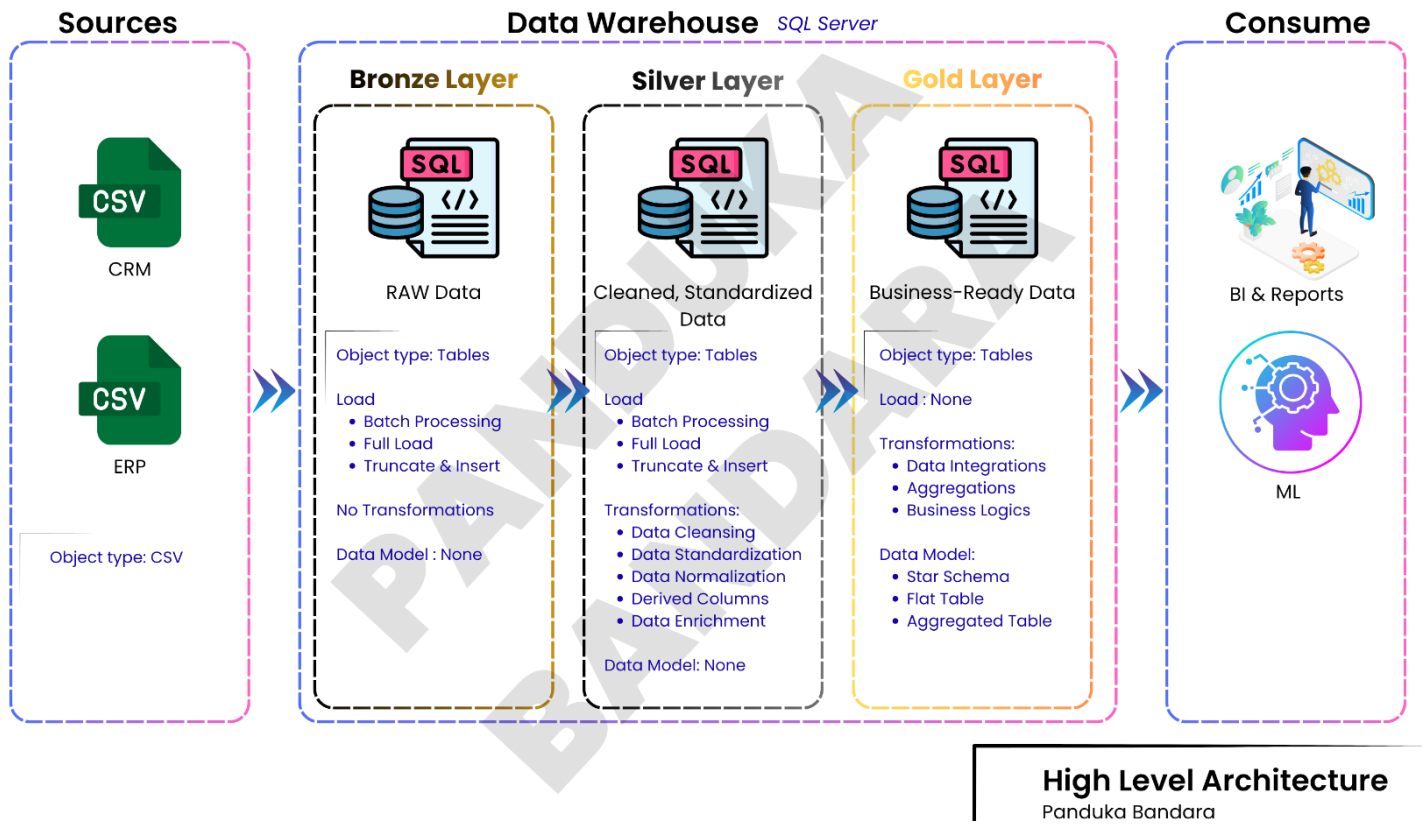
1. Data Warehouse

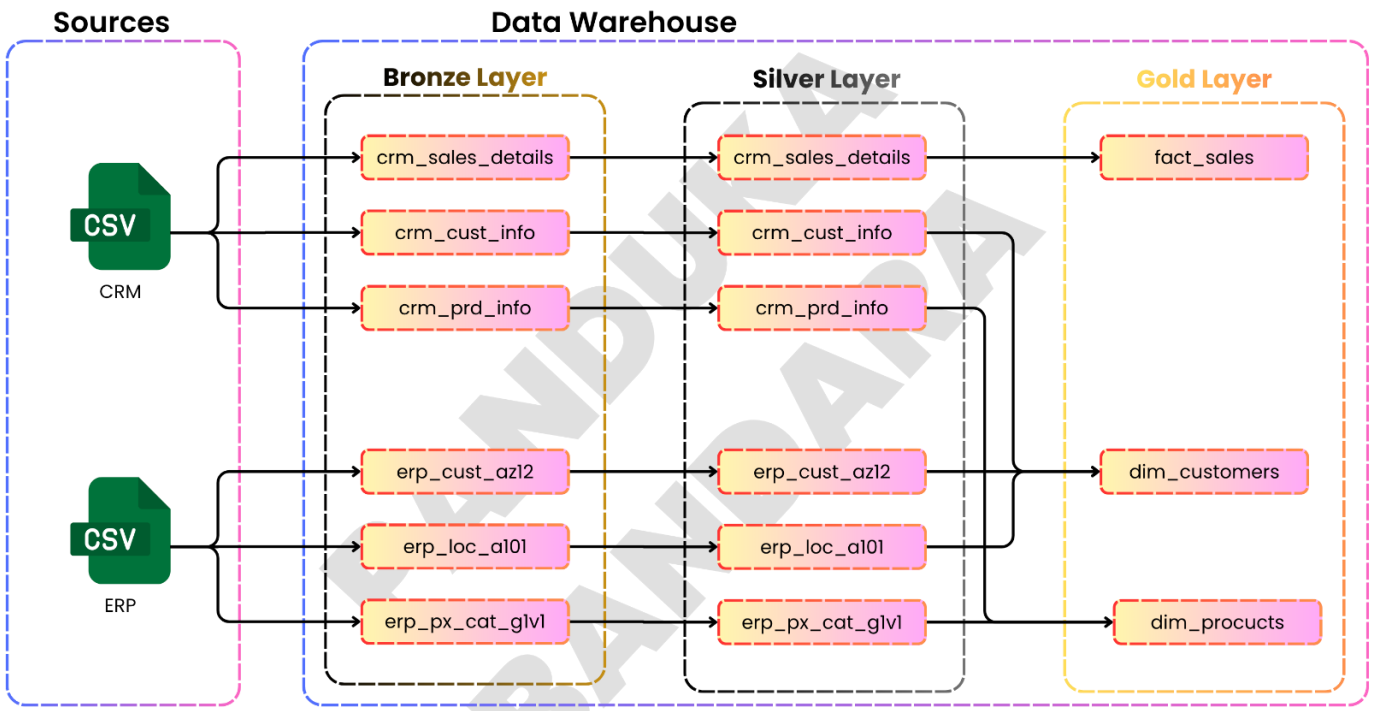
1.1 Overview

1.1.1 Data Sources Description

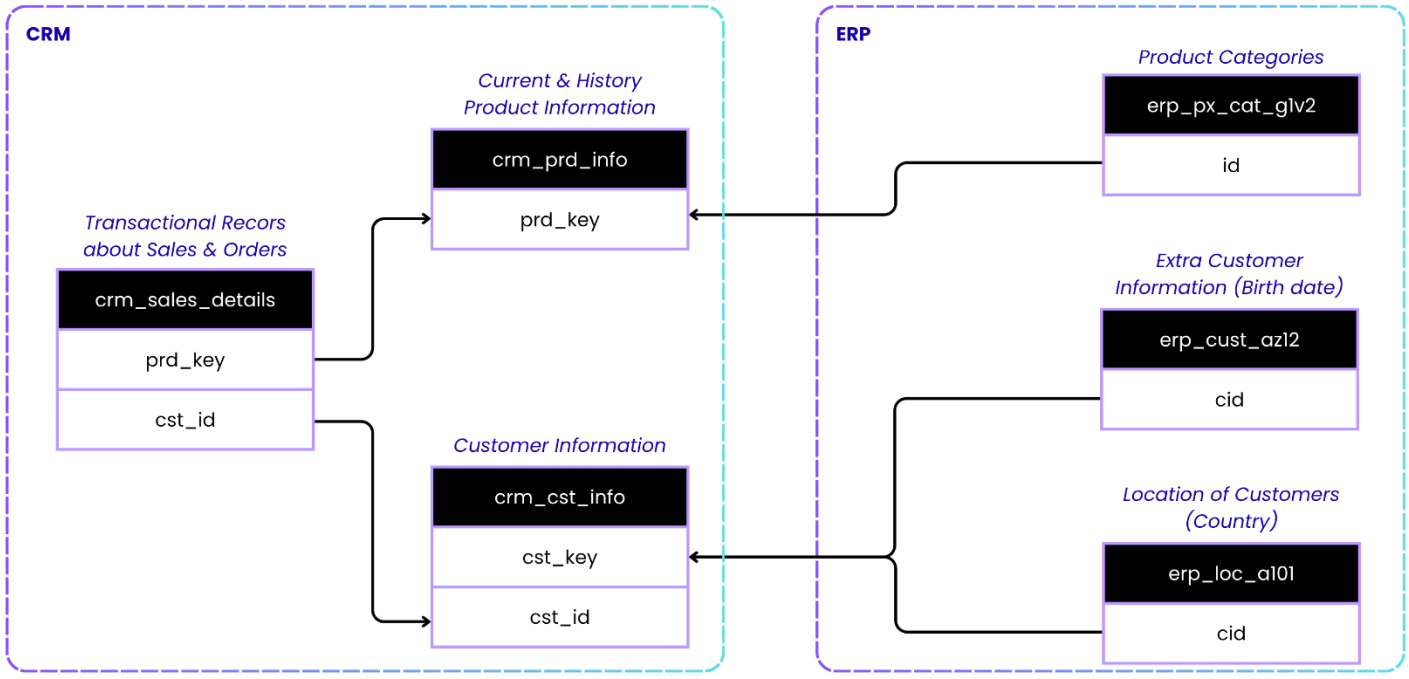
Source System	Full Form	Primary Function / Data Type	Example Tables / Data
CRM	**Customer Relationship Management**	Manages customer interactions, sales, marketing, and service data.	crm_cust_info, crm_prd_info, crm_sales_details
ERP	**Enterprise Resource Planning**	Handles operational, financial, and resource-related data like suppliers, manufacturing, and logistics.	erp_loc_a101, erp_cust_az12, erp_px_cat_g1v2

