbers from 40 to 65. There are also 'Ask' and 'Copy' buttons at the bottom right.

- Created a view for Staging tables

Create views for each of your staging tables to capture the latest data based on the LOAD_TIME column.

```

CREATE OR REPLACE VIEW staging_customers_latest_view AS
SELECT * FROM hima_db.staging.staging_customers WHERE LOAD_TIME = (SELECT
max(LOAD_TIME) FROM hima_db.staging.staging_customers);

```

```

CREATE OR REPLACE VIEW staging_products_latest_view AS
SELECT * FROM hima_db.staging.staging_products WHERE LOAD_TIME = (SELECT
max(LOAD_TIME) FROM hima_db.staging.staging_products);

```

```

CREATE OR REPLACE VIEW staging_stores_latest_view AS
SELECT * FROM hima_db.staging.staging_stores WHERE LOAD_TIME = (SELECT
max(LOAD_TIME) FROM hima_db.staging.staging_stores);

```

```

CREATE OR REPLACE VIEW staging_sales_latest_view AS
SELECT * FROM hima_db.staging.staging_sales WHERE LOAD_TIME = (SELECT
max(LOAD_TIME) FROM hima_db.staging.staging_sales);

```

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
Databases Worksheets
HIMA_DB.STAGING Settings
Q Search objects
> HIMA_DB
> SNOWFLAKE
> SNOWFLAKE_SAMPLE_DATA
56   CalendarDayInMonth INT, CalendarQuarter INT, CalendarQuarterStartDateId INT, CalendarQuarterEndDateId INT,
57   CalendarNumberOfDaysInQuarter INT, CalendarDayInQuarter INT, CalendarYear INT, CalendarYearStartDateId INT,
58   CalendarYearEndDateId INT, CalendarNumberOfDaysInYear INT, FiscalDay INT, FiscalWeek INT,
59   FiscalWeekStart DateId INT, FiscalWeekEnd DateId INT, FiscalDayInWeek INT, FiscalMonth INT,
60   FiscalMonthStart DateId INT, FiscalMonthEnd DateId INT, FiscalNumberOfDaysInMonth INT, FiscalDayInMonth INT,
61   FiscalQuarter INT, FiscalQuarterStart DateId INT, FiscalQuarterEnd DateId INT, FiscalNumberOfDaysInQuarter INT,
62   FiscalDayInQuarter INT, FiscalYear INT, FiscalYearStart DateId INT, FiscalYearEnd DateId INT,
63   FiscalNumberOfDaysInYear INT, LOAD_TIME TIMESTAMP
64 );
65
66
67 CREATE OR REPLACE VIEW staging_customers_latest_view AS
68 SELECT * FROM hima_db.staging.staging_customers WHERE LOAD_TIME = (SELECT max(LOAD_TIME) FROM
hima_db.staging.staging_customers);
69
70 CREATE OR REPLACE VIEW staging_products_latest_view AS
71 SELECT * FROM hima_db.staging.staging_products WHERE LOAD_TIME = (SELECT max(LOAD_TIME) FROM
hima_db.staging.staging_products);
72
73 CREATE OR REPLACE VIEW staging_stores_latest_view AS
74 SELECT * FROM hima_db.staging.staging_stores WHERE LOAD_TIME = (SELECT max(LOAD_TIME) FROM hima_db.staging.staging_stores);
75
76 CREATE OR REPLACE VIEW staging_sales_latest_view AS
77 SELECT * FROM hima_db.staging.staging_sales WHERE LOAD_TIME = (SELECT max(LOAD_TIME) FROM hima_db.staging.staging_sales);
78
79
80
Ask Copilot

```

- Load Data from S3 Bucket into Staging Tables

Below is a Python-based Snowflake stored procedure that loads data from the S3 bucket into the specified staging tables:

-- Stored Procedure for loading data from stage file to staging tables

```
CREATE OR REPLACE PROCEDURE hima_db.pipeline.loaddata_into_staging()
```

```
RETURNS STRING
```

```
LANGUAGE PYTHON
```

```
RUNTIME_VERSION = '3.8'
```

```
PACKAGES = ('snowflake-snowpark-python')
```

```
HANDLER = 'handler'
```

```
EXECUTE AS CALLER
```

```
AS
```

```
$$
```

```
import snowflake.snowpark as snowpark
```

```
def handler(session: snowpark.Session):
```

```
# Define the mapping from table names to file names
```

```
stage_table_mappings = {
```

```
    "staging.staging_customers": "customers.csv",
    "staging.staging_products": "products.csv",
    "staging.staging_stores": "stores.csv",
    "staging.staging_sales": "sales.csv",
    "staging.staging_dimdates": "dimdates.csv"
}
```

```
}
```

```
try:
```

```
    for table, file_name in stage_table_mappings.items():
```

```
        # Get columns for the table excluding 'load_time'
```

```

columns_query = f"""
    SELECT COLUMN_NAME
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = '{table.split('.')[0].upper()}' 
        AND TABLE_SCHEMA = 'STAGING'
        AND COLUMN_NAME != 'LOAD_TIME'
    ORDER BY ORDINAL_POSITION;
"""

columns_result = session.sql(columns_query).collect()
columns = [col["COLUMN_NAME"] for col in columns_result]

# Generate column placeholders and add a load timestamp
column_placeholders = [f"${i+1}" for i in range(len(columns))]
load_time_placeholder = "CURRENT_TIMESTAMP()"

# Build the COPY INTO query
query = f"""
    COPY INTO {table}
    FROM (
        SELECT ${", ".join(column_placeholders)}, {load_time_placeholder}
        FROM @staging.hima_stage_2024/{file_name}
    )
    FILE_FORMAT = (TYPE = 'CSV' FIELD_OPTIONALLY_ENCLOSED_BY = '')
    ON_ERROR = 'CONTINUE'
    FORCE = TRUE;
"""

# Execute the query
session.sql(query).collect()

return "Data loaded successfully"

except Exception as e:
    return f"Failed: {str(e)}"

$$;

```

This stored procedure pipeline.loaddata_into_staging() created in the pipeline schema is designed to automate the loading of data from the S3 bucket into the Snowflake staging tables. It handles the file-to-table mappings, error logging, and execution summary, making it an efficient approach to manage your data loading tasks.

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ... + ↻

ses Worksheets

HIMA_DB.PIPELINE ▾ Settings ▾

```

use warehouse HIMA_WH;
use database HIMA_DB;
create or replace schema HIMA_DB.PIPELINE;

-- Stored Procedure for loading data from stage file to staging tables
CREATE OR REPLACE PROCEDURE hima_db.pipeline.loaddata_into_staging()
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'handler'
EXECUTE AS CALLER
AS
$$
import snowflake.snowpark as snowpark

def handler(session: snowpark.Session):
    # Define the mapping from table names to file names
    stage_table_mappings = {
        "staging.staging_customers": "customers.csv",
        "staging.staging_products": "products.csv",
        "staging.staging_stores": "stores.csv",
        "staging.staging_sales": "sales.csv",
        "staging.staging_dimdates": "dimdates.csv"
    }

```

Code Version

here to search 70°F 11:56 08-09-2024

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ... + ↻

Databases Worksheets

HIMA_DB.PIPELINE ▾ Settings ▾

```

try:
    for table, file_name in stage_table_mappings.items():
        # Get columns for the table excluding 'load_time'
        columns_query = f"""
            SELECT COLUMN_NAME
            FROM INFORMATION_SCHEMA.COLUMNS
            WHERE TABLE_NAME = '{table.split('.')[1].upper()}' 
            AND TABLE_SCHEMA = 'STAGING'
            AND COLUMN_NAME != 'LOAD_TIME'
            ORDER BY ORDINAL_POSITION;
        """
        columns_result = session.sql(columns_query).collect()
        columns = [col["COLUMN_NAME"] for col in columns_result]

        # Generate column placeholders and add a load timestamp
        column_placeholders = [f"${i+1}" for i in range(len(columns))]
        load_time_placeholder = "CURRENT_TIMESTAMP()"

        # Build the COPY INTO query
        query = f"""
            COPY INTO {table}
            FROM (
                SELECT ${", ".join(column_placeholders)}, {load_time_placeholder}
                FROM @staging.hima_stage_2024/{file_name}
            )
            ORDER BY ORDINAL_POSITION;
        """

```

Code Version

Ask

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ... + ↻

Databases Worksheets

HIMA_DB.PIPELINE ▾ Settings ▾

```

        ...
        columns = [col["COLUMN_NAME"] for col in columns_result]
        # Execute the query
        session.sql(query).collect()

        return "Data Loaded successfully"

    except Exception as e:
        return f"Failed: {str(e)}"

$:

```

Code Versions

Ask

- Create a Scheduled Task

Task that runs the pipeline.loaddata_staging() stored procedure every day at 9 AM

-- Task for loading data from stage file to staging tables

```
CREATE OR REPLACE TASK pipeline.loaddata_into_staging_task
```

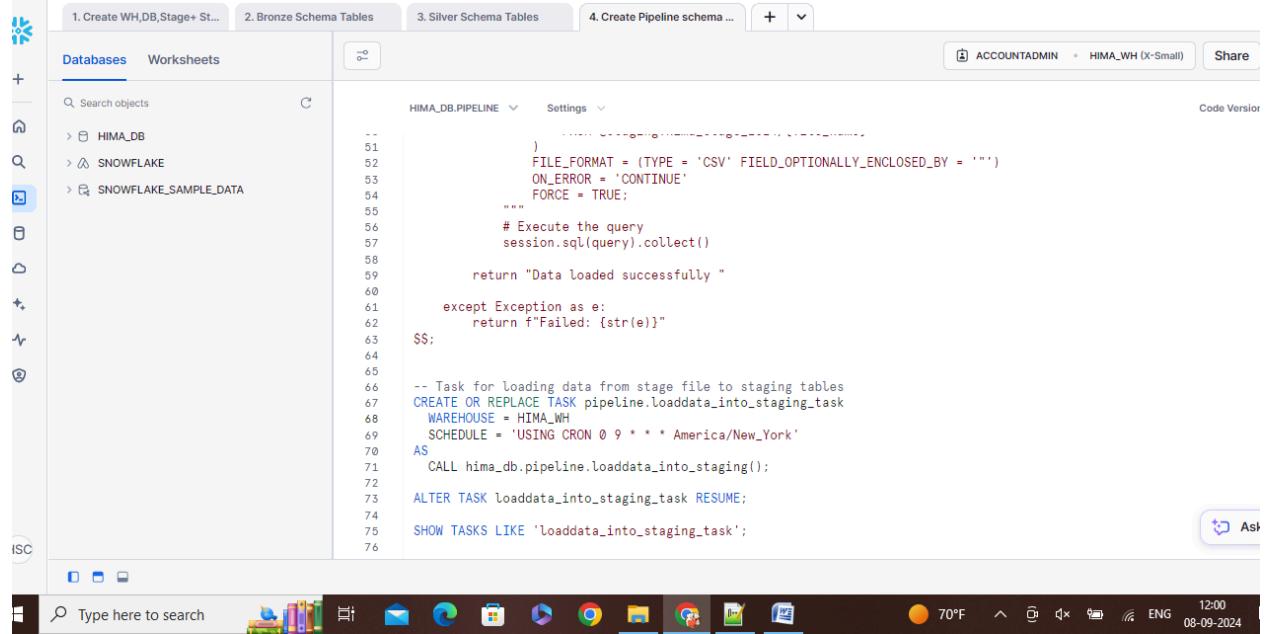
```
WAREHOUSE = HIMA_WH
```

```
SCHEDULE = 'USING CRON 0 9 * * * America/New_York'
```

AS

```
CALL hima_db.pipeline.loaddata_into_staging();
```

```
ALTER TASK loaddata_into_staging_task RESUME;
```



The screenshot shows the Snowflake UI with the code for the pipeline task. The code is as follows:

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
Databases Worksheets
HIMA_DB_PIPELINE Settings
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
)
FILE_FORMAT = (TYPE = 'CSV' FIELD_OPTIONALLY_ENCLOSED_BY = '')
ON_ERROR = 'CONTINUE'
FORCE = TRUE;
"""
# Execute the query
session.sql(query).collect()

return "Data loaded successfully"

except Exception as e:
    return f"Failed: {str(e)}"

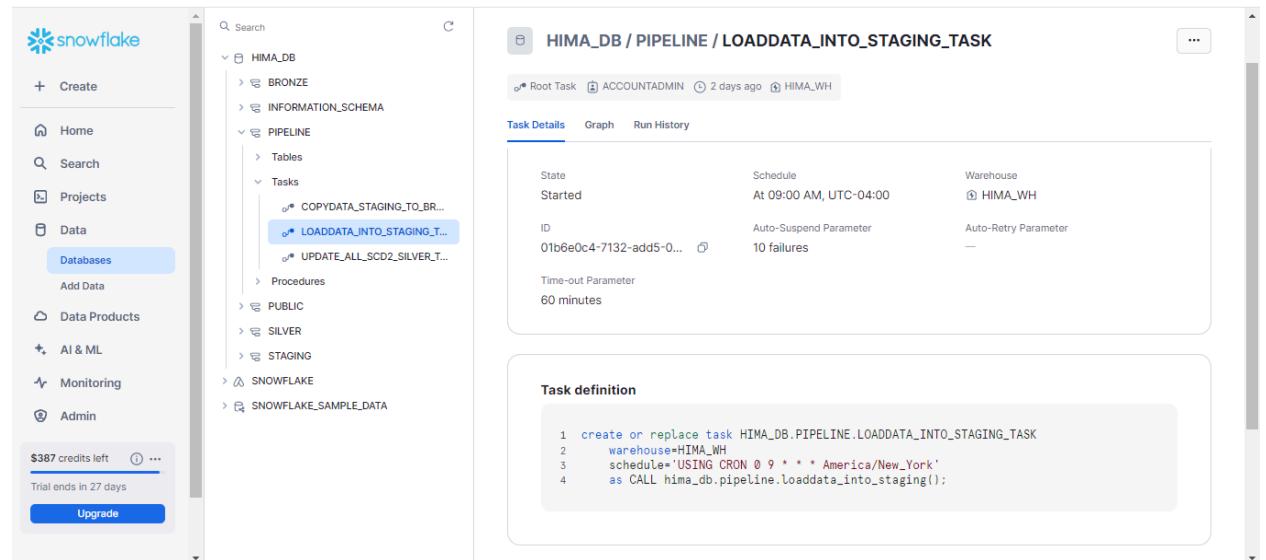
$$;

-- Task for loading data from stage file to staging tables
CREATE OR REPLACE TASK pipeline.loaddata_into_staging_task
WAREHOUSE = HIMA_WH
SCHEDULE = 'USING CRON 0 9 * * * America/New_York'
AS
CALL hima_db.pipeline.loaddata_into_staging();

ALTER TASK loaddata_into_staging_task RESUME;

SHOW TASKS LIKE 'loaddata_into_staging_task';

```



The screenshot shows the Snowflake UI with the task details and definition. The task details are as follows:

State	Schedule	Warehouse
Started	At 09:00 AM, UTC-04:00	HIMA_WH
ID	Auto-Suspend Parameter	Auto-Retry Parameter
01b6e0c4-7132-add5-0...	10 failures	—
Time-out Parameter 60 minutes		

The task definition is:

```

1 create or replace task HIMA_DB.PIPELINE.LOADDATA_INTO_STAGING_TASK
2 warehouse=HIMA_WH
3 schedule='USING CRON 0 9 * * * America/New_York'
4 as CALL hima_db.pipeline.loaddata_into_staging();

```

Process to Maintain Only the Latest Data in Bronze Tables

TASK: Ensure that each bronze table in Snowflake contains only the most recent data by

removing outdated records and inserting the latest data from the staging tables. Additionally, impose the following rules:

1. All primary keys of the tables should not be null or empty.
2. If a customer is deleted, their related sales should also be deleted.
3. If a store is deleted, related products should also be deleted.
4. Only data that qualifies these rules should be present in the bronze layer (today's data).

