

My First Attempt at Building a Data Warehouse

Contents:

- 1 Intro**
- 2 Project Resources & Plan Development**
- 3 Project Requirements**
- 4 Designing the Data Architecture**
 - a. Choosing the Data Management Approach
 - b. Define Architecture Layers
 - c. Build High-Level Architecture Diagram
- 5 Project Initialization**
 - a. Define Naming Conventions
 - b. Set up Git Repo
 - c. Create Schemas
- 6 Bronze Layer Build**
 - a. Analyzing: Source Systems
 - b. Coding: Data Ingestion
 - c. Validating: Data Completeness & Schema Checks
 - d. Document: Draw Data Flow Diagram
 - e. Commit Code to Git
- 7 Silver Layer Build**
 - a. Analyzing: Explore & Understand the Data
 - b. Document: High-Level Data Integration Diagram
 - c. Coding & Validating: Data Cleansing & Quality Checks
 - d. Document & Commit Code to Git
- 8 Gold Layer Build**
 - a. Analyzing: Explore Business Objects
 - b. Coding & Validating: Data Integration & Quality Checks
 - c. Document & Commit Code to Git
- 9 Exploratory SQL Data Analysis**
- 10 SQL Advanced Analytics**

1. INTRO

This project represents my journey in learning SQL and building a data warehouse in SQL Server through a guided project delivered by Baraa Khatib Salkini. The project was completed as part of Baraa's course '[The Complete SQL Bootcamp](#)'.

I strongly recommend the course for anyone seeking to learn SQL and the basics of data warehouse design.

Project Overview

I built a three-layer Medallion Architecture in SQL Server, ingesting and integrating data from fictitious CRM and ERP source systems and applying extensive data-quality checks, cleansing rules, and key-mapping logic.

In the Silver and Gold layers, I standardized and joined cross-system customer, product, and sales data, and designed a Star Schema with dimension and fact views to support analytics.

Throughout the project, I used SQL for modeling and transformation, Git for version control, and Draw.io for architectural and lineage documentation.

The final warehouse delivers a fully validated, analytics-ready dataset and demonstrates practical competency in SQL, data modeling, and data product development.

2. Project Resources & Plan Development

Project Resources:

Data

- o Cust_info.csv
- o Prd_info.csv
- o sales_details.csv
- o CUST_AZ12.csv
- o LOC_A101csv
- o PX_CAT_G1V2.csv

Data / Code Repository: [GitHub](#)

Project Management Tool: [Notion](#)

Diagram Development: [draw.io](#)

What I did to get started:

1. Obtained the project data from the instructor and saved locally on my machine
2. Created a Github account.
3. Downloaded and installed [draw.io](#) for desktop. [Draw.io](#) was used for the architecture diagram for this project.
4. Created a Notion account and set up high-level project plan with tasks for the project. Used the collection of Epics and tasks provided by the instructor, but made some tweaks as the project progressed..

Project Plan As Built in Notion

Aa. Name	Q. Progress	DWH Tasks
Requirements Analysis	100.00%	<ul style="list-style-type: none"> Analyze & Understand the Requirements
Design Data Architecture	100.00%	<ul style="list-style-type: none"> Choose Data Management Approach Brainstorm & Design the Layers Draw the Data Architecture (Draw.io)
Project Initialization	100.00%	<ul style="list-style-type: none"> Prepare the Git Repo & Prepare the Structure Create DB & Schemas Create Detailed Project Tasks (Notion) Define Project Naming Conventions
Build Bronze Layer	100.00%	<ul style="list-style-type: none"> Analyzing: Source Systems Coding: Data Ingestion Validating: Data Completeness & Schema Checks Document: Draw Data Flow (Draw.io) Commit Code in Git Repo
Build Silver Layer	100.00%	<ul style="list-style-type: none"> Analyzing: Explore & Understand Data Coding: Data Cleansing Validating: Data Correctness Checks Documenting & Versioning in GIT Commit Code in GIT Repo Document: Draw Data Integration (Draw.io)
Build Gold Layer	100.00%	<ul style="list-style-type: none"> Analyzing: Explore Business Objects Coding: Data Integration Validating: Data Integration Checks Document: Draw Data Model of Star Schema (Draw.io) Document: Create Data Catalog
Exploratory SQL Data Analysis	0.00%	<ul style="list-style-type: none"> Change Over Time Analysis Dimensions vs Measures Database Exploration Dimensions Exploration Date Exploration Measure Exploration Magnitude Analysis Ranking Analysis
SQL Advanced Analytics	0.00%	<ul style="list-style-type: none"> Change Over Time Analysis Cumulative Analysis Performance Analysis Part to Whole Analysis Data Segmentation Build Customers Report Build Products Report Document in GIT

3. Project Requirements

Objective

Develop a modern data warehouse using SQL Server to consolidate sales data, enabling analytical reporting and informed decision-making.

Specifications

- **Data Sources:** Import data from two source systems (ERP and CRM) provided as CSV files.
- **Data Quality:** Cleanse and resolve data quality issues prior to analysis.
- **Integration:** Combine both sources into a single, user-friendly data model designed for analytical queries.
- **Scope:** Focus on the latest dataset only; historization of data is not required.
- **Documentation:** Provide clear documentation of the data model to support both business stakeholders and analytics teams.

BI: Analytics & Reporting (Data Analysis)

Objective

Develop SQL-based analytics to deliver detailed insights into:

- Customer Behavior
- Product Performance
- Sales Trends

These insights empower stakeholders with key business metrics, enabling strategic decision-making.

4. Designing the Data Architecture

a. Choosing the Data Management Approach

In this task, I evaluated the requirements of the project and selected Data Warehouse as the appropriate approach from the following data architectures:

- Data Warehouse
- Data Lake
- Data Lakehouse
- Data Mesh

Next I learned about the types of data warehouse approaches, evaluated each against the requirements and selected the Medallion Architecture approach as it is most appropriate for this effort

- **Inmon:** Start with staging, next layer data is modeled in an EDW in 3NF, then broken into a third layer of data marts, then data BI tools are connected to the data marts
- **Kimbal:** Similar to Inmon, however, there is no EDW and data is moved from staging directly to the data marts as the EDW is time intensive. Kimbal is a faster approach than Inmon, but it can lead to chaos in the data mart with duplicate efforts
- **Data Vault:** Similar to Inmon however, the EDW layer is split into a Raw Vault (original data) and Business Vault (includes the business rules and transformations which prepare data for data marts)
- **Medallion Architecture:** Comprised of three layers (bronze, silver, and gold). Bronze Layer: Original data as-is. Enables traceability and issue identification; Silver: Transformed and cleansed data. Gold Layer: Similar to Data Mart in which business-ready objects are built and ready to share as products (machine learning, AI, reports) with business rules / logic applied.

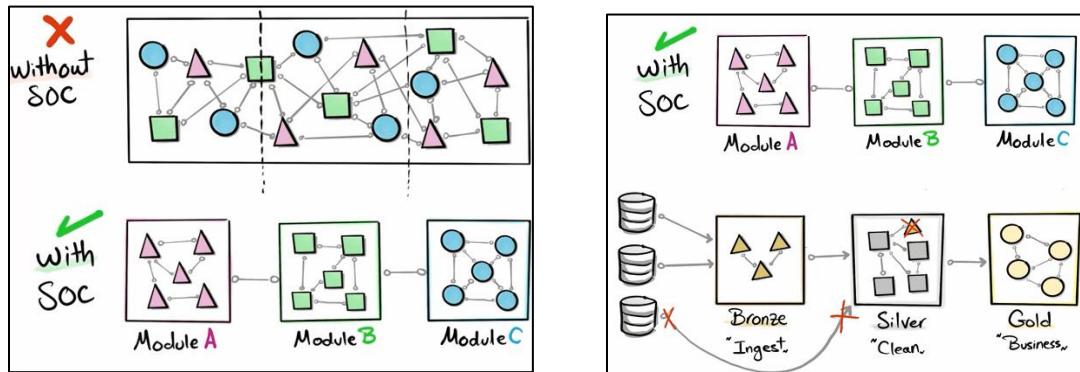
b. Define Architecture Layers

Summary:

Here I documented the definition and objective for each layer (Bronze, Silver, and Gold)

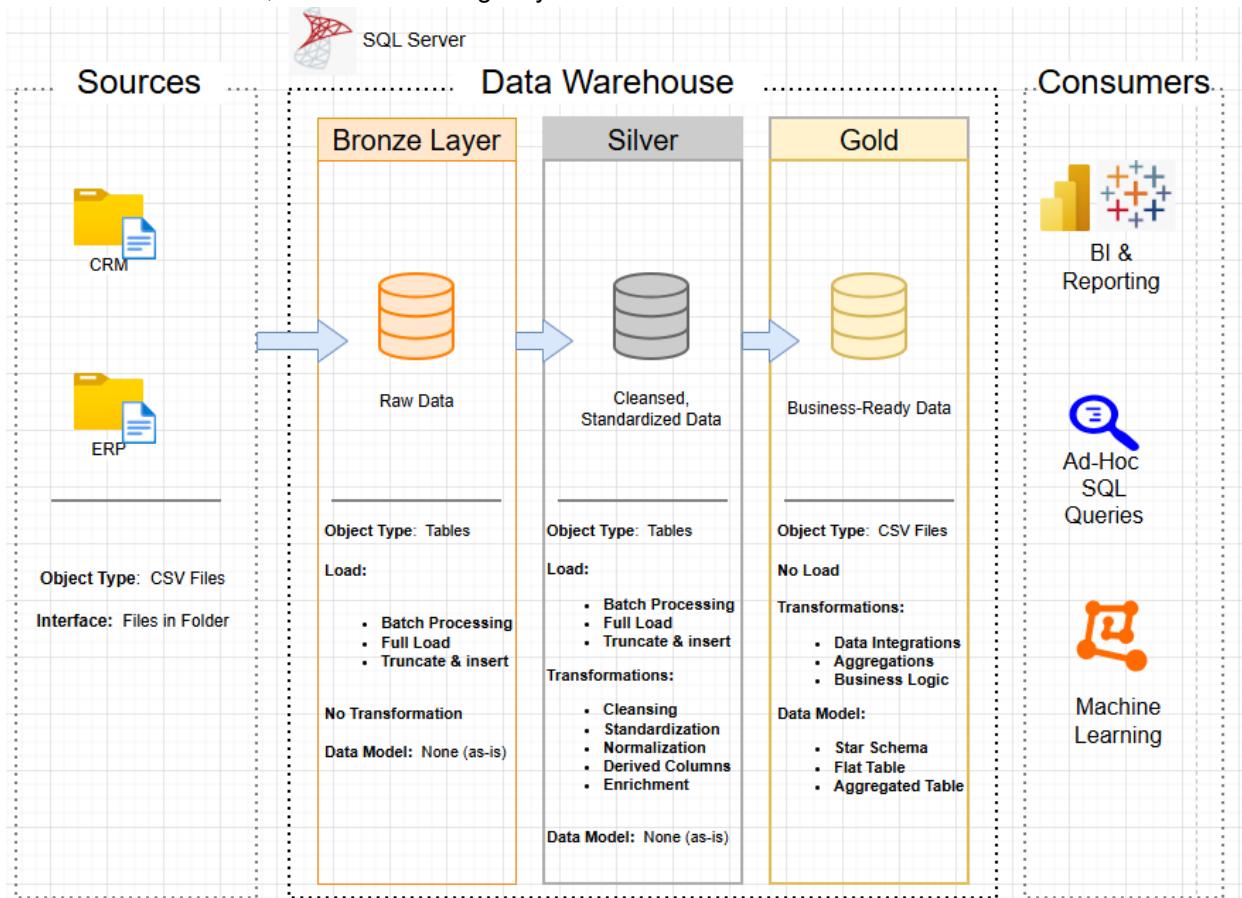
	Bronze	Silver	Gold Layer
Definition	Raw, unprocessed data as-is from sources	Cleansed and standardized data	Business-Ready Data
Objective	Traceability & Debugging	Intermediate Layer Prepare Data for Analysis	Provide data to be consumed for reporting & analytics
Object Type	Tables	Tables	Views
Load Method	Full Load (Truncate & Insert)	Full Load (Truncate & Insert)	None (not required as there are no tables)
Data Transformation	None (as-is)	<ul style="list-style-type: none">• Data Cleansing• Data Standardization• Data Normalization• Derived Columns• Data Enrichment	<ul style="list-style-type: none">• Data Integration• Data Aggregation• Business Logic & Rules
Data Modeling	None (as-is)	None (as-is)	<ul style="list-style-type: none">• Star Schema• Aggregated Objects• Flat Tables
Target Audience	Data Engineers	Data Analysts Data Engineers	Data Analysts Business Users

DOCUMENTING DETAIL IN THE TABLE ABOVE ENABLED SEPARATION OF CONCERNS (SOCs): Breaking the system into separate components with specific purposes with no overlap



c. Build High-Level Architecture Diagram

The objective here was to draw the data architecture using [Draw.io](#) according to the requirements, medallion architecture, and defined design layers within the architecture



5. Project Initialization

a. Define Naming Conventions

Here I defined the set of rules for naming everything in the project to ensure standardization so all members of the project are using the same naming conventions. This is done for:

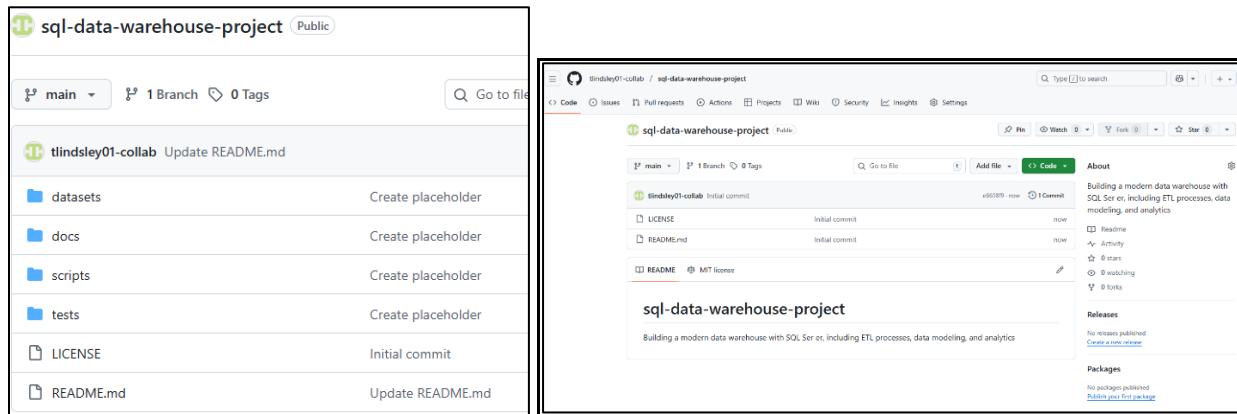
- The Database
- Schema
- Tables
- Stored Procedures, etc

b. Create Git Repo

For this task, I created a Github account, and learned how to create a repo structure for a project. A Git repo (repository) is a safe location for storing and tracking code used for a project and collaborating with the team. Additionally you can share your repository as part of your portfolio to show that you know how to build and maintain a well-documented git repository for your project.

I created my first repository in Git hub through this effort and gave it the name ‘sql-data-warehouse-project’. I then began creating folders and placeholders required for the project.

The project content structure in Git



c. Create Schemas

Below are the steps I took to create the database:

1. Open SQL Server Management Studio
2. Access the database master (system database) by creating a new query and entering: “USE MASTER”
3. Created the database with the following script in the same query window: “CREATE DATABASE DataWarehouse”;
4. I then went to the object explorer on the left hand side of SQL Server Management Studio, right clicked on ‘Databases’ and selected ‘Refresh’. The new database ‘DataWarehouse’ now appears in the collection of databases.

5. I now need to begin building the database. To do so, I entered 'USE DataWarehouse' in the same query window. The screenshot below shows the queries and outputs of steps 1-5

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left shows a connection to 'PF33Z2Q8\SQLEXPRESS (SQL Server 16.0.1150 - ANSWERTHINK)'. Under 'Databases', a new database named 'DataWarehouse' is selected. The 'SQLQuery16...lindsay (61)*' query window contains the following SQL code:

```

1 --CREATE Database 'Data Warehouse'
2 USE MASTER; --Entered this to access the database master (system database) in SQL Server
3
4 CREATE DATABASE DataWarehouse; --Once in the system database, I entered and ran this to create my new database
5
6 USE DataWarehouse; --Once the database was created, I entered and ran this to begin using the database for build-out of the db
7

```

The 'Messages' pane at the bottom right shows 'Commands completed successfully.' and a completion time of '2025-11-28T09:29:9980095-08:00'.

6. Next I executed the following query to create the schema for each layer of the medallion architecture (Bronze, Silver, and Gold)

Schema Creation SQL

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left shows a connection to 'PF33Z2Q8\SQLEXPRESS (SQL Server 16.0.1150 - ANSWERTHINK)'. Under 'Schemas', several schemas are listed: 'bronze', 'db_datareader', 'db_datawriter', 'db_ddladmin', 'db_denydatareader', 'db_denydatawriter', 'db_owner', 'db_securityadmin', 'dbo', 'gold', 'guest', 'INFORMATION_SCHEMA', 'silver', and 'sys'. The 'SQLQuery16...lindsay (61)*' query window contains the following SQL code:

```

1 --CREATE Database 'Data Warehouse'
2 USE MASTER; --Entered this to access the database master (system database) in SQL Server
3
4 CREATE DATABASE DataWarehouse; --Once in the system database, I entered and ran this to create my new database
5
6 USE DataWarehouse; --Once the database was created, I entered and ran this to begin using the database for build-out of the db
7
8
9 --Creates the schemas for each layer of the medallion architecture. They appear in
10 -- 'DataWarehouse' > 'Security' > 'Schemas'
11 CREATE SCHEMA bronze;
12 GO --This is the separator command that tells SQL to complete each command separately, then move to the next
13 CREATE SCHEMA silver;
14 GO --This is the separator command that tells SQL to complete each command separately, then move to the next
15 CREATE SCHEMA gold;
16 GO --This is the separator command that tells SQL to complete each command separately, then move to the next

```

The 'Messages' pane at the bottom right shows 'Commands completed successfully.' and a completion time of '2025-11-28T09:40:2450134-08:00'. A status bar at the bottom right indicates 'PF33Z2Q8\SQLEXPRESS (16.0 RTM) | ANSWERTHINK\tom.lindsay.'

7. Once Schemas were built for each layer, I then returned to Github to commit the code to my repo using the steps listed below. Completion of these steps also resulted in the completion of all tasks within the 'Project Initialization' EPIC in my project plan:

- a. Opened the 'scripts' folder in my sql-datawarehouse-project
- b. Selected 'Add file'
- c. Selected 'Create New File'
- d. Assigned file name 'init_database.sql'
- e. Pasted the code from SQL Server to document the steps taken to execute steps 1-6
- f. Added cautionary and informative commentary to the file to provide users of the script with an understanding of what the code will do and considerations before executing.
- g. Finally, I committed the code by selecting 'Commit Changes'

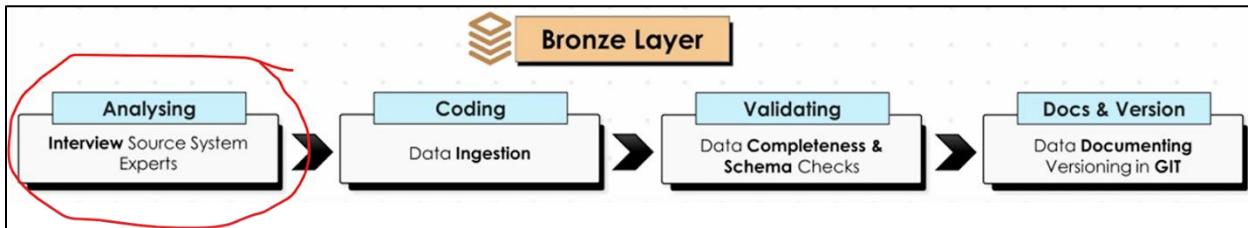
Code Commit in Git

The screenshot shows a GitHub repository interface. The top navigation bar includes 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main area is titled 'Code Commit in Git' and shows the file structure of 'sql-data-warehouse-project'. The 'Files' sidebar lists 'main', 'datasets', 'docs', 'scripts' (which contains 'init_database.sql' and 'placeholder'), 'tests', 'LICENSE', and 'README.md'. The right panel displays the contents of 'init_database.sql'. The code is as follows:

```
1  /*
2  =====
3  Create Database and Schemas
4  =====
5  Script Purpose:
6
7  This script creates a new database named 'DataWarehouse' after checking if it already exists.
8  If the database exists, it is dropped and recreated. Additionally, the script sets up three schemas
9  within the database: 'bronze', 'silver', and 'gold'.
10
11 --WARNING:
12 --Running this script will drop the entire 'DataWarehouse' database if it exists.
13 --All data in the database will be permanently deleted. Proceed with caution
14 --and ensure you have proper backups before running this script.
15 */
16 USE MASTER; --Entered this to access the database master (system database) in SQL Server
17 GO
18
19 --ONLY USE THE FOLLOWING IF RECREATING THE DATABASE. IF NOT...CONTINUE TO 'CREATE DATABASE SECTION BELOW.
20 --This will drop and recreate the 'DataWarehouse' database
21 IF EXISTS (SELECT 1 FROM sys.databases WHERE name = 'DataWarehouse')
22 BEGIN
23     ALTER DATABASE DataWarehouse SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
24     DROP DATABASE DataWarehouse;
25 END;
26 GO
```

6. Bronze Layer Build

a. Analyzing Source Systems



The first step in building the layer is to understand the source system. To do so in a normal project, I would conduct interviews with the source system owners / SMEs to understand the nature of the system to be connected to the warehouse. As this is a fictional project with fictional systems....no interviews were conducted

Conducting the interview allows you to understand:

- Business Context & Ownership
- Who owns the data
- What business processes it supports
- System and data documentation
- Data model and data catalog for the source system

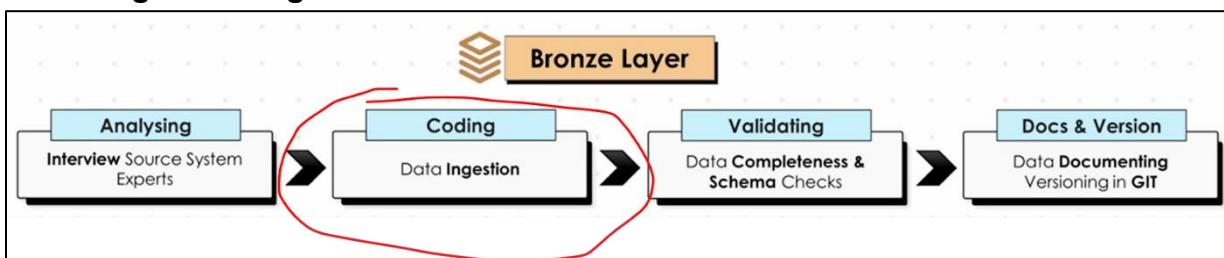
Architecture & Technology Stack

- How is the data stored? (On prem in SQL Server or Oracle. In the cloud on Azure or AWS)
- What are the integration capabilities of the source system? Is it through APIs, Kafka, File Extract, Direct DB connection?)

Extract & Load

- Can we do incremental loads vs full loads?
- Data scope & historical needs?
- Expected extract size (gigabytes / terabytes). Important for understanding whether we have the right tools and platforms to connect and extract source system data.
- Are there data volume limitations of the source system
- How do we avoid impacting the source system's performance in the connection and extraction
- What are the authentication and authorization requirements of the source system that need to be adhered to / made available to access and extract the data (tokens, SSH keys, VPN, IP whitelisting....)

b. Coding: Data Ingestion



The first step in this task is to understand the meta data, the structure, the schema of the incoming data before defining the structure of the tables for the Bronze layer in DDL (data definition language). This can be accomplished through the interviews with the source system SMEs / owners **and** through data

profiling / exploring the data on your own to identify column names and data types. If exploring on your own first, ensure that your assumptions are validated by the system owners / SMEs before building the bronze layer.

The following steps were used in this project to gain this understanding of the data sets provided for this project.