1.1.2 Diagrams



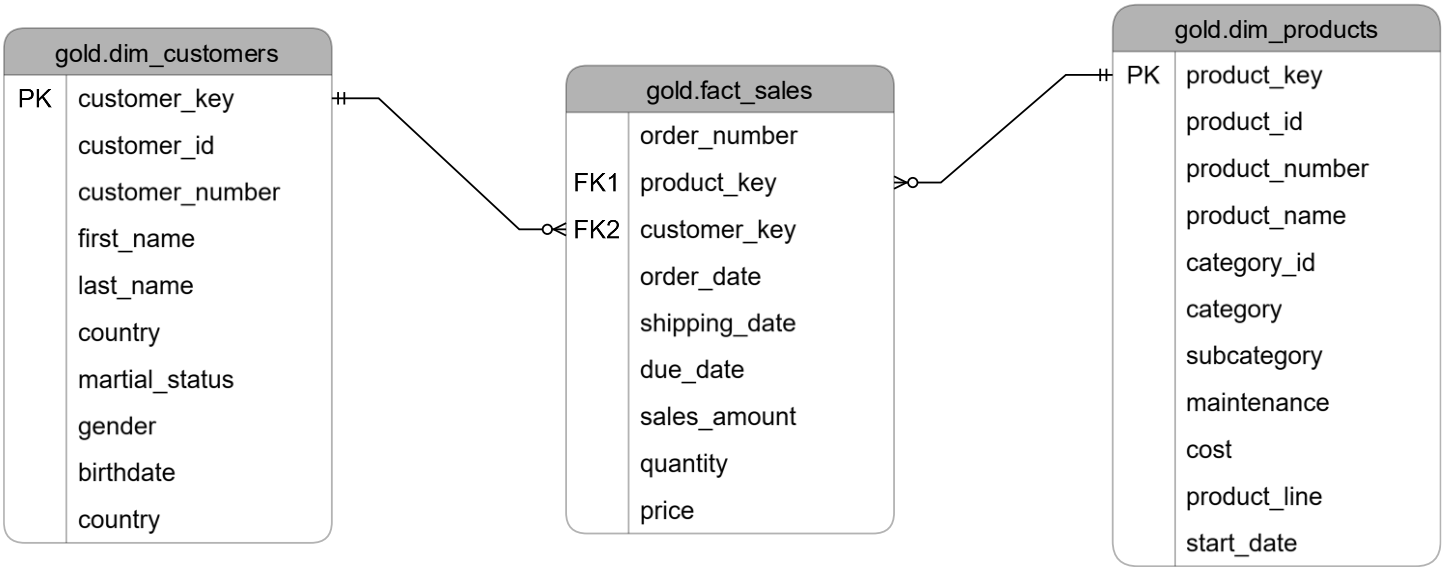


Data Flow
Panduka Bandara



Integration Model
Panduka Bandara

Sales Data Mart



1.2. Naming Conventions

1.2.1 General Principles

Naming Style: Use snake_case with all lowercase letters and underscores (_) to separate words.

Language: All names must be in English.

Clarity: Names should be meaningful, self-explanatory, and consistent across layers.

Avoid Reserved Words: Do not use SQL reserved keywords as identifiers (e.g., user, select, date).

Abbreviations: Avoid unnecessary abbreviations unless they are universally recognized (e.g., id, num, qty).

1.2.2 Table Naming Conventions

Each data layer (Bronze, Silver, Gold) follows a distinct naming standard to reflect data lineage and purpose.

Bronze Layer

Purpose: Raw, unprocessed data ingested directly from source systems.
Naming Pattern: <source_system>_<entity>
Rules:

- The <source_system> prefix identifies the origin (e.g., crm, erp, sap).
- The <entity> name must **match the original source table name** (no renaming).
- Do not alter column names from the source.

Examples:

- crm_customer_info → Raw customer information from CRM.
- erp_sales_order → Raw sales order data from ERP.

Silver Layer

Purpose: Cleaned, standardized, and conformed data derived from Bronze tables.
Naming Pattern: <source_system>_<entity>
Rules:

- Maintain the same structure as Bronze but include **transformed or standardized data**.
- Use consistent and descriptive entity names.
- Maintain the source system prefix to ensure lineage traceability.

Examples:

- crm_customer_master → Standardized customer dataset from CRM.
- erp_sales_order_clean → Cleaned sales order data from ERP.

Gold Layer

Purpose: Business-ready, analytical data models used for reporting and insights.
Naming Pattern: <category>_<entity>
Rules:

- Use business-aligned and domain-specific names.
- The <category> prefix identifies the table type (e.g., dim, fact, report).
- Avoid technical prefixes like source system names.

Examples:

- dim_customer → Customer dimension table.
- fact_sales → Sales fact table.
- report_sales_monthly → Monthly sales report dataset.

Category Glossary

Prefix	Purpose / Description	Example(s)
dim_	Dimension table (master entities)	dim_customer, dim_product
fact_	Fact table (transactions/measures)	fact_sales, fact_orders
report_	Aggregated/report-ready data	report_sales_summary

1.2.3 Column Naming Conventions

Surrogate Keys

Purpose: Uniquely identify records in dimension tables.
Naming Pattern: <entity>_key
Rules:

- The column must be a numeric, auto-generated key.
- Use the entity name as the prefix.

Examples:

- customer_key → Surrogate key in dim_customer.
- product_key → Surrogate key in dim_product.

Technical Columns

Purpose: Store system-generated metadata for auditing, lineage, and tracking.
Naming Pattern: dwh_<column_name>
Rules:

- Always prefix with dwh_ to distinguish from business columns.
- Clearly describe the column's purpose.

Examples:

- dwh_load_date → Date when the record was loaded.
- dwh_batch_id → Identifier for the data load batch.
- dwh_record_source → Indicates the source system.

Stored Procedure Naming Conventions

Purpose: Automate ETL/ELT processing for each data layer.
Naming Pattern: load_<layer>
Rules:

- <layer> must indicate the target layer (bronze, silver, or gold).
- Use consistent naming across all procedures.

Examples:

- load_bronze → Loads data into the Bronze layer.
- load_silver → Loads data into the Silver layer.
- load_gold → Loads data into the Gold layer.

