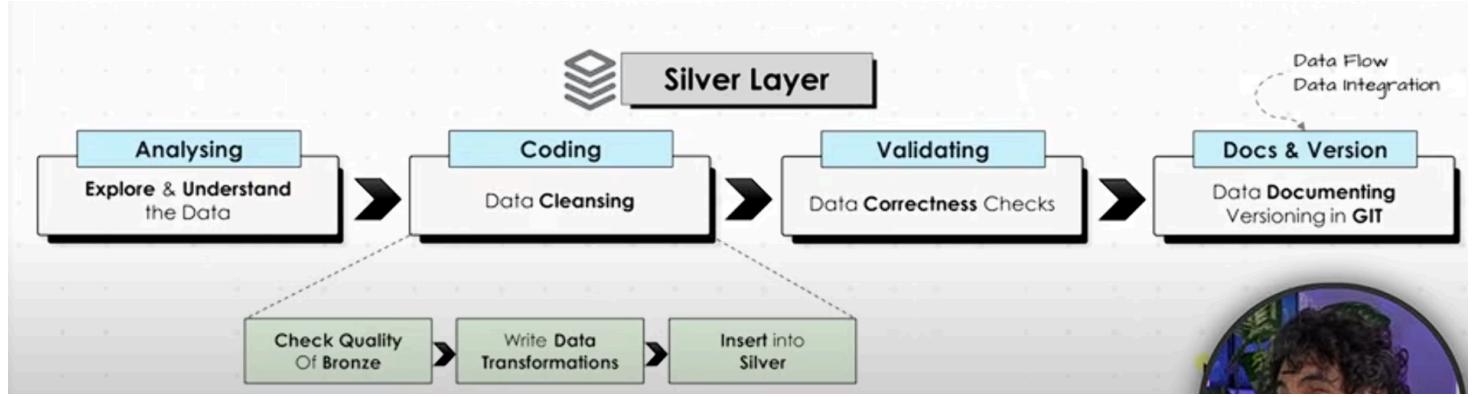


Building SILVER LAYER



Build Silver Layer

Explore & Understand The Data

A screenshot showing a task list and a SQL query editor.

Task List:

- Build Silver Layer: Analysing: explore & understand data
- Build Silver Layer: Coding: Data Cleansing
- Build Silver Layer: Validating: Data Correctness Checks
- Build Silver Layer: Documenting & Versioning in GIT
- Build Silver Layer: Commit Code in Git Repo

SQL Query Editor:

```

SELECT TOP (1000) [cst_id]
      ,[cst_key]
      ,[cst_firstname]
      ,[cst_lastname]
      ,[cst_marital_status]
      ,[cst_gndr]
      ,[cst_create_date]
  FROM [DataWarehouse].[bronze].[crm_cust_info]
  
```

Document & Visualize What You Understand from Data

A screenshot showing multiple SQL query windows analyzing data from the DataWarehouse database.

SQL Query 1:

```

SELECT TOP (1000) [cst_id]
      ,[cst_key]
      ,[cst_firstname]
      ,[cst_lastname]
      ,[cst_marital_status]
      ,[cst_gndr]
      ,[cst_create_date]
  FROM [DataWarehouse].[bronze].[crm_cust_info]
  
```

SQL Query 2:

```

SELECT TOP (1000) [prd_id]
      ,[prd_key]
      ,[prd_nm]
      ,[prd_cost]
      ,[prd_line]
      ,[prd_start_dt]
      ,[prd_end_dt]
  FROM [DataWarehouse].[bronze].[crm_prd_info]
  
```

SQL Query 3:

```

SELECT TOP (1000) * 
  FROM [DataWarehouse].[bronze].[crm_sales_details]
  
```

SQL Query 4:

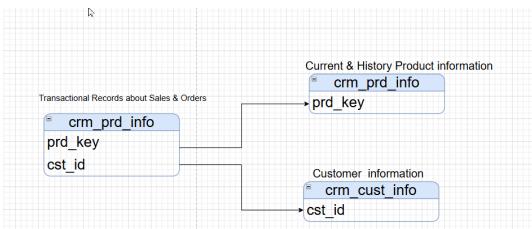
```

SELECT TOP (1000) * 
  FROM [DataWarehouse].[bronze].[erp_cust_az12]
  
```

Result Table:

cid	bdate	gen
1	1971-10-06	Male
2	1976-05-10	Male
3	1971-02-09	Male

We continue analyzing table by table and put them all in the same query.
We look at each table individually and see how we can join them.



```

SELECT TOP 1000 * FROM [DataWarehouse].[bronze].[crm_prd_info]
SELECT TOP 1000 * FROM [DataWarehouse].[bronze].[crm_sales_details]
SELECT TOP 1000 * FROM [DataWarehouse].[bronze].[erp_cust_az12]
SELECT TOP 1000 * FROM [DataWarehouse].[bronze].[crm_cust_info]

```

Little by little, we are joining the tables.

105 % No se encontraron problemas.

	cid	bdate	gen
1	NASAW00011000	1971-10-06	Male
2	NASAW00011001	1976-05-10	Male
3	NASAW00011002	1971-02-09	Male
4	NASAW00011003	1973-08-14	Female

	cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1	11000	AW00011000	Jon	Yang	M	M	2025-10-06
2	11001	AW00011001	Eugene	Huang	S	M	2025-10-06
3	11002	AW00011002	Ruben	Torres	M	M	2025-10-06

We can join these two tables using the **customer_key** ID.

CTRL+Enter adds a new row in Vaio.

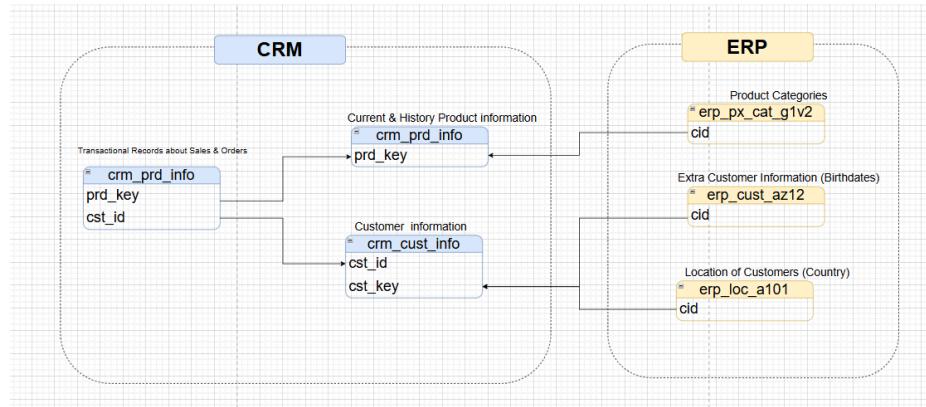
ERP PX catalog contains product categories and subcategories. We have special identifiers for that information.