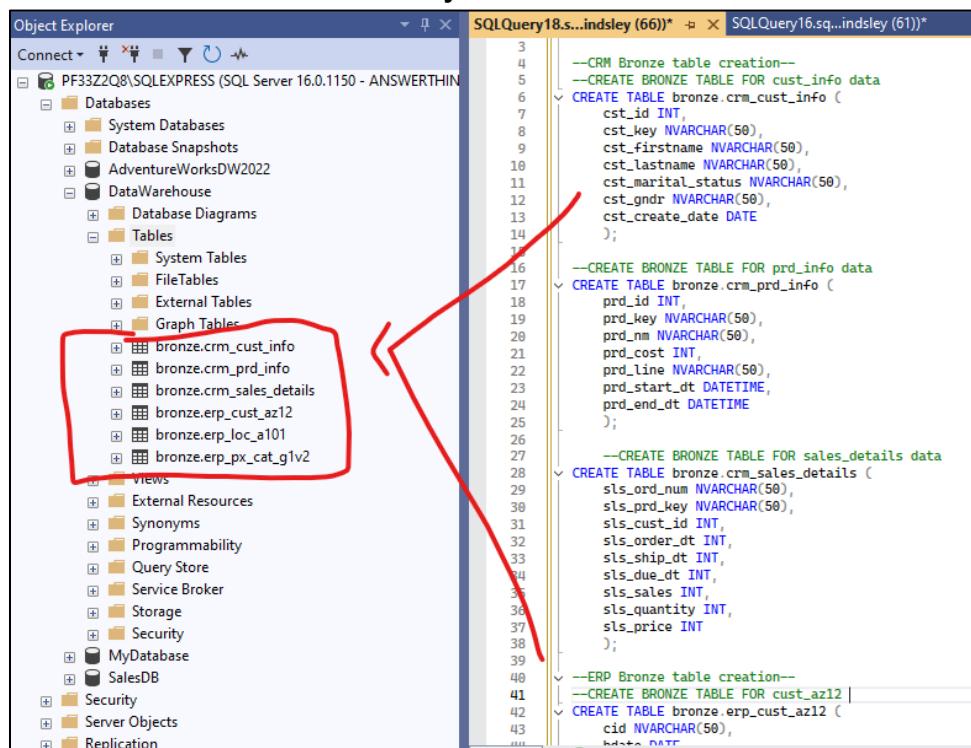
1. Open the 'cust_info.csv' data file for CRM and identify the column names, review the data and determine the data types
2. Open a new query in 'DataWarehouse' database and enter the following command to create the table in the bronze layer of the database (adhering to naming conventions defined previously in this project):

```
CREATE TABLE bronze.crm_cust_info
```
3. In the same query, create the following columns and assign their data types based on what is understood from the provided data set. The names should be identical to how they are defined in the data set

```
cst_id INT,
cst_key NVARCHAR(50),
cst_firstname NVARCHAR(50),
cst_lastname NVARCHAR(50),
cst_marital_status NVARCHAR(50),
cst_gndr NVARCHAR(50),
cst_create_date DATE
```

4. Execute the query, then refresh the tables in the database to confirm creation.
5. Execute steps 1 - 3 for all remaining files for both source systems

Bronze Layer Table Creation



The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer pane displays a tree structure of databases and objects. A red box highlights the 'Tables' node under the 'bronze' database. On the right, the SQL Query Editor pane contains several 'CREATE TABLE' statements for different bronze tables:

```

3  --CRM Bronze table creation--
4  --CREATE BRONZE TABLE FOR cust_info data
5
6  CREATE TABLE bronze.crm_cust_info (
7      cst_id INT,
8      cst_key NVARCHAR(50),
9      cst_firstname NVARCHAR(50),
10     cst_lastname NVARCHAR(50),
11     cst_marital_status NVARCHAR(50),
12     cst_gndr NVARCHAR(50),
13     cst_create_date DATE
14 );
15
16  --CREATE BRONZE TABLE FOR prd_info data
17  CREATE TABLE bronze.crm_prd_info (
18      prd_id INT,
19      prd_key NVARCHAR(50),
20      prd_nm NVARCHAR(50),
21      prd_cost INT,
22      prd_line NVARCHAR(50),
23      prd_start_dt DATETIME,
24      prd_end_dt DATETIME
25 );
26
27  --CREATE BRONZE TABLE FOR sales_details data
28  CREATE TABLE bronze.crm_sales_details (
29      sls_ord_num NVARCHAR(50),
30      sls_prd_key NVARCHAR(50),
31      sls_cust_id INT,
32      sls_order_dt INT,
33      sls_ship_dt INT,
34      sls_due_dt INT,
35      sls_sales INT,
36      sls_quantity INT,
37      sls_price INT
38 );
39
40  --ERP Bronze table creation--
41  --CREATE BRONZE TABLE FOR cust_az12 |
42  CREATE TABLE bronze.erpcust_az12 (
43      cid NVARCHAR(50),
44      bdate DATE
45 );

```

6. Loading the data: We are doing a bulk insert directly from files into the database. A single operation in one go. I wrote the script to load the cust_info data file into the bronze.crm_cust_info' table.

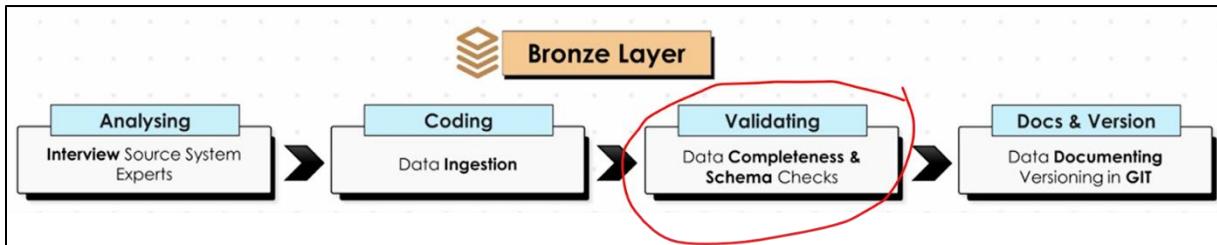
NOTE: I learned that this script will encounter an error if the file is open by a user. I had the file open when I attempted to execute the script and encountered just that. Closed the file and executed again successfully

--Script to load the cust_info data file into the bronze.crm_cust_info' table

```
BULK INSERT bronze.crm_cust_info
FROM 'C:\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
WITH (
    FIRSTROW = 2, --Tells SQL Server that the data starts on row 2 in the file
    FIELDTERMINATOR = ',', -- Tells SQL Server that commas are used as the separator between data for each column
    TABLOCK --Option to lock the table to improve performance while data is being loaded
);
```

c. Validating: Data Completeness & Schema Checks

This was essentially a continuation of the previous tasks conducted in 'Coding'



a. The next step was to run a query to view the data in the table to confirm that the data was pulled into the correct columns. I ran the following: SELECT * FROM bronze.crm_cust_info

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1	AW00011000	Jon	Yang	M	M	2025-10-06
2	AW00011001	Eugene	Huang	S	M	2025-10-06
3	AW00011002	Ruben	Tores	M	M	2025-10-06
4	AW00011003	Christy	Zhu	S	F	2025-10-06
5	AW00011004	Elizabeth	Johnson	S	F	2025-10-06
6	AW00011005	Julio	Ruiz	S	M	2025-10-06
7	AW00011006	Janet	Alvarez	S	F	2025-10-06
8	AW00011007	Marco	Mehta	M	M	2025-10-06
9	AW00011008	Rob	Verhoff	S	F	2025-10-06
10	AW00011009	Shannon	Carlson	S	M	2025-10-06
11	AW00011010	Jacquelyn	Suarez	S	F	2025-10-06
12	AW00011011	Curtis	Lu	M	M	2025-10-06
13	AW00011012	Lauren	Walker	M	F	2025-10-06
14	AW00011013	Ian	Jankine	M	M	2025-10-06

b. The previous steps were only a portion of the full code that needed to be written. The command needed to include a 'TRUNCATE' command to enable bulk load if the data is refreshed in the source file

without adding to the existing data in the table. Truncating will replace existing data with the new / refreshed data source. So....the command was rewritten and run again

```
--Script to load the cust_info data file into the bronze.crm_cust_info' table

TRUNCATE TABLE bronze.crm_cust_info; --Empties existing data in the table, then loads with refreshed data from scratch.
--This is a full load.

BULK INSERT bronze.crm_cust_info
FROM 'C:\Users\tom.lindsay\OneDrive - The Hackett Group, Inc\Desktop\Data Knowledge Development\SQL Bootcamp Files\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
WITH (
    FIRSTROW = 2, --Tells SQL Server that the data starts on row 2 in the file
    FIELDTERMINATOR = ',', -- Tells SQL Server that commas are used as the separator between data for each column
    TABLOCK --Option to lock the table to improve performance while data is being loaded
);

-- QUERY TO CHECK THE QUALITY OF THE DATA LOAD AND ENSURE DATA WENT TO THE CORRECT COLUMNS
SELECT * FROM bronze.crm_cust_info -- QUERY TO CHECK THE QUALITY OF THE DATA LOAD AND ENSURE DATA WENT TO THE
CORRECT COLUMNS

SELECT COUNT(*) FROM bronze.crm_cust_info ---- QUERY TO CHECK THAT THE TABLE ROW COUNT MATCHES THE CSV FILE
```

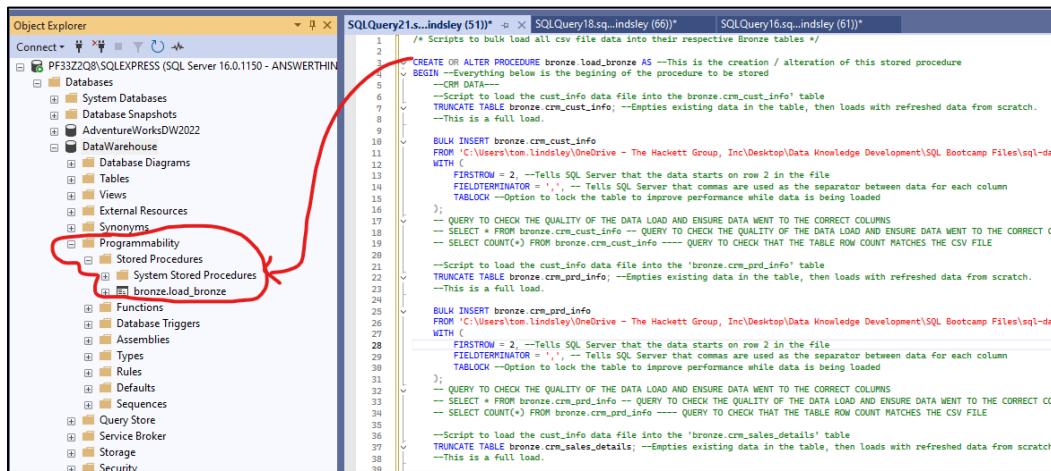
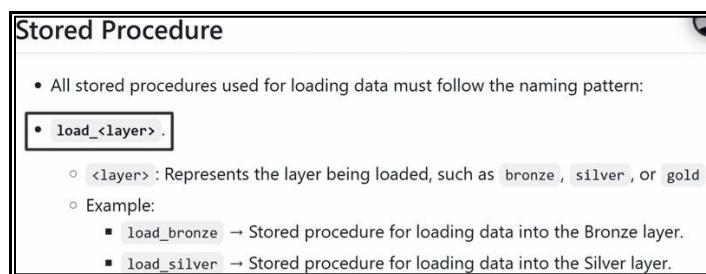
c. The same script was written and run for each data file for CRM and ERP data tables.

d. After running each script, I checked the results of the load against the source files to confirm that the data was placed into the correct columns and that the tables and the data sources had the same number of rows.

e. With the checks complete, I was good to move on to creating a stored procedure as this script would need to be run on a recurring basis.

f. Creating a stored procedure:

- I wrote a stored procedure for the loading of the data into the database as it is intended to be a recurring action to bring in the data.



- Once executing the stored procedure and refreshing the DB to visually confirm its existence, I then created a new query to test the execution of the procedure. The following SQL statement was used: EXEC bronze.load_bronze
- The message output in SQL Server Studio is fairly vague. I edited the script in the procedure to enhance the information produced in the message output for greater clarity / confirmation of successful load to the intended tables. The result of the enhancement:

The screenshot shows the 'Messages' window from SQL Server Studio. The log output is as follows:

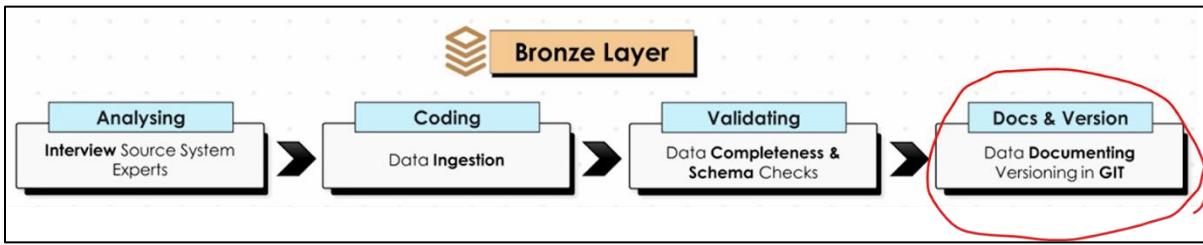
```

Messages
=====
Loading Bronze Layer
-----
Loading CRM Tables
-----
>>Truncating Table bronze.crm_cust_info
>>Inserting Data Into: bronze.crm_cust_info
(18493 rows affected)
>>Truncating Table bronze.crm_prd_info
>>Inserting Data Into: bronze.crm_prd_info
(397 rows affected)
>>Truncating Table bronze.crm_sales_details
>>Inserting Data Into: crm_sales_details
(60398 rows affected)
-----
Loading ERP Tables
-----
>>Truncating Table bronze.erp_cust_az12
>>Inserting Data Into: erp_cust_az12
(18483 rows affected)
>>Truncating Table bronze.erp_loc_a101
>>Inserting Data Into: erp_loc_a101
(18484 rows affected)
>>Truncating Table bronze.erp_px_cat_g1v2
>>Inserting Data Into: erp_px_cat_g1v2

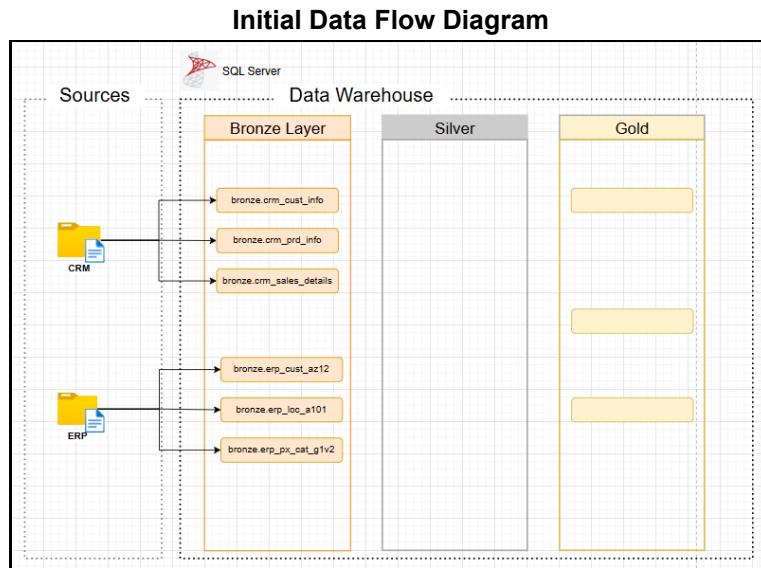
```

- The next step was to further enhance the stored procedure by adding a TRY CATCH command to handle errors that might occur in execution
- I also added script to include the duration of the loading effort for each table. To do this I declared two variables just before the 'BEGIN TRY' command for start time and end time. These variables will be used to output the execution duration that will be included in the message detail

d. Document: Draw Data Flow (Draw.io)

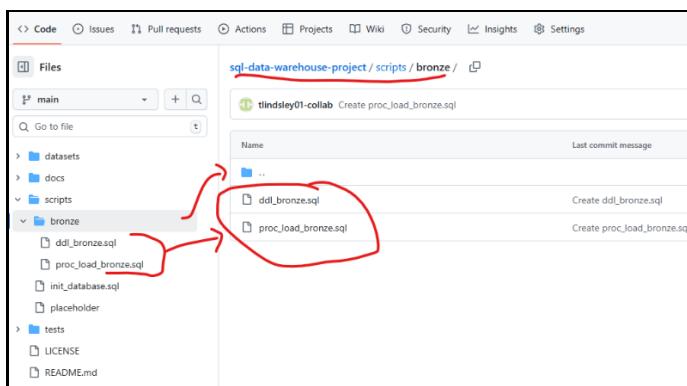


1. For the first step in this task, I drew a simple diagram in [Draw.io](#) of the flow of data from sources to the tables in the bronze layer to document data lineage. The output completed for the bronze layer is depicted below.

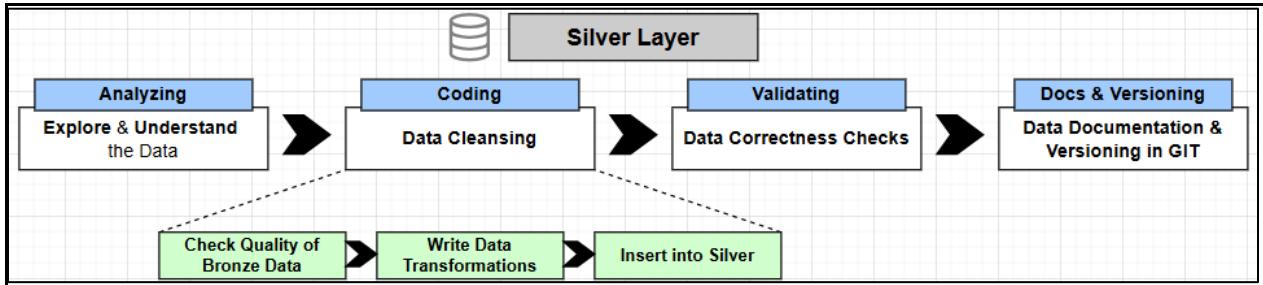


d. Commit Code to Git

The last task in this Epic is to commit my code into my Git repository. All of the code generated for table creation, and table loading for the bronze layer were added as separate files in the repository in a folder specific to the bronze layer.



7. Silver Layer Build

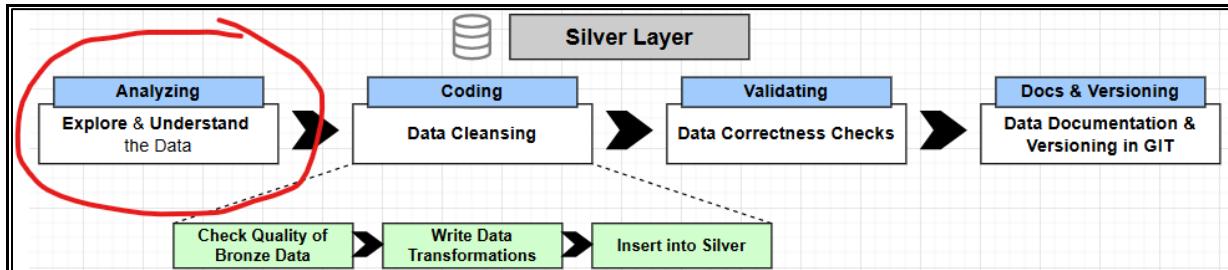


It is now time to build the Silver layer of the data warehouse now that the bronze layer has been built, documented, and its supporting code has committed to my Git repo.

This Epic is comprised of the following key tasks that will be completed in the order in which they are displayed below:

- Analyzing: Explore & Understand Data
- Coding: Data Cleansing
 - Check Quality of Bronze Data
 - Write Data Transformations
 - Insert into Silver Layer
- Validating: Data Correctness Checks
- Documenting & Versioning in GIT
- Commit Code in GIT Repo

a. Analyzing: Explore & Understand Data



Exploring and understanding the data is key to building out the Silver layer, the relationships, etc. To do this I explored each table one-by-one in the bronze layer.

For each table, I selected the top 1000 rows to get an initial understanding. This can be done by creating and executing the query below, or (in SQL Server) by right clicking on the desired table and selecting 'Select Top 1000 Rows'.

A key step in the data exploration task is to build an integration model to document what I observe / assume about how the tables are related. This is important so that I would not later forget what I observed

CRM Data Tables

- **bronze.crm_cust_info:** This table contains customer info with standard customer detail such as first name, last name, customer creation date. It includes a customer id (cust_id) as the primary key as well as a customer key (cust_key) which seems duplicative. An initial view of this table indicates that both the cust_id and the cust_key could be used to define relationships with other tables.
- **bronze.crm_prd_info:** This table contains basic product information along with historical information (historical product pricing). It includes a product id (prd_id) that appears to serve as the primary key, but it also includes a product key (prd_key). Like the cust_info table, an initial view of this table indicates that two columns (prd_id and prd_key) could be used to define relationships with other tables. I will have to determine which is most appropriate as I review the other tables.
- **bronze.crm_sales_details:** This table contains historical transactional records about sales and orders. Each record includes an order number (sls_ord_num), a product key for the product in the order (sls_prd_key), the id of the customer that placed the order (sls_cust_id), as well as order date, ship date, total order sales, sales quantity, and sales price. It appears that the sls_ord_num is the primary key for this table.

It appears that the sls_prd_key can be used to define the relationship with the crm_prd_info table but the value in the sls_prd_key for each record appears to only be a subset of the value listed for each record in the prd_key field of the crm_prd_info table.

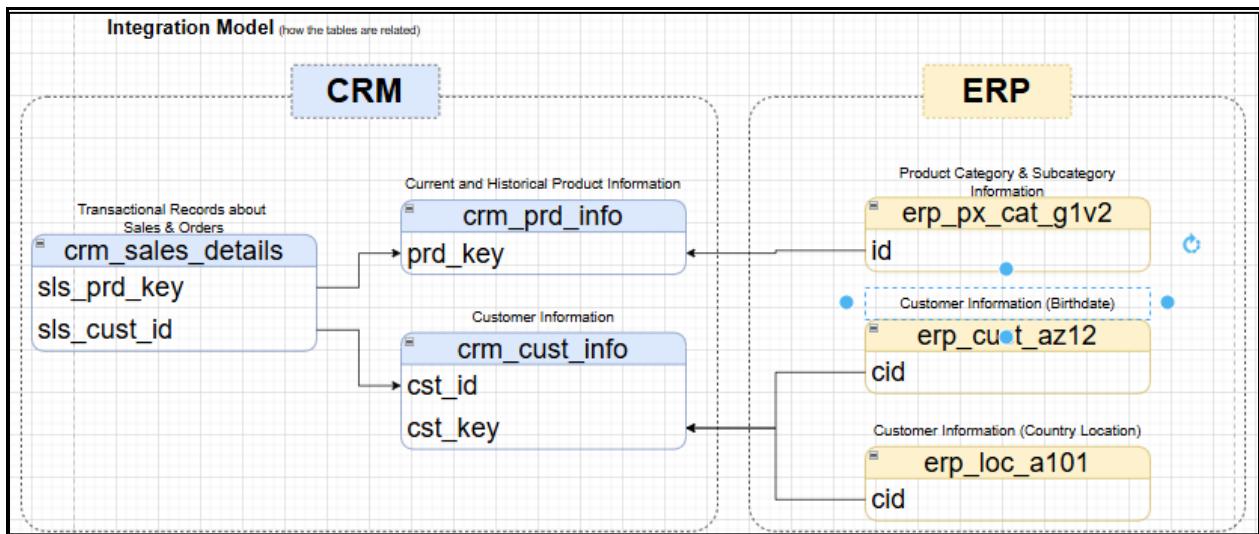
The data type and character count for records in the sls_cust_id field appear to be a match for the data type and character count for the cust_id field in the crm_cust_info table. Therefore I will use the sls_cust_id field to define the relationship with the crm_cust_info table matching to the cst_id field.

ERP Data Tables

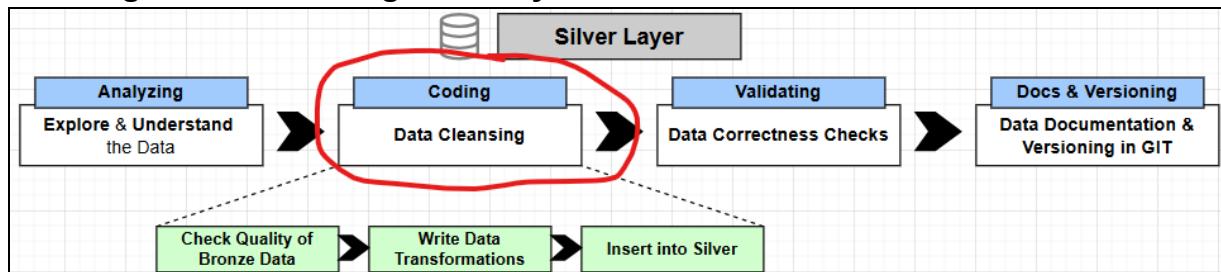
- **bronze.erp_cust_az12:** This table contains customer birth dates and gender. Its primary key appears to be the customer id field (cid). The values in this field for each record appear to be partial match to the customer key (cst_key) in the CRM table crm_cust_info, but it includes additional leading characters. No other field in this table could be used to create a relationship with any other table. So the customer id field in this table will be used to create a relationship with the crm_cust_info table by matching on its cst_key
- **bronze.erp_loc_a101:** This table contains country for each customer. Similar to the erp_cust_az12 table, its primary key appears to be the customer id field (cid) with partial match to the customer key (cst_key) in the CRM table crm_cust_info, but it includes “-” as separators for characters in this field. No other field in this table could be used to create a relationship with any other table. So the customer id (cid) field in this table will be used to create a relationship with the crm_cust_info table by matching on its cst_key
- **bronze.erp_px_cat_g1v2:** This table contains product categories, subcategories, and an identifier for “maintenance”. Not sure what the “maintenance” field is used for, but it only includes “Yes” or “No” as values for each record. The primary key appears to be the “id” field and is comprised of five (5) characters →Example: ‘AC_BR’. I assume this is the “id” for each product subcategory. The values in this field appear to be a match for the first five (5) characters in the prd_key field in the crm_prd_info table. This allows me to define a relationship between the two tables by matching on these fields. Doing so will require some transformation / manipulation first.

b. Document: High-Level Data Integration Diagram

As I explored the data in each table, I built the model below to help me visually understand the relationships that I will need to define.



c. Coding: Data Cleansing & Quality Checks



The first step in this task is to build the structure of the silver layer. It will be identical to the structure used for the Bronze layer. To accomplish this, I open the ddl script used for the bronze layer that I have saved in Git, paste it into a new query in SQL Server, then replace all references to 'bronze' with 'silver'.