1.2.4 Summary of Naming Standards

Object Type	Naming Pattern	Example
Bronze Table	<source_system>_<entity>	crm_customer_info
Silver Table	<source_system>_<entity>	erp_sales_order_clean
Gold Table	<category>_<entity>	fact_sales, dim_customer
Surrogate Key Column	<entity>_key	customer_key
Technical Column	dwh_<column_name>	dwh_load_date
Stored Procedure	load_<layer>	load_gold

1.3 Project Initialization

```sql
/*
=============================================================
Create Database and Schemas
=============================================================
Script Purpose:
    This script creates a new database named 'DataWarehouse' after checking if it already exists.
    If the database exists, it is dropped and recreated. Additionally, the script sets up three schemas
    within the database: 'bronze', 'silver', and 'gold'.

WARNING:
    Running this script will drop the entire 'DataWarehouse' database if it exists.
    All data in the database will be permanently deleted. Proceed with caution
    and ensure you have proper backups before running this script.
*/

USE master;
GO

-- Drop and recreate the 'DataWarehouse' database
IF EXISTS (SELECT 1 FROM sys.databases WHERE name = 'DataWarehouse')
BEGIN
    ALTER DATABASE DataWarehouse SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
    DROP DATABASE DataWarehouse;
END;
GO

-- Create the 'DataWarehouse' database
CREATE DATABASE DataWarehouse;
GO

USE DataWarehouse;
GO

-- Create Schemas
CREATE SCHEMA bronze;
GO

CREATE SCHEMA silver;
GO

CREATE SCHEMA gold;
GO
```

We create three seperate schemas other than defauld schema (dbo)
Where, schema is a logical container inside a database that group related objects like tables, views, and procedures.

- It helps organize, secure, and separate data logically within the same database.
- Each schema can have different permissions and prevents naming conflicts.

1.4 Build the Bronze Layer

1.4.1 Create Bronze Tables

```
/*
===============================================================================
DDL Script: Create Bronze Tables
===============================================================================
Script Purpose:
    This script creates tables in the 'bronze' schema, dropping existing tables
    if they already exist.
          Run this script to re-define the DDL structure of 'bronze' Tables
===============================================================================
*/

IF OBJECT_ID('bronze.crm_cust_info', 'U') IS NOT NULL
    DROP TABLE bronze.crm_cust_info;
GO

CREATE TABLE bronze.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
);
GO

IF OBJECT_ID('bronze.crm_prd_info', 'U') IS NOT NULL
    DROP TABLE bronze.crm_prd_info;
GO

CREATE TABLE bronze.crm_prd_info (
    prd_id       INT,
    prd_key      NVARCHAR(50),
    prd_nm       NVARCHAR(50),
    prd_cost     INT,
    prd_line     NVARCHAR(50),
    prd_start_dt DATETIME,
    prd_end_dt   DATETIME
);
GO

IF OBJECT_ID('bronze.crm_sales_details', 'U') IS NOT NULL
    DROP TABLE bronze.crm_sales_details;
GO

CREATE TABLE bronze.crm_sales_details (
    sls_ord_num  NVARCHAR(50),
    sls_prd_key  NVARCHAR(50),
    sls_cust_id  INT,
    sls_order_dt INT,
    sls_ship_dt  INT,
    sls_due_dt   INT,
    sls_sales    INT,
    sls_quantity INT,
    sls_price    INT
);
GO

IF OBJECT_ID('bronze.erp_loc_a101', 'U') IS NOT NULL
    DROP TABLE bronze.erp_loc_a101;
GO

CREATE TABLE bronze.erp_loc_a101 (
    cid    NVARCHAR(50),
    cntry  NVARCHAR(50)
);
GO

IF OBJECT_ID('bronze.erp_cust_az12', 'U') IS NOT NULL
    DROP TABLE bronze.erp_cust_az12;
GO
```

```sql
CREATE TABLE bronze.erp_cust_az12 (
    cid    NVARCHAR(50),
    bdate  DATE,
    gen    NVARCHAR(50)
);
GO

IF OBJECT_ID('bronze.erp_px_cat_g1v2', 'U') IS NOT NULL
    DROP TABLE bronze.erp_px_cat_g1v2;
GO

CREATE TABLE bronze.erp_px_cat_g1v2 (
    id           NVARCHAR(50),
    cat          NVARCHAR(50),
    subcat       NVARCHAR(50),
    maintenance  NVARCHAR(50)
);
GO
```

Table: bronze.crm_cust_info

Column Name	Data Type	Description
cst_id	INT	Unique identifier for each customer record
cst_key	NVARCHAR(50)	Unique customer key from source system
cst_firstname	NVARCHAR(50)	Customer's first name
cst_lastname	NVARCHAR(50)	Customer's last name
cst_marital_status	NVARCHAR(50)	Customer's marital status (e.g., Single, Married)
cst_gndr	NVARCHAR(50)	Customer gender
cst_create_date	DATE	Date the customer record was created

Table: bronze.crm_prd_info

Column Name	Data Type	Description
prd_id	INT	Unique identifier for each product
prd_key	NVARCHAR(50)	Unique product key from source system
prd_nm	NVARCHAR(50)	Product name
prd_cost	INT	Product cost or base price
prd_line	NVARCHAR(50)	Product line or category grouping
prd_start_dt	DATETIME	Date and time when product became active
prd_end_dt	DATETIME	Date and time when product was discontinued or ended

Table: bronze.crm_sales_details

Column Name	Data Type	Description
sls_ord_num	NVARCHAR(50)	Unique sales order number
sls_prd_key	NVARCHAR(50)	Product key associated with the order
sls_cust_id	INT	Customer ID associated with the order
sls_order_dt	INT	Order date (stored as integer or date key)
sls_ship_dt	INT	Shipping date (stored as integer or date key)
sls_due_dt	INT	Due date (stored as integer or date key)
sls_sales	INT	Total sales amount for the order
sls_quantity	INT	Quantity of products sold
sls_price	INT	Price per unit at time of sale

Table: bronze.erp_loc_a101

Column Name	Data Type	Description
cid	NVARCHAR(50)	Unique location or customer identifier
cntry	NVARCHAR(50)	Country name or code associated with the record

Table: bronze.erp_cust_az12

Column Name	Data Type	Description
cid	NVARCHAR(50)	Unique customer identifier
bdate	DATE	Customer's birth date
gen	NVARCHAR(50)	Customer gender information from ERP system

Table: bronze.erp_px_cat_g1v2

Column Name	Data Type	Description
id	NVARCHAR(50)	Unique product or category identifier
cat	NVARCHAR(50)	Product category
subcat	NVARCHAR(50)	Product subcategory
maintenance	NVARCHAR(50)	Maintenance type or flag for the product category

1.4.2 Load the Data into Bronze Tables

This stored procedure automates the data loading process into the Bronze layer of the Data Warehouse.It performs ETL ingestion from external CSV files into SQL Server tables under the bronze schema.