105 % No se encontraron problemas.

	id	cat	subcat	maintenance
1	AC_BR	Accessories	Bike Racks	Yes
2	AC_BS	Accessories	Bike Stands	No
3	AC_BC	Accessories	Bottles and Cages	No
4	AC_CL	Accessories	Cleaners	Yes
5	AC_FE	Accessories	Fenders	No

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01 00:00:00.000	NULL
2	211	CO-RF-FR-R92R-58	HL Road Frame - Red- 58	NULL	R	2003-07-01 00:00:00.000	NULL
3	212	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

We can join those two tables using the source system CRM.



Now it's much clearer how the tables are connected to each other.

EPICS		Tasks
Build Silver Layer	Analysing: Explore & Understand Data	<input checked="" type="checkbox"/>
Build Silver Layer	Document: Draw Data Integration (Draw.io)	<input checked="" type="checkbox"/>
Build Silver Layer	Coding: Data Cleansing	<input type="checkbox"/> OPEN <input type="checkbox"/>
Build Silver Layer	Validating: Data Correctness Checks	<input type="checkbox"/>

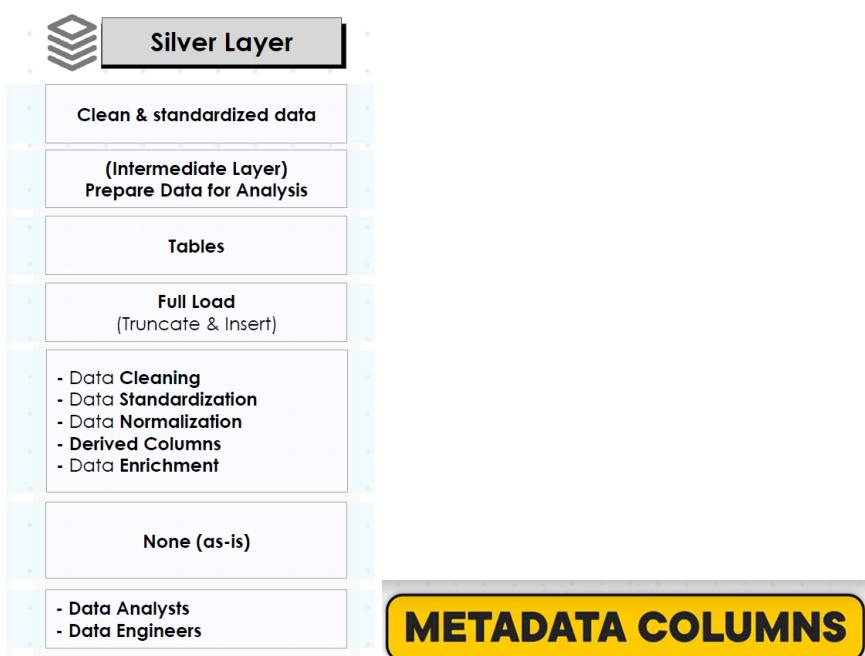


DATA JANITOR

Build Silver Layer	Document: Draw Data Integration (Draw.io)	<input checked="" type="checkbox"/>
Build Silver Layer	Coding: Data Cleansing	<input type="checkbox"/>
Build Silver Layer	Validating: Data Correctness Checks	<input type="checkbox"/> OPEN <input type="checkbox"/>
Build Silver Layer	Document: External Data Flow (Draw.io)	

Build Silver Layer

Create DDL for Tables



In this part, we have a lot of data transformation.

These are extra columns or fills that the data engineers add to each table and don't come directly from the source system.

create_date: The record's load timestamp.

update_date: The record's last update timestamp.

source_system: The origin system of the record.

file_location: The file source of the record.

It's good to see if I have gaps in my data. Specially if I have increments. It's like putting leaves on everything.

We will add a default value for the date. I want the database to generate this information automatically. The naming convention is very important. And then we repeat the same thing for all other tables.

```

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,
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
GO

```

AdventureWorksDW2014
DataWarehouse
Diagramas de base de datos
Tablas
Tablas del sistema
Tablas de archivos
Tablas externas
Tablas de grafos
bronze.crm_cust_info
bronze.crm_prd_info
bronze.crm_sales_details
bronze.erp_cust_az12
bronze.erp_loc_a101
bronze.erp_px_cat_g1v2
Silver.crm_cust_info
silver.crm_prd_info
silver.crm_sales_details
silver.erp_cust_az12
silver.erp_loc_a101
silver.erp_px_cat_g1v2
Vistas

The commands have completed successfully. End time: 2025-10-13T12:46:42.4945016-03:00. There we created 6 tables.



Before writing any query, we need to assess the quality of the Bronze layer. First, we look at the quality issues, and then we start writing our code.

We'll review all the tables in the Bronze layer, clean them, and then insert them into the Silver layer.

-- Check for nulls or duplicates in Primary Key

-- Expectation: no result

Quality Check
A Primary Key must be unique and not null

To detect duplicates in the PK, we aggregate the primary key. If we find any value that appears more than once, it means it's not unique and we have duplicates in the primary table.

We're interested in the values that are greater than 1.

```

3
4     SELECT
5         [cst_id],
6         COUNT(*)
7     FROM [bronze].[crm_cust_info]
8     GROUP BY cst_id
9     HAVING COUNT (*) > 1

```

SQLQuery7.sql...no conectado SQLQuery6.sql...no conectado SQLQuery5.sql...no conectado

Resultados Mensajes

cst_id	(Sin nombre de columna)
1	29449
2	29473
3	29433
4	NULL
5	29483
6	29466

```

1 -- Check for nulls or duplicates in Primary Key
2 --Expectation: no result
3
4     SELECT
5         [cst_id],
6         COUNT(*)
7     FROM [bronze].[crm_cust_info]
8     GROUP BY cst_id
9     HAVING COUNT (*) > 1 OR cst_id IS NULL

```

We have the following issues in this table because this indicates that we have duplicates. Those IDs appear more than once in the table.