- Created Bronze Schema and Tables

```
use warehouse HIMA_WH;
use database HIMA_DB;
create schema HIMA_DB.BRONZE;
```

-- Bronze Layer Table for Customers

```
CREATE OR REPLACE TABLE bronze_customers (
    CUSTOMER_ID TEXT NOT NULL PRIMARY KEY, CUSTOMER_NAME TEXT,
    DATE_OF_BIRTH DATE, LOAD_TIME TIMESTAMP
);
```

-- Bronze Layer Table for Products

```
CREATE OR REPLACE TABLE bronze_products (
    PRODUCT_ID TEXT NOT NULL PRIMARY KEY, PRODUCT_NAME TEXT, PRICE
    FLOAT, LOAD_TIME TIMESTAMP
);
```

-- Bronze Layer Table for Stores

```
CREATE OR REPLACE TABLE bronze_stores (
    STORE_ID TEXT NOT NULL PRIMARY KEY, STORE_NAME TEXT, LOAD_TIME
    TIMESTAMP
);
```

-- Bronze Layer Table for Sales

```
CREATE OR REPLACE TABLE bronze_sales (
    TRANSACTION_ID INTEGER PRIMARY KEY, PRODUCT_ID TEXT NOT NULL,
    CUSTOMER_ID TEXT NOT NULL,
    STORE_ID TEXT NOT NULL, T_DATE DATE, QUANTITY INTEGER, LOAD_TIME
    TIMESTAMP,
    FOREIGN KEY (PRODUCT_ID) REFERENCES bronze_products(PRODUCT_ID),
    FOREIGN KEY (CUSTOMER_ID) REFERENCES bronze_customers(CUSTOMER_ID),
    FOREIGN KEY (STORE_ID) REFERENCES bronze_stores(STORE_ID)
);
```

```

1 use warehouse HIMA_WH;
2 use database HIMA_DB;
3 create schema HIMA_DB.BRONZE;
4
5 -- Bronze Layer Table for Customers
6 CREATE OR REPLACE TABLE bronze_customers (
7     CUSTOMER_ID TEXT NOT NULL PRIMARY KEY, CUSTOMER_NAME TEXT, DATE_OF_BIRTH DATE, LOAD_TIME TIMESTAMP
8 );
9
10 -- Bronze Layer Table for Products
11 CREATE OR REPLACE TABLE bronze_products (
12     PRODUCT_ID TEXT NOT NULL PRIMARY KEY, PRODUCT_NAME TEXT, PRICE FLOAT, LOAD_TIME TIMESTAMP
13 );
14
15 -- Bronze Layer Table for Stores
16 CREATE OR REPLACE TABLE bronze_stores (
17     STORE_ID TEXT NOT NULL PRIMARY KEY, STORE_NAME TEXT, LOAD_TIME TIMESTAMP
18 );
19
20 -- Bronze Layer Table for Sales
21 CREATE OR REPLACE TABLE bronze_sales (
22     TRANSACTION_ID INTEGER PRIMARY KEY, PRODUCT_ID TEXT NOT NULL, CUSTOMER_ID TEXT NOT NULL,
23     STORE_ID TEXT NOT NULL, T_DATE DATE, QUANTITY INTEGER, LOAD_TIME TIMESTAMP,
24     FOREIGN KEY (PRODUCT_ID) REFERENCES bronze_products(PRODUCT_ID),
25     FOREIGN KEY (CUSTOMER_ID) REFERENCES bronze_customers(CUSTOMER_ID),
26     FOREIGN KEY (STORE_ID) REFERENCES bronze_stores(STORE_ID)
27 );

```

- Details of the DATA_QUALITY_RULES Table:

To ensure that data loaded into the bronze layer tables adheres to certain quality and consistency rules, you created a table named DATA_QUALITY_RULES in the pipeline schema. This table stores the rules that dictate how data should be validated and processed before it is inserted into the bronze tables.

-- Table for Logging Failures of data quality rules:

```

CREATE OR REPLACE TABLE HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS (
    RULE_ID INT,
    RULE_TYPE VARCHAR(255),
    RULE_DESCRIPTION VARCHAR(255),
    STATUS VARCHAR(255),
    ERROR_MESSAGE VARCHAR(255),
    EXECUTION_TIME TIMESTAMP,
    FOREIGN KEY (RULE_ID) REFERENCES
        HIMA_DB.PIPELINE.DATA_QUALITY_RULES(RULE_ID)
);

```

- Details of the RULE_EXECUTION_LOGS Table:

The logging table is used to capture and record any failures or issues that occur while applying data quality rules during the data loading process into the bronze layer. This helps in monitoring and troubleshooting data quality issues.

-- Table for Logging Failures of data quality rules:

```

CREATE OR REPLACE TABLE HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS (
    RULE_ID INT,

```

```

    RULE_TYPE VARCHAR(255),
    RULE_DESCRIPTION VARCHAR(255),
    STATUS VARCHAR(255),
    ERROR_MESSAGE VARCHAR(255),
    EXECUTION_TIME TIMESTAMP,
    FOREIGN KEY (RULE_ID) REFERENCES
HIMA_DB.PIPELINE.DATA_QUALITY_RULES(RULE_ID)
);

```

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
Databases Worksheets
HIMA_DB PIPELINE Settings
ACCOUNTADMIN HIMA_WH (X-Small) S
Code
-- TASK: make sure to impose these rules 1. All the primary keys of the tables should not be null or empty 2.
cusomer is deleted his sales should also be deleted 3. if the stores is deleted then the products should also
only data qualified these rules should be in bronze layer (today's data)

-- Create DATA_QUALITY_RULES Table while loading data from staging to bronze layer
CREATE OR REPLACE TABLE HIMA_DB.PIPELINE.DATA_QUALITY_RULES(
    RULE_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT start 1 increment 1 order,
    TABLE_NAME VARCHAR(255),
    RULE_TYPE VARCHAR(255) NOT NULL,
    RULE_DESCRIPTION VARCHAR(255),
    RULE_QUERY TEXT NOT NULL
);

-- Table for Logging Failures of data quality rules:
CREATE OR REPLACE TABLE HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS (
    RULE_ID INT,
    RULE_TYPE VARCHAR(255),
    RULE_DESCRIPTION VARCHAR(255),
    STATUS VARCHAR(255),
    ERROR_MESSAGE VARCHAR(255),
    EXECUTION_TIME TIMESTAMP,
    FOREIGN KEY (RULE_ID) REFERENCES HIMA_DB.PIPELINE.DATA_QUALITY_RULES(RULE_ID)
);

```

- To insert rules into the DATA_QUALITY_RULES table, you need to define the rules that will be applied to the data in the bronze tables.

```
-- Insert rules into DATA_QUALITY_RULES Table
INSERT INTO HIMA_DB.PIPELINE.DATA_QUALITY_RULES (RULE_ID, TABLE_NAME,
RULE_TYPE, RULE_DESCRIPTION, RULE_QUERY)
VALUES
(1, 'bronze_customers', 'insert', 'Customer ID should not be null or empty', 'CUSTOMER_ID IS
NOT NULL AND TRIM(CUSTOMER_ID) <> ""'),
(2, 'bronze_products', 'insert', 'Product ID should not be null or empty', 'PRODUCT_ID IS NOT
NULL AND TRIM(PRODUCT_ID) <> ""'),
(3, 'bronze_stores', 'insert', 'Store ID should not be null or empty', 'STORE_ID IS NOT NULL
AND TRIM(STORE_ID) <> ""'),
(4, 'bronze_sales', 'insert', 'Transaction ID, Product ID, Customer ID, and Store ID should not be
null or empty', 'TRANSACTION_ID IS NOT NULL AND TRIM(TRANSACTION_ID) <> ""
AND PRODUCT_ID IS NOT NULL AND CUSTOMER_ID IS NOT NULL AND STORE_ID
IS NOT NULL'),
(5, 'bronze_sales', 'delete', 'Delete sales for customers that do not exist in the bronze layer
anymore', 'CUSTOMER_ID NOT IN (SELECT CUSTOMER_ID FROM
staging.staging_customers_latest_view)'),
```

```
(6, 'bronze_products', 'delete', 'Delete products for stores that do not exist in the bronze layer anymore', 'PRODUCT_ID IN (SELECT PRODUCT_ID FROM staging.staging_sales_latest_view WHERE STORE_ID NOT IN (SELECT STORE_ID FROM staging.staging_stores_latest_view))');
```

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
HIMA_DB.PIPELINE v Settings v
Code Versions Q
ACCOUNTADMIN + HIMA_WH (X-Small) Share ▶ ▾
HIMA_DB PIPELINE
EXECUTION_TIME TIMESTAMP,
FOREIGN KEY (RULE_ID) REFERENCES HIMA_DB.PIPELINE.DATA_QUALITY_RULES(RULE_ID)
);

-- Insert rules into DATA_QUALITY_RULES Table
INSERT INTO HIMA_DB.PIPELINE.DATA_QUALITY_RULES (RULE_ID, TABLE_NAME, RULE_TYPE, RULE_DESCRIPTION, RULE_QUERY)
VALUES
(1, 'bronze_customers', 'insert', 'Customer ID should not be null or empty', 'CUSTOMER_ID IS NOT NULL AND TRIM(CUSTOMER_ID) <> '''),
(2, 'bronze_products', 'insert', 'Product ID should not be null or empty', 'PRODUCT_ID IS NOT NULL AND TRIM(PRODUCT_ID) <> '''),
(3, 'bronze_stores', 'insert', 'Store ID should not be null or empty', 'STORE_ID IS NOT NULL AND TRIM(STORE_ID) <> '''),
(4, 'bronze_sales', 'insert', 'Transaction ID, Product ID, Customer ID, and Store ID should not be null or empty', 'TRANSACTION_ID IS NOT NULL AND TRIM(TRANSACTION_ID) <> '' AND PRODUCT_ID IS NOT NULL AND CUSTOMER_ID IS NOT NULL AND STORE_ID IS NOT NULL'),
(5, 'bronze_sales', 'delete', 'Delete sales for customers that do not exist in the bronze layer anymore', 'CUSTOMER_ID NOT IN (SELECT CUSTOMER_ID FROM staging.staging_customers_latest_view)'),
(6, 'bronze_products', 'delete', 'Delete products for stores that do not exist in the bronze layer anymore', 'PRODUCT_ID IN (SELECT PRODUCT_ID FROM staging.staging_sales_latest_view WHERE STORE_ID NOT IN (SELECT STORE_ID FROM staging.staging_stores_latest_view)))');

SELECT * FROM DATA_QUALITY_RULES;
SELECT * FROM RULE_EXECUTION_LOGS;

```

- Stored Procedure for Loading Data from Staging to Bronze Tables

Below is a Python-based Snowflake stored procedure created in the pipeline schema to load data from staging tables to bronze tables after applying data quality rules. The procedure ensures that only data meeting these rules is transferred.

```
CREATE OR REPLACE PROCEDURE pipeline.copydata_staging_to_bronze()
```

```
RETURNS STRING
```

```
LANGUAGE PYTHON
```

```
RUNTIME_VERSION = '3.8'
```

```
PACKAGES = ('snowflake-snowpark-python')
```

```
HANDLER = 'apply_data_quality_rules'
```

```
EXECUTE AS CALLER
```

```
AS
```

```
$$
```

```
import snowflake.snowpark as sp
```

```
def apply_data_quality_rules(session: sp.Session) -> str:
```

```
# Fetch all the rules from the DATA_QUALITY_RULES table
```

```
rules_df = session.table("HIMA_DB.PIPELINE.DATA_QUALITY_RULES").collect()
```

```
# Define table mappings
```

```
table_mappings = {
```

```
    "bronze_customers": "staging.staging_customers_latest_view",
```

```
    "bronze_products": "staging.staging_products_latest_view",
```

```
    "bronze_stores": "staging.staging_stores_latest_view",
```

```

    "bronze_sales": "staging.staging_sales_latest_view"
}

# Initialize counters for successful and failed rules
success_count = 0
failure_count = 0

# Clear all data from the bronze tables before loading new data
for bronze_table in table_mappings.keys():
    session.sql(f"TRUNCATE TABLE bronze.{bronze_table};").collect()

# Apply insert rules
for bronze_table, staging_view_table in table_mappings.items():
    for rule in rules_df:
        if rule['TABLE_NAME'] == bronze_table and rule['RULE_TYPE'] == 'insert':
            query = f"""
                INSERT INTO bronze.{bronze_table}
                SELECT * FROM {staging_view_table} WHERE {rule['RULE_QUERY']};
            """
            try:
                session.sql(query).collect() # Execute the query
                success_count += 1
            except Exception as e:
                # Log the failure
                session.sql(f"""
                    INSERT INTO HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS
                    (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS,
                    ERROR_MESSAGE, EXECUTION_TIME)
                    VALUES
                    ({rule['RULE_ID']}, '{rule['RULE_TYPE']}', '{rule['RULE_DESCRIPTION']}',
                    'FAILED', '{str(e)}', CURRENT_TIMESTAMP)
                """).collect()
                failure_count += 1

# Apply delete rules
for rule in rules_df:
    if rule['RULE_TYPE'] == 'delete':
        query = f"""
            DELETE FROM bronze.{rule['TABLE_NAME']} WHERE {rule['RULE_QUERY']};
        """
        try:
            session.sql(query).collect() # Execute the query

```

```

success_count += 1
except Exception as e:
    # Log the failure
    session.sql(f"""
        INSERT INTO HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS
        (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS,
        ERROR_MESSAGE, EXECUTION_TIME)
        VALUES
        ({rule['RULE_ID']}, '{rule['RULE_TYPE']}','{rule['RULE_DESCRIPTION']}',
        'FAILED', '{str(e)}', CURRENT_TIMESTAMP)
    """).collect()
    failure_count += 1

# Return the execution summary
return f"Execution completed. Successful: {success_count}, Failed: {failure_count}"

```

\$\$;