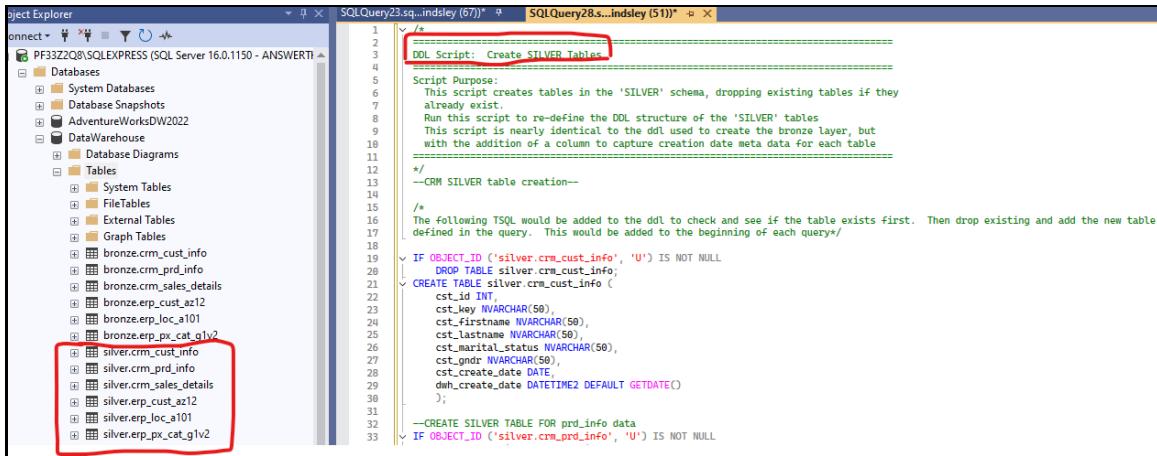
In addition to replicating the ddl to build the Silver layer, I also have to add metadata columns to each table to capture the following data. This information is needed to enable tracking and root cause analysis of corrupt data errors, data gaps in loads, etc:

- **create_date:** The record's load timestamp. Added "dwh_create_date DATETIME2 DEFAULT GETDATE()" in each CREATE TABLE command as an additional column for every table

These columns were not created in this step for each table. Will need to determine if these will be added later.

- **update_date:** The records last update timestamp
- **source_system:** The origin system of the record
- **file_location:** The file source of the record (where it is stored and made available for loading into the db)

Creation and execution of the ddl script to build the silver layer successfully complete resulted in the silver layer tables being added to the Data Warehouse as depicted below:



```

Object Explorer
PF3Z2Q\SQLEXPRESS (SQL Server 16.0.1150 - ANSWERTI)
Databases
Tables
System Tables
FileTables
External Tables
Graph Tables
bronze.crm_cust_info
bronze.crm_prd_info
bronze.crm_sales_details
bronze.emp_cust_az12
bronze.emp_loc_a101
bronze.emp_px_cat_g1v2
silver.crm_cust_info
silver.crm_prd_info
silver.crm_sales_details
silver.emp_cust_az12
silver.emp_loc_a101
silver.emp_px_cat_g1v2

SQLQuery23.sql...indsley (67)* SQLQuery28.s...indsley (51)*
=====
DDL Script: Create SILVER Tables
=====
Script Purpose:
This script creates tables in the 'SILVER' schema, dropping existing tables if they already exist.
Run this script to re-define the DDL structure of the 'SILVER' tables
This script is nearly identical to the ddl used to create the bronze layer, but with the addition of a column to capture creation date meta data for each table
=====
/*
--CRM SILVER table creation

The following TSQL would be added to the ddl to check and see if the table exists first. Then drop existing and add the new table defined in the query. This would be added to the beginning of each query*/
IF OBJECT_ID ('silver.crm_cust_info', 'U') IS NOT NULL
DROP TABLE silver.crm_cust_info;
CREATE TABLE silver.crm_cust_info (
    cst_id INT,
    cst_key NVARCHAR(50),
    cst_firstname NVARCHAR(50),
    cst_lastname NVARCHAR(50),
    cst_marital_status NVARCHAR(50),
    cst_gndr NVARCHAR(50),
    cst_create_date DATE,
    dim_create_date DATETIME2 DEFAULT GETDATE()
);

--CREATE SILVER TABLE FOR prd_info data
IF OBJECT_ID ('silver.crm_prd_info', 'U') IS NOT NULL

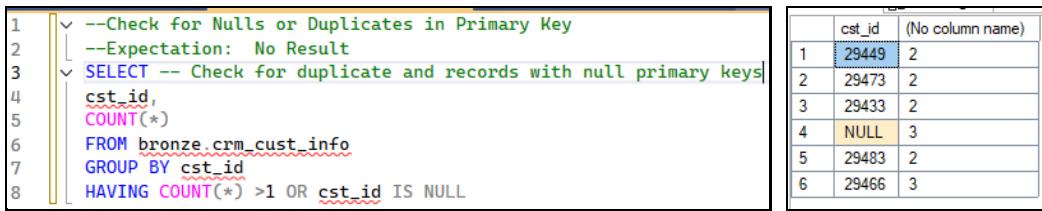
```

With the Silver layer tables built, the next step is to identify the data quality issues in each data set. This must be done before writing any transformation scripts. This exploration is done by evaluating each table in the bronze layer.

CRM → Customer Info Table

Checking for and Correcting Records with Duplicate and Null Primary Keys

Primary Key Duplicate and Null For each table, the primary key was evaluated to determine whether or not there were NULL values or duplicates using the following script. The result was the identification of 6 customer ids with two or more duplicates. Three (3) of those records had null values in the cst_id field (no customer id)



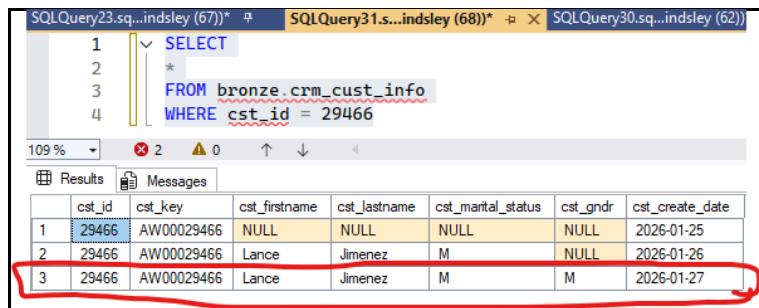
```

1 --Check for Nulls or Duplicates in Primary Key
2 ---Expectation: No Result
3 SELECT -- Check for duplicate and records with null primary keys
4 cst_id,
5 COUNT(*)
6 FROM bronze.crm_cust_info
7 GROUP BY cst_id
8 HAVING COUNT(*) >1 OR cst_id IS NULL

```

cst_id	(No column name)
29449	2
29473	2
29433	2
NULL	3
29483	2
29466	3

Next I needed to create a new query to clean the data to correct / account for the duplicates and null values. Before starting the transformation query, I needed to explore the data a bit more. I started by focusing on one of the IDs (29466) that appears more than once and running a query to view the associated records. This review allowed me to see that cst_id 29466 appears for three different records in the data set. Each record has a different creation date. The most recent creation date was selected as the primary record in the dataset.



```

SELECT *
FROM bronze.crm_cust_info
WHERE cst_id = 29466

```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1 29466	AW00029466	NULL	NULL	NULL	NULL	2026-01-25
2 29466	AW00029466	Lance	Jimenez	M	NULL	2026-01-26
3 29466	AW00029466	Lance	Jimenez	M	M	2026-01-27

I assumed that all other duplicates likely had different creation dates. So, to set the most recently added records for duplicate cst_ids, I needed to rank them according to their creation date and assign a ranking number. The following query was used to rank all records in order by their creation date with the assignment of “1” for the most recent. Records with the assignment of “1” will be used as the primary records from the data set.

```

1  SELECT
2  *
3  ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
4  FROM bronze.crm_cust_info
    
```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	flag_last
1	NULL	PO25	NULL	NULL	NULL	NULL	1
2	NULL	SF566	NULL	NULL	NULL	NULL	2
3	NULL	13451235	NULL	NULL	NULL	NULL	3
4	11000	AW00011000	Jon	Yang	M	2025-10-06	1
5	11001	AW00011001	Eugene	Huang	S	2025-10-06	1
6	11002	AW00011002	Ruben	Torres	M	2025-10-06	1
7	11003	AW00011003	Christy	Zhu	S	2025-10-06	1
8	11004	AW00011004	Elizabeth	Johnson	S	2025-10-06	1
9	11005	AW00011005	Julio	Ruiz	S	2025-10-06	1
10	11006	AW00011006	Janet	Alvarez	S	2025-10-06	1
11	11007	AW00011007	Marco	Mehta	M	2025-10-06	1
12	11008	AW00011008	Rob	Verhoff	S	2025-10-06	1
13	11009	AW00011009	Shannon	Carlson	S	2025-10-06	1
14	11010	AW00011010	Jacquelyn	Suarez	C	2025-10-06	1

Creating and running the query below allowed me to see all records that are duplicates (or nulls) with ranking identifiers not equal to “1”. These will not be used in my data set for this table.

```

1  SELECT
2  *
3  FROM (
4  SELECT
5  *
6  ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
7  FROM bronze.crm_cust_info
8  )t WHERE flag_last != 1
    
```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	flag_last
1	NULL	SF566	NULL	NULL	NULL	NULL	2
2	NULL	13451235	NULL	NULL	NULL	NULL	3
3	29433	AW00029433	NULL	NULL	M	2026-01-25	2
4	29449	AW00029449	NULL	Chen	S	2026-01-25	2
5	29465	AW00029465	Lance	Jimenez	M	2026-01-26	2
6	29466	AW00029466	NULL	NULL	NULL	2026-01-25	3
7	29473	AW00029473	Carmen	NULL	NULL	2026-01-25	2
8	29483	AW00029483	NULL	Navarro	NULL	2026-01-25	2

Creating and running the query below allowed me to see all records with ranking identifiers equal to “1”. These records will be used in my data set for this table.

```

1  /*Order all records and their cst_ids based on their creation date starting with
the most recent first and assign an order number to each.
Example: Every customer record with a cst_id that appears once, will be assigned a "1", while
every customer record with a cst_id that appears more than once will be ordered by the most recent
creation date and assigned a rank of 1, 2, 3, etc for each occurrence of that cst_id
*/
2
3  SELECT
4  *
5  FROM (
6  SELECT
7  *
8  ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
9  FROM bronze.crm_cust_info
10 )t WHERE flag_last = 1
    
```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	flag_last
1	NULL	PO25	NULL	NULL	NULL	NULL	1
2	11000	AW00011000	Jon	Yang	M	2025-10-06	1
3	11001	AW00011001	Eugene	Huang	S	2025-10-06	1
4	11002	AW00011002	Ruben	Torres	M	2025-10-06	1
5	11003	AW00011003	Christy	Zhu	S	2025-10-06	1
6	11004	AW00011004	Elizabeth	Johnson	S	2025-10-06	1
7	11005	AW00011005	Julio	Ruiz	S	2025-10-06	1
8	11006	AW00011006	Janet	Alvarez	S	2025-10-06	1
9	11007	AW00011007	Marco	Mehta	M	2025-10-06	1
10	11008	AW00011008	Rob	Verhoff	S	2025-10-06	1
11	11009	AW00011009	Shannon	Carlson	S	2025-10-06	1
12	11010	AW00011010	Jacquelyn	Suarez	C	2025-10-06	1
13	11011	AW00011011	Curtis	Lu	M	2025-10-06	1
14	11012	AW00011012	Lauren	Walker	M	2025-10-06	1

Checking for Unwanted Spaces in String Values

Each field with strings as data types (NVARCHAR) needs to be checked for unwanted spaces and have unwanted spaces removed (leading or trailing spaces). The following columns were identified as being strings and needed to be checked and corrected if applicable.

```
cst_key NVARCHAR(50),
cst_firstname NVARCHAR(50),
cst_lastname NVARCHAR(50),
cst_marital_status NVARCHAR(50),
cst_gndr NVARCHAR(50),
```

The following query was written and executed on the cst_firstname field to test the identification of records with unwanted spaces as depicted below. The same was done for each string value field. Only the 'cst_firstname' and 'cst_lastname' fields had unwanted spaces that required correction.

A screenshot of the SQL Server Management Studio interface. The top pane shows a query window with the following code:

```
1 /* Check string fields for unwanted spaces (leading or trailing)
2 */
3 /*
4  SELECT cst_firstname
5   FROM bronze.crm_cust_info
6  WHERE cst_firstname != TRIM(cst_firstname)
7  --identifies records where the original field and trimmed field are not equal. Those are the records
8  --with unwanted spaces.
```

The bottom pane shows the results of the query, which lists 15 rows of data from the 'cst_firstname' column. A red bracket is drawn under the first two rows, 'Jon' and 'Elizabeth', indicating they are the records identified as having unwanted spaces.

cst_firstname
1 Jon
2 Elizabeth
3 Lauren
4 Ian
5 Chloe
6 Destiny
7 Angela
8 Caleb
9 Wille
10 Ruben
11 Javier
12 Nicole
13 Maria
14 Allison
15 Adrián

The following query was written and executed to produce a table that included all fields from cust_info table, but corrects the wanted spaces in the cst_firstname and cst_lastname fields, and includes only unique records records (non-duplicates) with a ranking of "1" based on their creation date. The ranking identifier is not included in the output.

A screenshot of the SQL Server Management Studio interface. The top pane shows a query window with the following code:

```
1 /* Produce a table from cust_info that corrects unwanted spaces in
2  cst_firstname and cst_lastname, and excludes duplicate records while
3  keeping only the most recently added record for any duplicates
4 */
5 /*
6  SELECT
7   cst_id,
8   cst_key,
9   TRIM(cst_firstname) AS cst_firstname,
10  TRIM(cst_lastname) AS cst_lastname,
11  cst_marital_status,
12  cst_gndr,
13  cst_create_date
14
15  FROM (
16   SELECT
17   *,
18   ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
19   FROM bronze.crm_cust_info
20   WHERE cst_id IS NOT NULL
21 )t WHERE flag_last = 1
```

The bottom pane shows the results of the query, which is a table with 13 rows of corrected data. The columns are cst_id, cst_key, cst_firstname, cst_lastname, cst_marital_status, cst_gndr, and cst_create_date.

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1	11000	AW00011000	Jon	Yang	M	2025-10-05
2	11001	AW00011001	Eugene	Huang	S	2025-10-06
3	11002	AW00011002	Ruben	Torres	M	2025-10-06
4	11003	AW00011003	Christy	Zhu	S	2025-10-06
5	11004	AW00011004	Elizabeth	Johnson	S	2025-10-06
6	11005	AW00011005	Julio	Ruiz	S	2025-10-06
7	11006	AW00011006	Janet	Alvarez	S	2025-10-06
8	11007	AW00011007	Marco	Mehta	M	2025-10-06
9	11008	AW00011008	Rob	Verhoff	S	2025-10-06
10	11009	AW00011009	Shannon	Carlson	S	2025-10-05
11	11010	AW00011010	Jacquelyn	Suarez	S	2025-10-06
12	11011	AW00011011	Curtis	Lu	M	2025-10-06
13	11012	AW00011012	Lauren	Walker	F	2025-10-06

Checking for consistency of values in low cardinality columns (columns used for categorical data)

The columns capturing customer marital status and gender have low quantity of possible variations. The data in these columns needs to be assessed for consistency. Example: The cst_gndr column should only have two possible values → “M” for male and “F” for female. The values in this column should be “M” or “F” only (or maybe NULL). It should be list “Male”, “Female”, “Q”, “Duck”, or anything else. So I run the following query to identify inconsistencies.

The following query was run and produced results for three (3) possible values: M, F, and NULL as depicted below.

cst_gndr
NULL
F
M

Now...to make this data business / reader friendly, we will adjust our primary transformation query for this table so that M is replaced with “Male”, “F” is replaced with “Female” and “NULL” is replaced with “n/a”. The query will also be written so that any unwanted spaces are trimmed and we account for any possible lower case occurrences of “M” or “F”.

The same approach was applied to the marital status field (cst_marital_status) to replace “S”, “M”, and “NULL” with “Single”, “Married”, and “n/a” respectively.

```

4
5
6 /*/
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
    */
SELECT
    cst_id,
    cst_key,
    TRIM(cst_firstname) AS cst_firstname,
    TRIM(cst_lastname) AS cst_lastname,
    CASE WHEN UPPER(TRIM(cst_marital_status)) = 'S' THEN 'Single'
        WHEN UPPER(TRIM(cst_marital_status)) = 'M' THEN 'Married'
        ELSE 'n/a'
    END cst_marital_status,
    CASE WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
        WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
        ELSE 'n/a'
    END cst_gndr,
    cst_create_date
FROM (
    SELECT
        *,
        ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
    FROM bronze.crm_cust_info
    WHERE cst_id IS NOT NULL
) t WHERE flag_last = 1

```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	
1	11000	AW00011000	Jon	Yang	Mamed	Male	2025-10-06
2	11001	AW00011001	Eugene	Huang	Single	Male	2025-10-06
3	11002	AW00011002	Ruben	Tones	Mamed	Male	2025-10-06
4	11003	AW00011003	Christy	Zhu	Single	Female	2025-10-06
5	11004	AW00011004	Elizabeth	Johnson	Single	Female	2025-10-06
6	11005	AW00011005	Julio	Ruiz	Single	Male	2025-10-06
7	11006	AW00011006	Janet	Alvarez	Single	Female	2025-10-06
8	11007	AW00011007	Marco	Mehta	Mamed	Male	2025-10-06
9	11008	AW00011008	Rob	Verhoff	Single	Female	2025-10-06
10	11009	AW00011009	Shannon	Cadeen	Single	Male	2025-10-06

Inserting Customer Info data into Silver layer table

At this point, I am now ready to insert the cleansed data into the Silver layer table as the data quality checks and cleansing complete for the table `crm_cust_info`. An ‘`INSERT INTO`’ command was added to the data quality script so it would execute the load after the data cleansing script is complete.

```

SQLQuery28.sql...indsley (51)*
6   v [INSERT INTO silver.crm_cust_info (
7     cst_id,
8     cst_key,
9     cst_firstname,
10    cst_lastname,
11    cst_marital_status,
12    cst_gndr,
13    cst_create_date)
14
15   SELECT
16     cst_id,
17     cst_key,
18     TRIM(cst_firstname) AS cst_firstname,
19     TRIM(cst_lastname) AS cst_lastname,
20     CASE WHEN UPPER(TRIM(cst_marital_status)) = 'S' THEN 'Single'
21     WHEN UPPER(TRIM(cst_marital_status)) = 'M' THEN 'Married'
22     ELSE 'n/a'
23   END cst_marital_status,
24   CASE WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
25     WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
26     ELSE 'n/a'
27   END cst_gndr,
28   cst_create_date
29
30   FROM
31   *
32   ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_last
33   FROM bronze.crm_cust_info
34   WHERE cst_id IS NOT NULL
35   )t WHERE flag_last = 1
  
```

Messages

(18484 rows affected)

Compilation time: 2023-11-26T12:48:19.6286900-03:00

I now check that the data quality corrections and standardizations are correct by running a query for each correction / standardizationWith the load complete. The screenshot below depicts the results of the query to confirm values listed for `cst_gndr` are correctly converted from M, F, and NULL to “Male”, “Female”, and “n/a”....and that no other value types exist

```

SQLQuery28.sql...indsley (51)*
1  --Check for Nulls or Duplicates in Primary Key
2  --Expectation: No Result
3  v [SELECT -- Check for duplicate and records with null primary keys
4    cst_id,
5    COUNT(*)
6    FROM silver.crm_cust_info
7    GROUP BY cst_id
8    HAVING COUNT(*) >1 OR cst_id IS NULL
9
10 --Check for cst_key Unwanted Spaces
11 --Expectation: No Result
12 v [SELECT -- Check for duplicate and records with null primary keys
13   cst_key
14   FROM silver.crm_cust_info
15   WHERE cst_key != TRIM(cst_key)
16
17 --
18 v [SELECT
19   cst_firstname
20   FROM silver.crm_cust_info
21   WHERE cst_firstname != TRIM(cst_firstname)
22
23 --
24 v [SELECT
25   cst_lastname
26   FROM silver.crm_cust_info
27   WHERE cst_lastname != TRIM(cst_lastname)
28
29 --Check for data standardization
30 --Expectation: No Result
31 v [SELECT DISTINCT
32   cst_marital_status
33   FROM silver.crm_cust_info
34
35 v [SELECT DISTINCT
36   cst_gndr
37   FROM silver.crm_cust_info
  
```

Results

cst_gndr
n/a
Male
Female

CRM → Product Info Table

Checking for and Correcting Records with Duplicate and Null Primary Keys

Primary key was evaluated to determine whether or not there were NULL values or duplicates using the following script. This resulted in zero (0) duplicates or null values for the prd_id field

```

13  --> SELECT -- Check for duplicate and records with 'null primary keys'
14      prd_id,
15      COUNT(*)
16  FROM bronze.crm_prd_info
17  GROUP BY prd_id
18  HAVING COUNT(*) >1 OR prd_id IS NULL

```

Deriving New Columns to Enable Mapping

During the data exploration, I found that the prd_key field is a concatenation of the category id 'id' field from the 'erp_px_cat_g1v2' table and additional characters native to this table. The first five characters of each value in this field are the category id and the remaining are just....a bunch of other characters. So, this column needs to separate the first five characters into a new column so the 'erp_px_cat_g1v2' can be mapped to it.

prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
210	CO-RF-FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01 00:00:00.000	NULL

CAT ID

Additionally, I have to correct for the difference between the characters as the "id" in the 'erp_px_cat_g1v2' are separated by a "_", whereas the characters in the prd_key from the prd_info table are separated by a "-".

I also need to create a separate column for the prd_key without the category id characters. So...some standardization is needed here. This is accomplished by the script (with results below) in the following image:

```

2  --> SELECT
3      prd_id,
4      prd_key,
5      PARSENAME(SUBSTRING(prd_key,1,5), '-1', '_') AS cat_id, --Extracts the category id, replaces the "-" with "_",
6      --creates new column so that this table can be mapped to the category id ("id") field in the category table
7      --(erp_px_cat_g1v2)
8      SUBSTRING(prd_key,7,LEN(prd_key)) AS prd_key, -- Removes the first five characters that are the category id from 'erp_px_cat_g1v2'
9      prd_nm,
10     prd_cost,
11     prd_line,
12     prd_start_dt,
13     prd_end_dt
14  FROM bronze.crm_prd_info
15
16  --Check bronze.crm_prd_info for existence of duplicate prd_id

```

prd_id	prd_key	cat_id	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt		
1	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Read Frame - Black	58	NULL	R	2003-07-01 00:00:00.000	NULL
2	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Read Frame - Red	58	NULL	R	2003-07-01 00:00:00.000	NULL
3	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet Red	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	
4	AC-HE-HL-U509-B	AC_HE	HL-U509-B	Sport-100 Helmet Black	14	S	2011-07-01 00:00:00.000	2007-12-27 00:00:00.000	
5	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet Red	13	S	2011-07-01 00:00:00.000	NULL	
6	AC-HE-HL-U509-B	AC_HE	HL-U509-B	Sport-100 Helmet Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	
7	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet Black	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	
8	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet Black	13	S	2013-07-01 00:00:00.000	NULL	
9	CL-SO-SO-8909-M	CL_SO	SO-8909-M	Mountain Bike Socks- M	3	M	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	
10	CL-SO-SO-8909-L	CL_SO	SO-8909-L	Mountain Bike Socks- L	3	M	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	
11	AC-HE-HL-U509-B	AC_HE	HL-U509-B	Sport-100 Helmet Blue	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	
12	AC-HE-HL-U509-B	AC_HE	HL-U509-B	Sport-100 Helmet Blue	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	
13	AC-HE-HL-U509-B	AC_HE	HL-U509-B	Sport-100 Helmet Blue	13	S	2013-07-01 00:00:00.000	NULL	

Checking for unwanted spaces

I checked the prd_nm field for unwanted spaces and found none using the following query

```
SELECT  
prd_nm  
FROM bronze.crm_prd_info  
WHERE prd_nm != TRIM(prd_nm)
```

Checking for product costs <0 and Null values

I checked the prd_cost field to identify any products with negative values or nulls using the following query. No products had negative costs, but there are two products with null values for cost. In a business context, the business stakeholder would need to determine whether to retain the null or replace it with a "0" or some other value.

```
--Check for NULLS or negative numbers  
SELECT  
prd_cost  
FROM bronze.crm_prd_info  
WHERE prd_cost < 0 OR prd_cost IS NULL
```

prd_cost
1 NULL
2 NULL

As this is a fictional situation without actual stakeholders, I chose to replace the NULL values with a "0". I did this by adding the following to the transformation script for the prd_info table:

```
ISNULL(prd_cost,0) AS prd_cost, --Replaces null values in cost with 0
```

Checking 'prd_line' field for consistency of values (similar to the check conducted for marital status and gender in the 'crm_cust_info' table)

The 'prd_line' column has a low quantity of possible variations. The data in this column needs to be assessed for consistency. I ran 'SELECT DISTINCT' query on the prd_info table and found the following possible values to be: "NULL", "M", "R", "S", and "T".

```
--Data Standardization & Consistency  
SELECT DISTINCT prd_line  
FROM bronze.crm_prd_info
```

prd_line
1 NULL
2 M
3 R
4 S
5 T

In a real-world situation, I would ask the data owner / business stakeholder what those values represent so that I could convert them to a user / business friendly value. There are no stakeholders in this case as this is a fictional project. So...I added a CASE WHEN with UPPER(TRIM) to the transformation script to replace the values with the following per the instructor for this course:

```
CASE WHEN UPPER(TRIM(prd_line)) = 'M' THEN 'Mountain'
WHEN UPPER(TRIM(prd_line)) = 'R' THEN 'Road'
WHEN UPPER(TRIM(prd_line)) = 'S' THEN 'Other Sales'
WHEN UPPER(TRIM(prd_line)) = 'T' THEN 'Touring'
ELSE 'n/a'
END AS prd_line,
```

Checking the quality of Product Start and End Date fields and correcting quality issues.