We'll start by writing the query that will perform the data transformation and data cleaning.

```

SQLQuery6.s...David (88)* + X SQLQuery5.s...David (89)* | ddl_bronzes...N\David (61)*
1  SELECT
2  *
3  FROM [bronze].[crm_cust_info]

15 % No se encontraron problemas.
Resultados Mensajes
cst_id cst_key cst_firstname cst_lastname cst_marital_status cst_gndr cst_create_date
1 11000 AW00011000 Jon Yang M M 2025-10-06
2 11001 AW00011001 Eugene Huang S M 2025-10-06

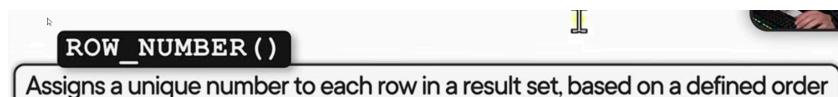
SQLQuery5.s...David (88)* + X SQLQuery6.s...David (88)* + X | ddl_bronzes...N\David (61)*
1  SELECT
2  *
3  FROM [bronze].[crm_cust_info]
4  WHERE [cst_id]= 29466

115 % No se encontraron problemas.
Resultados Mensajes
cst_id cst_key cst_firstname cst_lastname cst_marital_status cst_gndr cst_create_date
1 29466 AW00029466 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 focus on the issues. The ID is repeated 3 times.

Which one do we choose? It's better to choose the newest one because it has the most up-to-date information.



dividimos por customer id

```

SQLQuery5.s...David (89)* + X SQLQuery6.s...David (88)* + X | ddl_bronzes...N\David (61)* | ddl_silvers...N\David (56)
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]
5  WHERE [cst_id]= 29466

115 % No se encontraron problemas.
Resultados Mensajes
cst_id cst_key cst_firstname cst_lastname cst_marital_status cst_gndr cst_create_date flag_last
1 29466 AW00029466 Lance Jimenez M M 2026-01-27 1
2 29466 AW00029466 Lance Jimenez M NULL 2026-01-26 2
3 29466 AW00029466 NULL NULL NULL 2026-01-25 3

```

The data is sorted by the creation date.

```

David (89)* + X SQLQuery6.s...David (88)* + X | ddl_bronzes...N\David (61)* | ddl_silvers...N\David (56)
* 
FROM (
SELECT
*
ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) as flag_last
FROM [bronze].[crm_cust_info]
)t WHERE flag_last !=1

115 % No se encontraron problemas.
Resultados Mensajes
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 NULL 2
2 NULL 13451235 NULL NULL NULL NULL NULL 3
3 29433 AW00029433 NULL NULL M M 2026-01-25 2
4 29449 AW00029449 NULL Chen S NULL 2026-01-25 2
5 29466 AW00029466 Lance Jimenez M NULL 2026-01-26 2
6 29466 AW00029466 NULL NULL NULL NULL 2026-01-25 3
7 29473 AW00029473 Carmen NULL NULL NULL 2026-01-25 2
8 29483 AW00029483 NULL Navarro NULL NULL 2026-01-25 2

```

Here we have all the data that we don't need.

```

1 SELECT *
2 FROM (
3     SELECT *
4     ,ROW_NUMBER () OVER (PARTITION BY cst_id ORDER BY cst_create_date DESC) as flag_last
5     FROM [bronze].[crm_cust_info]
6     WHERE [cst_id] IS NOT NULL
7 ) t WHERE flag_last = 1

```

115 % No se encontraron problemas.

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	flag_last
11000	AW0001000	Jon	Yang	M	M	2025-10-06	1
11001	AW0001001	Eugene	Huang	S	M	2025-10-06	1


```

6 ROW_NUMBER () OVER (PARTITION BY cst_id ORDER BY cst_create_date
7 FROM [bronze].[crm_cust_info]
8 ) t WHERE flag_last = 1 AND cst_id = 29466

```

115 % No se encontraron problemas.

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	flag_last	
1	29466	AW00029466	Lance	Jimenez	M	M	2026-01-27	1

If we do it this way, we ensure that our PK doesn't repeat and that each value appears only once. This query removes any duplicates.



```

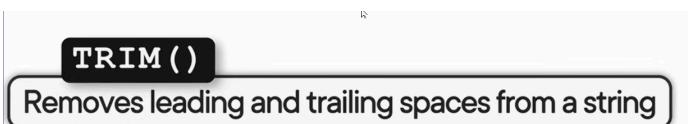
-- Check for unwanted Spaces
SELECT cst_firstname
FROM bronze.crm_cust_info

```

115 % No se encontraron problemas.

cst_firstname
Jon
Eugene
Ruben

Let's check for empty spaces. We apply a filter.



The TRIM function removes all empty spaces.

If the original value is not equal to the same value after trimming, it means there are spaces!

If the first value is not equal to the second value after applying TRIM, then we have a problem. The first name is not equal to the first name after trimming the values. We can also check the last name.

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

This is the list of all names where we have extra spaces.

We perform the same check for the last names. We can check all string values in the table.

```

1 -- Check for unwanted Spaces
2 -- Expectation: No result
3
4 SELECT cst_lastname
5 FROM bronze.crm_cust_info
6 WHERE cst_lastname != TRIM(cst_lastname)

1 -- Check for unwanted Spaces
2 -- Expectation: No result
3
4 SELECT cst_gndr
5 FROM bronze.crm_cust_info
6 WHERE cst_gndr != TRIM(cst_gndr)

```

We didn't get any results, which means the quality is better.

Now we're going to perform a transformation to clean these two columns.

```
--Limpiamos espacios innecesarios
SELECT
    [cst_id],
    [cst_key],
    [cst_firstname],
    [cst_lastname],
    [cst_marital_status],
    [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
```

```
--Limpiamos espacios innecesarios
SELECT
    [cst_id],
    [cst_key],
    TRIM ([cst_firstname]) AS cst_firstname,
    TRIM ([cst_lastname]) AS cst_lastname,
    [cst_marital_status],
    [cst_gndr],
    [cst_create_date]
FROM (
    SELECT
        *,
        ROW_NUMBER() OVER (PARTITION BY cst_id ORDER BY cst_cr
    FROM [bronze].[crm_cust_info]
    WHERE [cst_id] IS NOT NULL
) t WHERE flag_last = 1
```

Results

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1	11000	AW00011000	Jon	Yang	M	2025-10-06
2	11001	AW00011001	Eugene	Huang	S	2025-10-06
3	11002	AW00011002	Ruben	Torres	M	2025-10-06

With this, we've already cleaned unnecessary spaces.

CST create date

Quality Check

Check the consistency of values in low cardinality columns

Results Messages

Now we check the quality of the marital_status and gndr columns.

In our data warehouse, we aim to store clear and meaningful values rather than using abbreviated terms

BRING Consistency

```
-- Data Standardization & Consistency
SELECT DISTINCT cst_gndr
FROM bronze.crm_cust_info
```

Results

cst_gndr
NULL
F
M

We're going to change the values F and M to Female and Male.

We also need to check for NULLs.

In our data warehouse, we use the default value 'n/a' for missing values !

Apply UPPER() just in case mixed-case values appear later in your column.

```
TRIM( [cst_lastname]) AS cst_lastname,
[cst_marital_status],
CASE WHEN UPPER (cst_gndr) = 'F' THEN 'Female'
     WHEN UPPER (cst_gndr) = 'M' THEN 'Male'
     ELSE 'n/a'
END [cst_gndr],
[cst_create_date]
```

We'll convert them to UPPER just in case. We use the UPPER function to handle any values that might appear in lowercase.