The screenshot shows the Snowflake UI interface. On the left, there's a sidebar with icons for databases, worksheets, and other database-related functions. The main area is titled "HIMA_DB.PIPELINE". It contains a code editor with the following Python script:

```

117
118 -- Stored Procedure for loading data from staging tables to Bronze tables after data quality rules applied
119 CREATE OR REPLACE PROCEDURE pipeline.copydata_staging_to_bronze()
120 RETURNS STRING
121 LANGUAGE PYTHON
122 RUNTIME_VERSION = '3.8'
123 PACKAGES = ('snowflake-snowpark-python')
124 HANDLER = 'apply_data_quality_rules'
125 EXECUTE AS CALLER
126 AS
127 $$
128 import snowflake.snowpark as sp
129
130 def apply_data_quality_rules(session: sp.Session) -> str:
131     # Fetch all the rules from the DATA_QUALITY_RULES table
132     rules_df = session.table("HIMA_DB.PIPELINE.DATA_QUALITY_RULES").collect()
133
134     # Define table mappings
135     table_mappings = {
136         "bronze_customers": "staging.staging_customers_latest_view",
137         "bronze_products": "staging.staging_products_latest_view",
138         "bronze_stores": "staging.staging_stores_latest_view",
139         "bronze_sales": "staging.staging_sales_latest_view"
140     }
141
142     # Initialize counters for successful and failed rules
143     success_count = 0
144     failure_count = 0
145
146     # Clear all data from the bronze tables before loading new data
147     for bronze_table in table_mappings.keys():
148         session.sql(f"TRUNCATE TABLE bronze.{bronze_table};").collect()
149
150     # Apply insert rules
151     for bronze_table, staging_view_table in table_mappings.items():
152         for rule in rules_df:
153             if rule['TABLE_NAME'] == bronze_table and rule['RULE_TYPE'] == 'insert':
154                 query = f"""
155                     INSERT INTO bronze.{bronze_table}
156                     SELECT * FROM {staging_view_table} WHERE {rule['RULE_QUERY']}
157                 """
158
159             try:
160                 session.sql(query).collect() # Execute the query
161                 success_count += 1
162             except Exception as e:
163                 # Log the failure
164                 session.sql(f"""
165                     INSERT INTO HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS
166                     (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS, ERROR_MESSAGE, EXECUTION_TIME)
167                     (failure_count, 'failed', 'rule_type', 'failed', 'rule_description', 'failure')
168                 """)
169

```

The screenshot shows the same Snowflake UI interface. The code editor now contains the following Python script:

```

142
143     # Initialize counters for successful and failed rules
144     success_count = 0
145     failure_count = 0
146
147     # Clear all data from the bronze tables before loading new data
148     for bronze_table in table_mappings.keys():
149         session.sql(f"TRUNCATE TABLE bronze.{bronze_table};").collect()
150
151     # Apply insert rules
152     for bronze_table, staging_view_table in table_mappings.items():
153         for rule in rules_df:
154             if rule['TABLE_NAME'] == bronze_table and rule['RULE_TYPE'] == 'insert':
155                 query = f"""
156                     INSERT INTO bronze.{bronze_table}
157                     SELECT * FROM {staging_view_table} WHERE {rule['RULE_QUERY']}
158                 """
159
160             try:
161                 session.sql(query).collect() # Execute the query
162                 success_count += 1
163             except Exception as e:
164                 # Log the failure
165                 session.sql(f"""
166                     INSERT INTO HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS
167                     (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS, ERROR_MESSAGE, EXECUTION_TIME)
168                     (failure_count, 'failed', 'rule_type', 'failed', 'rule_description', 'failure')
169                 """)

```

```

170         failure_count += 1
171
172     # Apply delete rules
173     for rule in rules_df:
174         if rule['RULE_TYPE'] == 'delete':
175             query = f"""
176                 DELETE FROM bronze.{rule['TABLE_NAME']} WHERE {rule['RULE_QUERY']};
177             """
178             try:
179                 session.sql(query).collect() # Execute the query
180                 success_count += 1
181             except Exception as e:
182                 # Log the failure
183                 session.sql(f"""
184                     INSERT INTO HIMA_DB.PIPELINE.RULE_EXECUTION_LOGS
185                         (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS, ERROR_MESSAGE, EXECUTION_TIME)
186                         VALUES
187                         ({rule['RULE_ID']}, '{rule['RULE_TYPE']}', '{rule['RULE_DESCRIPTION']}', 'FAILED', '{str(e)}',
188                         CURRENT_TIMESTAMP)
189                 """).collect()
190                 failure_count += 1
191
192     # Return the execution summary
193     return f"Execution completed. Successful: {success_count}, Failed: {failure_count}"
194
$$;

```

- Created a task for bronze stored procedure

This task runs a bronze stored procedure daily after the staging task, ensures that the bronze data loading procedure only runs after the staging tables have been updated, keeping the data pipeline streamlined and synchronized.

-- Task for loading data from staging tables to Bronze tables

CREATE OR REPLACE TASK pipeline.copydata_staging_to_bronze_task

WAREHOUSE = HIMA_WH

AFTER hima_db.pipeline.loaddata_into_staging_task

AS

CALL hima_db.pipeline.copydata_staging_to_bronze();

ALTER TASK copydata_staging_to_bronze_task RESUME;

```

185         (RULE_ID, RULE_TYPE, RULE_DESCRIPTION, STATUS, ERROR_MESSAGE, EXECUTION_TIME)
186         VALUES
187         ({rule['RULE_ID']}, '{rule['RULE_TYPE']}', '{rule['RULE_DESCRIPTION']}', 'FAILED', '{str(e)}',
188         CURRENT_TIMESTAMP)
189     """).collect()
190     failure_count += 1
191
192     # Return the execution summary
193     return f"Execution completed. Successful: {success_count}, Failed: {failure_count}"
194
$$;

-- Task for loading data from staging tables to Bronze tables
CREATE OR REPLACE TASK pipeline.copydata_staging_to_bronze_task
    WAREHOUSE = HIMA_WH
    AFTER hima_db.pipeline.loaddata_into_staging_task
AS
    CALL hima_db.pipeline.copydata_staging_to_bronze();

ALTER TASK copydata_staging_to_bronze_task RESUME;

SHOW TASKS LIKE 'copydata_staging_to_bronze_task';

```

The screenshot shows the Snowflake web interface. On the left, the navigation sidebar is visible with options like Home, Search, Projects, Data, Databases (selected), Add Data, Data Products, AI & ML, Monitoring, and Admin. A message at the bottom says '\$387 credits left' and 'Trial ends in 27 days'. In the center, the main content area displays the 'HIMA_DB / PIPELINE / COPYDATA_STAGING_TO_BRONZE_TASK' page. The pipeline tree on the left shows nodes like BRONZE, INFORMATION_SCHEMA, PIPELINE, Tables, Tasks, Procedures, PUBLIC, SILVER, STAGING, and SNOWFLAKE. The 'Tasks' node is expanded, showing 'COPYDATA_STAGING_TO_BRONZE_TASK' (selected), 'LOADDATA_INTO_STAGING_T...', and 'UPDATE_ALL_SCD2_SILVER_T...'. The 'Task Details' tab is selected, showing the task's state (Started), ID (01b6e0c4-cbe0-3a87-0...), schedule (None), warehouse (HIMA_WH), and time-out parameter (60 minutes). Below this, the 'Task definition' tab shows the SQL code for the task:

```

1 create or replace task HIMA_DB.PIPELINE.COPYDATA_STAGING_TO_BRONZE_TASK
2   warehouse=HIMA_WH
3   after HIMA_DB.PIPELINE.LOADDATA_INTO_STAGING_TASK
4   as CALL hima_db.pipeline.copydata_staging_to_bronze();

```

Process for Implementing Silver Layer Tables with SCD Type 2

TASK: Created silver layer tables that implement Slowly Changing Dimension (SCD) Type 2.

The objective is to develop a proof of concept (POC) for deploying an SCD Type 2 table in Snowflake. The SCD Type 2 table should reflect changes made in the base table and incorporate the following features:

1. Record Updates: When a record is updated in the base table, the same record should be added as a new entry in the SCD Type 2 table. This new record should be marked as active.
2. Inactive Records: The previous version of the record should be marked as inactive in the SCD Type 2 table.
3. Record Deletions: If a record is deleted from the base table, the corresponding record in the SCD Type 2 table should be marked as inactive.
4. Date Tracking: Add START_DATE and END_DATE columns to the SCD Type 2 table to track the validity period of each record version. The START_DATE should be set when the record is inserted or updated, and the END_DATE should be updated when the record is deactivated or replaced.

- Created Silver Schema and Tables

use warehouse HIMA_WH;
use database HIMA_DB;
create or replace schema SILVER;

```

CREATE OR REPLACE TABLE silver.scd2_silver_customers (
  SURROGATE_KEY INTEGER AUTOINCREMENT PRIMARY KEY,
  CUSTOMER_ID TEXT,

```

```

CUSTOMER_NAME TEXT,
DATE_OF_BIRTH DATE,
START_DATE DATE DEFAULT CURRENT_DATE,
END_DATE DATE,
IS_ACTIVE BOOLEAN DEFAULT TRUE
);

```

```

CREATE OR REPLACE TABLE silver.scd2_silver_products (
    SURROGATE_KEY INTEGER AUTOINCREMENT PRIMARY KEY,
    PRODUCT_ID TEXT,
    PRODUCT_NAME TEXT,
    PRICE FLOAT,
    START_DATE DATE DEFAULT CURRENT_DATE,
    END_DATE DATE,
    IS_ACTIVE BOOLEAN DEFAULT TRUE
);

```

```

CREATE OR REPLACE TABLE silver.scd2_silver_stores (
    SURROGATE_KEY INTEGER AUTOINCREMENT PRIMARY KEY,
    STORE_ID TEXT,
    STORE_NAME TEXT,
    START_DATE DATE DEFAULT CURRENT_DATE,
    END_DATE DATE,
    IS_ACTIVE BOOLEAN DEFAULT TRUE
);

```

```

CREATE OR REPLACE TABLE silver.scd2_silver_sales (
    SURROGATE_KEY INTEGER AUTOINCREMENT PRIMARY KEY,
    TRANSACTION_ID INTEGER,
    PRODUCT_ID TEXT,
    CUSTOMER_ID TEXT,
    STORE_ID TEXT,
    T_DATE DATE,
    QUANTITY INTEGER,
    START_DATE DATE DEFAULT CURRENT_DATE,
    END_DATE DATE,
    IS_ACTIVE BOOLEAN DEFAULT TRUE
);

```

- Stored Procedure for SCD Type 2 in Silver Tables

Below is a Python-based Snowflake stored procedure for implementing Slowly Changing Dimension (SCD) Type 2 in silver layer tables. This procedure is located in the pipeline schema. It manages the SCD Type 2 functionality by performing the above mentioned task:

```
-- Stored Procedure for SCD2 Silver Tables
CREATE OR REPLACE PROCEDURE hima_db.pipeline.update_all_scd2_silver_tables()
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'run'
EXECUTE AS CALLER
AS
$$
import snowflake.snowpark as snowpark

def run_query(session, query):
    """Helper function to execute a SQL query."""
    session.sql(query).collect()

def update_scd2_table(session, table_name, view_name, id_column, columns):
    """
    Updates an SCD2 table by marking old records as inactive, inserting new ones,
    and handling deletions.

    :param session: Snowpark session
    :param table_name: The target SCD2 table name
    :param view_name: The staging view name with the latest data
    :param id_column: The primary key column of the table
    :param columns: List of column names to compare for changes
    """
    # Mark existing records as inactive and set the END_DATE for records missing in the latest
    # data view
    delete_query = f"""
        UPDATE {table_name} scd
        SET
            END_DATE = CURRENT_DATE,
            IS_ACTIVE = FALSE
        WHERE scd.IS_ACTIVE = TRUE
        AND scd.{id_column} NOT IN (SELECT {id_column} FROM {view_name});
    """
    run_query(session, delete_query)
```

```

# Mark existing records as inactive and set the END_DATE if they have changed
update_query = f"""
    UPDATE {table_name} scd
    SET
        END_DATE = CURRENT_DATE,
        IS_ACTIVE = FALSE
    FROM {view_name} base
    WHERE scd.{id_column} = base.{id_column}
        AND scd.IS_ACTIVE = TRUE
        AND (' OR 'join([f'scd.{col}' <>> base.{col}]' for col in columns]));
"""

run_query(session, update_query)

# Insert new or updated records into the SCD2 table
insert_query = f"""
    INSERT INTO {table_name} ({id_column}, {'join(columns)}, START_DATE,
END_DATE, IS_ACTIVE)
    SELECT
        base.{id_column},
        {'join([f'base.{col}' for col in columns])},
        CURRENT_DATE,
        '9999-12-31', -- END_DATE
        TRUE      -- Mark as active
    FROM {view_name} base
    LEFT JOIN {table_name} scd
    ON base.{id_column} = scd.{id_column}
        AND scd.IS_ACTIVE = TRUE
    WHERE scd.{id_column} IS NULL
        OR (' OR 'join([f'scd.{col}' <>> base.{col}]' for col in columns)));
"""

run_query(session, insert_query)

def run(session):
    """Main function that runs the SCD2 update process for all tables."""
    # Update the SCD2 Customers table
    update_scd2_table(
        session,
        table_name="silver.scd2_silver_customers",
        view_name="staging.staging_customers_latest_view",
        id_column="CUSTOMER_ID",
        columns=["CUSTOMER_NAME", "DATE_OF_BIRTH"]
    )

```

```
)  
  
# Update the SCD2 Products table  
update_scd2_table(  
    session,  
    table_name="silver.scd2_silver_products",  
    view_name="staging.staging_products_latest_view",  
    id_column="PRODUCT_ID",  
    columns=["PRODUCT_NAME", "PRICE"]  
)  
  
# Update the SCD2 Stores table  
update_scd2_table(  
    session,  
    table_name="silver.scd2_silver_stores",  
    view_name="staging.staging_stores_latest_view",  
    id_column="STORE_ID",  
    columns=["STORE_NAME"]  
)  
  
# Update the SCD2 Sales table  
update_scd2_table(  
    session,  
    table_name="silver.scd2_silver_sales",  
    view_name="bronze.bronze_sales",  
    id_column="TRANSACTION_ID",  
    columns=["PRODUCT_ID", "CUSTOMER_ID", "STORE_ID", "T_DATE",  
    "QUANTITY"]  
)  
  
return "SCD2 Update Complete for All Silver Tables"  
  
$$;
```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ... + v

Databases Worksheets

HIMA_DB.PIPELINE ▼ Settings ▼

```

213
214
215 -- Stored Procedure for SCD2 Silver Tables
216 CREATE OR REPLACE PROCEDURE hima_db.pipeline.update_all_scd2_silver_tables()
217 RETURNS STRING
218 LANGUAGE PYTHON
219 RUNTIME_VERSION = '3.8'
220 PACKAGES = ('snowflake-snowpark-python')
221 HANDLER = 'run'
222 EXECUTE AS CALLER
223 AS
224 $$

225 import snowflake.snowpark as snowpark
226
227 def run_query(session, query):
228     """Helper function to execute a SQL query."""
229     session.sql(query).collect()
230
231 def update_scd2_table(session, table_name, view_name, id_column, columns):
232     """
233         Updates an SCD2 table by marking old records as inactive, inserting new ones,
234         and handling deletions.
235
236         :param session: Snowpark session
237         :param table_name: The target SCD2 table name
238         :param view_name: The staging view name with the latest data
239     """