The first part of this effort is to identify records that have a product end date that is earlier than the product start date. To do this, I ran the following query and found 201 records in which this occurs.

```
SELECT * FROM bronze.crm_prd_info
WHERE prd_end_dt < prd_start_dt
```

I assumed that the start and end dates for these records were simply reversed and in the incorrect field. So, the start date is actually the end date and the end date is actually the start date. Assuming this is the case, I also found that several records had product end dates that overlap with the start date of the product with the same product key but next prd_id sequence. So....I had some corrections to make.

In a real-world scenario, these observations would need to be reviewed and verified with the data owner before making any decisions on transformations / clean-up. In this case, I pretend that I reviewed the issues with a data owner / business stakeholder and had them confirm the dates were reversed, and that the end dates should not overlap but end the day prior to the start date of the next product id having the same value for 'prd_key'

The below image summarizes observations against a sample set of records with approach for correction:

INITIAL REVIEW OF THE FIELDS									OBSERVATION / NOTE
prd_id	prd_key	cat_id	prd_key	prd_name	prd_cost	prd_line	prd_start_dt	prd_end_dt	
212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	Other Sales	7/1/2011 0:00	12/28/2007 0:00	Prod end dates is earlier than start. Date values are in reverse column order.
213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	Other Sales	7/1/2012 0:00	12/27/2008 0:00	
214	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	13	Other Sales	7/1/2013 0:00	NULL	

PASS AT TRANSPOSING DATES - OVERLAP IDENTIFIED									OBSERVATION / NOTE
prd_id	prd_key	cat_id	prd_key	prd_name	prd_cost	prd_line	prd_start_dt	prd_end_dt	
212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	Other Sales	12/28/2007 0:00	7/1/2011 0:00	Transposed start and end dates so start date is earlier than end date. This reveals an overlap between the end date for prd_id 212 and the start date for prd_id 213 (of the same prod key). Need to set end date to be day prior to start date of the next product id
213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	Other Sales	12/27/2008 0:00	7/1/2012 0:00	
214	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	13	Other Sales	7/1/2013 0:00	NULL	

CORRECTING OVERLAP OF END DATE AND START DATES									OBSERVATION / NOTE
prd_id	prd_key	cat_id	prd_key	prd_name	prd_cost	prd_line	prd_start_dt	prd_end_dt	
212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	Other Sales	7/1/2011 0:00	6/30/2012 0:00	Transposed start and end dates so start date is earlier than end date. Corrected overlap between the end date for prd_id 212 and the start date for prd_id 213 (of the same prod key) by setting end date = start date of next record - 1
213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	Other Sales	7/1/2012 0:00	6/30/2013 0:00	
214	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	13	Other Sales	7/1/2013 0:00	NULL	

The following commands were added to the script to correct the start and end dates, and to remove the time values from the date field as they were "00:00:00". Below is a screenshot of the full script and results.

```
CAST(prd_start_dt AS DATE) as prd_start_dt,--"CAST" removes time data
CAST(LEAD(prd_start_dt) OVER (PARTITION BY prd_key ORDER BY prd_start_dt)-1 AS DATE)
AS prd_end_dt
```

prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	
7	405	AC-FE-FE-6654	AC_FE	FE-6654	Fender Set - Mountain	8	Mountain	2013-07-01	NULL
8	218	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet: Black	12	Other Sales	2011-07-01	2012-06-30
9	216	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet: Black	14	Other Sales	2012-07-01	2013-06-30
10	217	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet: Black	13	Other Sales	2013-07-01	NULL
11	220	AC-HE-HL-U509-B	AC_HE	HL-U509	Sport-100 Helmet: Blue	12	Other Sales	2011-07-01	2012-06-30
12	221	AC-HE-HL-U509-B	AC_HE	HL-U509	Sport-100 Helmet: Blue	14	Other Sales	2012-07-01	2013-06-30
13	222	AC-HE-HL-U509-R	AC_HE	HL-U509	Sport-100 Helmet: Blue	13	Other Sales	2013-07-01	NULL
14	212	AC-HE-HL-U509-R	AC_HE	HL-U509	Sport-100 Helmet: Red	12	Other Sales	2011-07-01	2012-06-30
15	213	AC-HE-HL-U509-R	AC_HE	HL-U509	Sport-100 Helmet: Red	14	Other Sales	2012-07-01	2013-06-30
16	214	AC-HE-HL-U509-R	AC_HE	HL-U509	Sport-100 Helmet: Red	13	Other Sales	2013-07-01	NULL
17	487	AC-HP-HY-1023-70	AC_HP	HY-102...	Hydration Pack - 70 oz.	21	Other Sales	2013-07-01	NULL
18	451	AC-ULT-H902	AC_UL	LT-H902	Headlights - Dual-Beam	14	Road	2012-07-01	NULL
19	452	AC-ULT-H903	AC_UL	LT-H903	Headlights - Weatherp...	19	Road	2012-07-01	NULL
20	453	AC-ULT-H904	AC_UL	LT-H904	Headlights - Weatherp...	17	Road	2012-07-01	NULL

I needed to update the ddl for the prd_info table in the Silver layer to account for the removal of time values from the start and end date columns in the transformation script. I also added the cat_id field to the ddl as that is being created in the transformation script. Below is the updated ddl to account for this.

```
--CREATE SILVER TABLE FOR prd_info data
IF OBJECT_ID ('silver.crm_prd_info', 'U') IS NOT NULL
DROP TABLE silver.crm_prd_info;
CREATE TABLE silver.crm_prd_info (
    prd_id INT,
    cat_id NVARCHAR(50), ←
    prd_key NVARCHAR(50),
    prd_nm NVARCHAR(50),
    prd_cost INT,
    prd_line NVARCHAR(50),
    prd_start_dt DATE, ←
    prd_end_dt DATE, ←
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
```

Inserting and validating the transformation query results into the Silver layer prd_info table

It is now time to insert the transformed data now that the data quality corrections are complete for the prd_info table and the ddl has been updated.

To insert the corrected data, I wrote and executed an INSERT INTO script ahead of the transformation query for the prd_info table and executed checks against the output. Below is the INSERT INTO with the transformation script:

```

4   v INSERT INTO silver.crm_prd_info (
5     prd_id,
6     cat_id,
7     prd_key,
8     prd_nm,
9     prd_cost,
10    prd_line,
11    prd_start_dt,
12    prd_end_dt
13  )
14
15  SELECT
16    prd_id,
17    REPLACE(SUBSTRING(prd_key, 1, 5), '-' , '_') AS cat_id, --Extracts the category id, replaces the "-" with "_",
18    --creates new column so that this table can be mapped to the category id ("id") field in the category table
19    --(erp_px_cat_glv2)
20    SUBSTRING(prd_key, 7,LEN(prd_key)) AS prd_key, -- Removes the first five characters that are the category id from 'erp_px_cat_glv2'
21    --and creates new column of just the product key characters so we can map to the crm_sales_details table
22    prd_nm,
23    ISNULL(prd_cost,0) AS prd_cost, --Replaces null values in cost with 0
24    CASE UPPER(TRIM(prd_line)) --Replaces single letter values with business-friendly names
25    and removes unwanted spaces /*
26      WHEN 'M' THEN 'Mountain'
27      WHEN 'R' THEN 'Road'
28      WHEN 'O' THEN 'Other Sales'
29      WHEN 'T' THEN 'Touring'
30      ELSE 'n/a'
31    END AS prd_line,
32    CAST(prd_start_dt AS DATE) as prd_start_dt,--"CAST" removes time data
33    CAST(LEAD(prd_start_dt) OVER (PARTITION BY prd_key ORDER BY prd_start_dt)-1 AS DATE) AS prd_end_dt /* create new column
34    in which the end date is equal to original start date -1 to correct for the transposed data in these columns.*/
35  FROM bronze.crm_prd_info

```

The next step was to run checks against the transformed data and derived fields in the prd_info table in the silver table. Each of the following scripts were run and executed. All were successful.

```

50  --Check silver.crm_prd_info for existence of duplicate prd_id
51  v SELECT -- Check for duplicate and records with null primary keys
52    prd_id,
53    COUNT(*)
54  FROM silver.crm_prd_info
55  GROUP BY prd_id
56  HAVING COUNT(*) > 1 OR prd_id IS NULL
57
58  --Check for cst_key Unwanted Spaces
59  --Expectation: No Result
60  v SELECT
61    prd_nm
62    FROM silver.crm_prd_info
63    WHERE prd_nm != TRIM(prd_nm)
64
65  --Check for NULLS or negative numbers
66  --Expectation: No Result
67  v SELECT
68    prd_cost
69    FROM silver.crm_prd_info
70    WHERE prd_cost < 0 OR prd_cost IS NULL
71
72  --Data Standardization & Consistency
73  v SELECT DISTINCT prd_line
74    FROM silver.crm_prd_info
75
76  --Checking for invalid product start and end dates
77  v SELECT
78    *
79    FROM silver.crm_prd_info
80    WHERE prd_end_dt < prd_start_dt --Checks for records with end dates that are earlier than the prd start date
81
82  v SELECT
83    *
84    FROM silver.crm_prd_info --View full data set in the silver table to get a warm fuzzy|

```

Results

prd_id	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	dwh_create_date	
1	478	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01	NULL	2025-11-27 09:56:50.0500000
2	479	AC_BC	BC-R205	Road Bottle Cage	3	Road	2013-07-01	NULL	2025-11-27 09:56:50.0500000
3	477	AC_BC	WB-H098	Water Bottle - 30 oz.	2	Other Sales	2013-07-01	NULL	2025-11-27 09:56:50.0500000
4	483	AC_BR	RA-H123	Hitch Rack - 4-Bike	45	Other Sales	2013-07-01	NULL	2025-11-27 09:56:50.0500000
5	486	AC_BS	ST-1401	All-Purpose Bike Stand	59	Mountain	2013-07-01	NULL	2025-11-27 09:56:50.0500000
6	484	AC_CL	CL-9009	Bike Wash - Dissolver	3	Other Sales	2013-07-01	NULL	2025-11-27 09:56:50.0500000

DISCLAIMER: Now....I completely understand that this is best case scenario as I was following step-by-step with the instructor. I likely would have dorked this up completely and not known how to correct any issues if running this on my own. It would have been a disaster in all likelihood. But I'm learning....baby steps. This is fun.....so far.

CRM → Sales Details Table

Checking primary key (sls_ord_num) for unwanted spaces:

Executed the following query with no results for records with unwanted spaces:

```

Cust_Info_Fla...lindsay (68) | cust_info_tran...lindsay (60))
1  SELECT
2   sls_ord_num,
3   sls_prd_key,
4   sls_cust_id,
5   sls_order_dt,
6   sls_ship_dt,
7   sls_due_dt,
8   sls_sales,
9   sls_quantity,
10  sls_price
11  FROM bronze.crm_sales_details
12  WHERE sls_ord_num != TRIM(sls_ord_num)

```

Results

sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_quantity	sls_price

Checking the sls_prd_key to determine whether or not there are matches in the prd_key of the the crm_prd_info table: This is necessary as I determined earlier in the data diagram that I would need to using a portion of the values in this column to match to the prd_key field in the crm_prd_info table. The following was written and executed...resulting in zero (0) matches.

```

Cust_Info_Fla...lindsay (68) | cust_info_tran...lindsay (60))
1  SELECT
2   sls_ord_num,
3   sls_prd_key,
4   sls_cust_id,
5   sls_order_dt,
6   sls_ship_dt,
7   sls_due_dt,
8   sls_sales,
9   sls_quantity,
10  sls_price
11  FROM bronze.crm_sales_details
12  WHERE sls_prd_key NOT IN (SELECT prd_key FROM silver.crm_prd_info)

```

Results

sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_quantity	sls_price

Checking the sls_cust_id to determine whether or not there are matches in the cst_id field of the crm_cust_info table: This is necessary as I determined earlier in the data diagram that I would need to using a portion of the values in this column to match to the cst_id field in the crm_cust_info table. The following was written and executed...resulting in zero (0) matches.

```

Cust_Info_Fla...lindsay (68) | cust_info_tran...lindsay (60))
1  SELECT
2   sls_ord_num,
3   sls_prd_key,
4   sls_cust_id,
5   sls_order_dt,
6   sls_ship_dt,
7   sls_due_dt,
8   sls_sales,
9   sls_quantity,
10  sls_price
11  FROM bronze.crm_sales_details
12  WHERE sls_cust_id NOT IN (SELECT cst_id FROM silver.crm_cust_info)

```

Results

sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_quantity	sls_price

Check values in the following fields to identify non-date values: sls_order_dt; sls_ship_dt; and sls_due_dt. I ran the following query and it returned 17 records with a “0” in the sls_order_dt field

```
SELECT  
    sls_order_dt  
FROM bronze.crm_sales_details  
WHERE sls_order_dt <=0
```

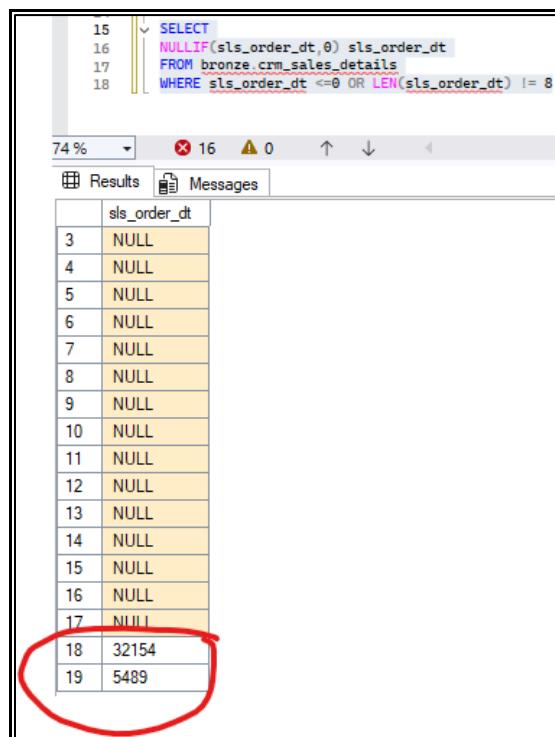
I needed to convert the “0” for these records to NULL. The following was written to correct for this:

```
SELECT  
    NULLIF(sls_order_dt,0) sls_order_dt  
FROM bronze.crm_sales_details  
WHERE sls_order_dt <=0
```

A further check of the sls_order_dt field was done to identify:

- Any additional non-date numerical values that can not be converted to a date
- Any dates beyond 20500101 (fictional upper boundary chosen for this exercise)
- Any dates earlier than 19990101 (fictional lower boundary chosen for this exercise).

This query was run and returned all NULLs as well as two records with non-date numerical values.



	sls_order_dt
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL
9	NULL
10	NULL
11	NULL
12	NULL
13	NULL
14	NULL
15	NULL
16	NULL
17	NULL
18	32154
19	5489

Converting the values in the following fields from INT to dates: sls_order_dt; sls_ship_dt; and sls_due_dt. This is necessary because the values should be dates but are written as "20101229" for example. They should be in this format "2010-12-29"

I added the following to the transformation query to convert non-date numerical values to NULL and set all others to dates:

```

Cust_Info_Fla...lindsay (68) | cust_info_tran...lindsay (60)

1 | SELECT
2 |     sls_ord_num,
3 |     sls_prd_key,
4 |     sls_cust_id,
5 |     CASE WHEN sls_order_dt = 0 OR LEN(sls_order_dt) != 8 THEN NULL
6 |     ELSE CAST(CAST(sls_order_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
7 |     --casting the VARCHAR to a date
8 |     END AS sls_order_dt,
9 |     sls_ship_dt,
10 |     sls_due_dt,
11 |     sls_sales,
12 |     sls_quantity,
13 |     sls_price
14 |     FROM bronze.crm_sales_details
15 |
16 | 74 % 18 ▲ 0 ↑ ↓
17 | Results Messages
18 |
```

	sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_quantity	sls_price
4...	SO69210	HL-U509	12426	2013-10-26	20131102	20131107	35	1	35
4...	SO69211	TI-M823	12432	2013-10-26	20131102	20131107	35	1	35
4...	SO69211	FE-6654	12432	2013-10-26	20131102	20131107	22	1	22
4...	SO69211	HL-U509-R	12432	2013-10-26	20131102	20131107	35	1	35
4...	SO69212	WB-H098	11495	2013-10-26	20131102	20131107	5	1	5
4...	SO69212	BC-R205	11495	2013-10-26	20131102	20131107	9	1	9
4...	SO69213	FE-6654	13368	2013-10-26	20131102	20131107	22	1	22
4...	SO69213	LJ-0192-S	13368	2013-10-26	20131102	20131107	50	1	50
4...	SO69214	BK-M68B-38	18595	2013-10-26	20131102	20131107	2295	1	2295
4...	SO69214	TI-M823	18595	2013-10-26	20131102	20131107	35	1	35
4...	SO69215	BK-M68B-46	16864	2013-10-26	20131102	20131107	2295	1	2295
4...	SO69215	TI-M823	16864	NULL	20131102	20131107	-35	1	35
4...	SO69215	TT-M928	16864	NULL	20131102	20131107	5	1	5
4...	SO69215	HL-U509	16864	2013-10-26	20131102	20131107	35	1	35

The same series of checks and corrections done on the sls_order_dt field were conducted for the remaining date fields and added to the transformation query. No quality issues were identified, but I added the same CASE WHEN transformation script to each date field to ensure similar potential quality issues that might arise in future data sets are automatically corrected.

```

Cust_Info_Fla...lindsay (68) | cust_info_tran...lindsay (60)

1 | SELECT
2 |     sls_ord_num,
3 |     sls_prd_key,
4 |     sls_cust_id,
5 |     sls_order_dt,
6 |     CASE WHEN sls_order_dt = 0 OR LEN(sls_order_dt) != 8 THEN NULL
7 |     ELSE CAST(CAST(sls_order_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
8 |     --casting the VARCHAR to a date
9 |     END AS sls_order_dt,
10 |     sls_ship_dt,
11 |     CASE WHEN sls_ship_dt = 0 OR LEN(sls_ship_dt) != 8 THEN NULL
12 |     ELSE CAST(CAST(sls_ship_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
13 |     --casting the VARCHAR to a date
14 |     END AS sls_ship_dt,
15 |     sls_due_dt,
16 |     CASE WHEN sls_due_dt = 0 OR LEN(sls_due_dt) != 8 THEN NULL
17 |     ELSE CAST(CAST(sls_due_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
18 |     --casting the VARCHAR to a date
19 |     END AS sls_due_dt,
20 |     sls_sales,
21 |     sls_quantity,
22 |     sls_price
23 |     FROM bronze.crm_sales_details
24 |
25 | 74 % 23 ▲ 0 ↑ ↓
26 | Results Messages
27 |
```

	sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_quantity	sls_price
1	SO43697	BK-R93R-62	21768	2010-12-29	2011-01-05	2011-01-10	3578	1	3578
2	SO43698	BK-M825-44	28389	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
3	SO43699	BK-M825-44	25863	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
4	SO43700	BK-R50B-62	14501	2010-12-29	2011-01-05	2011-01-11	699	1	699
5	SO43701	BK-M825-44	11003	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
6	SO43702	BK-R93R-44	27645	2010-12-30	2011-01-06	2011-01-11	3578	1	3578
7	SO43703	BK-R93R-62	16624	2010-12-30	2011-01-06	2011-01-11	3578	1	3578
8	SO43704	BK-M82B-48	11005	2010-12-30	2011-01-06	2011-01-11	3375	1	3375
9	SO43705	BK-M825-38	11011	2010-12-30	2011-01-06	2011-01-11	3400	1	3400
10	SO43706	BK-R93R-48	27621	2010-12-31	2011-01-07	2011-01-12	3578	1	3578
11	SO43707	BK-R93R-48	27616	2010-12-31	2011-01-07	2011-01-11	3578	1	3578
12	SO43708	BKR50R-52	20042	2010-12-31	2011-01-07	2011-01-12	699	1	699
13	SO43709	BK-R93R-52	16351	2010-12-31	2011-01-07	2011-01-12	3578	1	3578
14	SO43710	BK-R93R-56	16517	2010-12-31	2011-01-07	2011-01-12	3578	1	3578
15	SO43711	BK-R93R-56	27606	2011-01-01	2011-01-07	2011-01-11	3578	1	3578
16	SO43712	BK-R93B-44	13513	2011-01-01	2011-01-08	2011-01-13	3578	1	3578

Checking for invalid dates. The purpose of this check is to ensure that the ship dates and due dates are not earlier than the order dates. The following query was run and returned no results - confirming that order dates are not later than ship or due dates.

```

SELECT
*
FROM bronze.crm_sales_details
WHERE sls_order_dt > sls_ship_dt OR sls_order_dt > sls_due_dt

```

Checking quality of data for sales, quantity and price. In this fictional project, the following business rules have been defined for these fields:

- Sales must be equal to the product of quantity and price ($\text{sales} = \text{quantity} * \text{price}$).
- Negative values, zeros, and Nulls are not allowed

To check for data consistency, I ran the following query that returned results with records in all three fields that required correction.

	sls_sales	sls_quantity	sls_price
1	0	1	10
2	16	2	NULL
3	NULL	1	9
4	NULL	1	35
5	35	1	NULL
6	30	1	-30
7	70	2	NULL
8	10	2	NULL
9	NULL	1	24
10	21	1	-21
11	40	1	2
12	-18	1	9
13	NULL	1	10
14	100	10	NULL

In a real-world situation, I would need to meet with the data owners / business stakeholders to review the data quality issues and gain alignment on a decision to resolve them. The decision could be for them to fix the data in the source system or to correct them through transformations in the data warehouse until they are able to correct them in the source system.

As this is a fictional situation, the following rules were applied to correct the data quality issues identified for sales, price, and quantity.

- If Sales is negative, zero, or null, derive it using Quantity * Price
- If Price is zero or Null, calculate it using sales / quantity
- If Price is negative, convert to a positive number

The following corrections were added to the transformation query using the above rules.

```

sls_price AS old_sls_price,
CASE WHEN sls_sales IS NULL OR sls_sales <= 0 OR sls_sales != sls_quantity * ABS(sls_price)
      THEN sls_quantity * ABS(sls_price)
      ELSE sls_sales
END AS sls_sales,
sls_quantity,
CASE WHEN sls_price IS NULL OR sls_price <=0
      THEN sls_sales / NULLIF(sls_quantity,0)
      ELSE sls_price
END AS sls_price

```

Finalize transformation query for `crm_sales_details`, update ddl, and `INSERT` into the silver table. After adding the above to the transformation query, I also had to update the data types for the date fields in the ddl script for the `sales_details` table from INT to DATE and re-execute. I Then prepared the `INSERT INTO` script to insert the corrected data into the silver table

```

SQLQuery35.s...lindsley (72)* -> Silver_Layer_T...lindsley (51)* | Silver_Layer_C...lindsley (62) | prd_info_trans...
Cust_Info_Fla...lindsley (68) | cust_info_tran...lindsley (60)
1   1.  INSERT INTO silver.crm_sales_details (
2     sls_ord_num,
3     sls_prd_key,
4     sls_cust_id,
5     sls_order_dt,
6     sls_ship_dt,
7     sls_due_dt,
8     sls_sales,
9     sls_quantity,
10    sls_price
11  )
12
13  SELECT
14    sls_ord_num,
15    sls_prd_key,
16    sls_cust_id,
17    CASE WHEN sls_order_dt = 0 OR LEN(sls_order_dt) != 8 THEN NULL
18    ELSE CAST(CAST(sls_order_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
--casting the VARCHAR to a date
19  END AS sls_order_dt,
20  CASE WHEN sls_ship_dt = 0 OR LEN(sls_ship_dt) != 8 THEN NULL
21  ELSE CAST(CAST(sls_ship_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
--casting the VARCHAR to a date
22  END AS sls_ship_dt,
23  CASE WHEN sls_due_dt = 0 OR LEN(sls_due_dt) != 8 THEN NULL
24  ELSE CAST(CAST(sls_due_dt AS VARCHAR) AS DATE) --Order here is casting the integer values to VARCHAR then
--casting the VARCHAR to a date
25  END AS sls_due_dt,
26  CASE WHEN sls_sales IS NULL OR sls_sales <= 0 OR sls_sales != sls_quantity * ABS(sls_price)
27    THEN sls_quantity * ABS(sls_price)
28  ELSE sls_sales
29  END AS sls_sales,
30  sls_quantity,
31  CASE WHEN sls_price IS NULL OR sls_price <= 0
32    THEN sls_sales / NULLIF(sls_quantity, 0)
33  ELSE sls_price
34  END AS sls_price
35  FROM bronze.crm_sales_details
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
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
856
857
858
858
859
859
860
861
862
863
864
865
865
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
886
887
887
888
888
889
889
890
891
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
162
```

I wrote the following transformation query to remove the “NAS” from any records that included it in the cid field, then added the CASE WHEN to a WHERE statement to check if there were any records from the correction that did not appear in the silver table of `crm_cust_info`.

```

SQLQuery36...lndslay (51)* -> X
1  SELECT
2    cid,
3    CASE WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid, 4, LEN(cid))
4      ELSE cid
5    END AS cid,
6    bdate,
7    gen
8  FROM bronze.erp_cust_az12
99 % 14 0 ↑ ↓
Results Messages
cid cid bdate gen
1 NASAW00011000 AW00011000 1971-10-06 Male
2 NASAW00011001 AW00011001 1976-05-10 Male
3 NASAW00011002 AW00011002 1971-02-09 Male
4 NASAW00011003 AW00011003 1973-08-14 Female
5 NASAW00011004 AW00011004 1979-08-05 Female
6 NASAW00011005 AW00011005 1976-08-01 Male

```



```

SQLQuery36...lndslay (51)* -> X
1  SELECT
2    cid,
3    CASE WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid, 4, LEN(cid))
4      ELSE cid
5    END AS cid,
6    bdate,
7    gen
8  FROM bronze.erp_cust_az12
9  WHERE CASE WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid, 4, LEN(cid))
10    ELSE cid
11    END NOT IN (SELECT DISTINCT cst_key FROM silver.crm_cust_info)
12
99 % 14 0 ↑ ↓
Results Messages
cid cid bdate gen

```

Check birthdates for values outside of upper and lower date boundaries

The following query was run to identify anyone that is ≥ 100 years old or have birthdates in the future. The results returned records that meet both criteria and would require correction. In a real world situation, I would need to meet with the data owner / business stakeholder to determine how these should be treated. Since this is a fictional situation, I will pretend that I met with the stakeholder and was informed that only birthdates in the future would be adjusted in the data set and they would be replaced with NULL. This was accomplished using a CASE WHEN and GETDATE.

```

--Check for out-of-range-dates
13  SELECT DISTINCT
14    bdate
15  FROM bronze.erp_cust_az12
16  WHERE bdate < '1924-01-01' OR bdate > GETDATE()
17
99 % 12 0 ↑ ↓
Results Messages
bdate
1 1916-02-10
2 1917-02-09
3 1917-06-05
4 1917-09-20
5 1918-02-11
6 1918-11-08
7 1919-02-14
8 1919-03-10
9 1920-11-14
10 1922-01-02
11 1922-04-10
12 1922-06-06
13 1923-04-14
14 1923-08-16
15 1923-11-16
16 2038-10-17
17 2042-02-22
18 2045-03-03
19 2050-05-21

```

Check gen field for data standardization

This is a categorization field and should have standard values. A quick query on distinct values in the gen field resulted in inconsistent values that will require standardization ("F", "Female", "M", "Male", etc.). Additionally, there are empty values and nulls. In a real world situation, I would meet with the business stakeholder to obtain a decision on the appropriate business-friendly correction and how to account for / correct the NULLs and empty fields.