What it does

- ☑ Truncates existing Bronze tables to remove old data.
- ☑ Uses BULK INSERT commands to efficiently load new data from CSV files located in the project's dataset directories (source_crm and source_erp).
- ☑ Loads multiple source systems:
  - CRM data (cust_info, prd_info, sales_details)
  - ERP data (loc_a101, cust_az12, px_cat_g1v2)
- ☑ Tracks and prints start and end times for each table load and the total batch duration.
- ☑ Includes error handling to print detailed messages if any step fails.

```
/*
===============================================================================
Stored Procedure: Load Bronze Layer (Source -> Bronze)
===============================================================================
Script Purpose:
    This stored procedure loads data into the 'bronze' schema from external CSV files.
    It performs the following actions:
    - Truncates the bronze tables before loading data.
    - Uses the `BULK INSERT` command to load data from csv Files to bronze tables.

Parameters:
    None.
            This stored procedure does not accept any parameters or return any values.

Usage Example:
    EXEC bronze.load_bronze;
===============================================================================
*/
CREATE OR ALTER PROCEDURE bronze.load_bronze AS
BEGIN
          DECLARE @start_time DATETIME, @end_time DATETIME, @batch_start_time DATETIME, @batch_end_time DATETIME;
          BEGIN TRY
                    SET @batch_start_time = GETDATE();
                    PRINT '================================================';
                    PRINT 'Loading Bronze Layer';
                    PRINT '================================================';

                    PRINT '------------------------------------------------';
                    PRINT 'Loading CRM Tables';
                    PRINT '------------------------------------------------';

                    SET @start_time = GETDATE();
                    PRINT '>> Truncating Table: bronze.crm_cust_info';
                    TRUNCATE TABLE bronze.crm_cust_info;
                    PRINT '>> Inserting Data Into: bronze.crm_cust_info';
                    BULK INSERT bronze.crm_cust_info
                    FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
                    WITH (
                              FIRSTROW = 2,
                              FIELDTERMINATOR = ',',
                              TABLOCK
                    );
                    SET @end_time = GETDATE();
                    PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
                    PRINT '>> -------------';
                   SET @start_time = GETDATE();
                    PRINT '>> Truncating Table: bronze.crm_prd_info';
                    TRUNCATE TABLE bronze.crm_prd_info;

                    PRINT '>> Inserting Data Into: bronze.crm_prd_info';
                    BULK INSERT bronze.crm_prd_info
                    FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_crm\prd_info.csv'
                    WITH (
                              FIRSTROW = 2,
                              FIELDTERMINATOR = ',',
                              TABLOCK
                    );
                    SET @end_time = GETDATE();
                    PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
                    PRINT '>> -------------';

                     SET @start_time = GETDATE();
                    PRINT '>> Truncating Table: bronze.crm_sales_details';
                    TRUNCATE TABLE bronze.crm_sales_details;
                    PRINT '>> Inserting Data Into: bronze.crm_sales_details';
                    BULK INSERT bronze.crm_sales_details
                    FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_crm\sales_details.csv'
                    WITH (
                              FIRSTROW = 2,
                              FIELDTERMINATOR = ',',
                              TABLOCK
                    );
                    SET @end_time = GETDATE();
```

```
		PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
		PRINT '>> -------------';

		PRINT '------------------------------------------------';
		PRINT 'Loading ERP Tables';
		PRINT '------------------------------------------------';

		SET @start_time = GETDATE();
		PRINT '>> Truncating Table: bronze.erp_loc_a101';
		TRUNCATE TABLE bronze.erp_loc_a101;
		PRINT '>> Inserting Data Into: bronze.erp_loc_a101';
		BULK INSERT bronze.erp_loc_a101
		FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_erp\loc_a101.csv'
		WITH (
			FIRSTROW = 2,
			FIELDTERMINATOR = ',',
			TABLOCK
		);
		SET @end_time = GETDATE();
		PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
		PRINT '>> -------------';

		SET @start_time = GETDATE();
		PRINT '>> Truncating Table: bronze.erp_cust_az12';
		TRUNCATE TABLE bronze.erp_cust_az12;
		PRINT '>> Inserting Data Into: bronze.erp_cust_az12';
		BULK INSERT bronze.erp_cust_az12
		FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_erp\cust_az12.csv'
		WITH (
			FIRSTROW = 2,
			FIELDTERMINATOR = ',',
			TABLOCK
		);
		SET @end_time = GETDATE();
		PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
		PRINT '>> -------------';

		SET @start_time = GETDATE();
		PRINT '>> Truncating Table: bronze.erp_px_cat_g1v2';
		TRUNCATE TABLE bronze.erp_px_cat_g1v2;
		PRINT '>> Inserting Data Into: bronze.erp_px_cat_g1v2';
		BULK INSERT bronze.erp_px_cat_g1v2
		FROM 'C:\Users\Panduka Bandara\OneDrive\Documents\sql-data-warehouse-project\datasets\source_erp\px_cat_g1v2.csv'
		WITH (
			FIRSTROW = 2,
			FIELDTERMINATOR = ',',
			TABLOCK
		);
		SET @end_time = GETDATE();
		PRINT '>> Load Duration: ' + CAST(DATEDIFF(second, @start_time, @end_time) AS NVARCHAR) + ' seconds';
		PRINT '>> -------------';

		SET @batch_end_time = GETDATE();
		PRINT '=========================================='
		PRINT 'Loading Bronze Layer is Completed';
	PRINT '   - Total Load Duration: ' + CAST(DATEDIFF(SECOND, @batch_start_time, @batch_end_time) AS NVARCHAR) + ' seconds';
		PRINT '=========================================='
	END TRY
	BEGIN CATCH
		PRINT '=========================================='
		PRINT 'ERROR OCCURED DURING LOADING BRONZE LAYER'
		PRINT 'Error Message' + ERROR_MESSAGE();
		PRINT 'Error Message' + CAST (ERROR_NUMBER() AS NVARCHAR);
		PRINT 'Error Message' + CAST (ERROR_STATE() AS NVARCHAR);
		PRINT '=========================================='
	END CATCH
END
```