```
[cst_marital_status],
CASE WHEN UPPER (cst_gndr) = 'F' THEN 'Female'
     WHEN UPPER (cst_gndr) = 'M' THEN 'Male'
     ELSE 'n/a'
END [cst_gndr]
```

Something else you can add—if you don't trust the data and want to avoid extra spaces—is to include the TRIM function.

The screenshot shows a code editor with a query that uses the TRIM function to remove leading and trailing spaces from gender values ('F' and 'M'). It then uses a CASE WHEN statement to map these to 'Female' and 'Male' respectively, or 'n/a' if neither applies. The results table shows four rows of customer data with their gender columns now containing 'Female', 'Male', or 'n/a'.

```
7   TRIM([cst_gndr]) AS cst_gndr,
8   TRIM([cst_lastname]) AS cst_lastname,
9   [cst_marital_status],
10  CASE WHEN UPPER(Trim(cst_gndr)) = 'F' THEN 'Female'
11    WHEN UPPER(Trim(cst_gndr)) = 'M' THEN 'Male'
12    ELSE 'n/a'
13  END [cst_gndr],
14  [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	Male	2025-10-06
2	11001	AW00011001	Eugene	Huang	S	Male	2025-10-06
3	11002	AW00011002	Ruben	Torres	M	Male	2025-10-06
4	11003	AW00011003	Christy	Zhu	S	Female	2025-10-06

If we don't get any value, it will show N/A.

Now we'll do the same for marital_status.

The screenshot shows a code editor with a query that first selects distinct marital statuses from the bronze table. It then uses the TRIM function to remove spaces from the marital status values ('S', 'M', or 'n/a') and maps them to 'Single', 'Married', or 'n/a' using a CASE WHEN statement. The results table shows two rows of customer data with their marital status columns now containing 'Single', 'Married', or 'n/a'.

```
6 WHERE cst_key != TRIM(cst_key)
7
8 -- Data Standardization & Consistency
9 SELECT DISTINCT [cst_marital_status]
10 FROM bronze.crm_cust_info
11
```

cst_marital_status
1 S
2 NULL
3 M

```
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],
```

```
6 [cst_key],
7 TRIM([cst_firstname]) AS cst_firstname,
8 TRIM([cst_lastname]) AS cst_lastname,
9 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],
10 CASE WHEN UPPER(Trim(cst_gndr)) = 'F' THEN 'Female'
WHEN UPPER(Trim(cst_gndr)) = 'M' THEN 'Male'
```

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	
1	11000	AW00011000	Jon	Yang	Married	Male	2025-10-06
2	11001	AW00011001	Eugene	Huang	Single	Male	2025-10-06

Now, instead of showing S or M, it shows Single or Married.

Next, we'll work on the create_date.

Now we're going to write the INSERT statement.

The screenshot shows the beginning of an INSERT INTO statement for the silver table. It lists all the columns: cst_id, cst_key, cst_firstname, cst_lastname, cst_marital_status, cst_gndr, and cst_create_date. Below the INSERT statement, there is a SELECT statement that includes the cst_id and cst_key columns.

```
3
4
5 INSERT INTO silver.crm_cust_info (
6   [cst_id],
7   [cst_key],
8   [cst_firstname],
9   [cst_lastname],
10  [cst_marital_status],
11  [cst_gndr],
12  [cst_create_date])
13
14 SELECT
15   [cst_id],
16   [cst_key],
```

(18,484 rows affected)

End time: 2025-10-13T22:09:31.0686826-03:00

With this, we have inserted the clean data into the Silver table.

Quality Check of the Silver Table

```

SQLQuery5 c...David (62)* SQLQuery8.sq..(David (53)* SQLQuery6 pa...|David
1 -- Check for nulls or duplicates in Primary Key
2 --Expectation: no result
3
4 SELECT
5 [cst_id],
6 COUNT(*)
7 FROM silver.[crm_cust_info]
8 GROUP BY cst_id
9 HAVING COUNT(*) > 1 OR cst_id IS NULL

```

Resultados | Mensajes

cst_id	(Sin nombre de columna)

Quality of Silver

Re-run the quality check queries from the bronze layer to verify the quality of data in the silver layer.

```

10
11 -- Check for unwanted Spaces
12 -- Expectation: No result
13
14 SELECT [cst_firstname]
15 FROM silver.[crm_cust_info]
16 WHERE [cst_firstname] != TRIM([cst_firstname])
17

```

Resultados | Mensajes

cst_firstname

We don't have any duplicates in the PK.

```

16 WHERE [cst_firstname] != TRIM([cst_firstname])
17
18 -- Data Standardization & Consistency
19
20 SELECT DISTINCT cst_gndr
21 FROM silver.[crm_cust_info]
22
23

```

Resultados | Mensajes

cst_gndr
1 n/a
2 Male
3 Female

```

21
22 --Final Check
23 SELECT * FROM [silver].[crm_cust_info]

```

Resultados | Mensajes

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	dwh_create_date
1 11000	AW00011000	Jon	Yang	Married	Male	2025-10-06	2025-10-13 22:09:30.9466667
2 11001	AW00011001	Eugene	Huang	Single	Male	2025-10-06	2025-10-13 22:09:30.9466667
3 11002	AW00011002	Ruben	Torres	Married	Male	2025-10-06	2025-10-13 22:09:30.9466667
4 11003	AW00011003	Christy	Zhu	Single	Female	2025-10-06	2025-10-13 22:09:30.9466667

Everything is fine.

Remove Unwanted spaces

Removes unnecessary spaces to ensure data consistency, and uniformity across all records.

```

cst_key,
TRIM(cst_firstname) AS cst_firstname,
TRIM(cst_lastname) AS cst_lastname,
CASE

```

Maps coded values to meaningful, user-friendly descriptions

```

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 AS cst_marital_status, -- Normalize marital status values to
CASE
    WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'

```

Handling Missing Data

Fills in the blanks by adding a default value

```
    LST 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; -- Select the most recent record per customer
```

remove the duplicates

Remove Duplicates

Ensure only one record per entity by identifying and retaining the most relevant row.



```
1  SELECT  
2      [prd_id],  
3      [prd_key],  
4      [prd_nm],  
5      [prd_cost],  
6      [prd_line],  
7      [prd_start_dt],  
8      [prd_end_dt]  
9  
10     FROM [bronze].[crm_prd_info]
```

115 % - ● No se encontraron problemas.

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01 00:00:00.000	NULL
2	211	CO-RF-FR-R92R-58	HL Road Frame - Red- 58	NULL	R	2003-07-01 00:00:00.000	NULL
3	212	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

We start with the second table. The first thing we need to check is the PK.