```

Databases Worksheets

HIMA_DB.PIPELINE ▼ Settings ▼

```

242
243
244
245 # Mark existing records as inactive and set the END_DATE for records missing in the latest data view
246 delete_query = f"""
247     UPDATE {table_name} scd
248     SET
249         END_DATE = CURRENT_DATE,
250         IS_ACTIVE = FALSE
251     WHERE scd.IS_ACTIVE = TRUE
252         AND scd.{id_column} NOT IN (SELECT {id_column} FROM {view_name});
253 """
254 run_query(session, delete_query)

255
256 # Mark existing records as inactive and set the END_DATE if they have changed
257 update_query = f"""
258     UPDATE {table_name} scd
259     SET
260         END_DATE = CURRENT_DATE,
261         IS_ACTIVE = FALSE
262     FROM {view_name} base
263     WHERE scd.{id_column} = base.{id_column}
264         AND scd.IS_ACTIVE = TRUE
265         AND ({' OR '.join([f'scd.{col} <> base.{col}' for col in columns])});
266 """
267 run_query(session, update_query)

268 # Insert new or updated records into the SCD2 table
269 insert_query = f"""

```

Databases Worksheets

HIMA_DB.PIPELINE ▼ Settings ▼

```

265
266
267
268 # Insert new or updated records into the SCD2 table
269 insert_query = f"""
270     INSERT INTO {table_name} ({id_column}, {', '.join(columns)}, START_DATE, END_DATE, IS_ACTIVE)
271     SELECT
272         base.{id_column},
273         {', '.join(f'base.{col}' for col in columns)},
274         CURRENT_DATE,
275         '9999-12-31', -- END_DATE
276         TRUE -- Mark as active
277     FROM {view_name} base
278     LEFT JOIN {table_name} scd
279     ON base.{id_column} = scd.{id_column}
280         AND scd.IS_ACTIVE = TRUE
281     WHERE scd.{id_column} IS NULL
282         OR ({' OR '.join([f'scd.{col} <> base.{col}' for col in columns])});
283 """
284 run_query(session, insert_query)

285 def run(session):
286     """Main function that runs the SCD2 update process for all tables."""
287     # Update the SCD2 Customers table
288     update_scd2_table(
289         session,
290         table_name="silver.scd2_silver_customers",
291         view_name="staging.staging_customers_latest_view",

```

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
HIMA_DB.PIPELINE Settings
def run(session):
    """Main function that runs the SCD2 update process for all tables."""
    # Update the SCD2 Customers table
    update_scd2_table(
        session,
        table_name="silver.scd2_silver_customers",
        view_name="staging.staging_customers_latest_view",
        id_column="CUSTOMER_ID",
        columns=["CUSTOMER_NAME", "DATE_OF_BIRTH"]
    )

    # Update the SCD2 Products table
    update_scd2_table(
        session,
        table_name="silver.scd2_silver_products",
        view_name="staging.staging_products_latest_view",
        id_column="PRODUCT_ID",
        columns=["PRODUCT_NAME", "PRICE"]
    )

    # Update the SCD2 Stores table
    update_scd2_table(
        session,
        table_name="silver.scd2_silver_stores",
        view_name="staging.staging_stores_latest_view",
        id_column="STORE_ID",
        columns=[]
    )

```

```

300     id_column="PRODUCT_ID",
301     columns=["PRODUCT_NAME", "PRICE"]
302 )
303
304 # Update the SCD2 Stores table
305 update_scd2_table(
306     session,
307     table_name="silver.scd2_silver_stores",
308     view_name="staging.staging_stores_latest_view",
309     id_column="STORE_ID",
310     columns=["STORE_NAME"]
311 )
312
313 # Update the SCD2 Sales table
314 update_scd2_table(
315     session,
316     table_name="silver.scd2_silver_sales",
317     view_name="staging.staging_sales_latest_view",
318     id_column="TRANSACTION_ID",
319     columns=["PRODUCT_ID", "CUSTOMER_ID", "STORE_ID", "T_DATE", "QUANTITY"]
320 )
321
322 return "SCD2 Update Complete for All Silver Tables"
323
324 $$;
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339

```

- Created a task for silver stored procedure

This task has been created in the pipeline schema that runs every day after the staging task. This task executes the stored procedure for the SCD Type 2 tables, ensuring that the data in the silver layer is updated according to the SCD Type 2 logic.

```

1. Create WH,DB,Stage+ St... 2. Bronze Schema Tables 3. Silver Schema Tables 4. Create Pipeline schema ...
HIMA_DB.PIPELINE Settings
update_scd2_table(
    session,
    table_name="silver.scd2_silver_sales",
    view_name="staging.staging_sales_latest_view",
    id_column="TRANSACTION_ID",
    columns=["PRODUCT_ID", "CUSTOMER_ID", "STORE_ID", "T_DATE", "QUANTITY"]
)

return "SCD2 Update Complete for All Silver Tables"
$$;

-- Task for SCD2 Silver Customers Table
CREATE OR REPLACE TASK pipeline.update_all_scd2_silver_tables_task
WAREHOUSE = HIMA_WH
AFTER hima_db.pipeline.loaddata_into_staging_task
AS
CALL hima_db.pipeline.update_all_scd2_silver_tables();

ALTER TASK update_all_scd2_silver_tables_task RESUME;

SHOW TASKS LIKE 'update_all_scd2_silver_tables_task';

```

The screenshot shows the Snowflake interface. On the left, the sidebar has 'Databases' selected. The main area shows a tree view of 'HIMA_DB' with 'PIPLINE' expanded, revealing 'Tables', 'Tasks', and 'Procedures'. Under 'Tasks', 'UPDATE_ALL_SCD2_SILVER_TABLES_TASK' is selected. The right side displays the 'Task Details' page for this task. It includes sections for 'Details' (State: Started, Schedule: —, Warehouse: HIMA_WH), 'Run History' (with one run listed), and 'Task definition' (SQL code: 1 create or replace task HIMA_DB.PIPELINE.UPDATE_ALL_SCD2_SILVER_TABLES_TASK, 2 warehouse=HIMA_WH, 3 after HIMA_DB.PIPELINE.LOADDATA_INTO_STAGING_TASK, 4 as CALL hima_db.pipeline.update_all_scd2_silver_tables();). A 'Graph' tab is also present.

OUTPUT

Monitoring all layer tables—staging, bronze, and silver—for a period of three days. This monitoring involves checking for:

Data Consistency: Ensuring that data is correctly processed and reflects the expected changes in each layer.

Data Accuracy: Verifying that data transformations and rules are applied correctly.

Task Execution: Confirming that scheduled tasks, including data loading and SCD Type 2 updates, are running as planned.

Error Logs: Reviewing logs for any errors or issues that may arise during data processing.

DAY 1

Check Loaded Data with SELECT queries for layers tables

STAGING LAYER TABLES

1. STAGING_CUSTOMERS table

The screenshot shows the Snowflake interface with a query editor and results viewer. The query is: `SELECT * FROM staging_customers;`. The results table shows 11 rows of customer data. The columns are CUSTOMER_ID, CUSTOMER_NAME, DATE_OF_BIRTH, and LOAD_TIME. The last column, LOAD_TIME, is highlighted in yellow. The results table has 11 rows. The right side of the screen shows 'Query Details' including duration (81ms), rows (50), and Query ID (01b6e618-030a-5c40-0...). There are also 'Ask Copilot' and 'Customer_ID' status indicators.

CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	LOAD_TIME
1 C-1	Lezlie Trott	1980-03-15	2024-09-08 06:00:08.984
2 C-2	Joann Shortt	1995-07-22	2024-09-08 06:00:08.984
3 C-3	Violet Newingham	1982-11-05	2024-09-08 06:00:08.984
4 C-4	Shanika Bang	1978-09-10	2024-09-08 06:00:08.984
5 C-5	Tequila Carmen	1990-02-18	2024-09-08 06:00:08.984
6 C-6	Tomi Hati	1988-06-25	2024-09-08 06:00:08.984
7 C-8	Odella Thompkins	1985-12-14	2024-09-08 06:00:08.984
8 C-9	Bernardo Vancamp	1965-08-20	2024-09-08 06:00:08.984
9 C-10	Hilde Mustard	1999-01-08	2024-09-08 06:00:08.984
10 C-11	Charissa Critchfield	1997-10-02	2024-09-08 06:00:08.984

2. STAGING_PRODUCTS table

The screenshot shows the Snowflake UI interface. The left sidebar has a tree view with 'HIMA_DB' expanded, showing 'SNOWFLAKE' and 'SNOWFLAKE_SAMPLE_DATA'. The main area shows a query editor with the following code:

```
SELECT * FROM staging_products;
```

The results pane displays a table with the following data:

PRODUCT_ID	PRODUCT_NAME	PRICE	LOAD_TIME
1	P-1000	14.25	2024-09-08 06:00:12.281
2	P-1001	18.73	2024-09-08 06:00:12.281
3	P-1002	5.48	2024-09-08 06:00:12.281
4	P-1003	19.72	2024-09-08 06:00:12.281
5	P-1004	26.3	2024-09-08 06:00:12.281
6	P-1005	25.44	2024-09-08 06:00:12.281
7	P-1006	8.95	2024-09-08 06:00:12.281
8	P-1007	19.46	2024-09-08 06:00:12.281
9	P-1008	13.32	2024-09-08 06:00:12.281
10	P-1009	24.10	2024-09-08 06:00:12.281

The Query Details panel on the right shows a duration of 75ms and 100 rows.

3. STAGING_STORES table

The screenshot shows the Snowflake UI interface. The left sidebar has a tree view with 'HIMA_DB' expanded, showing 'SNOWFLAKE' and 'SNOWFLAKE_SAMPLE_DATA'. The main area shows a query editor with the following code:

```
SELECT * FROM staging_stores;
```

The results pane displays a table with the following data:

STORE_ID	STORE_NAME	LOAD_TIME
1	STORE_10	2024-09-08 06:00:14.463
2	S-10	2024-09-08 06:00:14.463
3	S-9	2024-09-08 06:00:14.463
4	S-8	2024-09-08 06:00:14.463
5	S-5	2024-09-08 06:00:14.463
6	S-1	2024-09-08 06:00:14.463
7	S-7	2024-09-08 06:00:14.463
8	S-6	2024-09-08 06:00:14.463
9	S-4	2024-09-08 06:00:14.463
10	S-9	2024-09-08 06:00:14.463

The Query Details panel on the right shows a duration of 549ms and 11 rows.

4. STAGING_SALES table

The screenshot shows the Snowflake UI interface. The left sidebar has a tree view with 'HIMA_DB' expanded, showing 'SNOWFLAKE' and 'SNOWFLAKE_SAMPLE_DATA'. The main area shows a query editor with the following code:

```
SELECT * FROM staging_sales;
```

The results pane displays a table with the following data:

TRANSACTION_ID	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	LOAD_TIME	
1	586	C-30	S-9	2023-12-31	10	2024-09-08 06:00:16.246	
2	156	P-1059	C-12	S-6	2023-12-31	7	2024-09-08 06:00:16.246
3	733	P-1022	C-37	S-4	2023-12-31	8	2024-09-08 06:00:16.246
4	1046	P-1015	C-23	S-6	2023-12-31	9	2024-09-08 06:00:16.246
5	1422	P-1090	C-36	S-8	2023-12-31	6	2024-09-08 06:00:16.246
6	2425	P-1016	C-46	S-2	2023-12-31	4	2024-09-08 06:00:16.246
7	2483	P-1066	C-32	S-6	2023-12-31	8	2024-09-08 06:00:16.246
8	2587	P-1062	C-47	S-3	2023-12-31	5	2024-09-08 06:00:16.246
9	3265	P-1006	C-12	S-9	2023-12-31	5	2024-09-08 06:00:16.246
10	2292	D-1067	C-40	S-0	2023-12-31	6	2024-09-08 06:00:16.246

The Query Details panel on the right shows a duration of 637ms and 11K rows.

5. STAGING_DIMDATES table

The screenshot shows the Snowflake interface with the following details:

- Top Navigation:** 1. Create WH,DB,Stage+ St..., 2. Bronze Schema Tables, 3. Silver Schema Tables, 4. Create Pipeline schema ...
- Left Sidebar:** Databases (selected), Worksheets, HIMA_DB, SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA.
- Current Database:** HIMA_DB.STAGING
- Code Editor:** SELECT * FROM staging_dimdates;
- Results View:** Results (selected), Chart. The results table has columns: TER, CALDAYINQUARTER, FISCALYEAR, YEARSTARTDATEID, FISCALYEARENDDATEID, IYEAR, LOAD_TIME. The data shows rows from 1 to 9, all with LOAD_TIME: 2024-09-08 06:00:1E.
- Query Details:** Query duration: 895ms, Rows: 18.3K, Query ID: 01b6e61f-030a-5c12-0...
- Metrics:** ID, DATE, Ask Copilot.

BRONZE LAYER TABLES

1. BRONZE_CUSTOMER table

The screenshot shows the Snowflake interface with the following details:

- Top Navigation:** 1. Create WH,DB,Stage+ St..., 2. Bronze Schema Tables, 3. Silver Schema Tables, 4. Create Pipeline schema ...
- Left Sidebar:** Databases (selected), Worksheets, HIMA_DB, SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA.
- Current Database:** HIMA_DB.BRONZE
- Code Editor:** SELECT * FROM bronze_customers;
- Results View:** Results (selected), Chart. The results table has columns: CUSTOMER_ID, CUSTOMER_NAME, DATE_OF_BIRTH, LOAD_TIME. The data shows rows from 1 to 11, all with LOAD_TIME: 2024-09-08 06:00:08.984.
- Query Details:** Query duration: 39ms, Rows: 50, Query ID: 01b6e628-030a-5c40-0...
- Metrics:** CUSTOMER_ID, CUSTOMER_NAME, Ask Copilot.

2. BRONZE_PRODUCTS table

The screenshot shows the Snowflake interface with the following details:

- Top Navigation:** 1. Create WH,DB,Stage+ St..., 2. Bronze Schema Tables, 3. Silver Schema Tables, 4. Create Pipeline schema ...
- Left Sidebar:** Databases (selected), Worksheets, HIMA_DB, SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA.
- Current Database:** HIMA_DB.BRONZE
- Code Editor:** SELECT * FROM bronze_products;
- Results View:** Results (selected), Chart. The results table has columns: PRODUCT_ID, PRODUCT_NAME, PRICE, LOAD_TIME. The data shows rows from 1 to 10, all with LOAD_TIME: 2024-09-08 06:00:12.281.
- Query Details:** Query duration: 183ms, Rows: 100, Query ID: 01b6e62a-030a-5c40-0...
- Metrics:** PRODUCT_ID, PRODUCT_NAME, Ask Copilot.

3. BRONZE_STORES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Code Editor:** SELECT * FROM bronze_stores;
- Results:** A table with columns STORE_ID, STORE_NAME, and LOAD_TIME. The data includes 11 rows of store information.
- Query Details:** Query duration: 228ms, Rows: 11, Query ID: 01b6e62a-030a-5c40-0...
- Metrics:** STORE_ID (100% filled), STORE_NAME (Ask Copilot)

4. BRONZE_SALES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Code Editor:** SELECT * FROM bronze_sales;
- Results:** A table with columns TRANSACTION_ID, PRODUCT_ID, CUSTOMER_ID, STORE_ID, T_DATE, QUANTITY, and LOAD_TIME. The data includes 9 rows of transaction information.
- Query Details:** Query duration: 407ms, Rows: 9.4K, Query ID: 01b6e62b-030a-5c40-0...
- Metrics:** TRANSACTION_ID (Ask Copilot), PRODUCT_ID (Ask Copilot)

SILVER LAYER TABLES

1. SCD2_SILVER_CUSTOMERS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.SILVER
- Code Editor:** SELECT * FROM scd2_silver_customers;
- Results:** A table with columns SURROGATE_KEY, CUSTOMER_ID, CUSTOMER_NAME, DATE_OF_BIRTH, START_DATE, END_DATE, and IS_ACTIVE. The data includes 11 rows of customer information.
- Query Details:** Query duration: 53ms, Rows: 50, Query ID: 01b6e62d-030a-5c40-0...
- Metrics:** SURROGATE_KEY (Ask Copilot), CUSTOMER_ID (Ask Copilot)

2. SCD2_SILVER_PRODUCTS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.SILVER
- Code Versions:** ACCOUNTADMIN • HIMA_WH (X-Small)
- Query:** SELECT * FROM scd2_silver_products;
- Results:** A table with 10 rows of data.
- Query Details:**
 - Query duration: 295ms
 - Rows: 100
 - Query ID: 01b6e634-030a-5c12-0...
- Ask Copilot:** A button in the bottom right corner.

SURROGATE_KEY	PRODUCT_ID	PRODUCT_NAME	PRICE	START_DATE	END_DATE	IS_ACTIVE
1	P-1000	Asparagus	14.25	2024-09-08	9999-12-31	TRUE
2	P-1001	Broccoli	18.73	2024-09-08	9999-12-31	TRUE
3	P-1002	Carrots	5.48	2024-09-08	9999-12-31	TRUE
4	P-1003	Cauliflower	19.72	2024-09-08	9999-12-31	TRUE
5	P-1004	Celery	26.3	2024-09-08	9999-12-31	TRUE
6	P-1005	Corn	25.44	2024-09-08	9999-12-31	TRUE
7	P-1006	Cucumbers	8.95	2024-09-08	9999-12-31	TRUE
8	P-1007	Lettuce / Greens	19.46	2024-09-08	9999-12-31	TRUE
9	P-1008	Mushrooms	13.32	2024-09-08	9999-12-31	TRUE
10	P-1009	Onions	28.10	2024-09-08	9999-12-31	TRUE

3. SCD2_SILVER_STORES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.SILVER
- Code Versions:** ACCOUNTADMIN • HIMA_WH (X-Small)
- Query:** SELECT * FROM scd2_silver_stores;
- Results:** A table with 10 rows of data.
- Query Details:**
 - Query duration: 302ms
 - Rows: 11
 - Query ID: 01b6e636-030a-5c12-0...
- Ask Copilot:** A button in the bottom right corner.

SURROGATE_KEY	STORE_ID	STORE_NAME	START_DATE	END_DATE	IS_ACTIVE
1	STORE_1	STORE_NAME	2024-09-08	9999-12-31	TRUE
2	S-10	Henderson	2024-09-08	9999-12-31	TRUE
3	S-9	Whangaparaora	2024-09-08	9999-12-31	TRUE
4	S-8	Westgate	2024-09-08	9999-12-31	TRUE
5	S-5	Massey	2024-09-08	9999-12-31	TRUE
6	S-1	Queen St.	2024-09-08	9999-12-31	TRUE
7	S-7	East Auckland	2024-09-08	9999-12-31	TRUE
8	S-6	West Auckland	2024-09-08	9999-12-31	TRUE
9	S-4	St. James	2024-09-08	9999-12-31	TRUE
10	S-2	Alders	2024-09-08	9999-12-31	TRUE

4. SCD2_SILVER_SALES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.SILVER
- Code Versions:** ACCOUNTADMIN • HIMA_WH (X-Small)
- Query:** SELECT * FROM silver.scd2_silver_sales;
- Results:** A table with 9 rows of data.
- Query Details:**
 - Query duration: 804ms
 - Rows: 11,11K
 - Query ID: 01b6e637-030a-5c12-0...
- Ask Copilot:** A button in the bottom right corner.

PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	START_DATE	END_DATE	IS_ACTIVE
P-1043	C-30	S-9	2023-12-31	10	2024-09-08	9999-12-31	TRUE
P-1059	C-12	S-6	2023-12-31	7	2024-09-08	9999-12-31	TRUE
P-1022	C-37	S-4	2023-12-31	8	2024-09-08	9999-12-31	TRUE
P-1015	C-23	S-6	2023-12-31	9	2024-09-08	9999-12-31	TRUE
P-1090	C-36	S-8	2023-12-31	6	2024-09-08	9999-12-31	TRUE
P-1016	C-46	S-2	2023-12-31	4	2024-09-08	9999-12-31	TRUE
P-1066	C-32	S-6	2023-12-31	8	2024-09-08	9999-12-31	TRUE
P-1062	C-47	S-3	2023-12-31	5	2024-09-08	9999-12-31	TRUE
P-1006	C-12	S-9	2023-12-31	5	2024-09-08	9999-12-31	TRUE

DAY 2

STAGING LAYER TABLES

1. STAGING_CUSTOMERS table

For example:

On Day 1, the staging_customers table includes a customer with the Customer ID C-9. However, on Day 2, this Customer ID C-9 is no longer present in the table.

Databases Worksheets

HIMA_DB STAGING