The screenshot shows a SQL query results window. The query is:`--Check "gen" field for consistency
SELECT DISTINCT
 gen
FROM bronze.erp_cust_az12`

The results table has one column 'gen' with the following data:

gen
NULL
F
Male
Female
M

The following query was written to standardize values in the gen field.

The screenshot shows a SQL query results window. The query is:`--Check "gen" field for consistency
SELECT DISTINCT
 gen,
 CASE WHEN UPPER(TRIM(gen)) IN ('F', 'FEMALE') THEN 'Female'
 WHEN UPPER(TRIM(gen)) IN ('M', 'MALE') THEN 'Male'
 ELSE 'n/a'
 END AS gen
FROM bronze.erp_cust_az12`

The results table has two columns: 'gen' and 'gen'. The 'gen' column contains the original values, and the second 'gen' column contains the standardized values. Red arrows highlight the 'gen' column and the second 'gen' column in the results table.

gen	gen
Female	Female
F	Female
	n/a
M	Male
NULL	n/a
Male	Male

Insert erp_cust_az12 into Silver Layer

The corrections identified for erp_cust_az12 were consolidated into a single transformation query along with an INSERT INTO command to insert the corrected data into the silver layer with quality checks completed to ensure data fed correctly

```
--INSERT transformation for erp_cust_az12 into the silver layer
1  INSERT INTO silver.erp_cust_az12 (
2      cid,
3      bdate,
4      gen
5
6
7      SELECT
8          CASE WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid, 4, LEN(cid))
9              ELSE cid
10         END AS cid,
11         CASE WHEN bdate > GETDATE() THEN NULL
12             ELSE bdate
13         END AS bdate,
14         CASE WHEN UPPER(TRIM(gen)) IN ('F', 'FEMALE') THEN 'Female'
15             WHEN UPPER(TRIM(gen)) IN ('M', 'MALE') THEN 'Male'
16             ELSE 'n/a'
17         END AS gen
18     FROM bronze.erp_cust_az12
19
20
21     --Check for out-of-range-dates
22     SELECT DISTINCT
23         bdate

```

99% ▾

✖ 22 ⚠ 0 ↑ ↓ ↻

Messages

(18483 rows affected)

Completion time: 2025-11-27T22:23:36.8184966-05:00

ERP → loca101 (The table with customer id and country)

Checking primary key (cid) to determine what needs to be done to connect to crm_cust_info via the cst_key

Running separate queries against both tables allows me to see that the cid field in the loc_a101 table has a “-” between the first two string characters and the remaining numerical characters in the field. The “-” will need to be removed to enable mapping between the two tables.

The following script was written to remove the “-” from the cid field in loc_a101. I also ran the same query with a NOT IN subquery to determine whether any of the transformed values did not have matches in the cust_info_table

```
SQLQuery37...indsley (65)* -> X erp_cust_az12...indsley
1   v SELECT
2     REPLACE(cid, '-', '') cid,
3     cntry
4   FROM bronze.erp_loc_a101
5

99 %  x 6  ! 0  ↑  ↓  ←
Results Messages


|   | cid         | cntry     |
|---|-------------|-----------|
| 1 | AW000011000 | Australia |
| 2 | AW000011001 | Australia |
| 3 | AW000011002 | Australia |
| 4 | AW000011003 | Australia |


```

Checking the country field to identify consistency / needs for standardization

I ran a SELECT DISTINCT query to identify all possible values and identify needs for standardization. There is a mix of abbreviations, fully spelled out country names, NULLs, and empty values.

```

SQLQuery37...lindsay (65)* -> X erp_cust_az12...lindsay (51)*
1   SELECT
2     REPLACE(cid, '-', '') cid,
3     ctry
4   FROM bronze.erp_loc_a101;
5
6   SELECT DISTINCT
7     ctry
8   FROM bronze.erp_loc_a101
9

```

	cntry
1	DE
2	USA
3	Germany
4	United States
5	NULL
6	Australia
7	United Kingdom
8	
9	Canada
10	France
11	US

The following query was written and added to the transformation query to correct for the inconsistent country identifiers. I also added the same CASE WHEN to the previous query to identify how the matches mapped to their original values to confirm correction.

```

SQLQuery37.s...lindsay (65)* -> X erp_cust_az12...lindsay (51)*
1   SELECT
2     REPLACE(cid, '-', '') cid,
3     CASE WHEN TRIM(ctry) = 'DE' THEN 'Germany'
4       WHEN TRIM(ctry) IN ('US', 'USA') THEN 'United States'
5       WHEN TRIM(ctry) = '' or ctry IS NULL THEN 'n/a'
6       ELSE TRIM(ctry)
7     END AS ctry
8   FROM bronze.erp_loc_a101;

```

```

SQLQuery37.s...lindsay (65)* -> X erp_cust_az12...lindsay (51)*
9
10
11
12   --CHECK COUNTRY FOR CONSTITENCY & STANDARDIZATION--
13
14   SELECT DISTINCT
15     ctry AS old_ctry,
16     CASE WHEN TRIM(ctry) = 'DE' THEN 'Germany'
17       WHEN TRIM(ctry) IN ('US', 'USA') THEN 'United States'
18       WHEN TRIM(ctry) = '' or ctry IS NULL THEN 'n/a'
19       ELSE TRIM(ctry)
20     END AS ctry
21   FROM bronze.erp_loc_a101

```

	old_ctry	ctry
1	US	United States
2	United States	United States
3	DE	Germany
4	Canada	Canada
5	Australia	Australia
6	France	France
7	USA	United States
8	NULL	n/a
9	United Kingdom	United Kingdom
10		n/a
11	Germany	Germany

Insert loc_a101 into silver layer

I wrote an INSERT INTO query to insert the transformed data from the bronze layer into the silver layer and ran a query to review and confirm that corrections to cid and cntry fields appeared correctly in the silver layer

```
1  ✓ INSERT INTO silver.erp_loc_a101 (
2    cid,
3    cntry
4  )
5  SELECT
6    REPLACE(cid, '-', '') cid,
7    CASE WHEN TRIM(cntry) = 'DE' THEN 'Germany'
8      WHEN TRIM(cntry) IN ('US', 'USA') THEN 'United States'
9      WHEN TRIM(cntry) = '' or cntry IS NULL THEN 'n/a'
10     ELSE TRIM(cntry)
11   END AS cntry
12  FROM bronze.erp_loc_a101;
```

	cid	cntry	dwh_create_date
1...	AW00029465	Australia	2025-11-27 22:48:44.3600000
1...	AW00029466	Germany	2025-11-27 22:48:44.3600000
1...	AW00029467	Germany	2025-11-27 22:48:44.3600000
1...	AW00029468	United King...	2025-11-27 22:48:44.3600000
1...	AW00029469	United King...	2025-11-27 22:48:44.3600000
1...	AW00029470	France	2025-11-27 22:48:44.3600000
1...	AW00029471	France	2025-11-27 22:48:44.3600000
1...	AW00029472	France	2025-11-27 22:48:44.3600000
1...	AW00029473	United King...	2025-11-27 22:48:44.3600000
1...	AW00029474	Germany	2025-11-27 22:48:44.3600000
1...	AW00029475	Germany	2025-11-27 22:48:44.3600000

ERP → px_cat_g1v2 (The table with product category ids, product categories, subcategories, and maintenance identifier)

Checking primary key (id) to determine what needs to be done to connect to crm_prod_info via the cat_id that was created earlier in for the silver layer of this table

a. Running separate queries against both tables allows me to see the values cat_id in crm_prod_info appears to be in the same format and structure as the values listed in the id field of px_cat_g1v2 and is ready to be used for mapping

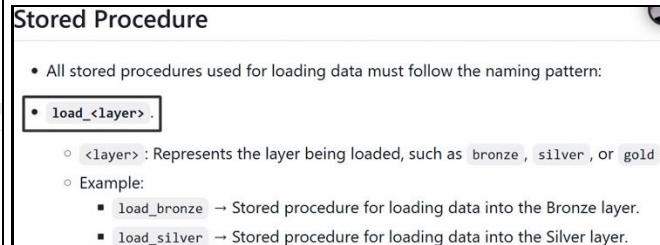
Checking the “cat” field to ensure consistency, removal of unwanted spaces, etc.

- Running a query to identify unwanted spaces across all fields returned zero results
- Running a query to identify inconsistencies in the cat field resulted in no consistent value usage
- Running a query to identify inconsistencies in the subcat field resulted in no consistent value usage
- Running a query to identify inconsistencies in the maintenance field resulted in no consistent value usage

With all checks complete, I wrote the INSERT INTO query to place the data into the silver layer. Then built a query that consolidated the INSERT queries for all tables in the silver layer with TRUNCATE TABLE commands for each.

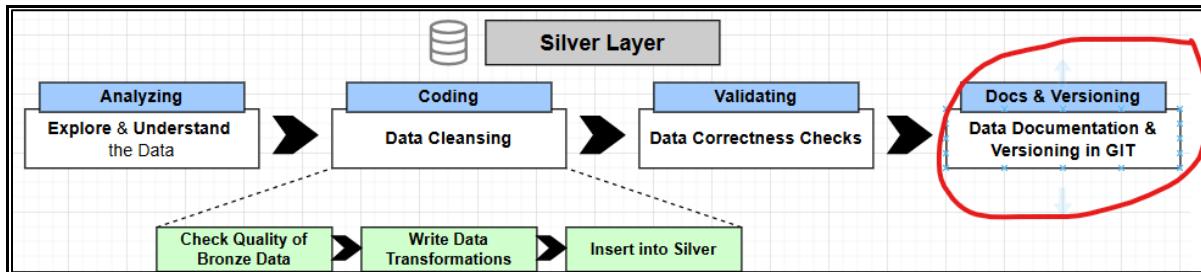
Creating a stored procedure:

a. I wrote a stored procedure for the loading of the data into the database as it is intended to be a recurring action to bring in the data.

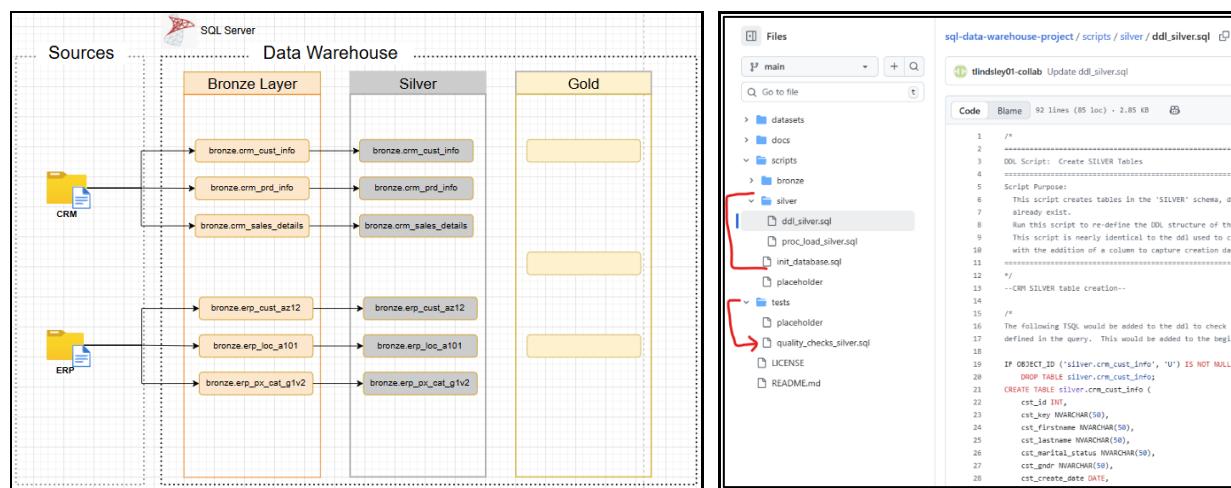


Once executing the stored procedure and refreshing the DB to visually confirm its existence, I then created a new query to test the execution of the procedure. The following SQL statement was used: EXEC silver..load silver

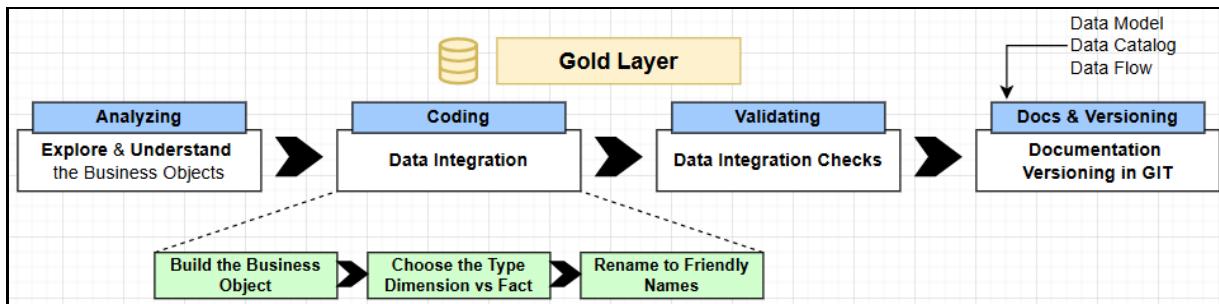
d. Documenting & Commit Code in GIT



I updated the data flow to reflect the lineage of data from source systems to bronze to silver. I also loaded and committed my ddl script, stored procedure, and quality checks for the silver layer to my Git repo.



8. Build Gold Layer



a. Analyzing: Explore Business Objects

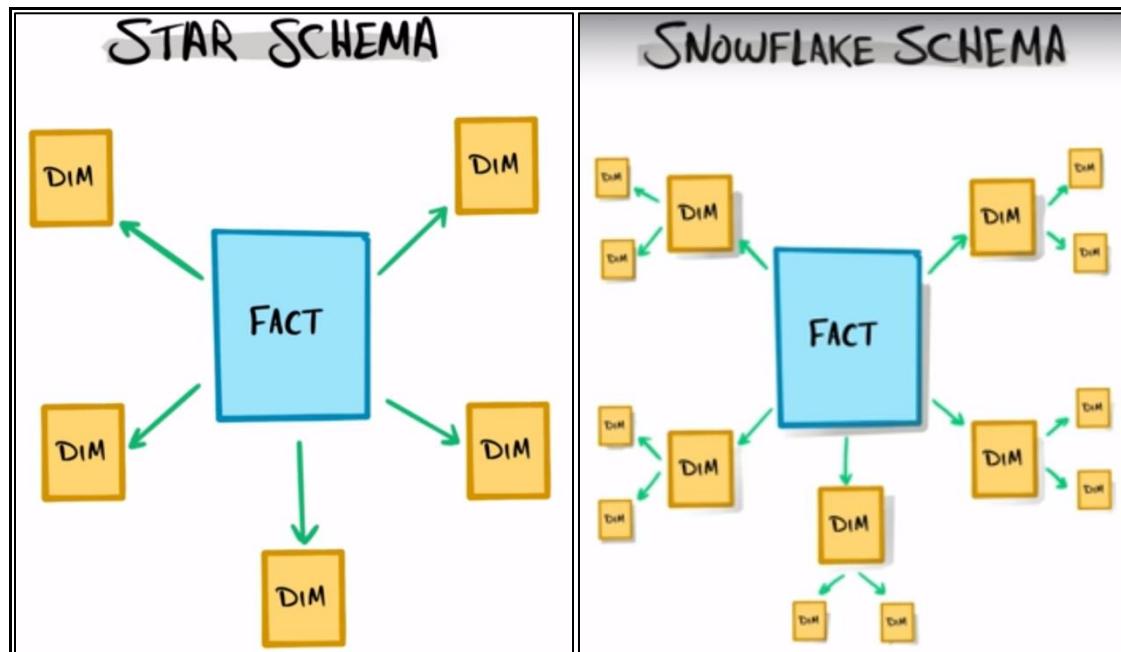
Data Model Selection

The first step in this task is to select the logical data model to enable development of relationships between data sets. Additionally, it serves as a reference for future users of the data warehouse.

The data model needs to be optimized for data analytics. It also needs to be flexible, scalable and easy to understand. The types of data models typically used are Star Schema and Snowflake Schema.

Star Schema: A central fact table in the center and surrounded by dimensions. The fact table contains transactional / event-oriented data, and the dimensions contain descriptive information and the relationship between the fact table and the dimensions around it form a star shape when depicted visually....hence “Star Schema”. It is simple in design and easy to query; however dimensions might have duplicates and they get larger over time. Commonly used and ideal for BI tools like Tableau. Star Schema will be used for this project

Snowflake Schema: Similar to the Star Schema in that the fact table is the central component of the model and is surrounded by dimensions; however, each dimension is broken out further into sub-dimensions. This extension into sub-dimensions results in a snowflake pattern when depicted visually. Hence...the name “Snowflake Schema”. The Snowflake Schema is more complex, requires deeper knowledge of the underlying data sets and requires more effort to query. The main advantage comes through normalization from breaking redundancies into smaller tables which optimizes storage.

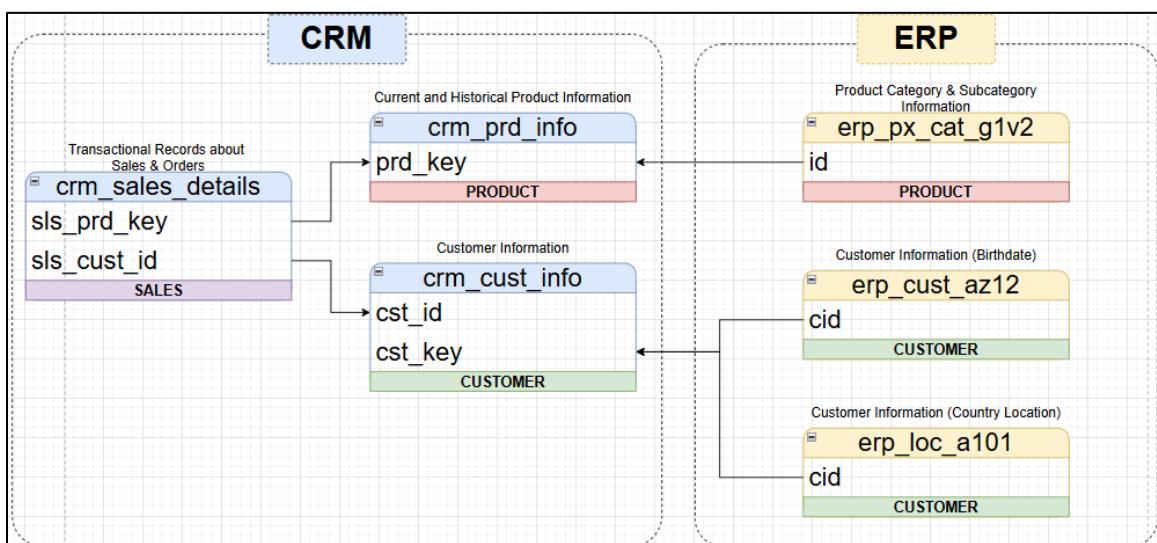


Data Model Design & Build

The dimensions and facts need to be defined now that Star Schema has been selected. Dimension tables will contain descriptive information to give context to the data (example: product info). If the data is oriented around who, what, when and where...then it belongs in the dimension table. Fact tables contain quantitative information that represent events / transactions (Example: sales orders, shipments, returns with dates, and other numerical data). If the data is oriented around how much, when, etc....then it belongs in the Fact table.

The first step here is to understand the business context of the data to ensure data is correctly identified / assigned to dimension tables and fact table. In a real-world situation, I would have met with the business stakeholders / process SMEs / process owners to walk-through the data and its alignment to relevant business processes.

Before doing so, it is important to explore the data and develop an initial understanding and documenting it to enable review and validation with the business SMEs. As this is a fictional situation without stakeholders to review, I merely prepared the below visual representation of the data objects and their relationships that will serve as the foundation for building out the Star Schema for the Gold Layer.



b. Coding & Validating: Data Integration & Quality Checks

Build of the Gold Layer is focused on creating the Star Schema in the data warehouse with dimensions and facts defined to build Views. This will be done by integrating Silver Layer data into business-friendly objects to enable data consumption for reporting & analytics.

Integrating Customer Data

There are three tables that need to be integrated to bring customer data together into a single view in the Gold layer from the Silver layer. Those are:

- silver.crm_cust_info
- silver.erp_cust_az12
- silver.erp_loc_a101

I created a join of these three tables with 'silver.crm_cust_info' serving as the master table.

Once created, I ran the query and ran a check to ensure there are no duplicates in the primary keys. If duplicates appear as a result of the check, then that is an indication that there is an issue with the Silver layer data that would need to be resolved before moving forward with the Gold Layer.

Initial Query to Join Customer Data

```

SQLQuery41...lndley (51)*  □ X
1   --Integrate Customer Data From Silver Layer per the data model
2   ✓ SELECT
3       ci.cst_id,
4       ci.cst_key,
5       ci.cst_firstname,
6       ci.cst_lastname,
7       ci.cst_marital_status,
8       ci.cst_gndr,
9       ci.cst_create_date,
10      ca.bdate,
11      ca.gen,
12      la.cntry
13  ✓ FROM silver.crm_cust_info ci -- silver.crm_cust_info serves as the master table to which the other customer tables will be joined
14      -- "ci" is the alias being assigned to this table. To enable use in the query
15  ✓ LEFT JOIN silver.erp_cust_az12 ca -- "ca" is the alias being assigned to this table. To enable use in the query
16      ON ci.cst_key = ca.cid -- joining on these two fields in alignment with the data model and data clean-up work that was done in the Silver layer
17  ✓ LEFT JOIN silver.erp_loc_az01 la -- "la" is the alias being assigned to this table. To enable use in the query
18      ON ci.cst_key = la.cid -- joining on these two fields in alignment with the data model and data clean-up work that was done in the Silver layer

```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	bdate	gen	cntry
29469	AW00029469	Dominique	Saunders	Married	Female	2026-01-25	1977-10-20	n/a	United King...
29470	AW00029470	Nathan	Roberts	Single	n/a	2026-01-25	1971-08-22	n/a	France
29471	AW00029471	Dana	Oteaga	Single	n/a	2026-01-25	1965-09-23	n/a	France
29472	AW00029472	Lacey	Shama	Married	n/a	2026-01-25	1965-09-11	n/a	France
29473	AW00029473	Carmen	Subram	Single	n/a	2026-01-26	1965-11-19	n/a	United King...
29474	AW00029474	Jane	Raje	Married	n/a	2026-01-25	1965-04-02	n/a	Germany
29475	AW00029475	Jaed	Ward	Single	n/a	2026-01-25	1976-03-18	n/a	Germany
29476	AW00029476	Elizabeth	Bradley	Married	n/a	2026-01-25	1964-12-30	n/a	Germany
29477	AW00029477	Neil	Ruiz	Married	n/a	2026-01-25	1965-01-02	n/a	United King...
29478	AW00029478	Daren	Carlson	Single	n/a	2026-01-25	1964-11-21	n/a	United King...
29479	AW00029479	Tomas	Tapo	Married	n/a	2026-01-25	1965-06-30	n/a	France

Query & Results Checking for Duplicate Primary Keys

```

SQLQuery41...lndley (51)*  □ X
1   --Integrate Customer Data From Silver Layer per the data model
2   ✓ SELECT cst_id, COUNT(*) FROM
3       (SELECT
4           ci.cst_id,
5           ci.cst_key,
6           ci.cst_firstname,
7           ci.cst_lastname,
8           ci.cst_marital_status,
9           ci.cst_gndr,
10          ci.cst_create_date,
11          ca.bdate,
12          ca.gen,
13          la.cntry
14      ✓ FROM silver.crm_cust_info ci -- silver.crm_cust_info serves as the master table to which the other customer tables will be joined
15      -- "ci" is the alias being assigned to this table. To enable use in the query
16  ✓ LEFT JOIN silver.erp_cust_az12 ca -- "ca" is the alias being assigned to this table. To enable use in the query
17      ON ci.cst_key = ca.cid -- joining on these two fields in alignment with the data model and data clean-up work that was done in the Silver layer
18  ✓ LEFT JOIN silver.erp_loc_az01 la -- "la" is the alias being assigned to this table. To enable use in the query
19      ON ci.cst_key = la.cid -- joining on these two fields in alignment with the data model and data clean-up work that was done in the Silver layer
20
21  ✓ GROUP BY cst_id
22  ✓ HAVING COUNT(*) > 1

```

cst_id	(No column name)
	1

The fields that contain customer gender in tables 'crm_cust_info' and 'silver.erp_cust_az12' were not cleansed for consistency during the build of the Silver layer. I would think that the correct approach would be to go back to the silver layer and make the corrections there, but....as I am merely following along with the instructor, I will cleanse them for consistency in the gold layer as he does.