1.5 Build Silver Layer

1.5.1 Create Silver Tables

```sql
/*
===============================================================================
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 'bronze' Tables
===============================================================================
*/

IF OBJECT_ID('silver.crm_cust_info', 'U') IS NOT NULL
    DROP TABLE silver.crm_cust_info;
GO

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

IF OBJECT_ID('silver.crm_prd_info', 'U') IS NOT NULL
    DROP TABLE silver.crm_prd_info;
GO

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()
);
GO

IF OBJECT_ID('silver.crm_sales_details', 'U') IS NOT NULL
    DROP TABLE silver.crm_sales_details;
GO

CREATE TABLE silver.crm_sales_details (
    sls_ord_num     NVARCHAR(50),
    sls_prd_key     NVARCHAR(50),
    sls_cust_id     INT,
    sls_order_dt    DATE,
    sls_ship_dt     DATE,
    sls_due_dt      DATE,
    sls_sales       INT,
    sls_quantity    INT,
    sls_price       INT,
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
GO

IF OBJECT_ID('silver.erp_loc_a101', 'U') IS NOT NULL
    DROP TABLE silver.erp_loc_a101;
GO

CREATE TABLE silver.erp_loc_a101 (
    cid             NVARCHAR(50),
    cntry           NVARCHAR(50),
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
```

```
GO

IF OBJECT_ID('silver.erp_cust_az12', 'U') IS NOT NULL
    DROP TABLE silver.erp_cust_az12;
GO

CREATE TABLE silver.erp_cust_az12 (
    cid             NVARCHAR(50),
    bdate           DATE,
    gen             NVARCHAR(50),
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
GO

IF OBJECT_ID('silver.erp_px_cat_g1v2', 'U') IS NOT NULL
    DROP TABLE silver.erp_px_cat_g1v2;
GO

CREATE TABLE silver.erp_px_cat_g1v2 (
    id              NVARCHAR(50),
    cat             NVARCHAR(50),
    subcat          NVARCHAR(50),
    maintenance     NVARCHAR(50),
    dwh_create_date DATETIME2 DEFAULT GETDATE()
);
GO
```

1.5.2 Identify Quality Issues

Check for Duplicates for Possible PKs

```
-- Check for Nulls
SELECT
cst_id,
COUNT(*)
FROM bronze.crm_cust_info
GROUP BY cst_id
HAVING COUNT(*) > 1 OR cst_id IS NULL
```

Results | Messages

	cst_id	(No column name)
1	29449	2
2	29473	2
3	29433	2
4	NULL	3
5	29483	2
6	29466	3

Check what caused the duplicates

```
-- Check the reasons
SELECT *
FROM bronze.crm_cust_info
WHERE cst_id = 29466
```

Results | Messages

	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

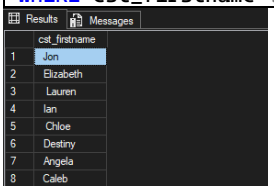
So, the reason for duplicates is some data are old and they also included. Need to rank and get most recently updated values.

```
 SELECT
*,
ROW_NUMBER() OVER(PARTITION BY cst_id ORDER BY cst_create_date DESC) as flag_last
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	NULL	1
2	NULL	SF566	NULL	NULL	NULL	NULL	NULL	2
3	NULL	13451235	NULL	NULL	NULL	NULL	NULL	3

Check for Unwanted Spaces

```sql
SELECT cst_firstname
FROM bronze.crm_cust_info
WHERE cst_firstname != TRIM(cst_firstname)
```



Results | Messages

	cst_firstname
1	Jon
2	Elizabeth
3	Lauren
4	Ian
5	Chloe
6	Destiny
7	Angela
8	Caleb

Do the same for last name & all string values.

Overall

```sql
/*
===============================================================================
Quality Checks
===============================================================================
Script Purpose:
    This script performs various quality checks for data consistency, accuracy,
    and standardization across the 'silver' layer. It includes checks for:
    - Null or duplicate primary keys.
    - Unwanted spaces in string fields.
    - Data standardization and consistency.
    - Invalid date ranges and orders.
    - Data consistency between related fields.