```
SELECT * FROM staging_customers;
```

Results

CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	LOAD_TIME
51	C-1	1980-03-15	2024-09-09 06:00:20.136
52	C-2	1995-07-22	2024-09-09 06:00:20.136
53	C-3	1982-11-05	2024-09-09 06:00:20.136
54	C-4	1978-09-10	2024-09-09 06:00:20.136
55	C-5	1990-02-18	2024-09-09 06:00:20.136
56	C-6	1988-06-25	2024-09-09 06:00:20.136
57	C-8	1985-12-14	2024-09-09 06:00:20.136
58	C-10	1999-01-08	2024-09-09 06:00:20.136
59	C-11	1987-10-03	2024-09-09 06:00:20.136

Query Details

- Query duration: 75ms
- Rows: 99
- Query ID: 01b6ea8-020a-5c12-0..

Ask Copilot

2. STAGING_PRODUCTS table

Databases Worksheets

HIMA_DB STAGING

```
SELECT * FROM staging_products;
```

Results

PRODUCT_ID	PRODUCT_NAME	PRICE	LOAD_TIME
98	P-1097	20.8	2024-09-08 06:00:12.281
99	P-1098	18.2	2024-09-08 06:00:12.281
100	P-1099	10.51	2024-09-08 06:00:12.281
101	P-1000	14.25	2024-09-09 06:00:22.242
102	P-1001	18.73	2024-09-09 06:00:22.242
103	P-1002	5.48	2024-09-09 06:00:22.242
104	P-1003	19.72	2024-09-09 06:00:22.242
105	P-1004	26.3	2024-09-09 06:00:22.242
106	P-1005	25.44	2024-09-09 06:00:22.242

Query Details

- Query duration: 1.3s
- Rows: 200
- Query ID: 01b6ea8-020a-5c12-0..

Ask Copilot

3. STAGING_STORES table

Databases Worksheets

HIMA_DB STAGING

```
SELECT * FROM staging_stores;
```

Results

STORE_ID	STORE_NAME	LOAD_TIME
S-1	Henderson	2024-09-09 06:00:24.151
S-9	Whangaparaoa	2024-09-09 06:00:24.151
S-8	Westgate	2024-09-09 06:00:24.151
S-5	Massey	2024-09-09 06:00:24.151
S-1	Queen St.	2024-09-09 06:00:24.151
S-7	East Auckland	2024-09-09 06:00:24.151
S-6	West Auckland	2024-09-09 06:00:24.151
S-4	St. James	2024-09-09 06:00:24.151
S-2	Albany	2024-09-09 06:00:24.151
S-3	Manukau	2024-09-09 06:00:24.151

Query Details

- Query duration: 373ms
- Rows: 22
- Query ID: 01b6aac-020a-5d45-0..

Ask Copilot

4. STAGING_SALES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB
- Tables:** BRONZE, INFORMATION_SCHEMA, PIPELINE
- Tasks:** COPYDATA_STAGING_TO_BR..., LOADDATA_INTO_STAGING_T...
- Procedures:** UPDATE_ALL_SCD2_SILVER...
- Public:** PUBLIC
- Silver:** SILVER
- Staging:** STAGING
- Snowflake:** SNOWFLAKE
- Snowflake Sample Data:** SNOWFLAKE_SAMPLE_DATA

The main area displays the results of the query:

```
102
103   SELECT * FROM staging_sales;
104
105
106
107
```

ANSACTION_ID	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	LOAD_TIME
11082	11082	P-1020	C-41	S-6	2024-09-08	7 2024-09-08 06:00:16.246
11083	586	P-1043	C-30	S-9	2023-12-31	10 2024-09-09 06:00:26.101
11084	156	P-1059	C-12	S-6	2023-12-31	7 2024-09-09 06:00:26.101
11085	733	P-1022	C-37	S-4	2023-12-31	8 2024-09-09 06:00:26.101
11086	1046	P-1015	C-23	S-6	2023-12-31	9 2024-09-09 06:00:26.101
11087	1422	P-1090	C-36	S-8	2023-12-31	6 2024-09-09 06:00:26.101
11088	2425	P-1016	C-46	S-2	2023-12-31	4 2024-09-09 06:00:26.101
11089	2483	P-1066	C-32	S-6	2023-12-31	8 2024-09-09 06:00:26.101
11090	2667	P-1029	C-17	S-3	2023-12-31	5 2024-09-09 06:00:26.101

Query Details: Rows: 22.2K, Query ID: 01b6eac3-020a-5c12-0...

5. STAGING_DIMDATES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB
- Tables:** BRONZE, INFORMATION_SCHEMA, PIPELINE
- Tasks:** COPYDATA_STAGING_TO_BR..., LOADDATA_INTO_STAGING_T...
- Procedures:** UPDATE_ALL_SCD2_SILVER...
- Public:** PUBLIC
- Silver:** SILVER
- Staging:** STAGING
- Snowflake:** SNOWFLAKE
- Snowflake Sample Data:** SNOWFLAKE_SAMPLE_DATA

The main area displays the results of the query:

```
96
97   SELECT * FROM staging_dimdates;
98
99
100
```

LYEAR	FISCALYEARSTARTDATEID	FISCALYEARENDDATEID	FISCALNUMBEROFDAYSINYEAR	LOAD_TIME
18250	2049	490001	491131	365 2024-09-08 06:00:18
18256	2049	490001	491131	365 2024-09-09 06:00:24
18257	2000	1	1130	365 2024-09-09 06:00:24
18258	2000	1	1130	365 2024-09-09 06:00:24
18259	2000	1	1130	365 2024-09-09 06:00:24
18260	2000	1	1130	365 2024-09-09 06:00:24
18261	2000	1	1130	365 2024-09-09 06:00:24
18262	2000	1	1130	365 2024-09-09 06:00:24
18263	2000	1	1130	365 2024-09-09 06:00:24

Query Details: Rows: 26.5K, Query ID: 01b6eac7-020a-5c40-0...

BRONZE LAYER TABLES

1. BRONZE_CUSTOMERS table

For example:

On Day 1, the `staging_customers` table contains a customer with Customer ID 'C-9'. However, on Day 2, this Customer ID 'C-9' is no longer present in the staging table.

Since the bronze layer tables are designed to hold only the latest data, the Customer ID 'C-9' is not present in the `bronze_customers` table. Consequently, in the `bronze_sales` table, any sales records related to Customer ID 'C-9' are also not present, reflecting the enforcement of data quality rules where related sales data is removed when a customer no longer exists.

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Worksheets:** Results
- Code:**

```
28
29
30 | SELECT * FROM bronze_customers;
31
32
33
```
- Results:** A table with 11 rows of data.
- Query Details:**
 - Query duration: 52ms
 - Rows: 19
 - Query ID: 01b6eac9-020a-5c12-0...
- Filter:** CUSTOMER_ID (100% filled)
- Sort:** CUSTOMER_NAME (Ask Copilot)

CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	LOAD_TIME
C-1	Lezlie Trott	1980-03-15	2024-09-09 06:00:20.136
C-2	Joann Shortt	1995-07-22	2024-09-09 06:00:20.136
C-3	Violet Newingham	1982-11-05	2024-09-09 06:00:20.136
C-4	Shanika Bang	1978-09-10	2024-09-09 06:00:20.136
C-5	Tequila Carmen	1990-02-18	2024-09-09 06:00:20.136
C-6	Tomi Hatt	1988-06-25	2024-09-09 06:00:20.136
C-8	Odella Thompkins	1985-12-14	2024-09-09 06:00:20.136
C-10	Hilde Mustard	1999-01-08	2024-09-09 06:00:20.136
C-11	Shenika Critchfield	1987-10-03	2024-09-09 06:00:20.136
C-14	Adalio Kinch	1992-06-01	2024-09-09 06:00:20.136

2. BRONZE_PRODUCTS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Worksheets:** Results
- Code:**

```
32
33
34 | SELECT * FROM bronze_products;
35
36
37
```
- Results:** A table with 10 rows of data.
- Query Details:**
 - Query duration: 246ms
 - Rows: 100
 - Query ID: 01b6eaca-020a-5c40-0...
- Filter:** PRODUCT_ID (100% filled)
- Sort:** PRODUCT_NAME (Ask Copilot)

PRODUCT_ID	PRODUCT_NAME	PRICE	LOAD_TIME
P-1000	Asparagus	14.25	2024-09-09 06:00:22.242
P-1001	Broccoli	18.73	2024-09-09 06:00:22.242
P-1002	Carrots	5.48	2024-09-09 06:00:22.242
P-1003	Cauliflower	19.72	2024-09-09 06:00:22.242
P-1004	Celery	26.3	2024-09-09 06:00:22.242
P-1005	Corn	25.44	2024-09-09 06:00:22.242
P-1006	Cucumbers	8.95	2024-09-09 06:00:22.242
P-1007	Lettuce / Greens	19.46	2024-09-09 06:00:22.242
P-1008	Mushrooms	13.32	2024-09-09 06:00:22.242
P-1009	Onions	24.10	2024-09-09 06:00:22.242

3. BRONZE_STORES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Worksheets:** Results
- Code:**