SQLQuery2.s...David (58)* SQLQuery1.s...David (90)*

```

1  --Check For Nulls or Duplicates in Primary Key
2  --Expectation: No result
3  SELECT
4      [prd_id],
5      COUNT(*)
6  FROM [bronze].[crm_prd_info]
7  GROUP BY [prd_id]
8  HAVING COUNT(*) > 1 OR [prd_id] IS NULL

```

115 % No se encontraron problemas.

Resultados Mensajes

prd_id	(Sin nombre de columna)
--------	-------------------------

Perfect, the PK doesn't repeat.

Now we continue with the `prd_key`; we need to split the string into two pieces of information.



We use the `SUBSTRING` function to extract part of the string.

First, we specify the column from which we want to extract information. Then we define the starting position. Finally, we specify the length.

SQLQuery2 ch...N,David (58) SQLQuery1.s...David (90)*

```

1  SELECT
2      [prd_id],
3      [prd_key],
4      SUBSTRING(prd_key, 1, 5) AS cat_id,
5      [prd_nm],
6      [prd_cost],
7      [prd_line],
8      [prd_start_dt],
9      [prd_end_dt]
10     FROM [bronze].[crm_prd_info]

```

115 % No se encontraron problemas.

Resultados Mensajes

prd_id	prd_key	cat_id	prd_nm	prd_line
1	210	CO-RF-FR-R92B-58	CO-RF	HL Road Frame - Black- 58
2	211	CO-RF-FR-R92R-58	CO-RF	HL Road Frame - Red- 58
3	212	AC-HE-HL-U509-R	AC-HE	Sport-100 Helmet- Red
4	213	AC-HE-HL-U509-R	AC-HE	Sport-100 Helmet- Red
5	214	AC-HE-HL-U509-R	AC-HE	Sport-100 Helmet- Red

115 % No se encontraron problemas.

Resultados Mensajes

id	
1	AC_RF
2	AC_BR
3	AC_BS
4	AC_CL
5	AC_FE
6	AC_HE

We have a new column called `cat_id` that contains the first part of the string.

Let's do a double check to see if we can join the data. We'll check if we can join both tables.



We have a problem: one uses an underscore and the other a hyphen.

We're going to use the `REPLACE` function.

```

SQLQuery2 ch...N\David (58) SQLQuery1.s...David (90)* ✘
1  SELECT
2    [prd_id],
3    [prd_key],
4    REPLACE (SUBSTRING (prd_key, 1, 5), '-', '_') AS cat_id,
5    [prd_nm],
6    [prd_cost],
7    [prd_line],
8    [prd_start_dt],
9    [prd_end_dt]
10   FROM [bronze].[crm_prd_info]
11
12  SELECT distinct id FROM bronze.[erp_px_cat_g1v2]

```

115 % No se encontraron problemas.

	prd_id	prd_key	cat_id	prd_nm	prd_cost	prd_line	prd_start_dt
1	210	CO-RF-FR-R92B-58	CO_RF	HL Road Frame - Black- 58	NULL	R	2003-07-01 00
2	211	CO-RF-FR-R92R-58	CO_RF	HL Road Frame - Red- 58	NULL	R	2003-07-01 00
3	212	AC-HE-HL-U509-R	AC_HE	Sport-100 Helmet- Red	12	S	2011-07-01 00
4	213	AC-HF-HI-I1509-R	AC_HF	Sport-100 Helmet- Red	14	S	2012-07-01 00

Perfect, we replaced the hyphen with an underscore.

**filters out unmatched data
after applying transformation**

Now we want to check that there isn't a matching value in the second table.

```

8  [prd_start_dt],
9  [prd_end_dt]
10  FROM [bronze].[crm_prd_info]
11 WHERE REPLACE (SUBSTRING (prd_key, 1, 5), '-', '_') NOT IN
12   (SELECT distinct id FROM bronze.[erp_px_cat_g1v2])

```

115 % No se encontraron problemas.

	prd_id	prd_key	cat_id	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	542	CO-PE-PD-M282	CO_PE	LL Mountain Pedal	18	M	2013-07-01 00:00:00.000	NULL
2	543	CO-PE-PD-M340	CO_PE	ML Mountain Pedal	28	M	2013-07-01 00:00:00.000	NULL
3	544	CO-PE-PD-M562	CO_PE	HL Mountain Pedal	36	M	2013-07-01 00:00:00.000	NULL
4	545	CO-PE-PD-R347	CO_PE	LL Road Pedal	18	R	2013-07-01 00:00:00.000	NULL
5	546	CO-PF-PD-R563	CO_PF	MI Road Pedal	28	R	2013-07-01 00:00:00.000	NULL

We only found one category that isn't in the second table. Our check is fine.

We already have the first part; now we'll do the same for the second part.

This time, we don't start at position 1 but at position 7. To make it dynamic and capture the full length, we use the LEN function to get all the characters each item has without losing information.

prd_line,
prd_len()

Returns the number of characters in a string

```

2   [prd_id],
3   [prd_key],
4   REPLACE (SUBSTRING (prd_key, 1, 5), '-', '_') AS cat_id,
5   SUBSTRING (prd_key, 7, LEN(prd_key)) AS prd_key,
6   [prd_nm],
7   [prd_cost],
8   [prd_line],
9   [prd_start_dt],
10  [prd_end_dt]
11  FROM [bronze].[crm_prd_info]

```

115 % No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line
1	210	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Road Frame - Black- 58	NULL	R
2	211	CO-RF-FR-R92R-58	CO_RF	FR-R92R-58	HL Road Frame - Red- 58	NULL	R
3	212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	S
4	213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	S
5	214	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	13	S
6	215	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet- Black	12	S
7	216	AC-HE-HL-U509	AC_HE	HL-U509	Sport-100 Helmet- Black	14	S

Now we extract the second part from the `prd_key` string.

We do this to join it with the `sales_details` table.

```

11  lprd_enu_ate
12
13   SELECT [sls_prd_key] FROM [bronze].[crm_sales_details]
14

```

115 % No se encontraron problemas.

Resultados Mensajes

	sls_prd_key
1	BK-R93R-62
2	BK-M82S-44
3	BK-M82S-44
4	BK-R50B-62
5	BK-M82S-44
6	BK-R93R-44
7	BK-R93R-62
8	BK-M82B-48
9	BK-M82S-38

We check the data in the table we want to join.