Usage Notes:
    - Run these checks after data loading Silver Layer.
    - Investigate and resolve any discrepancies found during the checks.
===============================================================================
*/

-- ====================================================================
-- Checking 'silver.crm_cust_info'
-- ====================================================================
-- Check for NULLs or Duplicates in Primary Key
-- Expectation: No Results
SELECT
    cst_id,
    COUNT(*)
FROM silver.crm_cust_info
GROUP BY cst_id
HAVING COUNT(*) > 1 OR cst_id IS NULL;

-- Check for Unwanted Spaces
-- Expectation: No Results
SELECT
    cst_key
FROM silver.crm_cust_info
WHERE cst_key != TRIM(cst_key);

-- Data Standardization & Consistency
SELECT DISTINCT
    cst_marital_status
FROM silver.crm_cust_info;

-- ====================================================================
-- Checking 'silver.crm_prd_info'
-- ====================================================================
-- Check for NULLs or Duplicates in Primary Key
-- Expectation: No Results
SELECT
    prd_id,
    COUNT(*)
FROM silver.crm_prd_info
GROUP BY prd_id
HAVING COUNT(*) > 1 OR prd_id IS NULL;

-- Check for Unwanted Spaces
-- Expectation: No Results
SELECT
```

```sql
    prd_nm
FROM silver.crm_prd_info
WHERE prd_nm != TRIM(prd_nm);

-- Check for NULLs or Negative Values in Cost
-- Expectation: No Results
SELECT
    prd_cost
FROM silver.crm_prd_info
WHERE prd_cost < 0 OR prd_cost IS NULL;

-- Data Standardization & Consistency
SELECT DISTINCT
    prd_line
FROM silver.crm_prd_info;

-- Check for Invalid Date Orders (Start Date > End Date)
-- Expectation: No Results
SELECT
    *
FROM silver.crm_prd_info
WHERE prd_end_dt < prd_start_dt;

-- ====================================================================
-- Checking 'silver.crm_sales_details'
-- ====================================================================
-- Check for Invalid Dates
-- Expectation: No Invalid Dates
SELECT
    NULLIF(sls_due_dt, 0) AS sls_due_dt
FROM bronze.crm_sales_details
WHERE sls_due_dt <= 0
    OR LEN(sls_due_dt) != 8
    OR sls_due_dt > 20500101
    OR sls_due_dt < 19000101;

-- Check for Invalid Date Orders (Order Date > Shipping/Due Dates)
-- Expectation: No Results
SELECT
    *
FROM silver.crm_sales_details
WHERE sls_order_dt > sls_ship_dt
   OR sls_order_dt > sls_due_dt;

-- Check Data Consistency: Sales = Quantity * Price
-- Expectation: No Results
SELECT DISTINCT
    sls_sales,
    sls_quantity,
    sls_price
FROM silver.crm_sales_details
WHERE sls_sales != sls_quantity * sls_price
   OR sls_sales IS NULL
   OR sls_quantity IS NULL
   OR sls_price IS NULL
   OR sls_sales <= 0
   OR sls_quantity <= 0
   OR sls_price <= 0
ORDER BY sls_sales, sls_quantity, sls_price;

-- ====================================================================
-- Checking 'silver.erp_cust_az12'
-- ====================================================================
-- Identify Out-of-Range Dates
-- Expectation: Birthdates between 1924-01-01 and Today
SELECT DISTINCT
    bdate
FROM silver.erp_cust_az12
WHERE bdate < '1924-01-01'
   OR bdate > GETDATE();

-- Data Standardization & Consistency
SELECT DISTINCT
    gen
FROM silver.erp_cust_az12;

-- ====================================================================
```

```sql
-- Checking 'silver.erp_loc_a101'
-- ====================================================================
-- Data Standardization & Consistency
SELECT DISTINCT
    cntry
FROM silver.erp_loc_a101
ORDER BY cntry;

-- ====================================================================
-- Checking 'silver.erp_px_cat_g1v2'
-- ====================================================================
-- Check for Unwanted Spaces
-- Expectation: No Results
SELECT
    *
FROM silver.erp_px_cat_g1v2
WHERE cat != TRIM(cat)
   OR subcat != TRIM(subcat)
   OR maintenance != TRIM(maintenance);

-- Data Standardization & Consistency
SELECT DISTINCT
    maintenance
FROM silver.erp_px_cat_g1v2;
```

1.5.3 Load Silver Layer

```sql
/*
===============================================================================
Stored Procedure: Load Silver Layer (Bronze -> Silver)
===============================================================================
Script Purpose:
    This stored procedure performs the ETL (Extract, Transform, Load) process to
    populate the 'silver' schema tables from the 'bronze' schema.
        Actions Performed:
                - Truncates Silver tables.
                - Inserts transformed and cleansed data from Bronze into Silver tables.

Parameters:
    None.
          This stored procedure does not accept any parameters or return any values.

Usage Example:
    EXEC Silver.load_silver;
===============================================================================
*/
CREATE OR ALTER PROCEDURE silver.load_silver AS
BEGIN
    DECLARE @start_time DATETIME, @end_time DATETIME, @batch_start_time DATETIME, @batch_end_time DATETIME;
    BEGIN TRY
        SET @batch_start_time = GETDATE();
        PRINT '================================================';
        PRINT 'Loading Silver Layer';
        PRINT '================================================';

                PRINT '------------------------------------------------';
                PRINT 'Loading CRM Tables';
                PRINT '------------------------------------------------';

                -- Loading silver.crm_cust_info
        SET @start_time = GETDATE();
                PRINT '>> Truncating Table: silver.crm_cust_info';
                TRUNCATE TABLE silver.crm_cust_info;
                PRINT '>> Inserting Data Into: silver.crm_cust_info';
                INSERT INTO silver.crm_cust_info(
                        cst_id,
                        cst_key,
                        cst_firstname,
                        cst_lastname,
                        cst_marital_status,
                        cst_gndr,
                        cst_create_date
                )
                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 AS cst_marital_status, -- Normalize marital status values to readable format
                CASE
                        WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
                        WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
                        ELSE 'n/a'
                END AS cst_gndr, -- Normalize gender values to readable format
                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; -- Select the most recent record per customer
        SET @end_time = GETDATE();
PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + ' seconds';
PRINT '>> -------------';