```
44
45
46 | SELECT * FROM bronze_stores;
47
48
```
- Results:** A table with 11 rows of data.
- Query Details:**
 - Query duration: 207ms
 - Rows: 11
 - Query ID: 01b6eacd-020a-5c40-0...
- Filter:** STORE_ID (100% filled)
- Sort:** STORE_NAME (Ask Copilot)

STORE_ID	STORE_NAME	LOAD_TIME
S-10	Henderson	2024-09-09 06:00:24.151
S-9	Whangaparaora	2024-09-09 06:00:24.151
S-8	Westgate	2024-09-09 06:00:24.151
S-5	Massey	2024-09-09 06:00:24.151
S-1	Queen St.	2024-09-09 06:00:24.151
S-7	East Auckland	2024-09-09 06:00:24.151
S-6	West Auckland	2024-09-09 06:00:24.151
S-4	St. James	2024-09-09 06:00:24.151
S-2	Albany	2024-09-09 06:00:24.151
S-3	Manukau	2024-09-09 06:00:24.151

4. BRONZE_SALES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Code Versions:** ACCOUNTADMIN - HIMA_WH (X-Small)
- Share:** Share button
- Results:** The query `SELECT * FROM bronze_sales;` was run, returning 9 rows of data.
- Table Headers:** TRANSACTION_ID, PRODUCT_ID, CUSTOMER_ID, STORE_ID, T_DATE, QUANTITY, LOAD_TIME
- Sample Data:**

TRANSACTION_ID	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	LOAD_TIME
1	733	P-1022	C-37	S-4	2023-12-31	8 2024-09-09 06:00:26.101
2	2483	P-1066	C-32	S-6	2023-12-31	8 2024-09-09 06:00:26.101
3	3283	P-1067	C-40	S-9	2023-12-31	5 2024-09-09 06:00:26.101
4	3853	P-1018	C-44	S-6	2023-12-31	8 2024-09-09 06:00:26.101
5	5842	P-1081	C-26	S-1	2023-12-31	8 2024-09-09 06:00:26.101
6	6641	P-1094	C-32	S-2	2023-12-31	1 2024-09-09 06:00:26.101
7	7270	P-1035	C-43	S-4	2023-12-31	4 2024-09-09 06:00:26.101
8	8452	P-1065	C-17	S-10	2023-12-31	2 2024-09-09 06:00:26.101
9	8059	P-1022	C-31	S-2	2023-12-31	5 2024-09-09 06:00:26.101
- Query Details:**
 - Query duration: 33ms
 - Rows: 6.2K
 - Query ID: 01b6eaca-020a-5c40-0...
- Charts:** Two charts are shown: one for TRANSACTION_ID and one for PRODUCT_ID.

SELECT * FROM bronze_sales WHERE customer_id = 'C-9';

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.BRONZE
- Code Versions:** ACCOUNTADMIN - HIMA_WH (X-Small)
- Share:** Share button
- Results:** The query `SELECT * FROM bronze_sales WHERE customer_id = 'C-9';` was run, returning 0 rows.
- Query produced no results.**
- Query Details:**
 - Query duration: 125ms
 - Rows: 0
 - Query ID: 01b6eacc-020a-5c40-0...
- Ask Copilot:** Ask Copilot button

SILVER LAYER TABLES

1. SCD2_SILVER_CUSTOMERS table

For example:

On Day 1, the `staging_customers` table contains a customer with Customer ID 'C-9'. However, on Day 2, this Customer ID 'C-9' is no longer present in the staging table.

In the silver layer tables, which implement Slowly Changing Dimension Type 2 (SCD2), any data that is deleted, updated, or inserted in the base table is managed by marking the old record as inactive and the new record as active. Since Customer ID 'C-9' is no longer present in the `staging_customers` table, the corresponding record in the `silver_customers` table is marked as inactive, reflecting the change in the staging layer.

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB
- Worksheets:** 3. Silver Schema Tables
- Code:**

```
HIMA_DB.SILVER
SELECT * FROM scd2_silver_customers;
```
- Results:** A table with 10 rows of data.

SURROGATE_KEY	CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	START_DATE	END_DATE	IS_ACTIVE
1	C-1	Lezlie Trott	1980-03-15	2024-09-08	9999-12-31	TRUE
2	C-2	Joann Shortt	1995-07-22	2024-09-08	9999-12-31	TRUE
3	C-3	Violet Newingham	1982-11-05	2024-09-08	9999-12-31	TRUE
4	C-4	Shanika Bang	1978-09-10	2024-09-08	9999-12-31	TRUE
5	C-5	Tequila Carmen	1990-02-18	2024-09-08	9999-12-31	TRUE
6	C-6	Tomi Hatt	1988-06-25	2024-09-08	9999-12-31	TRUE
7	C-8	Odelia Thompkins	1985-12-14	2024-09-08	9999-12-31	TRUE
8	C-9	Bernardo Vancamp	1965-08-20	2024-09-08	2024-09-09	FALSE
9	C-10	Hilde Mustard	1999-01-08	2024-09-08	9999-12-31	TRUE
10	C-11	Charissa Critchfield	1997-10-02	2024-09-08	9999-12-31	TRUE

Query Details:

- Query duration: 46ms
- Rows: 50
- Query ID: 01b6eacd-020a-5c40-0...

2. SCD2_SILVER_PRODUCTS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB
- Worksheets:** 3. Silver Schema Tables
- Code:**

```
HIMA_DB.SILVER
SELECT * FROM scd2_silver_products;
```
- Results:** A table with 100 rows of data.

SURROGATE_KEY	PRODUCT_ID	PRODUCT_NAME	PRICE	START_DATE	END_DATE	IS_ACTIVE
1	P-1090	Basil	8.05	2024-09-08	9999-12-31	TRUE
92	P-1091	Black pepper	20	2024-09-08	9999-12-31	TRUE
93	P-1092	Cilantro	12.89	2024-09-08	9999-12-31	TRUE
94	P-1093	Cinnamon	8.63	2024-09-08	9999-12-31	TRUE
95	P-1094	Garlic	12.07	2024-09-08	9999-12-31	TRUE
96	P-1095	Ginger	24.6	2024-09-08	9999-12-31	TRUE
97	P-1096	Mint	22.47	2024-09-08	9999-12-31	TRUE
98	P-1097	Oregano	20.8	2024-09-08	9999-12-31	TRUE
99	P-1098	Paprika	18.2	2024-09-08	9999-12-31	TRUE
100	P-1099	Parsley	10.51	2024-09-08	9999-12-31	TRUE

Query Details:

- Query duration: 371ms
- Rows: 100
- Query ID: 01b6eace-020a-5d45-0...

3. SCD2_SILVER_STORES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB
- Worksheets:** 3. Silver Schema Tables
- Code:**

```
HIMA_DB.SILVER
SELECT * FROM scd2_silver_stores;
```
- Results:** A table with 11 rows of data.

SURROGATE_KEY	STORE_ID	STORE_NAME	START_DATE	END_DATE	IS_ACTIVE
2	S-10	Henderson	2024-09-08	9999-12-31	TRUE
3	S-9	Whangaparaora	2024-09-08	9999-12-31	TRUE
4	S-8	Westgate	2024-09-08	9999-12-31	TRUE
5	S-5	Masey	2024-09-08	9999-12-31	TRUE
6	S-1	Queen St.	2024-09-08	9999-12-31	TRUE
7	S-7	East Auckland	2024-09-08	9999-12-31	TRUE
8	S-6	West Auckland	2024-09-08	9999-12-31	TRUE
9	S-4	St. James	2024-09-08	9999-12-31	TRUE
10	S-2	Albany	2024-09-08	9999-12-31	TRUE
11	S-3	Manukau	2024-09-08	9999-12-31	TRUE

Query Details:

- Query duration: 375ms
- Rows: 11
- Query ID: 01b6ead0-020a-5c40-0...

4. SCD2_SILVER_SALES table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.SILVER
- Code Editor:** SELECT * FROM silver.scd2_silver_sales;
- Results:** A table with 9 rows of data.

	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	START_DATE	END_DATE	IS_ACTIVE
1	P-1043	C-30	S-9	2023-12-31	10	2024-09-08	9999-12-31	TRUE
2	P-1059	C-12	S-6	2023-12-31	7	2024-09-08	9999-12-31	TRUE
3	P-1022	C-37	S-4	2023-12-31	8	2024-09-08	9999-12-31	TRUE
4	P-1015	C-23	S-6	2023-12-31	9	2024-09-08	9999-12-31	TRUE
5	P-1090	C-36	S-8	2023-12-31	6	2024-09-08	9999-12-31	TRUE
6	P-1016	C-46	S-2	2023-12-31	4	2024-09-08	9999-12-31	TRUE
7	P-1066	C-32	S-6	2023-12-31	8	2024-09-08	9999-12-31	TRUE
8	P-1062	C-47	S-3	2023-12-31	5	2024-09-08	9999-12-31	TRUE
9	P-1006	C-12	S-9	2023-12-31	5	2024-09-08	9999-12-31	TRUE

Query Details: Rows: 11.1K, Query ID: 01b6ead1-020a-5d45-0...

DAY 3

STAGING LAYER TABLES

1. STAGING_CUSTOMERS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.STAGING
- Code Editor:** SELECT * FROM staging_customers;
- Results:** A table with 107 rows of data.

	CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	LOAD_TIME
98	C-76	Melissa Williams	1974-09-14	2024-09-09 06:00:20.136
99	C-77	Samantha Price	1972-09-21	2024-09-09 06:00:20.136
100	C-1	Lezlie Trott	1980-03-15	2024-09-10 06:00:05.540
101	C-2	Joeann Shortt	1995-07-22	2024-09-10 06:00:05.540
102	C-3	Violet Newingham	1982-11-05	2024-09-10 06:00:05.540
103	C-4	Shanika Bang	1978-09-10	2024-09-10 06:00:05.540
104	C-5	Tequila Carmen	1990-02-18	2024-09-10 06:00:05.540
105	C-6	Tomi Hatt	1988-06-25	2024-09-10 06:00:05.540
106	C-8	Odela Thompkins	1985-12-14	2024-09-10 06:00:05.540
107	C-10	Hilde Mustard	1999-01-08	2024-09-10 06:00:05.540

Query Details: Rows: 148, Query ID: 01b6f041-020a-5e5f-00...

2. STAGING_PRODUCTS table

The screenshot shows the Snowflake interface with the following details:

- Databases:** HIMA_DB.STAGING
- Code Editor:** SELECT * FROM staging_products;
- Results:** A table with 108 rows of data.

	PRODUCT_ID	PRODUCT_NAME	PRICE	LOAD_TIME
199	P-1098	Paprika	18.2	2024-09-09 06:00:22.242
200	P-1099	Parsley	10.51	2024-09-09 06:00:22.242
201	P-1000	Asparagus	14.25	2024-09-10 06:00:09.712
202	P-1001	Broccoli	18.73	2024-09-10 06:00:09.712
203	P-1002	Carrots	5.48	2024-09-10 06:00:09.712
204	P-1003	Cauliflower	19.72	2024-09-10 06:00:09.712
205	P-1004	Celery	26.3	2024-09-10 06:00:09.712
206	P-1005	Corn	25.44	2024-09-10 06:00:09.712
207	P-1006	Cucumbers	8.95	2024-09-10 06:00:09.712
208	P-1007	Lettuce / Greens	10.18	2024-09-10 06:00:09.712

Query Details: Rows: 300, Query ID: 01b6f045-020a-5c40-0...

3. STAGING_STORES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM staging_stores;` executed. The results table has columns `STORE_ID`, `STORE_NAME`, and `LOAD_TIME`. The data shows 32 rows of store information, with the last row highlighted in yellow. The right sidebar displays query details (duration: 382ms, rows: 32) and a preview of the `STORE_ID` column.

STORE_ID	STORE_NAME	LOAD_TIME
S-10	Henderson	2024-09-10 06:00:12.388
S-9	Whangaparaoa	2024-09-10 06:00:12.388
S-8	Westgate	2024-09-10 06:00:12.388
S-5	Massey	2024-09-10 06:00:12.388
S-1	Queen St.	2024-09-10 06:00:12.388
S-7	East Auckland	2024-09-10 06:00:12.388
S-6	West Auckland	2024-09-10 06:00:12.388
S-4	St. James	2024-09-10 06:00:12.388
S-2	Albany	2024-09-10 06:00:12.388

4. STAGING_SALES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM staging_sales;` executed. The results table has columns `ISACTION_ID`, `PRODUCT_ID`, `CUSTOMER_ID`, `STORE_ID`, `T_DATE`, `QUANTITY`, and `LOAD_TIME`. The data shows 22 rows of transaction information, with the last row highlighted in yellow. The right sidebar displays query details (duration: 1.3s, rows: 33K) and a preview of the `TRANSACTION_ID` column.

ISACTION_ID	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	LOAD_TIME
22169	11087	P-1023	C-14	S-2	2024-09-09	5 2024-09-09 06:00:26.101
22170	11088	P-1037	C-64	S-7	2024-09-09	3 2024-09-09 06:00:26.101
22171	586	P-1043	C-30	S-9	2023-12-31	10 2024-09-10 06:00:14.902
22172	156	P-1059	C-12	S-6	2023-12-31	7 2024-09-10 06:00:14.902
22173	733	P-1022	C-37	S-4	2023-12-31	8 2024-09-10 06:00:14.902
22174	1046	P-1015	C-23	S-6	2023-12-31	9 2024-09-10 06:00:14.902
22175	1422	P-1090	C-36	S-8	2023-12-31	6 2024-09-10 06:00:14.902
22176	2425	P-1016	C-46	S-2	2023-12-31	4 2024-09-10 06:00:14.902
22177	2483	P-1066	C-32	S-6	2023-12-31	8 2024-09-10 06:00:14.902

5. STAGING_DIMDATES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM staging_dimdates;` executed. The results table has columns `TER`, `DAYINQUARTER`, `ISCALYEAR`, `FISCALYEARSTARTDATEID`, `ARENDDATEID`, `FDAYSINYEAR`, and `LOAD_TIME`. The data shows 17 rows of dimension date information, with the last row highlighted in yellow. The right sidebar displays query details (duration: 1.5s, rows: 54.8K) and a preview of the `ID` column.

TER	DAYINQUARTER	ISCALYEAR	FISCALYEARSTARTDATEID	ARENDDATEID	FDAYSINYEAR	LOAD_TIME
36511	91	86	2049	490001	491131	365 2024-09-09 06:00:26
36512	91	87	2049	490001	491131	365 2024-09-09 06:00:26
36513	90	3	2000	1	1130	365 2024-09-10 06:00:17
36514	90	4	2000	1	1130	365 2024-09-10 06:00:17
36515	90	5	2000	1	1130	365 2024-09-10 06:00:17
36516	90	6	2000	1	1130	365 2024-09-10 06:00:17
36517	90	7	2000	1	1130	365 2024-09-10 06:00:17
36518	90	8	2000	1	1130	365 2024-09-10 06:00:17
36519	90	9	2000	1	1130	365 2024-09-10 06:00:17

BRONZE LAYER TABLES

1. BRONZE_CUSTOMERS table

The screenshot shows the Snowflake interface with the query `SELECT * FROM bronze_customers;` executed. The results are displayed in a table with columns: CUSTOMER_ID, CUSTOMER_NAME, DATE_OF_BIRTH, and LOAD_TIME. The data includes 11 rows of customer information. The right panel shows Query Details: duration 247ms, 49 rows, and a Query ID.

CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	LOAD_TIME
C-1	Lezlie Trott	1980-03-15	2024-09-10 06:00:09.540
C-2	Joann Shortt	1995-07-22	2024-09-10 06:00:09.540
C-3	Violet Newingham	1982-11-05	2024-09-10 06:00:09.540
C-4	Shanika Bang	1978-09-10	2024-09-10 06:00:09.540
C-5	Tequila Carmen	1990-02-18	2024-09-10 06:00:09.540
C-6	Tomi Hatt	1988-06-25	2024-09-10 06:00:09.540
C-8	Odella Thompkins	1985-12-14	2024-09-10 06:00:09.540
C-10	Hilde Mustard	1999-01-08	2024-09-10 06:00:09.540
C-11	Shenika Critchfield	1987-10-03	2024-09-10 06:00:09.540
C-1A	Adalita Kinch	1992-08-01	2024-09-10 06:00:09.540

2. BRONZE_PRODUCTS table

The screenshot shows the Snowflake interface with the query `SELECT * FROM bronze_products;` executed. The results are displayed in a table with columns: PRODUCT_ID, PRODUCT_NAME, PRICE, and LOAD_TIME. The data includes 10 rows of product information. The right panel shows Query Details: duration 30ms, 100 rows, and a Query ID.

PRODUCT_ID	PRODUCT_NAME	PRICE	LOAD_TIME
P-1000	Asparagus	14.25	2024-09-10 06:00:09.712
P-1001	Broccoli	18.73	2024-09-10 06:00:09.712
P-1002	Carrots	5.48	2024-09-10 06:00:09.712
P-1003	Cauliflower	19.72	2024-09-10 06:00:09.712
P-1004	Celery	26.3	2024-09-10 06:00:09.712
P-1005	Corn	25.44	2024-09-10 06:00:09.712
P-1006	Cucumbers	8.95	2024-09-10 06:00:09.712
P-1007	Lettuce / Greens	19.46	2024-09-10 06:00:09.712
P-1008	Mushrooms	13.32	2024-09-10 06:00:09.712
P-1009	Onions	24.10	2024-09-10 06:00:09.712

3. BRONZE_STORES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM bronze_stores;` executed. The results are displayed in a table with columns: STORE_ID, STORE_NAME, and LOAD_TIME. The data includes 11 rows of store information. The right panel shows Query Details: duration 298ms, 11 rows, and a Query ID.

STORE_ID	STORE_NAME	LOAD_TIME
S-1U	Henderson	2024-09-10 06:00:12.388
S-9	Whangaparaoa	2024-09-10 06:00:12.388
S-8	Westgate	2024-09-10 06:00:12.388
S-5	Massey	2024-09-10 06:00:12.388
S-1	Queen St.	2024-09-10 06:00:12.388
S-7	East Auckland	2024-09-10 06:00:12.388
S-6	West Auckland	2024-09-10 06:00:12.388
S-4	St. James	2024-09-10 06:00:12.388
S-2	Albany	2024-09-10 06:00:12.388
S-3	Manukau	2024-09-10 06:00:12.388

4. BRONZE_SALES table

Databases Worksheets

HIMA_DB.BRONZE