We have many products that don't have any orders.

```

7   lpra_costi,
8   [prd_line],
9   [prd_start_dt],
10  [prd_end_dt]
11  FROM [bronze].[crm_prd_info]
12  WHERE SUBSTRING (prd_key, 7, LEN(prd_key)) NOT IN (
13    SELECT [sls_prd_key] FROM [bronze].[crm_sales_details])
14

```

115 % No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01 00:00:00.000	NULL
2	211	CO-RF-FR-R92R-58	CO_RF	FR-R92R-58	HL Road Frame - Red- 58	NULL	R	2003-07-01 00:00:00.000	NULL
3	218	CL-SO-SO-B909-M	CL_SO	SO-B909-M	Mountain Bike Socks- M	3	M	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000
4	219	CL-SO-SO-B909-L	CL_SO	SO-B909-L	Mountain Bike Socks- L	3	M	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000
5	238	CO-RF-FR-R92R-62	CO_RF	FR-R92R-62	HL Road Frame - Red- 62	748	R	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000
6	239	CO-RF-FR-R92R-62	CO_RF	FR-R92R-62	HL Road Frame - Red- 62	722	R	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
7	240	CO-RF-FR-R92R-62	CO_RF	FR-R92R-62	HL Road Frame - Red- 62	869	R	2013-07-01 00:00:00.000	NULL

We don't have any orders for the product that starts with FK.

```

10
11     lpru_ema_nac
12     FROM [bronze].[crm_prd_info]
13     WHERE SUBSTRING (prd_key, 7, LEN(prd_key) ) NOT IN (
14         SELECT [sls_prd_key] FROM [bronze].[crm_sales_details]
15         WHERE [sls_prd_key] LIKE 'FK-1%')

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

sls_prd_key

```

10     [prd_end_dt]
11     FROM [bronze].[crm_prd_info]
12     WHERE SUBSTRING (prd_key, 7, LEN(prd_key) ) IN (
13         SELECT [sls_prd_key] FROM [bronze].[crm_sales_details])

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	212	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
2	213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
3	214	AC-HE-HL-U509-R	AC HE	HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01 00:00:00.000	NULL

We need to remove the products that don't have any orders. Everything is fine. There are only products without orders, so we're good.

Next, we move on to `product_name`, which has unwanted spaces.

```

9
10    --Check for unwanted spaces
11    -- Expectation: No Results
12    SELECT [prd_nm]
13        FROM [bronze].[crm_prd_info]
14        WHERE prd_nm != TRIM(prd_nm)

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

```

15
16    --Check for NULLS or Negative Spaces
17    -- Expectation: No Results
18    SELECT [prd_cost]
19        FROM [bronze].[crm_prd_info]
20        WHERE [prd_cost] < 0 OR [prd_cost] IS NULL

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

prd_cost
1 NULL
2 NULL

We'll check that there are no `NULLs` or negative numbers. There are no negative values, but we do have `NULLs`.

We can replace the `NULLs` with zeros.

`ISNULL` is used to replace `NULL` values with zeros.

`ISNULL(expression,replacement_value)` RETURNS <any type>

ISNULL()

You can use `COALESCE` as well

Replaces `NULL` values with a specified replacement value

If it's `NULL`, then replace that value with zero.

```

1  SELECT
2      [prd_id],
3      [prd_key],
4      REPLACE (SUBSTRING (prd_key, 1, 5 ), '-' , '_') AS cat_id,
5      SUBSTRING (prd_key, 7, LEN(prd_key) ) AS prd_key,
6      [prd_nm],
7      ISNULL([prd_cost], 0) AS prd_cost,
8      [prd_line],
9      [prd_start_dt],
10     [prd_end_dt]
11  FROM [bronze].[crm_prd_info]

```

15 % ✓ No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line
1	210	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Road Frame - Black- 58	0	R
2	211	CO-RF-FR-R92R-58	CO_RF	FR-R92R-58	HL Road Frame - Red- 58	0	R
3	212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	S
4	213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	S

We no longer have **NULLs**; we have zeros. This will help us in the future when performing aggregates like AVG.

```

20 WHERE [prd_cost] < 0 OR [prd_cost] IS NULL
21
22 -- Data Standardization & Consistency
23
24 SELECT DISTINCT [prd_line]
25   FROM [bronze].[crm_prd_info]

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

	prd_line
1	NULL
2	M
3	R
4	S
5	T


```

6      [prd_nm],
7      ISNULL([prd_cost], 0) AS prd_cost,
8      CASE WHEN UPPER(TRIM([prd_line])) = 'M' THEN 'Mountain'
9          WHEN UPPER(TRIM([prd_line])) = 'R' THEN 'Road'
10         WHEN UPPER(TRIM([prd_line])) = 'S' THEN 'Other Sales'
11         WHEN UPPER(TRIM([prd_line])) = 'T' THEN 'Touring'
12         ELSE 'N/A'
13     END AS prd_line,
14     [prd_start_dt],
15     [prd_end_dt]
16   FROM [bronze].[crm_prd_info]
17

```

115 % ✓ No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	CO_RF	FR-R92B-58	HL Road Frame - Black- 58	0	Road	2010-01-01	2010-01-31
2	211	CO-RF-FR-R92R-58	CO_RF	FR-R92R-58	HL Road Frame - Red- 58	0	Road	2010-01-01	2010-01-31
3	212	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	12	Other Sales	2010-01-01	2010-01-31
4	213	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	14	Other Sales	2010-01-01	2010-01-31
5	214	AC-HE-HL-U509-R	AC_HE	HL-U509-R	Sport-100 Helmet- Red	13	Other Sales	2010-01-01	2010-01-31

We continue with the next column, **prd_line**, and review all the options.

We transform all the values into more user-friendly labels.

'ing'

Quick CASE WHEN
 Ideal for simple value mapping

prc_cost prc_line prc_start_dt

```

6      [prd_nm],
7      ISNULL([prd_cost], 0) AS prd_cost,
8      CASE UPPER(TRIM([prd_line]))
9          WHEN 'M' THEN 'Mountain'
10         WHEN 'R' THEN 'Road'
11         WHEN 'S' THEN 'Other Sales'
12         WHEN 'T' THEN 'Touring'
13         ELSE 'N/A'
14     END AS prd_line.
15

```

We can write it this way to avoid repeating the same code all the time.

Next, we move on to the last two columns.

s End date must not be earlier than the start date

We have a problem because some end dates occur before the start dates.

```

27
28
29 -- Check for Invalid Date Orders
30
31     SELECT *
  FROM [bronze].[crm_prd_info]
 WHERE [prd_end_dt]<[prd_start_dt]

115 % - ● No se encontraron problemas.

```

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	212	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
2	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
3	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000

It should have returned a NULL value. We have problems.

For complex transformations in SQL, I typically narrow it down to a specific example and brainstorm multiple solution approaches

	A	B	C	D	E	F	G
1	212	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
2	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
3	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01 00:00:00.000	NULL
4							
5	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000
5	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
7	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01 00:00:00.000	NULL
8							

We take these values and do a brainstorming session.

#1 Solution

Switch End Date and Start Date

#2 Solution

Derive the End Date from the Start Date

#2 Solution

End Date = Start Date of the 'NEXT' Record

#2 Solution

End Date = Start Date of the 'NEXT' Record -1

212 AC-HE-HL-U509-R	Sport-100 Helmet- Red	12 S	2011-07-01 00:00:00.000	2012-06-30 00:00:00.000
213 AC-HE-HL-U509-R	Sport-100 Helmet- Red	14 S	2012-07-01 00:00:00.000	2013-06-30 00:00:00.000
214 AC-HE-HL-U509-R	Sport-100 Helmet- Red	13 S	2013-07-01 00:00:00.000	NULL

We set the end date equal to the start date of the other order. Then we subtract one day to avoid overlap.