Below is a view of the inconsistency that needs to be corrected. In a real world situation, I would have the business stakeholder / data owner confirm the system that serves as the source of truth for customer gender. In this fictitious situation, the crm is the source of truth. So, I will need to create a query that applies consistency and build it into the primary query for customer data.

The query written to apply consistency to customer gender data will leverage a CASE WHEN to use the value in silver.crm_cust_info if the value is not 'n/a'. If the value is 'n/a', the it will pull the gender from silver.erp_cust_az12. If it is n/a in both, then the value remains as 'n/a'.

The query built to apply consistency was added to the customer info data integration query. I then applied business-friendly names to each field, reordered the columns to improve readability.

The integrated customer info is a dimension as it contains descriptive data rather than events, transactions, or any measurements.

As dimension object, it will need a primary key defined for each record. In this project, a surrogate key will be created using a Window function (Row_Number) so that there is no dependency on use of the primary keys of integrated tables.

Surrogate Key = System-generated unique identifier assigned to each record in a table. They can be defined in the ddl or they can be query based using a Window function (Row_Number)

Query Identifying Inconsistency in Gender Identifiers

The screenshot shows two tabs in SSMS: 'SQLQuery42.s...indsley (61)*' and 'SQLQuery41.sq...indsley (51)*'. The code in the first tab is:

```

1  SELECT DISTINCT
2      ci.cst_gndr,
3      ca.gen
4  FROM silver.crm_cust_info ci -- silver.crm_cu
-- "ci" is t
5      LEFT JOIN silver.erp_cust_az12 ca -- "ca" is
6          ON ci.cst_key = ca.cid -- joining on t
7      LEFT JOIN silver.erp_loc_a101 la -- "la" is t
8          ON ci.cst_key = la.cid
9
10     ORDER BY 1, 2

```

The results grid shows the following data:

	cst_gndr	gen
1	Female	Female
2	Female	Male
3	Female	n/a
4	Male	Female
5	Male	Male
6	Male	n/a
7	n/a	NULL
8	n/a	Female
9	n/a	Male
10	n/a	n/a

Query Applying Consistency to Gender Data

The screenshot shows two tabs in SSMS: 'SQLQuery42.s...indsley (61)*' and 'SQLQuery41.sq...indsley (51)*'. The code in the first tab is:

```

1  SELECT DISTINCT
2      ci.cst_gndr,
3      ca.gen
4      CASE WHEN ci.cst_gndr != 'n/a' THEN ci.cst_gndr -- CRM is master
5          ELSE COALESCE(ca.gen, 'n/a') END AS new_gen
6
7  FROM silver.crm_cust_info ci -- silver.crm_cust_info serves as the
-- "ci" is the alias being assign
8      LEFT JOIN silver.erp_cust_az12 ca -- "ca" is the alias being assign
9          ON ci.cst_key = ca.cid -- joining on these two fields in al
10     LEFT JOIN silver.erp_loc_a101 la -- "la" is the alias being assign
11         ON ci.cst_key = la.cid
12
13     ORDER BY 1, 2

```

The results grid shows the following data:

	cst_gndr	gen	new_gen
1	Female	Female	Female
2	Female	Male	Female
3	Female	n/a	Female
4	Male	Female	Male
5	Male	Male	Male
6	Male	n/a	Male
7	n/a	NULL	n/a
8	n/a	Female	Female
9	n/a	Male	Male
10	n/a	n/a	n/a

CREATE CUSTOMER VIEW

The screenshot shows the Object Explorer on the left with a tree view of database objects. A red box highlights the 'gold.dim_customer' view under the 'DataWarehouse' database. The main pane displays the T-SQL code for creating the view:

```
CREATE VIEW gold.dim_customer AS
SELECT
    ROW_NUMBER() OVER (ORDER BY cst_id) AS customer_key, --Assigns surrogate key (primary key to all records)
    ci.cst_id AS customer_id,
    ci.cst_key AS customer_number,
    ci.cst_firstname AS first_name,
    ci.cst_lastname AS last_name,
    la.ctry AS country,
    ci.cst_marital_status AS marital_status,
    CASE WHEN ci.cst_gndr != 'n/a' THEN ci.cst_gndr -- CRM is master for gender info
        ELSE COALESCE(ca.gen, 'n/a')
    END AS gender,
    ca.bdate AS birthdate,
    ci.cst_create_date AS create_date

FROM silver.crm_cust_info ci -- silver.crm_cust_info serves as the master table to which the other customer tab
-- "ci" is the alias being assigned to this table. To enable use in the query
LEFT JOIN silver.erp_cust_ar12 ca -- "ca" is the alias being assigned to this table. To enable use in the quer
ON ci.cst_key = ca.cid -- joining on these two fields in alignment with the data model and data clean-up
LEFT JOIN silver.erp_loc_ar01 la -- "la" is the alias being assigned to this table. To enable use in the quer
ON ci.cst_key = la.cid -- joining on these two fields in alignment with the data model and data clean-up
```

The status bar at the bottom indicates 'Commands completed successfully.' and 'Completion time: 2020-12-01T09:07:41.0146617-05:00'.

CREATE CUSTOMER VIEW

The final step in the creation of the Customer View is to conduct a quality check of the data. To do so, I ran a 'SELECT *' query and conducted a quick scan to ensure all data appeared in the correct order and that there are no unexpected NULL values. I also ran a 'SELECT DISTINCT' query on the 'gender' field to ensure that there are no values in the field other than 'Male', 'Female', or 'n/a'. Results of both queries indicated success.

Integrating Product Data

1. Product data exists in both the CRM and the ERP data sources. Integrating these into a single, usable view is necessary to enable analytics on the data.

The product info table contains historical and current product information. In a real-world scenario, a decision would be needed as to whether the view must contain both historical and current product information for analysis. In this fictitious case, only the current product data will be used in the view. The product key is being used in the mapping of product data between tables. This also allows for the historical data to be filtered out so only current product data is included.

To filter out historical data, a WHERE condition was applied to the query on the prd_end_dt field to filter out records where the value != NULL. All records with NULL in the prd_end_dt field are active products because the product end date has not occurred.

Query Joining Product Data from CRM and ERP tables

The screenshot shows a SQL Server Management Studio window with three tabs: 'SQLQuery43.s...lindsay (62)*', 'SQLQuery42.sq...lindsay (61)*', and 'gold_layer_cre...lindsay (51)'. The main query window contains the following code:

```

1  SELECT
2      pn_prd_id,
3      pn_cat_id,
4      pn_prd_key,
5      pn_prd_nm,
6      pn_prd_cost,
7      pn_prd_line,
8      pn_prd_start_dt,
9      pc_cat,
10     pc_subcat,
11     pc_maintenance
12  FROM silver.crm_prd_info pn -- silver.crm_prd_info serves as the master table to which the ERP product table will be joined
13      -- "pn" is the alias being assigned to this table. To enable use in the query
14  LEFT JOIN silver.erp_px_cat_glv2 pc -- "pc" is the alias being assigned to this table. To enable use in the query
15  ON pn.cat_id = pc.id
16  WHERE prd_end_dt IS NULL -- Filter out all historical data

```

The results grid displays 15 rows of product data, including columns like prd_id, cat_id, prd_key, prd_nm, prd_cost, prd_line, prd_start_dt, cat, subcat, and maintenance.

Conduct Data Quality Checks

The check for data quality, I wrote a `SELECT COUNT(*)` query with grouping by `prd_key` for records with `prd_key` appearing more than once. There were no records displayed in results the quality check query. That tells me that:

- There are no duplicates resulting from the join
- Each product has only one record
- There are no historical data

The screenshot shows a SQL Server Management Studio window with three tabs: 'SQLQuery43.s...lindsay (62)*', 'SQLQuery42.sq...lindsay (61)*', and 'gold_layer_cre...lindsay (51)'. The main query window contains the following code:

```

2  SELECT prd_key, COUNT(*) FROM (
3      SELECT
4          pn_prd_id,
5          pn_cat_id,
6          pn_prd_key,
7          pn_prd_nm,
8          pn_prd_cost,
9          pn_prd_line,
10         pn_prd_start_dt,
11         pc_cat,
12         pc_subcat,
13         pc_maintenance
14    FROM silver.crm_prd_info pn -- silver.crm_prd_info serves as the master table to which the ERP product table will be joined
15        -- "pn" is the alias being assigned to this table. To enable use in the query
16    LEFT JOIN silver.erp_px_cat_glv2 pc -- "pc" is the alias being assigned to this table. To enable use in the query
17    ON pn.cat_id = pc.id
18    WHERE prd_end_dt IS NULL -- Filter out all historical data
19 )t
20 GROUP BY prd_key HAVING COUNT(*) > 1

```

The results grid shows a single row with the column `prd_key` and value `(No column name)`.

The field names were then assigned business-friendly names and columns were reordered for readability. Additionally a the query was enhanced to include the creation of a surrogate key (primary key) for each product using the `ROW_NUMBER` window function (similar to what was done for the customer info dimension).

CREATE VIEW was added to the query, then the query was executed...resulting in the creation of the view in the data warehouse.

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists databases, including 'PF32ZQ8SQLEXPRESS' (SQL Server 16.0) which contains 'gold.dim_products'. The query editor window on the right displays a T-SQL CREATE VIEW statement for 'gold.dim_products'. A red arrow points from the 'gold.dim_products' entry in Object Explorer to the corresponding table reference in the query editor.

```
CREATE VIEW gold.dim_products AS
BEGIN
    SELECT
        ROW_NUMBER() OVER (ORDER BY pn.prd_start_dt, pn.prd_key) AS product_key, --Create and assign primary key
        pn.prd_id AS product_id,
        pn.prd_key AS product_number,
        pn.prd_nm AS product_name,
        pn.cat_id AS category_id,
        pc.cat AS category,
        pc.subcat AS subcategory,
        pc.maintenance,
        pn.prd_cost AS cost,
        pn.prd_line AS product_line,
        pn.prd_start_dt AS start_date
    FROM silver.crm_prd_info pn -- silver.crm_prd_info serves as the master table to which the ERP product table will be joined
        -- "pn" is the alias being assigned to this table. To enable use in the query
    LEFT JOIN silver.erp_px_cat_glv2 pc -- "pc" is the alias being assigned to this table. To enable use in the query
        ON pn.cat_id = pc.id
    WHERE prd_end_dt IS NULL -- Filter out all historical data
END
```

Building the View of Sales Data as a Facts Table

What I did - connected Customer and Product dimensions (product ID, customer ID) to the sales data

DIM keys **DATES** **MEASURES**

```
--Create Sales Facts Table Joining Customer and Product Dimension to Sales Data
CREATE VIEW gold_fact_sales AS
SELECT
    sd.sls_ord_num AS order_number,
    pr.product_key,
    cu.customer_key,
    sd.sls_order_dt AS order_date,
    sd.sls_ship_dt AS shipping_date,
    sd.sls_due_dt AS due_date,
    sd.sls_sales AS sales_amount,
    sd.sls_quantity AS quantity,
    sd.sls_price AS price
FROM silver_crm_sales_details sd
LEFT JOIN gold.dim_product pr
ON sd.sls_prd_key = pr.product_number
LEFT JOIN gold.dim_customer cu
ON sd.sls_cust_id = cu.customer_id

--Create Sales Facts Table Joining Customer and Product Dimension to Sales Data
CREATE VIEW gold_fact_sales AS
SELECT
    sd.sls_ord_num AS order_number,
    pr.product_key,
    cu.customer_key,
    sd.sls_order_dt AS order_date,
    sd.sls_ship_dt AS shipping_date,
    sd.sls_due_dt AS due_date,
    sd.sls_sales AS sales_amount,
    sd.sls_quantity AS quantity,
    sd.sls_price AS price
FROM silver_crm_sales_details sd
LEFT JOIN gold.dim_product pr
ON sd.sls_prd_key = pr.product_number
LEFT JOIN gold.dim_customer cu
ON sd.sls_cust_id = cu.customer_id
```

	order_number	product_key	customer_key	order_date	shipping_date	due_date	sales_amount	quantity	price
1	SO43697	20	10769	2010-12-29	2011-01-05	2011-01-10	3578	1	3578
2	SO43698	9	17390	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
3	SO43699	9	14864	2010-12-29	2011-01-20	2011-01-10	3400	1	3400
4	SO43700	41	3502	2010-12-29	2011-01-05	2011-01-10	699	1	699
5	SO43701	9	4	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
6	SO43702	16	16646	2010-12-30	2011-01-05	2011-01-11	3578	1	3578
7	SO43703	20	5625	2010-12-30	2011-01-06	2011-01-11	3578	1	3578
8	SO43704	6	6	2010-12-30	2011-01-06	2011-01-11	3375	1	3375
9	SO43705	7	12	2010-12-30	2011-01-06	2011-01-11	400	1	400

Checking Foreign Key Integrity (dimensions) to see if all dimension tables can successfully join to the fact table: No bad results....good to go

```
SQLQuery46.s...indsley (63)* ✎ x SQLQuery45.sql...indsley (61)* gold_layer_cre...  
1 ---Foreign Key Integrity (Dimensions)  
2 SELECT * FROM gold_fact_sales f  
3 LEFT JOIN gold_dim_customer c  
4 ON c.customer_key = f.customer_key  
5 LEFT JOIN gold_dim_products p  
6 ON p.product_key = f.product_key  
7 WHERE p.product_key IS NULL  
  
90% ▾ 8 ✖ 0 ⚠ 0 ↑ ↓ ◀  
Results Messages  

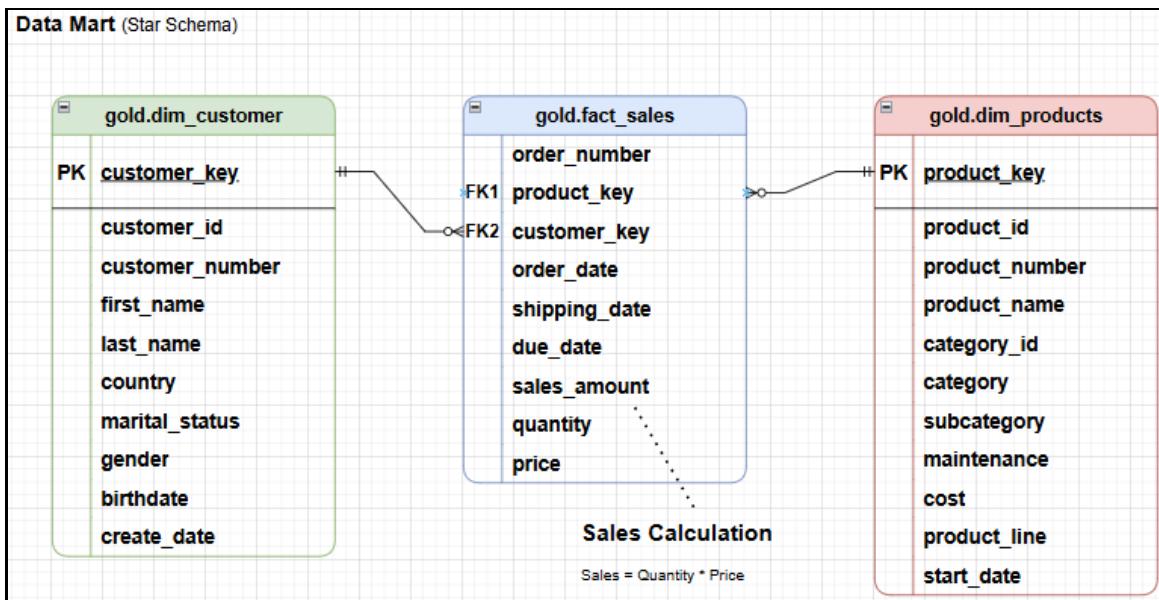

| order_number | product_key | customer_key | order_date | shipping_date | due_date | sales_ |
|--------------|-------------|--------------|------------|---------------|----------|--------|
|--------------|-------------|--------------|------------|---------------|----------|--------|


```

c. Document & Commit Code to Git

Document Star Schema

The Star Schema was modeled to visually depict the relationships defined between the customer dimension, the products dimension and the sales Facts views in the Gold Layer. It includes identification of primary keys, foreign keys, and the relationships (many to one / one to many)



Create Data Catalog & Document Versioning in Git

A Data Catalog is needed to support the data product we have created (the data warehouse) to be developed based on the data warehouse. The Data Catalog describes the data model (it's columns, tables, relationships, etc) to enable users to derive meaningful insights and to avoid repeat questions from users. The data catalog is documented in an md file in my Git repository.

The Data Catalog has a section for each table in the Gold layer (`gold.dim_customer`, `gold.dim_products`, and `gold.facts_sales`). Each section includes the purpose of the table, the column names, data types, and column descriptions (with examples).

I defined and documented the data catalog in a markdown file and committed it to my Git repo for this project under the 'docs' folder.

Data Dictionary for Gold Layer

Overview

The Gold layer is the business-level data representation, structure to support analytical and reporting use cases. It consists of dimension tables and fact tables for specific business metrics.

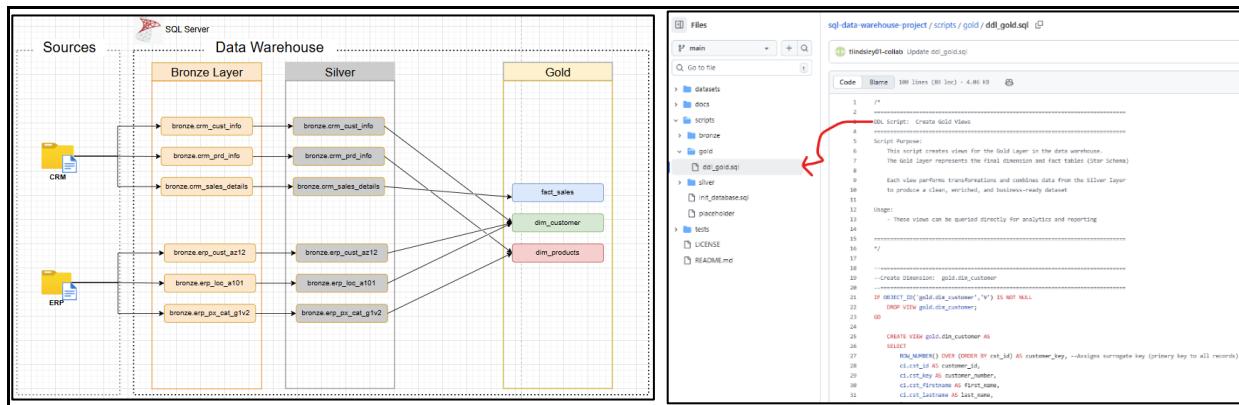
1. gold.dim_customer

- Purpose: Stores customer details enriched with demographic and geographic data
- Columns:

Column Name	Data Type	Description
customer_key	INT	Surrogate key uniquely identifying each customer record in the dimension table
customer_id	INT	Unique numerical identifier assigned to each customer
customer_number	NVARCHAR(50)	Alphanumeric identifier representing the customer, used for tracking and referencing
first_name	NVARCHAR(50)	The customer's first name as recorded in the CRM
last_name	NVARCHAR(50)	The customer's last name as recorded in the CRM
country	NVARCHAR(50)	The customer's country of residence (e.g., 'France')
marital_status	NVARCHAR(50)	The marital status of the customer (e.g. 'Married', 'Single')
gender	NVARCHAR(50)	The customer's gender (e.g., 'Male', 'Female', 'n/a')

Update the Data Flow Diagram to show Gold Layer and commit Gold Layer view ddl to Git: I updated the data flow diagram to include the dimension tables for customer and product data along with the fact table for sales data. This view provides a visual representation of the data lineage for the data warehouse.

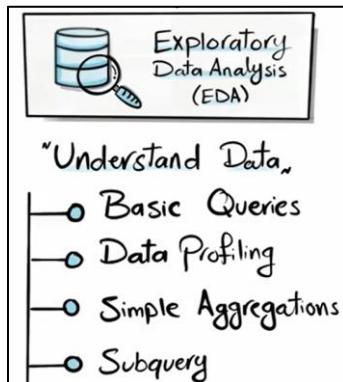
I also combined all gold layer creation scripts into a single ddl sql and uploaded to git with the scripts written for quality checks in the gold layer and all final diagrams and data flows created throughout this project.



9. Exploratory SQL Data Analysis

a. Overview

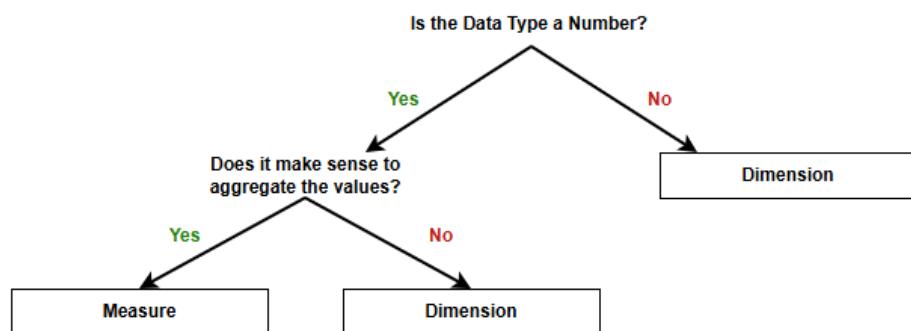
In this stage of the project, I will leverage the basic SQL skills I've learned throughout this effort to conduct data profiling, simple aggregation, and use of subqueries to further understand the data sets.



b. Dimensions vs Measures

The data in the gold layer of my database is separated into dimensions (dim tables) and measures (the fact table).

The data sets were evaluated according to the following to enable determination as to whether they should be defined as dimensions or tables.



Dimension Example: Looking at the 'category' field in the gold.dim_products view within the Gold layer, each of the values are strings (not numbers). Therefore, these are correctly categorized as dimension values and reside in a 'dim' view of the Gold layer.

SQLQuery48.s...indsley (51)*

```
1 | SELECT DISTINCT
2 | category
3 | FROM gold.dim_products
```

Results Messages

category
1. NULL
2. Accessories
3. Bikes
4. Clothing
5. Components

NOT Number

Measure Example: Looking at the 'sales_amount' field in the 'gold.fact_sales' view within the Gold layer, each of the values are numbers. As such, they can be aggregated and are therefore correctly categorized as measures and reside in a 'fact' view of the Gold layer for analysis.

```
SQLQuery48.s...indsley (51) 1 2 3
1 | SELECT DISTINCT
2 | sales_amount
3 | FROM gold.fact_sales
```

sales_amount
29
75
699
9
742
1701
3375
35
55
2182

Numbers
can
Aggregate

I reviewed data all fields across each data set in the gold layer to ensure I have each field correctly identified as a dimension or measure. The table below provides a consolidated view for my reference.

View	Column Name	Number?	Can be Aggregated?	Dimension or Measure	Rationale
gold.dim_customer	customer_key	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	customer_id	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	customer_number	No	No	Dimension	Values are combination of text and numbers. Not measurable values.
	first_name	No	No	Dimension	Values are text. Cannot measure.
	last_name	No	No	Dimension	Values are text. Cannot measure.
	country	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	marital_status	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	gender	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	birthdate	No	No	Dimension	Values are dates. Cannot measure, but can derive 'age' from this column to measure
	create_date	No	No	Dimension	Values are dates. Cannot measure, but can derive 'years as customer' from this column to measure
gold.dim_products	product_key	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	product_id	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	product_number	No	No	Dimension	Values are combination of text and numbers. Not measurable values.
	product_name	No	No	Dimension	Values are combination of text and numbers. Not measurable values.
	category_id	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	category	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	subcategory	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	maintenance	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
	cost	Yes	Yes	Measure	Values are numbers and measurable values
	product_line	No	No	Dimension	Values are text. Cannot measure, but can be categorized / grouped for analysis
gold.fact_sales	start_date	No	No	Dimension	Values are dates. Cannot measure, but can derive 'years as active product' from this column to measure
	order_number	No	No	Dimension	Values are combination of text and numbers. Not measurable values.

View	Column Name	Number?	Can be Aggregated?	Dimension or Measure	Rationale
	product_key	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	customer_key	Yes	Yes	Dimension	Values are numbers, but not measurable values.
	order_date	No	No	Dimension	Values are dates. Cannot measure, but can derive 'days from order to ship' and 'days from order to due date' from this column to measure
	shipping_date	No	No	Dimension	Values are dates. Cannot measure, but can derive 'days from order to ship' and 'days aging past due' from this column to measure
	due_date	No	No	Dimension	Values are dates. Cannot measure, but can derive 'days from order to due date' and 'days aging past due' from this column to measure
	sales_amount	Yes	Yes	Measure	Can measure sales across a variety of dimensions and derive additional measures
	quantity	Yes	Yes	Measure	Can measure quantity sold across a variety of dimensions and derive additional measures
	price	Yes	Yes	Measure	Can measure sales price across a variety of dimensions and derive additional measures (revenue using cost)

Identifying dimensions and measures allows me to identify which fields can be used to group / categorize data to answer questions on measurable values. For example, if I want to know the total sales by product category....I know that 'category' is a dimension and 'sales_amount' is the measure. Having this clarity enables effective analysis.

c. Database Exploration

I already know the layout and underlying structure of the data warehouse, however, if I were not familiar with it...I would need to explore it to develop a quick understanding. Running a handful of queries and reviewing the content and structure would give me this understanding.