```
36
37
38
39 | SELECT * FROM bronze_sales;
40
```

Results

TRANSACTION_ID	PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	LOAD_TIME
1	733	P-1022	C-37	S-4	2023-12-31	8 2024-09-10 06:00:14.902
2	2483	P-1066	C-32	S-6	2023-12-31	8 2024-09-10 06:00:14.902
3	3283	P-1067	C-40	S-9	2023-12-31	5 2024-09-10 06:00:14.902
4	3853	P-1018	C-44	S-6	2023-12-31	8 2024-09-10 06:00:14.902
5	5842	P-1081	C-26	S-1	2023-12-31	8 2024-09-10 06:00:14.902
6	6641	P-1094	C-32	S-2	2023-12-31	1 2024-09-10 06:00:14.902
7	7270	P-1035	C-43	S-4	2023-12-31	4 2024-09-10 06:00:14.902
8	8452	P-1065	C-17	S-10	2023-12-31	2 2024-09-10 06:00:14.902
9	8059	P-1022	C-31	S-2	2023-12-31	5 2024-09-10 06:00:14.902
10	0421	P-1019	C-42	S-6	2023-12-31	4 2024-09-10 06:00:14.902

Query Details

- Query duration: 450ms
- Rows: 6.2K
- Query ID: 01b6f052-020a-5c40-0...

Ask Copilot

SILVER LAYER TABLES

1. SCD2_SILVER_CUSTOMERS table

Databases Worksheets

HIMA_DB.SILVER

```
49
50
51 | SELECT * FROM scd2_silver_customers;
52
53
```

Results

SURROGATE_KEY	CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	START_DATE	END_DATE	IS_ACTIVE
1	C-1	Lezlie Trott	1980-03-15	2024-09-08	9999-12-31	TRUE
2	C-2	Joann Shortt	1995-07-22	2024-09-08	9999-12-31	TRUE
3	C-3	Violet Newingham	1982-11-05	2024-09-08	9999-12-31	TRUE
4	C-4	Shanika Bang	1978-09-10	2024-09-08	9999-12-31	TRUE
5	C-5	Tequila Carmen	1990-02-18	2024-09-08	9999-12-31	TRUE
6	C-6	Tomi Hatt	1988-06-25	2024-09-08	9999-12-31	TRUE
7	C-8	Odelia Thompkins	1985-12-14	2024-09-08	9999-12-31	TRUE
8	C-9	Bernardo Vancamp	1965-08-20	2024-09-08	2024-09-09	FALSE
9	C-10	Hilde Mustard	1999-01-08	2024-09-08	9999-12-31	TRUE
10	C-11	Shanika Critchfield	1987-10-02	2024-09-08	9999-12-31	TRUE

Query Details

- Query duration: 281ms
- Rows: 50
- Query ID: 01b6f054-020a-5e5f-0...

Ask Copilot

2. SCD2_SILVER_PRODUCTS table

On Day 3, in the staging_products table, the price value for Product ID P-1079 was updated.

This change was reflected in the SCD2 silver table, where the old record was marked as inactive, and the updated record was inserted as active with appropriate start and end dates.

Databases Worksheets

HIMA_DB.SILVER

```
55
56
57 | SELECT * FROM scd2_silver_products;
```

Results

SURROGATE_KEY	PRODUCT_ID	PRODUCT_NAME	PRICE	START_DATE	END_DATE	IS_ACTIVE
77	77	Rice	4.28	2024-09-08	9999-12-31	TRUE
78	78	Tea	12.27	2024-09-08	9999-12-31	TRUE
79	79	Vegetable oil	4.57	2024-09-08	9999-12-31	TRUE
80	P-1079	Vinegar	9	2024-09-08	2024-09-10	FALSE
81	81	Applesauce	16.55	2024-09-08	9999-12-31	TRUE
82	82	Baked beans	7.74	2024-09-08	9999-12-31	TRUE
83	83	Chili	13.23	2024-09-08	9999-12-31	TRUE
84	84	Fruit	7.02	2024-09-08	9999-12-31	TRUE
85	85	Olives	1.28	2024-09-08	9999-12-31	TRUE

Query Details

- Query duration: 283ms
- Rows: 101
- Query ID: 01b6f055-020a-5c40-0...

Ask Copilot

The screenshot shows the Snowflake interface with the query `SELECT * FROM scd2_silver_products;` executed. The results table has columns: SURROGATE_KEY, PRODUCT_ID, PRODUCT_NAME, PRICE, START_DATE, END_DATE, IS_ACTIVE. The data includes various spices and their details.

SURROGATE_KEY	PRODUCT_ID	PRODUCT_NAME	PRICE	START_DATE	END_DATE	IS_ACTIVE
92	P-1091	Black pepper	2.0	2024-09-08	9999-12-31	TRUE
93	P-1092	Cilantro	12.88	2024-09-08	9999-12-31	TRUE
94	P-1093	Cinnamon	8.63	2024-09-08	9999-12-31	TRUE
95	P-1094	Garlic	12.07	2024-09-08	9999-12-31	TRUE
96	P-1095	Ginger	24.6	2024-09-08	9999-12-31	TRUE
97	P-1096	Mint	22.47	2024-09-08	9999-12-31	TRUE
98	P-1097	Oregano	20.8	2024-09-08	9999-12-31	TRUE
99	P-1098	Paprika	18.2	2024-09-08	9999-12-31	TRUE
100	P-1099	Parsley	10.51	2024-09-08	9999-12-31	TRUE
101	P-1079	Vinegar	9.82	2024-09-10	9999-12-31	TRUE

3. SCD2_SILVER_STORES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM scd2_silver_stores;` executed. The results table has columns: SURROGATE_KEY, STORE_ID, STORE_NAME, START_DATE, END_DATE, IS_ACTIVE. The data includes various store locations and their details.

SURROGATE_KEY	STORE_ID	STORE_NAME	START_DATE	END_DATE	IS_ACTIVE
2	S-10	Henderson	2024-09-08	9999-12-31	TRUE
3	S-9	Whangaparaora	2024-09-08	9999-12-31	TRUE
4	S-8	Westgate	2024-09-08	9999-12-31	TRUE
5	S-5	Massey	2024-09-08	9999-12-31	TRUE
6	S-1	Queen St.	2024-09-08	9999-12-31	TRUE
7	S-7	East Auckland	2024-09-08	9999-12-31	TRUE
8	S-6	West Auckland	2024-09-08	9999-12-31	TRUE
9	S-4	St. James	2024-09-08	9999-12-31	TRUE
10	S-2	Albany	2024-09-08	9999-12-31	TRUE

4. SCD2_SILVER_SALES table

The screenshot shows the Snowflake interface with the query `SELECT * FROM silver.scd2_silver_sales;` executed. The results table has columns: PRODUCT_ID, CUSTOMER_ID, STORE_ID, T_DATE, QUANTITY, START_DATE, END_DATE, IS_ACTIVE. The data includes sales transactions with various products, customers, and stores.

PRODUCT_ID	CUSTOMER_ID	STORE_ID	T_DATE	QUANTITY	START_DATE	END_DATE	IS_ACTIVE
P-1043	C-30	S-9	2023-12-31	10	2024-09-08	9999-12-31	TRUE
P-1059	C-12	S-6	2023-12-31	7	2024-09-08	9999-12-31	TRUE
P-1022	C-37	S-4	2023-12-31	8	2024-09-08	9999-12-31	TRUE
P-1015	C-23	S-6	2023-12-31	9	2024-09-08	9999-12-31	TRUE
P-1090	C-36	S-8	2023-12-31	6	2024-09-08	9999-12-31	TRUE
P-1016	C-46	S-2	2023-12-31	4	2024-09-08	9999-12-31	TRUE
P-1066	C-32	S-6	2023-12-31	8	2024-09-08	9999-12-31	TRUE
P-1062	C-47	S-3	2023-12-31	5	2024-09-08	9999-12-31	TRUE
P-1006	C-12	S-9	2023-12-31	5	2024-09-08	9999-12-31	TRUE

TASK Status for LOADDATA_INTO_STAGING_TASK over 3 days

This task is responsible for loading data into the staging tables. The status will indicate whether the task is currently running, completed, scheduled, or if there are any errors or issues during execution.

HIMA_DB / PIPELINE / LOADDATA_INTO_STAGING_TASK

Root Task ACCOUNTADMIN 2 days ago HIMA_WH

Task Details Graph Run History

Last 7 days Task Status All

Succeeded (6)

6 Task Runs

SCHEDULED TIME	STATUS	RETURN VA...	DURATION
Sep 10, 2024, 9:00:00 AM	Succeeded	—	20s
Sep 9, 2024, 9:00:00 AM	Succeeded	—	28s
Sep 8, 2024, 9:00:00 AM	Succeeded	—	17s
Sep 7, 2024, 4:16:00 PM	Succeeded	—	14s
Sep 7, 2024, 4:07:00 PM	Succeeded	—	6.1s

Query - 01b6f00c-020a-5e5f-0008-7dff00010006

HIMA_WH SYSTEM

Query Details Query Profile

Details

Status Success	Duration 19s	Driver Status N/A
Start Time 9/10/2024, 9:00:00 AM	Query ID 01b6f00c-020a-5e5f-0008-7dff00010006	Client Driver SYSTEM-computeService
End Time 9/10/2024, 9:00:19 AM	Query Tag —	Session ID 36473651205
Warehouse Size X-Small		

SQL Text

```
CALL hima_db.pipeline.loaddata_into_staging()
```

Results 1 of 1 Row • Cluster 1

LOADDATA_INTO_STAGING
1 Data loaded successfully

TASK Status for COPYDATA_STAGING_TO_BRONZE_TASK over 3 Days After

LOADDATA_INTO_STAGING_TASK Execution. This task runs daily after the

`LOADDATA_INTO_STAGING_TASK` completes. The purpose of this task is to copy data from the staging tables to the bronze tables while enforcing the defined data quality rules.

The figure consists of three vertically stacked screenshots from the Snowflake interface.

Screenshot 1: Pipeline Task Status

This screenshot shows the `HIMA_DB / PIPELINE / COPYDATA_STAGING_TO_BRONZE_TASK`. The task has a status of "Succeeded (6)". A timeline chart shows successful runs from Sep 7 to Sep 10. Below the chart is a table of "6 Task Runs" with columns: SCHEDULED TIME, STATUS, RETURN VA..., and DURATION. The runs are listed as follows:

SCHEDULED TIME	STATUS	RETURN VA...	DURATION
Sep 10, 2024, 9:00:20 AM	Succeeded	—	10.0s
Sep 9, 2024, 9:00:29 AM	Succeeded	—	5.4s
Sep 8, 2024, 9:00:20 AM	Succeeded	—	6.3s
Sep 7, 2024, 4:16:15 PM	Succeeded	—	7.5s
Sep 7, 2024, 4:07:07 PM	Succeeded	—	6.5s

Screenshot 2: Query History - Success

This screenshot shows a query history entry for a successful query. The details include:

- Status: Success
- Duration: 8.6s
- Start Time: 9/10/2024, 9:00:21 AM
- End Time: 9/10/2024, 9:00:30 AM
- Warehouse Size: X-Small
- Query ID: 01b6f00c-020a-5c40-0000-00087dfffc6bd
- Client Driver: SYSTEM-computeService
- Session ID: 36473639381

Screenshot 3: Query History - SQL Text

This screenshot shows the SQL text for the query:

```
CALL hima_db.pipeline.copydata_staging_to_bronze()
```

Screenshot 4: Query History - Results

This screenshot shows the results of the query, which completed successfully with 0 failures:

	COPYDATA_STAGING_TO_BRONZE
1	Execution completed. Successful: 6, Failed: 0

`TASK Status for UPDATE_ALL_SCD2_SILVER_TABLES_TASK Over 3 Days.` This task runs automatically every day, immediately following the completion of the

`LOADDATA_INTO_STAGING_TASK`. The task is designed to update the SCD Type 2 silver tables with the latest data from the base staging tables.

HIMA_DB / PIPELINE / UPDATE_ALL_SCD2_SILVER_TABLES_TASK

Child Task: ACCOUNTADMIN 2 days ago HIMA_WH

Task Details Graph Run History

Last 7 days Task Status All

Succeeded (6)

3

0 Sep 4 Sep 5 Sep 6 Sep 7 Sep 8 Sep 9 Sep 10

6 Task Runs

SCHEDULED TIME	STATUS	RETURN VA...	DURATION
Sep 10, 2024, 9:00:20 AM	Succeeded	—	13s
Sep 9, 2024, 9:00:29 AM	Succeeded	—	7.4s
Sep 8, 2024, 9:00:20 AM	Succeeded	—	6.4s
Sep 7, 2024, 4:16:15 PM	Succeeded	—	7.4s
Sep 7, 2024, 4:07:07 PM	Succeeded	—	7.4s

Query - 01b6f00c-020a-5c40-0000-00087dfffc6c1

HIMA_WH SYSTEM

Query Details Query Profile

Details

Status	Duration	Driver Status
Success	12s	N/A
Start Time	Query ID	Client Driver
9/10/2024, 9:00:21 AM	01b6f00c-020a-5c40-0000-00087dfffc6c1	SYSTEM-computeService
End Time	Query Tag	Session ID
9/10/2024, 9:00:33 AM	—	36473639377
Warehouse Size		
X-Small		

SQL Text