```

1  SELECT
2   [prd_id],
3   [prd_key],
4   [prd_nm],
5   [prd_cost],
6   [prd_line],
7   [prd_start_dt],
8   [prd_end_dt]
9   FROM [bronze].[crm_prd_info]
10  WHERE [prd_key] IN ('AC-HE-HL-U509-R', 'AC-HE-HL-U509')

```

115 % No se encontraron problemas.

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	212	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
2	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
3	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01 00:00:00.000	NULL
4	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000
5	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000
6	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01 00:00:00.000	NULL

We're going to work with those two values. Now we build our logic.

LEAD ()

Access values from the next row within a window

We're going to use the LEAD and LAG functions.

We'll apply the window function per product. We partition the data by prd_key and then need to order the data.

```

7  |  [prd_start_dt],
8  |  [prd_end_dt],
9  |  LEAD([prd_start_dt]) OVER ( PARTITION BY [prd_key] ORDER BY [prd_start_dt]) AS [prd_end_dt_test]
10 |  FROM [bronze].[crm_prd_info]
11 |  WHERE [prd_key] IN ('AC-HE-HL-U509-R', 'AC-HE-HL-U509')

```

105 % No se encontraron problemas.

Línea: 9

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	prd_end_dt_test
1	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	2012-07-01 00:00:00.000
2	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	2013-07-01 00:00:00.000
3	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01 00:00:00.000	NULL	NULL
4	212	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	2012-07-01 00:00:00.000
5	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	2013-07-01 00:00:00.000
6	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01 00:00:00.000	NULL	NULL

prd_id	prd_key	prd_nm	prd_start_dt	prd_end_dt	prd_end_dt_test
215	AC-HE-HL-U509	Sport-100 Helmet- Black	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	2012-07-01 00:00:00.000
216	AC-HE-HL-U509	Sport-100 Helmet- Black	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	2013-07-01 00:00:00.000
217	AC-HE-HL-U509	Sport-100 Helmet- Black	2013-07-01 00:00:00.000	NULL	NULL

It's more organized now.

We made the end date come from the start date of the next item.

```

9    LEAD([prd_start_dt]) OVER ( PARTITION BY [prd_key] ORDER BY [prd_start_dt]) -1 AS [prd_end_dt_test]
10   FROM [bronze].[crm_prd_info]
11   WHERE [prd_key] IN ('AC-HE-HL-U509-R', 'AC-HE-HL-U509')

```

105 % ✓ No se encontraron problemas. Línea: 9

Resultados Mensajes

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	prd_end_dt_test
1	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01 00:00:00.000	2007-12-28 00:00:00.000	2012-06-30 00:00:00.000
2	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	2013-06-30 00:00:00.000
3	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01 00:00:00.000	NULL	NULL
4	212	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	2012-06-30 00:00:00.000
5	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01 00:00:00.000	2008-12-27 00:00:00.000	2013-06-30 00:00:00.000
6	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01 00:00:00.000	NULL	NULL

Then we subtract 1, and now it works.

Now we replace what we wrote in our main query.

SQLQuery3.sql... (David (52)) SQLQuery 4 n... (David (89)) SQLQuery3 h... David (59) X

1 SELECT
2 [prd_id],
3 [prd_key],
4 REPLACE (SUBSTRING (prd_key, 1, 5), '-' , '_') AS cat_id,
5 SUBSTRING (prd_key, 7, LEN(prd_key)) AS prd_key,
6 [prd_nm],
7 ISNULL([prd_cost], 0) AS prd_cost,
8 CASE UPPER(STRIM([prd_line]))
9 WHEN 'M' THEN 'Mountain'
10 WHEN 'R' THEN 'Road'
11 WHEN 'S' THEN 'Other Sales'
12 WHEN 'T' THEN 'Touring'
13 ELSE 'N/A'
14 END AS prd_line,
15 [prd_start_dt],
16 LEAD([prd_start_dt]) OVER (PARTITION BY [prd_key] ORDER BY [prd_start_dt]) -1 AS [prd_end_dt]
17 FROM [bronze].[crm_prd_info]

105 % ✓ No se encontraron problemas. Línea: 16

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	478	AC-BC-BC-M005	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01 00:00:00.000	NULL
2	479	AC-BC-BC-R205	AC_BC	BC-R205	Road Bottle Cage	3	Road	2013-07-01 00:00:00.000	NULL
3	477	AC-BC-WB-H098	AC_BC	WB-H098	Water Bottle - 30 oz.	2	Other Sales	2013-07-01 00:00:00.000	NULL

We don't like the zeros in the start_date.

14 WHILE @i < @max - 1 BEGIN
13 SET @i = @i + 1
12 ELSE 'N/A'
11 END AS prd_line,
10 CAST ([prd_start_dt] AS DATE) AS [prd_start_dt] ,
9 CAST (LEAD([prd_start_dt]) OVER (PARTITION BY [prd_key]
8 ORDER BY [prd_start_dt]) -1 AS DATE) AS [prd_end_dt]
7 FROM [bronze].[crm_prd_info]

105 % ✓ No se encontraron problemas.

Resultados Mensajes

	prd_id	prd_key	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	478	AC-BC-BC-M005	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01	NULL
2	479	AC-BC-BC-R205	AC_BC	BC-R205	Road Bottle Cage	3	Road	2013-07-01	NULL
3	477	AC-BC-WB-H098	AC_BC	WB-H098	Water Bottle - 30 oz.	2	Other Sales	2013-07-01	NULL
4	483	AC-BR-RA-H123	AC_BR	RA-H123	Hitch Rack - 4 Bike	45	Other Sales	2013-07-01	NULL
5	486	AC-BS-ST-1401	AC_BS	ST-1401	All-Purpose Bike Stand	59	Mountain	2013-07-01	NULL
6	484	AC-CL-CL-9009	AC_CL	CL-9009	Bike Wash - Dissolver	3	Other Sales	2013-07-01	NULL
7	485	AC-FE-FE-6654	AC_FE	FE-6654	Fender Set - Mountain	8	Mountain	2013-07-01	NULL
8	215	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

Now it looks much better. We now have clean product information.

```

LQuery3.sq...\\David (52)* SQLQuery 4 n...\\David (89)* SQLQuery3 h...\\David
1  v CREATE TABLE [silver].[crm_prd_info] (
2      [prd_id] INT,
3      [prd_key] NVARCHAR(50),
4      [prd_nm] NVARCHAR(50)
5      [prd_cost] INT,
6      [prd_line] NVARCHAR(50)
7      [prd_start_dt] DATETIME,
8      [prd_end_dt] DATETIME,
9      [dwh_create_date] DATETIME2 DEFAULT GETDATE()
10     );
11
12     SELECT

```

Everything is fine now, but before creating the tables, we need to do one more step.