Running a 'SELECT *' query on the 'INFORMATION_SCHEMA.TABLES' gives me a high-level understanding of the tables, schema, and table types. Here I can see the medallion architecture, the tables within each layer, and the object type within each layer (base vs view).

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1 DataWarehouse	bronze	crm_cust_info	BASE TABLE
2 DataWarehouse	bronze	crm_prd_info	BASE TABLE
3 DataWarehouse	bronze	crm_sales_details	BASE TABLE
4 DataWarehouse	bronze	erp_cust_az12	BASE TABLE
5 DataWarehouse	bronze	erp_loc_a101	BASE TABLE
6 DataWarehouse	bronze	erp_px_cat_g1v2	BASE TABLE
7 DataWarehouse	silver	crm_cust_info	BASE TABLE
8 DataWarehouse	silver	erp_cust_az12	BASE TABLE
9 DataWarehouse	silver	erp_loc_a101	BASE TABLE
10 DataWarehouse	silver	erp_px_cat_g1v2	BASE TABLE
11 DataWarehouse	silver	crm_prd_info	BASE TABLE
12 DataWarehouse	silver	crm_sales_details	BASE TABLE
13 DataWarehouse	gold	dim_customer	VIEW
14 DataWarehouse	gold	dim_products	VIEW
15 DataWarehouse	gold	fact_sales	VIEW

I can also explore all of the columns in the database by executing a ‘SELECT *’ query on the ‘INFORMATION_SCHEMA.COLUMNS’.

The results of this query provides me with the full inventory of all columns in the database with key metadata for each column, and I can see that there are 101 columns across the entire data warehouse.

```

SQLQuery48s...lindsey(511)  » X
1 -- Explore All Objects in the Database
2 SELECT *
3   FROM INFORMATION_SCHEMA.TABLES
4
5 --Explore All Columns in the Database
6 SELECT *
7   FROM INFORMATION_SCHEMA.COLUMNS

```

Results Messages

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHARACTER_OCTET_LENGTH	Numeric_Precision
1 DataWarehouse	bronze	cmn_cust_info	cust_id	1	NULL	YES	int	NULL	NULL	10
2 DataWarehouse	bronze	cmn_cust_info	cust_key	2	NULL	YES	nvarchar	50	100	NULL
3 DataWarehouse	bronze	cmn_cust_info	cust_flnname	3	NULL	YES	nvarchar	50	100	NULL
4 DataWarehouse	bronze	cmn_cust_info	cust_lname	4	NULL	YES	nvarchar	50	100	NULL
5 DataWarehouse	bronze	cmn_cust_info	cust_marital_status	5	NULL	YES	nvarchar	50	100	NULL
6 DataWarehouse	bronze	cmn_cust_info	cust_gndr	6	NULL	YES	nvarchar	50	100	NULL
7 DataWarehouse	bronze	cmn_cust_info	cust_create_dt	7	NULL	YES	date	NULL	NULL	NULL
8 DataWarehouse	bronze	cmn_cust_info	prod_id	1	NULL	YES	int	NULL	NULL	10
9 DataWarehouse	bronze	cmn_cust_info	prod_key	2	NULL	YES	nvarchar	50	100	NULL
10 DataWarehouse	bronze	cmn_cust_info	prod_nm	3	NULL	YES	nvarchar	50	100	NULL
11 DataWarehouse	bronze	cmn_cust_info	prod_crt	4	NULL	YES	int	NULL	NULL	10
12 DataWarehouse	bronze	cmn_cust_info	prod_line	5	NULL	YES	nvarchar	50	100	NULL
13 DataWarehouse	bronze	cmn_cust_info	prod_stat_dt	6	NULL	YES	datetime	NULL	NULL	NULL
14 DataWarehouse	bronze	cmn_cust_info	prod_end_dt	7	NULL	YES	datetime	NULL	NULL	NULL
15 DataWarehouse	bronze	cmn_sales_details	sle_ordr_num	1	NULL	YES	nvarchar	50	100	NULL
16 DataWarehouse	bronze	cmn_sales_details	sle_prod_key	2	NULL	YES	nvarchar	50	100	NULL
17 DataWarehouse	bronze	cmn_sales_details	sle_cust_id	3	NULL	YES	int	NULL	NULL	10
18 DataWarehouse	bronze	cmn_sales_details	sle_order_dt	4	NULL	YES	int	NULL	NULL	10
19 DataWarehouse	bronze	cmn_sales_details	sle_ship_dt	5	NULL	YES	int	NULL	NULL	10
20 DataWarehouse	bronze	cmn_sales_details	sle_dscr	6	NULL	YES	text	NULL	NULL	10
21 DataWarehouse	bronze	cmn_sales_details	sle_dscr	7	NULL	YES	text	NULL	NULL	10
22 DataWarehouse	bronze	cmn_sales_details	sle_dscr	8	NULL	YES	text	NULL	NULL	10
23 DataWarehouse	bronze	cmn_sales_details	sle_dscr	9	NULL	YES	text	NULL	NULL	10
24 DataWarehouse	bronze	cmn_sales_details	sle_dscr	10	NULL	YES	text	NULL	NULL	10

Query executed successfully.

To see column metadata for a specific table I can filter using a WHERE clause as depicted in the example below for the ‘dim_customer’ table.

```

5 --Explore All Columns in the Database
6 SELECT *
7   FROM INFORMATION_SCHEMA.COLUMNS
8 WHERE TABLE_NAME = 'dim_customer'

```

Results Messages

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHARACTER_OCTET_LENGTH	Numeric_Precision
1 DataWarehouse	gold	dim_customer	customer_key	1	NULL	YES	bigint	NULL	NULL	19
2 DataWarehouse	gold	dim_customer	customer_id	2	NULL	YES	int	NULL	NULL	10
3 DataWarehouse	gold	dim_customer	customer_number	3	NULL	YES	nvarchar	50	100	NULL
4 DataWarehouse	gold	dim_customer	first_name	4	NULL	YES	nvarchar	50	100	NULL
5 DataWarehouse	gold	dim_customer	last_name	5	NULL	YES	nvarchar	50	100	NULL
6 DataWarehouse	gold	dim_customer	country	6	NULL	YES	nvarchar	50	100	NULL
7 DataWarehouse	gold	dim_customer	marital_status	7	NULL	YES	nvarchar	50	100	NULL
8 DataWarehouse	gold	dim_customer	gender	8	NULL	YES	nvarchar	50	100	NULL
9 DataWarehouse	gold	dim_customer	birthday	9	NULL	YES	date	NULL	NULL	NULL
10 DataWarehouse	gold	dim_customer	create_date	10	NULL	YES	date	NULL	NULL	NULL

d. Dimensions Exploration

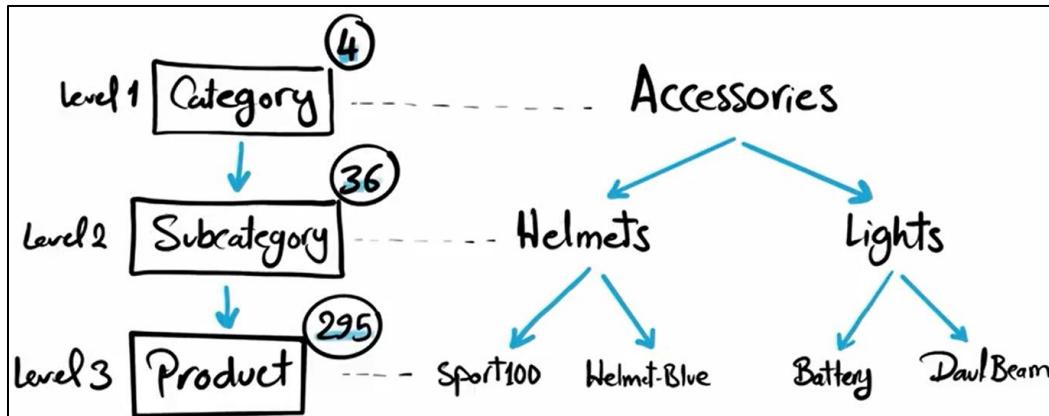
The purpose of dimension exploration is to identify the unique values (or categories) within each dimension so that you can develop an understanding of how data can be grouped or segmented for analysis.

I explored each of the dimensions identified earlier using ‘SELECT DISTINCT’, and prepared the table below that consolidates the results of customer dimensions for my own reference.

Table	Dimension (column name)	Distinct Values
gold.dim_customer	country	n/a Australia Canada France Germany United Kingdom United States
	marital_status	Single Married
	gender	n/a Male Female

A similar exploration was done on the gold.dim_products table to understand the underlying distinct values, their relationships and hierarchy. This was accomplished by including the 'category', 'subcategory', and 'product_name' columns in the SELECT DISTINCT query. Results of the query revealed the various distinct values, the hierarchy, and relationships.

SQLQuery49.s...indsley (61)* -> SQLQuery48.sq...indsley (51))*		
1	2	3
4	0	↑ ↓
90 %		
Results	Messages	
category	subcategory	product_name
1	NULL	HL Mountain Pedal
2	NULL	HL Road Pedal
3	NULL	LL Mountain Pedal
4	NULL	LL Road Pedal
5	NULL	ML Mountain Pedal
6	NULL	ML Road Pedal
7	NULL	Touring Pedal
8	Accessories	Bike Racks
9	Accessories	Bike Stands
10	Accessories	Bottles and Cages
11	Accessories	Bottles and Cages
12	Accessories	Bottles and Cages
13	Accessories	Cleaners
14	Accessories	Fenders
15	Accessories	Helmets
16	Accessories	Helmets
17	Accessories	Helmets
18	Accessories	Hydration Packs



e. Date Exploration

Date exploration involves identifying the time boundaries in the data set (the earliest and latest dates), and understanding the scope of the data and timespan.

The MIN and MAX SQL functions applied to date columns will be used to identify date boundaries, while the DATEDIFF will be used to identify the timespan.

Identifying order date boundaries (first and last orders) and time span in years

SQLQuery49.s...indsley (61)* -> SQLQuery48.sq...indsley (51))*		
1	2	3
4	5	6
90 %		
Results	Messages	
first_order_date	last_order_date	order_range_in_years
2010-12-29	2014-01-28	4

Identifying date boundaries for customer birthdates (oldest and youngest customer)

```

1  SELECT DISTINCT
2      MIN(birthdate) AS oldest_birthdate,
3      DATEDIFF(year, MIN(birthdate), GETDATE()) AS oldest_age,
4      MAX(birthdate) AS youngest_birthdate,
5      DATEDIFF(year, MAX(birthdate), GETDATE()) AS youngest_age
6  FROM gold.dim_customer
7
  
```

The screenshot shows a SQL Server Management Studio window with two tabs: 'Results' and 'Messages'. The 'Results' tab displays a single row of data:

	oldest_birthdate	oldest_age	youngest_birthdate	youngest_age
1	1916-02-10	109	1986-06-25	39

Identifying date boundaries for customer creation (first customer and most recent customer)

```

1  SELECT DISTINCT
2      MIN(create_date) AS first_customer_created_date,
3      MAX(create_date) AS latest_customer_created_date,
4      DATEDIFF(year, MIN(create_date), MAX(create_date)) AS years_creating_customers
5  FROM gold.dim_customer
6
  
```

The screenshot shows a SQL Server Management Studio window with two tabs: 'Results' and 'Messages'. The 'Results' tab displays a single row of data:

	first_customer_created_date	latest_customer_created_date	years_creating_customers
1	2025-10-06	2026-01-27	1

f. Measure Exploration

The purpose of this exploration is to calculate and identify the key metrics of the business at the highest and lowest levels of aggregation.

The SUM, AVG, COUNT, and COUNT(DISTINCT) functions will be used will be used to identify the following at the highest level of aggregation. Each was written separately as an initial pass to identify and pressure test values, then consolidated into a single query using UNION ALL to produce key business metrics into a single view:

```

35  --Generate a Report that Shows all key metrics of the business
36  SELECT 'Total Sales' AS measure_name, SUM(sales_amount) AS measure_value FROM gold.fact_sales
37  UNION ALL
38  SELECT 'Total Orders' AS measure_name, COUNT(DISTINCT(order_number)) AS measure_value FROM gold.fact_sales
39  UNION ALL
40  SELECT 'Total Items Sold' AS measure_name, SUM(quantity) AS measure_value FROM gold.fact_sales
41  UNION ALL
42  SELECT 'Average Selling Price' AS measure_name, AVG(price) AS measure_value FROM gold.fact_sales
43  UNION ALL
44  SELECT 'Total Products' AS measure_name, COUNT(product_key) AS measure_value FROM gold.dim_products
45  UNION ALL
46  SELECT 'Total Customers' AS measure_name, COUNT(customer_key) AS measure_value FROM gold.dim_customer
47  UNION ALL
48  SELECT 'Total Customers That Placed Orders' AS measure_name, COUNT(DISTINCT(customer_key)) AS measure_value FROM gold.fact_sales
  
```

The screenshot shows a SQL Server Management Studio window with two tabs: 'Results' and 'Messages'. The 'Results' tab displays a table of key business metrics:

measure_name	measure_value
1 Total Sales	29356250
2 Total Orders	27659
3 Total Items Sold	60423
4 Average Selling Price	486
5 Total Products	295
6 Total Customers	18484
7 Total Customers That Placed Orders	18484

g. Magnitude Analysis

Magnitude analysis involves drilling further into the data at a lower level. As an example, let's say I wanted to identify the total revenue generated by each customer, the total revenue generated for each category, or the distribution of sold items across countries. Those questions would be answered in this type of analysis.

Total Sales by Customer

Viewing the total revenue for each customer requires building a LEFT JOIN from gold.fact_sales on the gold.dim_customer table with matching on the customer_key from both fields. It also requires obtaining the sum of sales for each customer from the fact_sales table and the customer_key, first_name, and last_name from the dim_customer table....then grouping by customer and ordering by total sales in DESC. Below is the query I wrote to generate the desired view. Results included all 18,484 customers and their total sales.

Total Sales by Customer

```
2   SELECT
3       c.customer_key,
4       c.first_name,
5       c.last_name,
6       SUM(sales_amount) AS total_revenue
7   FROM gold.fact_sales f
8   LEFT JOIN gold.dim_customer| c
9   ON c.customer_key = f.customer_key
10  GROUP BY
11      c.customer_key,
12      c.first_name,
13      c.last_name
14  ORDER BY total_revenue DESC
```

	customer_key	first_name	last_name	total_revenue
1	1302	Nichole	Nara	13294
2	1133	Kaitlyn	Henderson	13294
3	1309	Margaret	He	13268
4	1132	Randall	Dominguez	13265
5	1301	Adriana	Gonzalez	13242
6	1322	Rosa	Hu	13215
7	1125	Brandi	Gill	13195

Total Items Sold by Country

Viewing the total items sold by country involves a similar approach using a LEFT JOIN. The following query was written to provide the desired view of total items sold by country.

Total Items Sold by Country

```
13      c.last_name
14
15      ORDER BY total_revenue DESC
16
17  --What is the distribution of sold items by country
18
19  SELECT
20      c.country,
21      SUM(quantity) AS total_items_sold
22  FROM gold.fact_sales f
23  LEFT JOIN gold.dim_customer c
24  ON c.customer_key = f.customer_key
25  GROUP BY
26      c.country
27  ORDER BY total_items_sold DESC
```

	country	total_items_sold
1	United States	20481
2	Australia	13346
3	Canada	7630
4	United Kingdom	6910
5	Germany	5626
6	France	5559
7	n/a	871

Total Revenue by Product Category

Viewing the total items sold by product category is similar the previous analysis using a LEFT JOIN. The following query was written to provide the desired view of total revenue by product category.

Total Revenue by Product Category

```
--What is the total revenue by product category
SELECT
    p.category,
    SUM(sales_amount) AS total_revenue
FROM gold.fact_sales f
LEFT JOIN gold.dim_products p
ON p.product_key = f.product_key
GROUP BY
    p.category
ORDER BY total_revenue DESC
```

	category	total_revenue
1	Bikes	28316272
2	Accessories	700262
3	Clothing	339716

h. Ranking Analysis

Ranking analysis involves ordering the values of dimensions by measure to identify top N and bottom N within the domain of interest.

Example:

- Top 5 Products by Revenue Generation
- Rank Countries by Total Sales and view the top 3
- Bottom 3 Customers by Total Orders

Which Top 5 Products Generate the Highest / Lowest Revenue

In this case, the measure is revenue which is captured as 'sales_amount' in the gold.fact_sales' table. The dimension is products which is captured as the 'product_name' in the gold.dim_products' table. Both tables are related as the 'product_key' is found in both tables. To limit the results of the query only requires including 'TOP 5' immediately after 'SELECT' in the query.

In a similar manner, identifying the top 5 products with the lowest sales requires sorting in ascending order by total_revenue or just leaving it blank as SQL will order in ascending order by default.

Identifying Top 5 Products by Total Sales / Revenue

```
--Which 5 products generate the highest revenue?
SELECT TOP 5
    p.product_name,
    SUM(sales_amount) AS total_revenue
FROM gold.fact_sales f
LEFT JOIN gold.dim_products p
ON p.product_key = f.product_key
GROUP BY
    p.product_name
ORDER BY total_revenue DESC
```

	product_name	total_revenue
1	Mountain-200 Black- 46	1373454
2	Mountain-200 Black- 42	1363128
3	Mountain-200 Silver- 38	1339394
4	Mountain-200 Silver- 46	1301029
5	Mountain-200 Black- 38	1294854

Assigning Rank identifiers to Enable More Complex Analysis

Assigning a rank ID to each record in the results enables more complex analysis. Doing so involves using WINDOW functions and generating a subquery.

```
61  --Assigning Rank Identifiers to View Products by Specific Rank Range
62  SELECT *
63  FROM (
64    SELECT
65      p.product_name,
66      SUM(sales_amount) AS total_revenue,
67      ROW_NUMBER() OVER (ORDER BY SUM(sales_amount) DESC) AS rank_products
68
69    FROM gold.fact_sales f
70    LEFT JOIN gold.dim_products p
71    ON p.product_key = f.product_key
72    GROUP BY p.product_name
73  )t
74  WHERE rank_products <= 5
```

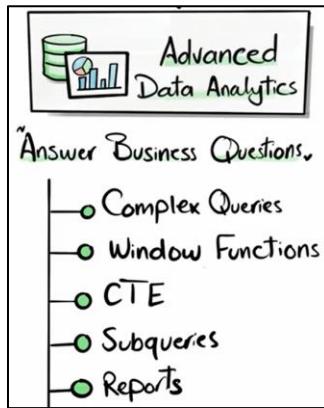
90 % ▾ 46 ▲ 0 ↑ ↓

	product_name	total_revenue	rank_products
1	Mountain-200 Black- 46	1373454	1
2	Mountain-200 Black- 42	1363128	2
3	Mountain-200 Silver- 38	1339394	3
4	Mountain-200 Silver- 46	1301029	4
5	Mountain-200 Black- 38	1294854	5

10. SQL Advanced Analytics

a. Overview

The purpose of this stage of the project is to use and understand SQL syntax that enables answering advanced business questions. I will leverage the SQL skills I've learned throughout up to this point to further my knowledge of analytics in SQL. I learned to apply the following in this portion of the project:



b. Change Over Time Analysis

This is a technique for analyzing how a measure evolves over time and helps track trends and identify seasonality in data. Doing so involves aggregating a measure by a date dimension to understand how they change over time.

For example:

- Total Sales By Year
- Average Cost by Month

Analyzing Sales Performance Over Time

Bringing together the order dates, sum of sales, customer count, and item quantities ordered into a single query allows me to see sales performance over time by year. The results reveal that sales increased from 2010 to 2013, then had a sharp drop in 2014. Additionally, you can see that total customers and total products sold took a nose dive between 2013 and 2014.

Sales Performance Over Time

```
SQLQuery51...indsley (60)* -> X SQLQuery50.sq...indsley (62))" go!
1 | --Change Over Time Analysis
2 | <-- SELECT
3 |   YEAR(order_date) AS order_year,
4 |   SUM(sales_amount) AS total_sales,
5 |   COUNT(DISTINCT(customer_key)) AS total_customers,
6 |   SUM(quantity) AS total_quantity
7 | FROM gold.fact_sales
8 | WHERE order_date IS NOT NULL
9 | GROUP BY YEAR(order_date)
10| ORDER BY YEAR(order_date)
```

	order_year	total_sales	total_customers	total_quantity
1	2010	43419	14	14
2	2011	7075088	2216	2216
3	2012	5842231	3255	3397
4	2013	16344878	17427	52807
5	2014	45642	834	1970

The sales performance data can be further broken-down by month to understand the seasonality of sales and provide greater clarity / insight on the data. By adjusting the query to include the month and the year, it becomes obvious that we were wrong to assume that sales took a nose-dive in 2014 as the data only captures the first month of 2014 rather than a full year. So....it would only make sense to compare the sales of the first month of 2014 with the sales of the same month for previous years.

Additionally, it becomes apparent that December and February are the best and worst sales months respectively. This makes sense as December tends to be a month in which purchasing activity increases in consumer goods due to Christmas and other holidays during this period. Sales typically decrease sharply after Christmas...so...it makes sense that they are lowest in February.

A screenshot of a SQL query execution interface. The top half shows the SQL code for a "Change Over Time Analysis". The bottom half shows the results grid with 12 rows of data. The last row, representing the 12th month, is highlighted with a red border.

```

1 --Change Over Time Analysis
2
3 SELECT
4     MONTH(order_date) AS order_month
5     ,SUM(sales_amount) AS total_sales
6     ,COUNT(DISTINCT(customer_key)) AS total_customers
7     ,SUM(quantity) AS total_quantity
8
9 FROM gold_fact_sales
10 WHERE order_date IS NOT NULL
11 GROUP BY MONTH(order_date)
12 ORDER BY MONTH(order_date)

```

	order_month	total_sales	total_customers	total_quantity
1	1	1868558	1818	4043
2	2	1744517	1765	3858
3	3	1908375	1982	4449
4	4	1948226	1916	4355
5	5	2204969	2074	4781
6	6	2935883	2430	5573
7	7	2412838	2154	5107
8	8	2684313	2312	5335
9	9	2536520	2210	5070
10	10	2916550	2533	5838
11	11	2979113	2500	5756
12	12	3211396	2656	6239

c. Cumulative Analysis

Cumulative analysis involves aggregating data progressively over time to understand how the values grow or decline. Aggregate WINDOW functions are used for this in SQL

Example:

- Running total sales
- Moving average sales

Running Total Sales

Creating the following query with subquery generates a view of total sales for each month and a view of the running total sales by month.

Running Total Sales (cont'd)

The image below shows the query and the results that show cumulative running total sales. The total sales for Dec 2010 (2010-12-01) was 43,419 and the running total sales for that month is the same as the total sales generated as it is the first month in which sales occurred. The following month, Jan 2011 (2011-01-01) has a monthly total sales of 469,795 and a running total sales of 513,214. This running total sales for Jan 2011 is the sum of sales for Dec 2010 and Jan 2011. The running total sum calculation continues for all remaining rows to provide visibility into the change in total sales over time.

```

4   SELECT
5     order_date,
6     total_sales,
7     SUM(total_sales) OVER (ORDER BY order_date) AS running_total_sales
8   FROM
9     (
10    SELECT
11      DATETRUNC(month, order_date) AS order_date,
12      SUM(sales_amount) AS total_sales
13    FROM gold.fact_sales
14    WHERE order_date IS NOT NULL
15    GROUP BY DATETRUNC(month, order_date)
16  ) t

```

	order_date	total_sales	running_total_sales
1	2010-12-01	43419	43419
2	2011-01-01	469795	513214
3	2011-02-01	466307	979521
4	2011-03-01	485165	1464686
5	2011-04-01	502042	1966728
6	2011-05-01	561647	2528375
7	2011-06-01	737793	3266168
8	2011-07-01	596710	3862878
9	2011-08-01	614516	4477394
10	2011-09-01	603047	5080441
11	2011-10-01	708164	5788605

The query used to view running total sales provides continuous running total sales from the first month and year of sales through the current month and year of sales. This query can be refined by using PARTITION BY to view the running total by month for each year. Doing so will reset the cumulative sum of sales each at the beginning of each year and only run through the 12 months of that year. So, analysis can be done to compare cumulative sales over a 12 month period for each year. The image below depicts the query adjustment and the cumulative results produced through the PARTITION BY.