```
CALL hima_db.pipeline.update_all_scd2_silver_tables()
```

Results 1 of 1 Row • Cluster 1

Export Results

	UPDATE_ALL_SCD2_SILVER_TABLES
1	SCD2 Update Complete for All Silver Tables

POWER BI

In Power BI, I developed an interactive sales dashboard utilizing data from the Silver schema (SCD Type 2 tables) to visualize key business metrics and trends. The data model integrated sales transactions with customer, product, and store details, allowing for historical analysis and current state tracking.

1. Data Merging and Enrichment: I enriched the sales data by merging the scd2_silver_sales table with the scd2_silver_customers, scd2_silver_products, and scd2_silver_stores tables, using foreign key relationships (such as CUSTOMER_ID, PRODUCT_ID, and STORE_ID) to bring in relevant details like CUSTOMER_NAME, PRODUCT_NAME, and STORE_NAME.

2. Slicers for Dynamic Filtering: Implemented a Price Range slicer using the PRICE field to filter products based on their prices.

Added a Time Range slicer using the T_DATE (transaction date) field, allowing dynamic filtering of sales transactions by date range.

3. KPI Cards: Created Total Users (Customers) KPI by counting distinct CUSTOMER_IDs, showing the total number of active users in the dataset.

Added a Total Products KPI by counting distinct PRODUCT_IDs, reflecting the number of unique products available.

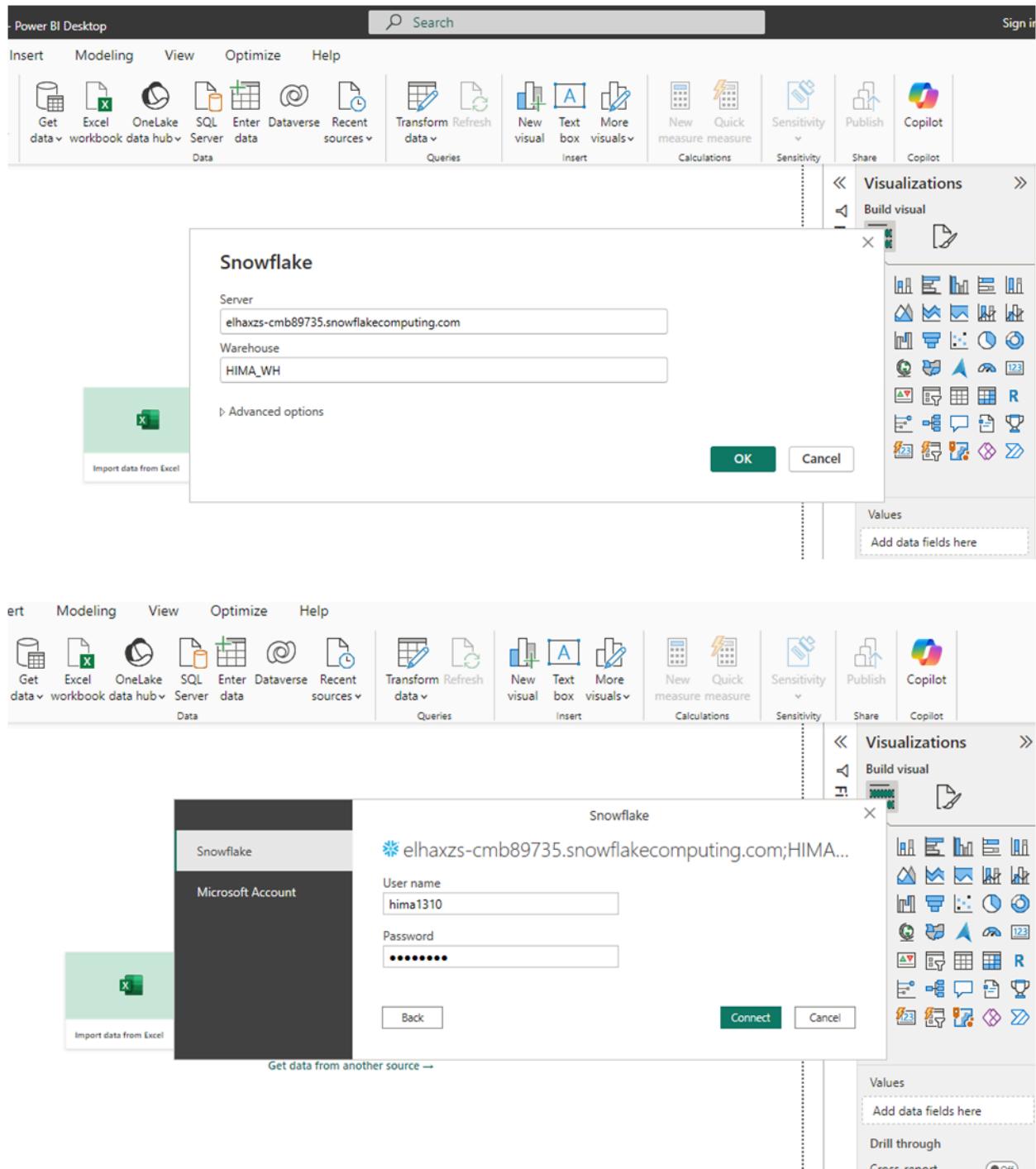
Additional KPI metrics such as Total Sales, Total Revenue, and Customers Left were added, utilizing the respective sales and customer data.

4. Top 10 Sold Products Table: Built a table to display the Top 10 Sold Products by sales quantity, using PRODUCT_NAME and QUANTITY fields. Applied a top 10 filter to showcase the best-performing products in terms of sales volume.

5. Bar Chart for Daily Sales: Created a Daily Sales Bar Chart, visualizing units sold per day. T_DATE was used on the X-axis to represent dates, and QUANTITY on the Y-axis to display the number of units sold. This chart provides insights into sales trends over time.

6. Table for Transaction Data: Added a Table Visualization to display individual transaction details. Fields such as CUSTOMER_NAME, PRODUCT_NAME, PRICE, T_DATE, and QUANTITY were included, allowing users to inspect and explore specific transactions in the dataset.

7. Data Integrity: Ensured that the data used in these visualizations adhered to the Silver schema's principles of historical and current state analysis, with Slowly Changing Dimension (SCD) Type 2 logic for capturing changes over time.



Power BI Desktop

Navigator

Display Options

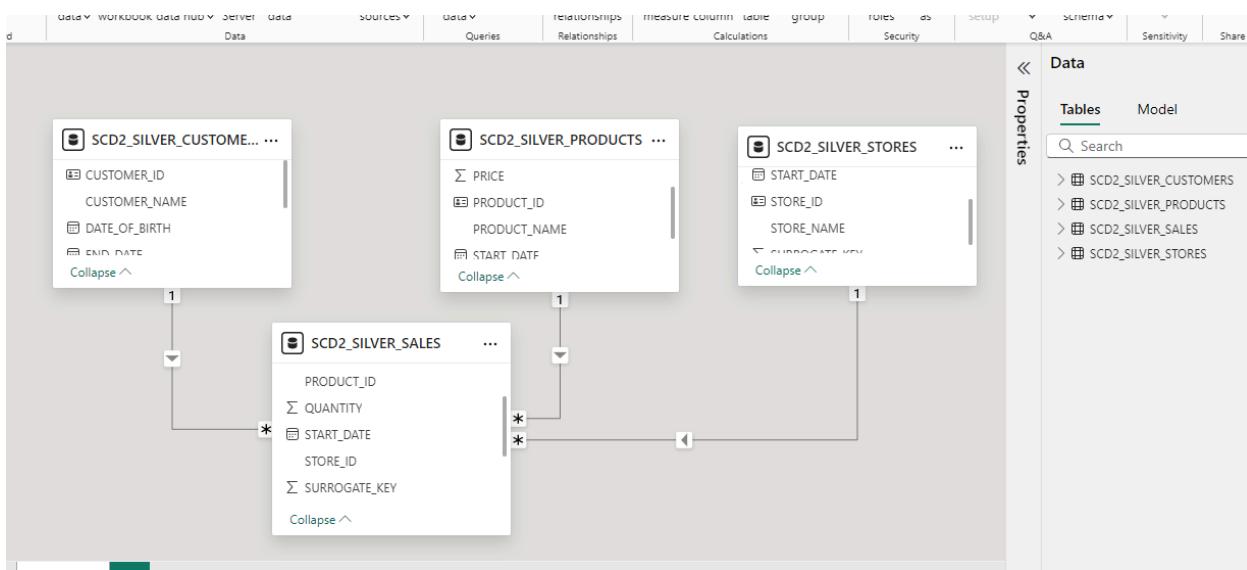
- elhaxzs-cmb89735.snowflakecomputing.com:HIMA_DB [5]
 - BRONZE
 - PIPELINE
 - PUBLIC
 - SILVER [4]
 - SCD2_SILVER_CUSTOMERS
 - SCD2_SILVER_PRODUCTS
 - SCD2_SILVER_SALES
 - SCD2_SILVER_STORES
 - STAGING
 - SALES_DWH
 - SNOWFLAKE
 - SNOWFLAKE_SAMPLE_DATA

SCD2_SILVER_CUSTOMERS

SURROGATE_KEY	CUSTOMER_ID	CUSTOMER_NAME	DATE_OF_BIRTH	START_DATE
1	C-1	Lezlie Trott	15-03-1980	
2	C-2	Joann Shortt	22-07-1995	
3	C-3	Violet Newingham	05-11-1982	
4	C-4	Shanika Bang	10-09-1978	
5	C-5	Tequila Carmen	18-02-1990	
6	C-6	Tomi Hatt	25-06-1988	
7	C-8	Odelia Thompkins	14-12-1985	
8	C-9	Bernardo Vancamp	20-08-1965	
9	C-10	Hilde Mustard	08-01-1999	
10	C-11	Shenika Critchfield	03-10-1987	
11	C-14	Adolfo Kinch	01-09-1983	
12	C-17	Lachelle Cianciolo	07-05-1998	
13	C-18	Enoch Whitford	23-01-1970	
14	C-22	Maren Paulos	03-07-1984	
15	C-24	Ben Eby	05-10-1966	
16	C-26	Ellie Kegley	31-01-1996	
17	C-28	Chaya Stalcup	02-04-1989	
18	C-31	Elsy Tusing	21-04-1973	
19	C-32	Demetra Tesar	06-11-1969	
20	C-35	Dominick Smartt	25-09-1978	
21	C-37	Winston Ohara	28-12-1967	
22	C-39	Alba Weideman	30-04-1983	
23	C-40	Clara Kozel	15-11-1972	

Load Transform Data Cancel

Model View



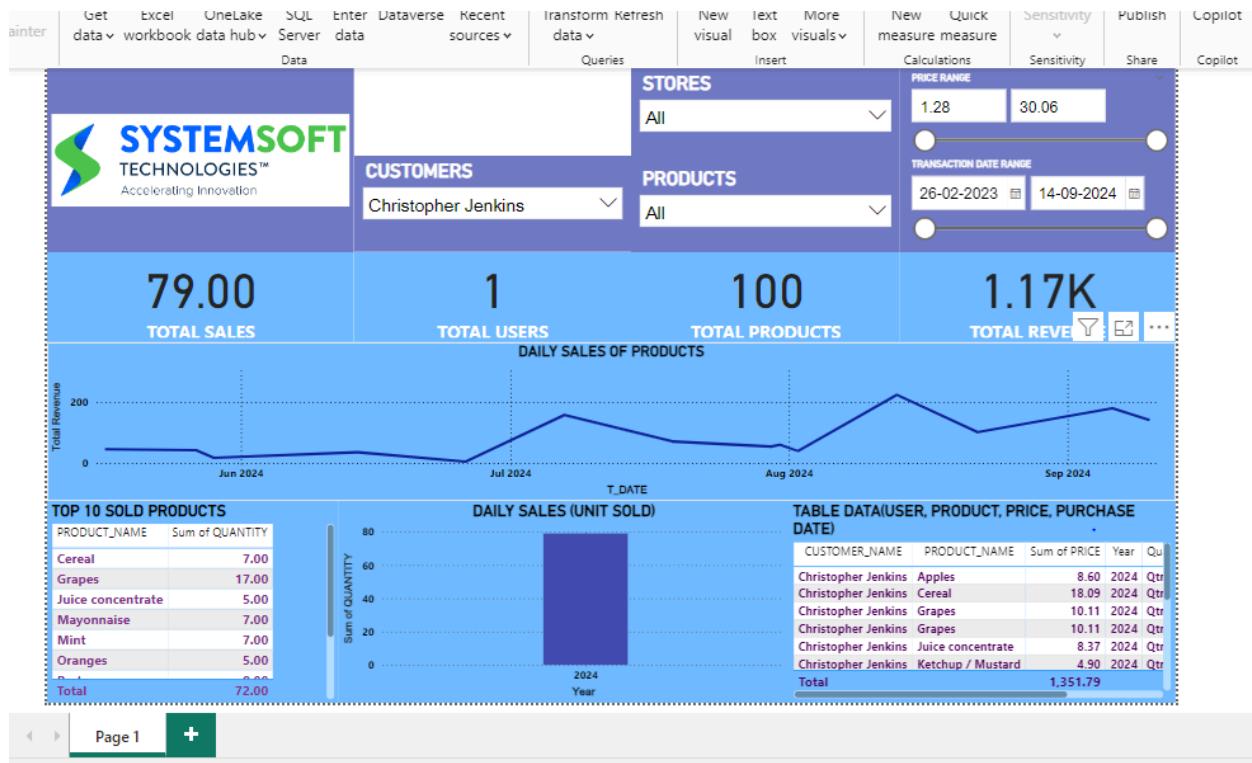
Report View

The screenshot shows the Power BI Report View with a dashboard titled "SYSTEMSOFT TECHNOLOGIES™ Accelerating Innovation". The dashboard includes the following elements:

- KPIs:** Total Sales (34.44K), Total Users (49), Total Products (100), and Total Revenue (468.17K).
- Line Chart:** Daily Sales of Products over time (T_DATE) from April 2023 to July 2024.
- Bar Chart:** Daily Sales (Unit Sold) by Year (2023 and 2024).
- Table:** TOP 10 SOLD PRODUCTS (e.g., Cherries, Fries / Tater tots, Ginger, Kiwis, Mac and cheese, Pasta sauce, Total: 4,740.00).
- Table:** TABLE DATA(USER, PRODUCT, PRICE, PURCHASE DATE) showing purchases for Adolfo Kinch (Apples, Applesauce) across different quarters.

The screenshot shows the Power BI Report View with the same dashboard, but now filtered for a single customer. The visualizations have been updated to reflect this change, such as the daily sales chart showing activity for a specific customer.

I applied a filter to the Power BI report to focus on a single customer. This dynamically updated all the visualizations, including KPIs, tables, and charts, to display data exclusively related to that customer. The transaction table now shows the selected customer's purchases, including product details, prices, transaction dates, and quantities. Other elements, such as the top sold products and daily sales chart, were also refined to reflect the specific customer's activity.



Adding Store and Product Details:

To provide a comprehensive view, we extended the visualizations to include details of products and stores associated with the selected customer. The transaction table now displays relevant product names and store names, reflecting the customer's purchases. KPIs and charts, such as those showing top-selling products and daily sales by store, are dynamically updated to highlight metrics specifically for the selected customer.