```

SQLQuery3.sq...\\David (52)* SQLQuery 4 n...\\David (89)* SQLQuery3 h...\\David (59)
1  v IF OBJECT_ID ('silver.crm_prd_info', 'U') IS NOT NULL
2      DROP TABLE [silver].[crm_prd_info];
3  v CREATE TABLE [silver].[crm_prd_info] (
4      [prd_id] INT,
5      [cat_id] NVARCHAR(50),
6      [prd_key] NVARCHAR(50),
7      [prd_nm] NVARCHAR(50),
8      [prd_cost] INT,
9      [prd_line] NVARCHAR(50),
10     [prd_start_dt] DATE,
11     [prd_end_dt] DATE,
12     [dwh_create_date] DATETIME2 DEFAULT GETDATE()
13     );
14

105 %  ✓ No se encontraron problemas.
Mensajes  Los comandos se han completado correctamente.
Hora de finalización: 2025-10-14T13:09:20.8569775-03:00

```

We create our table with the correct formats. Sometimes we need to adjust the metadata if the quality isn't good. It's similar to the Bronze layer but with some modifications.

Now we insert the clean data into the Silver table.

```

a...\\David (88)* SQLQuery3.sq...\\David (52)*
v INSERT INTO silver.crm_prd_info (
    [prd_id],
    [cat_id],
    [prd_key],
    [prd_nm],
    [prd_cost],
    [prd_line],
    [prd_start_dt],
    [prd_end_dt]
)
SELECT

```

We inserted the data into the Silver layer.

(397 rows affected)

End time: 2025-10-14T13:19:23.8391306-03:00

Quality Check of the Silver Table

```

1  Checamos cada columna
2
3  --Check For Nulls or Duplicates in Primary Key
4  --Expectatio: No result
5  SELECT
6      [prd_id],
7      COUNT(*)
8  FROM [silver].[crm_prd_info]
9  GROUP BY [prd_id]
10 HAVING COUNT(*) > 1 OR [prd_id] IS NULL
11
12 --Check for unwanted spaces
13 -- Expectation: No Results
14
15
16
17
18
19
20
21
22
23
24
25
26
27

```

105 % No se encontraron problemas.

	prd_nm
prd_id	(Sin nombre de columna)

1. We check the PK.

```

15  FROM [silver].[crm_prd_info]
16  WHERE prd_nm != TRIM(prd_nm)
17
18  --Check for NULLS or Negative Spaces
19  -- Expectation: No Results
20
21  SELECT [prd_cost]
22  FROM [silver].[crm_prd_info]
23  WHERE [prd_cost] < 0 OR [prd_cost] IS NULL
24
25
26
27

```

105 % No se encontraron problemas.

	prd_line
1	Mountain
2	N/A
3	Other Sales
4	Road
5	Touring

```

27
28  -- Check for Invalidad Date Orders
29
30  SELECT *
31  FROM [silver].[crm_prd_info]
32  WHERE [prd_end_dt] < [prd_start_dt]
33
34
35
36

```

105 % No se encontraron problemas.

prd_id	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt

```

31  WHERE [prd_end_dt] < [prd_start_dt]
32
33  -- Final Look
34
35  SELECT *
36  FROM [silver].[crm_prd_info]

```

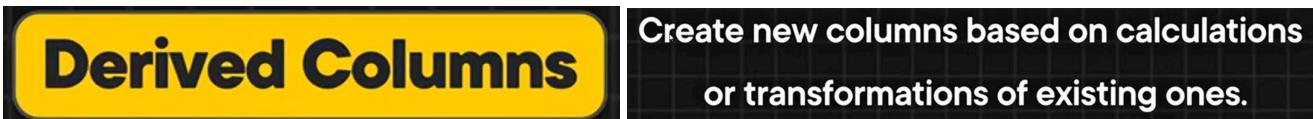
105 % No se encontraron problemas.

prd_id	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	dwh_create_date
1	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01	NULL	2025-10-14 13:28:49.2833333
2	479	AC_BC	BC-R205	3	Road	2013-07-01	NULL	2025-10-14 13:28:49.2833333
3	477	AC_BC	WB-H098	2	Other Sales	2013-07-01	NULL	2025-10-14 13:28:49.2833333
4	483	AC_BR	RA-H123	45	Other Sales	2013-07-01	NULL	2025-10-14 13:28:49.2833333
5	486	AC_BS	ST-1401	59	Mountain	2013-07-01	NULL	2025-10-14 13:28:49.2833333
6	484	AC_CL	CL-9009	3	Other Sales	2013-07-01	NULL	2025-10-14 13:28:49.2833333
7	485	AC_FE	FE-6654	8	Mountain	2013-07-01	NULL	2025-10-14 13:28:49.2833333

The last column is automatically generated by the DDL, indicating when we loaded this table.

```
SELECT
    [prd_id],
    REPLACE (SUBSTRING (prd_key, 1, 5 ), '-' , '_') AS cat_id, -- Extract category ID
    SUBSTRING (prd_key, 7, LEN(prd_key) ) AS prd_key,           -- Extract product key
    [prd_nm],
    ISNULL([prd_cost], 0) AS prd_cost,
    CASE UPPER(TRIM([prd_line]))
        WHEN 'M' THEN 'Mountain'
        WHEN 'R' THEN 'Road'
        WHEN 'S' THEN 'Other Sales'
        WHEN 'T' THEN 'Touring'
        ELSE 'N/A'
    END AS prd_line,                                         -- Map product Line codes to descriptive values
    CAST ([prd_start_dt] AS DATE) AS [prd_start_dt] ,
    CAST (
        LEAD([prd_start_dt]) OVER ( PARTITION BY [prd_key]
        ORDER BY [prd_start_dt]) -1
        AS DATE) AS [prd_end_dt]                                -- Calculate end date as one day before the next start date.
FROM [bronze].[crm_prd_info]
```

We performed the following transformations.



prd_nm,

ISNULL(prd_cost, 0) AS prd_cost,

CASE

ISNULL(prd_cost, 0) AS prd_cost.

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, -- Map product line codes to descriptive values

CAST(prd_start_dt AS DATE) AS prd_start_dt,

CAST(

CAST(prd_start_dt AS DATE) AS prd_start_dt,
CAST(
 LEAD(prd_start_dt) OVER (PARTITION BY prd_key ORDER BY prd_start_dt) - 1
 AS DATE
) AS prd_end_dt -- Calculate end date as one day before the next start date

We handled missing information.

data normalization

data type casting. We convert one data type to another