Cumulative Sales Partitioned by Year

```

4   SELECT
5     order_date,
6     total_sales,
7     SUM(total_sales) OVER (PARTITION BY order_date ORDER BY order_date) AS running_total_sales
8   FROM
9     (
10    SELECT
11      DATETRUNC(month, order_date) AS order_date,
12      SUM(sales_amount) AS total_sales
13    FROM gold.fact_sales
14    WHERE order_date IS NOT NULL
15    GROUP BY DATETRUNC(month, order_date)
16  ) t

```

	order_date	total_sales	running_total_sales
1	2010-12-01	43419	43419
2	2011-01-01	469795	469795
3	2011-02-01	466307	466307
4	2011-03-01	485165	485165
5	2011-04-01	502042	502042
6	2011-05-01	561647	561647
7	2011-06-01	737793	737793
8	2011-07-01	596710	596710
9	2011-08-01	614516	614516
10	2011-09-01	603047	603047
11	2011-10-01	708164	708164
12	2011-11-01	660507	660507
13	2011-12-01	669395	669395
14	2012-01-01	495363	495363
15	2012-02-01	506992	506992

Moving Average of Price

Adding the following to the query provides a view of the moving average price over time partitioned by year. The newly added column 'avg_price' must also be declared / created in the subquery for this to work.

```
AVG(avg_price) OVER (PARTITION BY order_date ORDER BY order_date) AS moving_average_price'
```

```

19   SELECT
20     order_date,
21     total_sales,
22     SUM(total_sales) OVER (PARTITION BY order_date ORDER BY order_date) AS running_total_sales,
23     AVG(avg_price) OVER (PARTITION BY order_date ORDER BY order_date) AS moving_average_price
24   (
25     SELECT
26       DATETRUNC(YEAR, order_date) AS order_date,
27       SUM(sales_amount) AS total_sales,
28       AVG(price) AS avg_price
29     FROM gold_fact_sales
30     WHERE order_date IS NOT NULL
31     GROUP BY DATETRUNC(YEAR, order_date)
32   )t
33

```

order_date	total_sales	running_total_sales	moving_average_price
2010-01-01	43419	43419	3101
2011-01-01	7075088	7075088	3192
2012-01-01	5842231	5842231	1719
2013-01-01	16344878	16344878	309
2014-01-01	45642	45642	23

d. Performance Analysis

Performance analysis is the comparison of current values to a target value. This is helpful when evaluating the performance of a specific category / dimension. The formula here is:

Current[Measure] – Target[Measure]

Examples:

- Current Sales – Average Sales
- Current Year sales – Previous Year Sales ← Year over Year Analysis
- Current Sales – Lowest Sales

Window Functions such as MAX, MIN, LEAD, and LAG are typically used here in conjunction with SUM, and AVG. Additionally, CASE WHEN is also needed for this.

```

58   SELECT *
59   FROM yearly_product_sales ps
60
61   SELECT
62     YEAR(order_date) AS order_year,
63     p.product_name,
64     SUM(sales_amount) AS current_sales
65   FROM gold_fact_sales f
66   LEFT JOIN dim_products p
67   ON f.product_key = p.product_key
68   WHERE order_date IS NOT NULL
69   GROUP BY YEAR(order_date),
70           p.product_name
71
72   SELECT
73     order_year,
74     product_name,
75     current_sales,
76     AVG(current_sales) OVER(PARTITION BY product_name) AS avg_sales,
77     (current_sales - AVG(current_sales)) OVER(PARTITION BY product_name) AS diff_avg,
78     CASE WHEN current_sales - AVG(current_sales) OVER(PARTITION BY product_name) > 0 THEN 'Above Avg'
79     ELSE 'Below Avg'
80     END AS avg_change,
81     LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) previous_year_sales,
82     current_sales - LAG(current_sales) OVER(PARTITION BY product_name ORDER BY order_year) AS diff_previous_year_sales,
83     CASE WHEN current_sales - LAG(current_sales) OVER(PARTITION BY product_name ORDER BY order_year) > 0 THEN 'Increasing'
84     ELSE 'Decreasing'
85     END AS py_change
86   END ProductSales
87   FROM yearly_product_sales
88   ORDER BY product_name, order_year
89

```

order_year	product_name	current_sales	avg_sales	diff_avg	avg_change	previous_year_sales	diff_previous_year_sales	py_change
1 2012	All-Purpose Bike Stand	159	13197	-10308	Below Avg	NULL	NULL	No Change
2 2013	All-Purpose Bike Stand	37683	13197	24406	Above Avg	159	37524	Increasing
3 2014	All-Purpose Bike Stand	1749	13197	-11448	Below Avg	37683	-35934	Decreasing
4 2012	AWC Logo Cap	72	6570	-6498	Below Avg	NULL	NULL	No Change
5 2013	AWC Logo Cap	18891	6570	12321	Above Avg	72	18819	Increasing
6 2014	AWC Logo Cap	747	6570	-5823	Below Avg	18891	-18144	Decreasing
7 2013	Bike Wash - Dissolver	6960	3636	3324	Above Avg	NULL	NULL	No Change
8 2014	Bike Wash - Dissolver	312	3636	-3324	Below Avg	6960	-6648	Decreasing

e. Part-to-Whole Analysis

This type of analysis is used to analyze how an individual part is performing compared to the overall. For example: What is the total sales for the Bikes category as a percentage of all product categories? Think...pie chart. This is helpful for understanding the contribution of each category to the whole.

The formula here is simple:

([Measure] / Total[Measure]) * 100 by Dimension

The screenshot below shows how I wrote a query that provides total sales for each product category, the overall sales for all product categories, and the percentage of sales that each product category contributes to total sales.

A screenshot of SQL Server Management Studio (SSMS) showing a T-SQL query and its results. The code uses a WITH clause to define a CTE named 'category_sales' which calculates total sales per category. It then selects from this CTE along with an overall sales sum to calculate the percentage of total sales for each category. A red arrow points from the 'percentage_of_total' column in the results table to the formula in the query.

```
74    WITH category_sales AS (
75        SELECT
76            category,
77            SUM(sales_amount) total_sales
78        FROM gold.fact_sales f
79        LEFT JOIN gold.dim_products p
80        ON f.product_key = p.product_key
81        GROUP BY category
82    )
83
84    SELECT
85        category,
86        total_sales,
87        SUM(total_sales) OVER () overall_sales,
88        CONCAT(ROUND((CAST(total_sales AS FLOAT) / SUM(total_sales) OVER ()) * 100,2), '%') AS percentage_of_total
89    FROM category_sales
90    ORDER BY total_sales DESC
91
92
```

category	total_sales	overall_sales	percentage_of_total
1 Bikes	28316272	29356250	96.46%
2 Accessories	700262	29356250	2.39%
3 Clothing	339716	29356250	1.16%

f. Data Segmentation

Data segmentation groups data into defined ranges to reveal relationships between measures. For example, total products by sales range or customers by age. For dimension items, you first create a measure (such as total products) and then segment it by a value range, effectively turning the measure into a dimension. I'll share a more detailed example built on the gold-layer data next.

Segmenting products into cost ranges and determining the quantity of products by segment

In this case I wrote the base query to categorize products into cost range categories using a CASE WHEN. I then built a CTE using WITH to generate a query on the base query to obtain the total number of products for each cost range category.

Segmenting Costs into Cost Range Categories

A screenshot of SSMS showing a T-SQL query for segmenting products into cost ranges. The code uses a WITH clause to define a CTE named 'product_segments'. It includes a CASE statement to categorize products into four cost ranges: 'Below 100', '100 - 500', '501 - 1000', and 'Above 1000'. It then counts the total number of products for each range. A red arrow points from the 'cost_range' column in the results table to the CASE statement in the query.

```
--Segmenting products into cost ranges and determining the quantity of products by segment
1
2
3    WITH product_segments AS (
4        SELECT
5            product_key,
6            product_name,
7            cost,
8            CASE WHEN cost < 100 THEN 'Below 100'
9                WHEN cost between 100 AND 500 THEN '100 - 500'
10               WHEN cost between 501 AND 1000 THEN '501 - 1000'
11               ELSE 'Above 1000'
12           END cost_range
13
14        FROM gold.dim_products)
15
16        SELECT
17            cost_range,
18            COUNT(product_key) AS total_products
19        FROM product_segments
20        GROUP BY cost_range
21        ORDER BY total_products DESC
```

cost_range	total_products
1 Below 100	110
2 100 - 500	101
3 501 - 1000	45
4 Above 1000	39

f. Data Segmentation (cont'd)

Segmenting Customers by Lifespan and Total Spending

In this example, I am segmenting customers into three group according to the number of months between their first and most recent order, and the total sales from all orders. The segmentation is:

- **VIP:** Custs with ≥ 12 months of history and spending more than 5,000
- **Regular:** Customeres with ≥ 12 months of history but spending $\leq 5,000$
- **New:** Customers with a lifespan less than 12 months

The first step in accomplishing this is a CTE that has the following (see image below for initial results):

- A left join on gold.fact_sales and gold.dim_customer tables on 'customer_key'
- Determining the customer lifespan using DATEDIFF, MIN, and MAX on 'order_date'
- Capturing customer sum of sales
- Assigning segmentation to VIP, Regular, or New using CASE WHEN on the lifespan criteria and the total sales criteria defined above.

Query to and Results for Initial Segmentation

```

gold_layer_exp_lindsay (51)
29   ✓ WITH customer_spending AS (
30     SELECT
31       c.customer_key,
32       SUM(f.sales_amount) AS total_spending,
33       MIN(order_date) AS first_order,
34       MAX(order_date) AS last_order,
35       DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS lifespan
36     FROM gold.fact_sales f
37     LEFT JOIN gold.dim_customer c
38     ON f.customer_key = c.customer_key
39     GROUP BY c.customer_key
40   )
41
42   SELECT
43     customer_key,
44     total_spending,
45     lifespan,
46     CASE
47       WHEN lifespan <= 12 AND total_spending > 5000 THEN 'VIP'
48       WHEN lifespan >= 12 AND total_spending <= 5000 THEN 'Regular'
49       ELSE 'New'
50     END customer_segment
51   FROM customer_spending
52 
```

customer_key	total_spending	lifespan	customer_segment
5621	5895	35	VIP
5632	4471	25	Regular
451	8173	24	VIP
4696	8324	21	VIP
1447	5944	20	VIP
15107	5768	20	VIP
16435	36	0	New
9202	134	0	New
10530	30	0	New
4957	5	0	New
13518	87	0	New

The next step is to find the total number of customers for each category. To accomplish this, I revised the CTE by adding a subquery, adding 'COUNT(customer_key) AS total_customers' and grouping by customer_segment to capture total customers by customer segment. This allows me to see the total quantity of customers by the categories I defined in the CTE.

```

29   ✓ WITH customer_spending AS (
30     SELECT
31       c.customer_key,
32       SUM(f.sales_amount) AS total_spending,
33       MIN(order_date) AS first_order,
34       MAX(order_date) AS last_order,
35       DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS lifespan
36     FROM gold.fact_sales f
37     LEFT JOIN gold.dim_customer c
38     ON f.customer_key = c.customer_key
39     GROUP BY c.customer_key
40   )
41
42   SELECT
43     customer_segment,
44     COUNT(customer_key) AS total_customers
45   FROM (
46     SELECT
47       customer_key,
48       CASE WHEN lifespan >= 12 AND total_spending > 5000 THEN 'VIP'
49         WHEN lifespan >= 12 AND total_spending <= 5000 THEN 'Regular'
50         ELSE 'New'
51       END customer_segment
52     FROM customer_spending
53   )t
54   GROUP BY customer_segment
55   ORDER BY total_customers DESC
56 
```

customer_segment	total_customers
New	14631
Regular	2198
VIP	1655

g. Build Customers Report

The purpose of this report is to consolidate key customer metrics and behaviors within the data sets in the data warehouse.

It should:

- Gather essential fields such as names, ages, and transaction details.
- Segments customers into categories (VIP, Regular, New) and age groups.
- Aggregate customer-level metrics:
 - total orders
 - total sales
 - total quantity purchased
 - total products
 - lifespan (in months)
- Calculate valuable KPIs:
 - recency (months since last order)
 - average order value
 - average monthly spend

Creating this report requires a well-crafted approach for developing complexed queries. The first thing to do here is to build the base data by selecting the data from the database. In this case, I will start with the fact table, then join the dimension tables and filter only on the columns needed. This base data will serve as the foundation for the remainder of the effort. I will build starting with a CTE since there are multiple steps in the overall effort (deriving new columns, executing some transformations).

Retrieving Core Columns from the Tables:

Here I am retrieving the core columns, conducting quick eval to determine whether or not any additional transformations are required to prepare them for aggregation.

Retrieve Core Columns

```
/*
1) Build the Base Query: Retrieve core columns from tables
*/
SELECT
    f.order_number,
    f.product_key,
    f.order_date,
    f.sales_amount,
    f.quantity,
    c.customer_key,
    c.customer_number,
    c.first_name,
    c.last_name,
    c.birthdate
FROM gold_fact_sales f
LEFT JOIN gold_dim_customer c
ON c.customer_key = f.customer_key
WHERE order_date IS NOT NULL
```

The screenshot shows a SQL query in a code editor. The code is a SELECT statement that joins the gold_fact_sales table with the gold_dim_customer table on their customer_key columns. It retrieves columns such as order_number, product_key, order_date, sales_amount, quantity, customer_key, customer_number, first_name, last_name, and birthdate. A WHERE clause filters out rows where the order_date is null. The code is annotated with a comment at the top indicating it's step 1 to build the base query for retrieving core columns from tables.

order_number	product_key	order_date	sales_amount	quantity	customer_key	customer_number	first_name	last_name	birthdate	
1	SO43697	20	2010-12-29	3578	1	10769	AW00021760	Cole	Watson	1952-02-19
2	SO43698	9	2010-12-29	3400	1	17390	AW00028389	Rachael	Martinez	1970-06-17
3	SO43699	9	2010-12-29	3400	1	14864	AW00025863	Sydney	Wright	1952-06-01
4	SO43700	41	2010-12-29	699	1	3502	AW00014501	Ruben	Prasad	1943-11-10
5	SO43701	9	2010-12-29	3400	1	4	AW00011003	Christy	Zhu	1973-08-14
6	SO43702	16	2010-12-30	3578	1	16646	AW00027645	Colin	Anand	1972-08-14
7	SO43703	20	2010-12-30	3578	1	5625	AW00016524	Albert	Alvarez	1983-07-24
8	SO43704	6	2010-12-30	3375	1	6	AW00011005	Julio	Ruiz	1976-08-01
9	SO43705	7	2010-12-30	3400	1	12	AW00011011	Curtis	Lu	1969-05-03
10	SO43706	17	2010-12-31	3578	1	16622	AW00027621	Edward	Brown	1975-01-23
11	SO43707	17	2010-12-31	3578	1	16617	AW00027616	Emma	Brown	1971-06-10

Upon evaluating the core columns, I determine that the first and last names should be concatenated so they are in a single column that reads 'First Name Last Name'.

I also need to capture the age for each customer. I use DATEDIFF for this. I also add syntax to begin creating the CTE.

Name and Age Transformations (with CTE added but not executed)

```

gold_layer_exp...lindsay (51)
23   WITH base_query AS (
24     /* */
25     -- 1) Build the Base Query: Retrieve core columns from tables
26     --
27   SELECT
28     f.order_number,
29     f.product_key,
30     f.order_date,
31     f.sales_amount,
32     f.quantity,
33     c.customer_key,
34     c.customer_number,
35     CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
36     DATEDIFF(year, c.birthdate, GETDATE()) age
37   FROM gold_fact_sales f
38   LEFT JOIN gold_dim_customer c
39   ON c.customer_key = f.customer_key

```

	order_number	product_key	order_date	sales_amount	quantity	customer_key	customer_number	customer_name	age
1	S043697	20	2010-12-29	3578	1	10769	AW00021768	ColeWatson	73
2	S043698	9	2010-12-29	3400	1	17390	AW00028389	RachaelMartinez	55
3	S043699	9	2010-12-29	3400	1	14864	AW00025963	SydneyWright	73
4	S043700	41	2010-12-29	699	1	3502	AW00014501	RubenPrasad	82
5	S043701	9	2010-12-29	3400	1	4	AW00011003	ChristyZhu	52
6	S043702	16	2010-12-30	3578	1	16646	AW00027645	ColinAnand	53
7	S043703	20	2010-12-30	3578	1	5625	AW00016624	AlbertAlvarez	42
8	S043704	6	2010-12-30	3375	1	6	AW00011005	JulioRuiz	49
9	S043705	7	2010-12-30	3400	1	12	AW00011011	CurtisLu	56
10	S043706	17	2010-12-31	3578	1	16622	AW00027621	EdwardBrown	50
11	S043707	17	2010-12-31	3578	1	16617	AW00027616	EmmaBrown	54
12	S043708	44	2010-12-31	699	1	9043	AW00020042	BradDeng	46
13	S043709	18	2010-12-31	3578	1	5352	AW00016351	MarthaXu	65
14	S043710	19	2010-12-31	3578	1	5518	AW00016517	KatrinaRaji	44
15	S043711	19	2011-01-01	3578	1	16607	AW00027606	CourtneyEdwar...	63

With the intermediate results prepared in the CTE, I am now able to begin aggregating the results to needed for the required report.

The first aggregation identifies the total orders, the total sales, the total quantity of products ordered, and the different products ordered. It also sets the foundations for identifying customers by lifespan according to age by adding a DATEDIFF on MIN and MAX order_date.

Initial Aggregations

```

gold_layer_exp...lindsay (51)
24   WITH base_query AS (
25     /* */
26     -- 1) Build the Base Query: Retrieve core columns from tables
27     --
28   SELECT
29     f.order_number,
30     f.product_key,
31     f.order_date,
32     f.sales_amount,
33     f.quantity,
34     c.customer_key,
35     c.customer_number,
36     CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
37     DATEDIFF(year, c.birthdate, GETDATE()) age
38   FROM gold_fact_sales f
39   LEFT JOIN gold_dim_customer c
40   ON f.customer_key = c.customer_key
41   WHERE order_date IS NOT NULL
42
43   SELECT
44     customer_key,
45     customer_number,
46     customer_name,
47     COUNT(DISTINCT order_number) AS total_orders,
48     SUM(quantity) AS total_quantity,
49     COUNT(DISTINCT product_key) AS total_products,
50     MIN(order_date) AS min_order_date,
51     MAX(order_date) AS max_order_date,
52     DATEDIFF(year, min_order_date, max_order_date) AS lifespan
53   GROUP BY
54     customer_key,
55     customer_number,
56     customer_name,
57     age
58

```

	customer_key	customer_number	customer_name	age	total_orders	total_sales	total_quantity	total_products	last_order_date	lifespan
1	AW00011000	JonYang	54	3	8249	8	8	28	2013-06-03	28
2	AW00011001	EugeneWang	49	3	6384	11	10	35	2013-12-10	35
3	AW00011002	PeterTorres	54	3	8114	4	4	35	2013-06-03	35
4	AW00011003	OliviaZhu	52	3	8139	9	9	29	2013-09-10	29
5	AW00011004	ElizabethJohnson	46	3	8196	6	6	28	2013-05-01	28
6	AW00011005	JuliRuiz	49	3	8121	6	6	29	2013-05-02	29
7	AW00011006	JanetNguyen	49	3	8119	5	5	28	2013-05-14	28

At this point, the query is getting fairly large. So, descriptions of its evolution will follow. The final script will be committed to Git for reference.

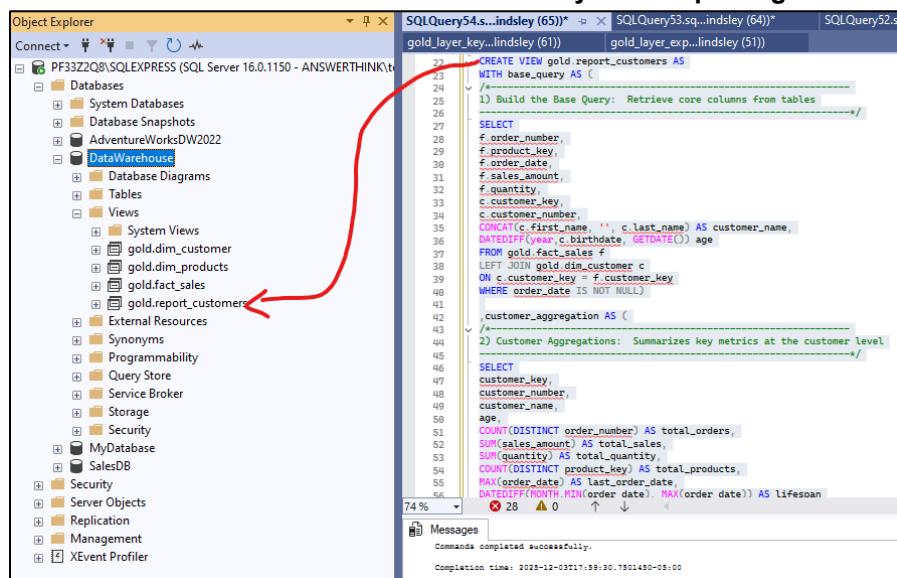
The aggregations are now ready to prepare the final result with the customer segmentation by age and customer category (VIP, Regular, and New) as well as required KPIs.

At this point I built in the following by creating an additional CTE:

- Customer Category (VIP, Regular, and New)
- Customer Age Category
- Recency (months since last order)
- Average order value
- Average monthly spend

With the query complete, I then created a view in the Gold layer to enable (fictional) data analysts to access the data for analysis and reporting.

Creation of Customer View for Analysis & Reporting



```

CREATE VIEW gold.report_customers AS
WITH base_query AS (
    /* 1) Build the Base Query: Retrieve core columns from tables */
    SELECT
        f.order_number,
        f.product_key,
        f.order_date,
        f.sales_amount,
        f.quantity,
        c.customer_key,
        c.customer_number,
        CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
        DATEDIFF(YEAR, c.birthdate, GETDATE()) AS age
    FROM gold_fact_sales f
    LEFT JOIN gold.dim_customer c
    ON c.customer_key = f.customer_key
    WHERE order_date IS NOT NULL
),
customer_aggregation AS (
    /* 2) Customer Aggregations: Summarizes key metrics at the customer level */
    SELECT
        customer_key,
        customer_number,
        customer_name,
        age,
        COUNT(DISTINCT order_number) AS total_orders,
        SUM(sales_amount) AS total_sales,
        SUM(quantity) AS total_quantity,
        COUNT(DISTINCT product_key) AS total_products,
        MAX(order_date) AS last_order_date,
        DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS lifespan
)
SELECT
    *
FROM base_query
    
```

Messages
Commands completed successfully.
Completion time: 2023-12-03T17:59:30.7501450-05:00

h. Build Products report

The purpose of this report is to consolidate key product metrics and behaviors.

It should:

1. Gather essential fields such as product name, category, subcategory, and cost.
2. Segment products by revenue to identify High-Performers, Mid-Range, or Low-Performers.
3. Aggregate product-level metrics:
 - total orders
 - total sales
 - total quantity sold
 - total customers (unique)
 - lifespan (in months)
4. Calculate valuable KPIs:
 - recency (months since last sale)
 - average order revenue (AOR)
 - average monthly revenue

The SQL query was prepared using a CTE and CREATE VIEW. It was added to the Gold layer views.

This was a tough one as I had to build it on my own rather than following along step-by-step. It worked for the most part, but I continued to run into errors that I had trouble resolving. The instructor did provide an example of how it should look. So, after several attempts...and getting it 80% correct, I referred to his example to finalize my query. It turns out that portions of my query were calling columns that had not been declared in preceding CTE queries. Was able to correct for those and get the desired output.

The below screenshot depicts the report location, the query (only a portion is visible) and the output (a portion of the output only).

'CREATE VIEW' Query for Products Report in Gold Layer

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure, including the 'DataWarehouse' schema which contains the 'gold.report_products' view. A red arrow points from the 'gold.report_products' entry in the object list to the query results window. The central pane shows the T-SQL code for creating the view, which includes a Common Table Expression (CTE) named 'base_query' and a main 'CREATE VIEW' statement. The right pane shows the results of the query, displaying a table with 6 rows of product data. The columns include: product_key, product_name, category, subcategory, cost, last_sale_date, recency_in_months, product_segment, total_orders, total_sales, total_quantity, avg_order_revenue, and avg_monthly_revenue.

product_key	product_name	category	subcategory	cost	last_sale_date	recency_in_months	product_segment	total_orders	total_sales	total_quantity	avg_order_revenue	avg_monthly_revenue	
1	105	Road Bike Cage	Accessories	Bottles and Cages	3	2014-01-25	143	Mid-Range	1711	15399	1711	9	1184
2	114	Mountain-500 Silver-48	Bikes	Mountain Bikes	308	2013-12-27	144	Mid-Range	50	28250	50	565	2568
3	144	Touring-3000 Blue-44	Bikes	Touring Bikes	461	2013-12-28	144	Mid-Range	53	39326	53	742	3575
4	285	LL Road Tire	Accessories	Tires and Tubes	8	2014-01-28	143	Mid-Range	1043	21903	1043	21	1684
5	155	Touring-2000 Blue-50	Bikes	Touring Bikes	755	2013-12-27	144	High-Performer	106	128790	106	1215	10732
6	159	Touring-1000 Blue-50	Bikes	Touring Bikes	1482	2013-12-28	144	High-Performer	150	357600	150	2384	29800

i. Document Work in Git

All sql code developed in the Exploratory and Advanced SQL analytics was loaded to my Git repo. It includes all scripts for exploratory analysis, advanced analysis, and the reports / views created for Customers and Products in the Gold Layer.

The screenshot shows a GitHub repository interface for the 'sql-data-warehouse-project / scripts / gold /' directory. The left sidebar shows the repository navigation bar with options like Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area displays the file structure under 'gold'. A yellow box highlights the 'data_exploration_and_analysis' folder, which contains several SQL files: 01_database_exploration.sql, 02_dimensions_exploration.sql, 03_date_range_exploration.sql, 04_measures_exploration.sql, 05_magnitude_analysis.sql, 06_ranking_analysis.sql, 07_change_over_time_analysis.sql, 08_cumulative_analysis.sql, 09_performance_analysis.sql, 10_data_segmentation.sql, and 11_part_to_whole_analysis.sql. Below this folder is a 'placeholder' folder containing three files: gold_layer_explore_schema_and_table_metadata.sql, gold_layer_key_business_metrics_report.sql, and gold_layer_report_products.sql. To the right of the file tree is a list of recent commits:

Name	Last commit message	Last commit date
..		
data_exploration_and_analysis	Add files via upload	4 minutes ago
gold_layer_reports	Delete scripts/gold/gold_layer_reports/placeholder	7 minutes ago
dd_gold.sql	Update dd_gold.sql	3 days ago

END