        -- Loading silver.crm_prd_info
SET @start_time = GETDATE();
        PRINT '>> Truncating Table: silver.crm_prd_info';
        TRUNCATE TABLE silver.crm_prd_info;
        PRINT '>> Inserting Data Into: silver.crm_prd_info';
        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
                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
                        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(
                        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;
SET @end_time = GETDATE();
PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + ' seconds';
PRINT '>> -------------';

        -- Loading crm_sales_details
SET @start_time = GETDATE();
        PRINT '>> Truncating Table: silver.crm_sales_details';
        TRUNCATE TABLE silver.crm_sales_details;
        PRINT '>> Inserting Data Into: silver.crm_sales_details';
        INSERT INTO silver.crm_sales_details (
                sls_ord_num,
                sls_prd_key,
                sls_cust_id,
                sls_order_dt,
                sls_ship_dt,
                sls_due_dt,
```

```
                        sls_sales,
                        sls_quantity,
                        sls_price
                )
                SELECT
                        sls_ord_num,
                        sls_prd_key,
                        sls_cust_id,
                        CASE
                                WHEN sls_order_dt = 0 OR LEN(sls_order_dt) != 8 THEN NULL
                                ELSE CAST(CAST(sls_order_dt AS VARCHAR) AS DATE)
                        END AS sls_order_dt,
                        CASE
                                WHEN sls_ship_dt = 0 OR LEN(sls_ship_dt) != 8 THEN NULL
                                ELSE CAST(CAST(sls_ship_dt AS VARCHAR) AS DATE)
                        END AS sls_ship_dt,
                        CASE
                                WHEN sls_due_dt = 0 OR LEN(sls_due_dt) != 8 THEN NULL
                                ELSE CAST(CAST(sls_due_dt AS VARCHAR) AS DATE)
                        END AS sls_due_dt,
                        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, -- Recalculate sales if original value is missing or incorrect
                        sls_quantity,
                        CASE
                                WHEN sls_price IS NULL OR sls_price <= 0
                                        THEN sls_sales / NULLIF(sls_quantity, 0)
                                ELSE sls_price  -- Derive price if original value is invalid
                        END AS sls_price
                FROM bronze.crm_sales_details;
        SET @end_time = GETDATE();
        PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + ' seconds';
        PRINT '>> -------------';

        -- Loading erp_cust_az12
        SET @start_time = GETDATE();
                PRINT '>> Truncating Table: silver.erp_cust_az12';
                TRUNCATE TABLE silver.erp_cust_az12;
                PRINT '>> Inserting Data Into: silver.erp_cust_az12';
                INSERT INTO silver.erp_cust_az12 (
                        cid,
                        bdate,
                        gen
                )
                SELECT
                        CASE
                                WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid, 4, LEN(cid)) -- Remove 'NAS' prefix if
present
                                ELSE cid
                        END AS cid,
                        CASE
                                WHEN bdate > GETDATE() THEN NULL
                                ELSE bdate
                        END AS bdate, -- Set future birthdates to NULL
                        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 -- Normalize gender values and handle unknown cases
                FROM bronze.erp_cust_az12;
            SET @end_time = GETDATE();
        PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + ' seconds';
        PRINT '>> -------------';

                PRINT '------------------------------------------------';
                PRINT 'Loading ERP Tables';
                PRINT '------------------------------------------------';

        -- Loading erp_loc_a101
        SET @start_time = GETDATE();
                PRINT '>> Truncating Table: silver.erp_loc_a101';
                TRUNCATE TABLE silver.erp_loc_a101;
                PRINT '>> Inserting Data Into: silver.erp_loc_a101';
                INSERT INTO silver.erp_loc_a101 (
```

```
                        cid,
                        cntry
                )
                SELECT
                        REPLACE(cid, '-', '') AS cid,
                        CASE
                                WHEN TRIM(cntry) = 'DE' THEN 'Germany'
                                WHEN TRIM(cntry) IN ('US', 'USA') THEN 'United States'
                                WHEN TRIM(cntry) = '' OR cntry IS NULL THEN 'n/a'
                                ELSE TRIM(cntry)
                        END AS cntry -- Normalize and Handle missing or blank country codes
                FROM bronze.erp_loc_a101;
            SET @end_time = GETDATE();
        PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + ' seconds';
        PRINT '>> -------------';

                -- Loading erp_px_cat_g1v2
                SET @start_time = GETDATE();
                PRINT '>> Truncating Table: silver.erp_px_cat_g1v2';
                TRUNCATE TABLE silver.erp_px_cat_g1v2;
                PRINT '>> Inserting Data Into: silver.erp_px_cat_g1v2';
                INSERT INTO silver.erp_px_cat_g1v2 (
                        id,
                        cat,
                        subcat,
                        maintenance
                )
                SELECT
                        id,
                        cat,
                        subcat,
                        maintenance
                FROM bronze.erp_px_cat_g1v2;
                SET @end_time = GETDATE();
                PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS NVARCHAR) + '
seconds';
        PRINT '>> -------------';

                SET @batch_end_time = GETDATE();
                PRINT '==========================================='
                PRINT 'Loading Silver Layer is Completed';
        PRINT '   - Total Load Duration: ' + CAST(DATEDIFF(SECOND, @batch_start_time, @batch_end_time) AS
NVARCHAR) + ' seconds';
                PRINT '==========================================='

        END TRY
        BEGIN CATCH
                PRINT '==========================================='
                PRINT 'ERROR OCCURED DURING LOADING BRONZE LAYER'
                PRINT 'Error Message' + ERROR_MESSAGE();
                PRINT 'Error Message' + CAST (ERROR_NUMBER() AS NVARCHAR);
                PRINT 'Error Message' + CAST (ERROR_STATE() AS NVARCHAR);
                PRINT '==========================================='
        END CATCH
END
```

The Silver Layer in a data warehouse represents cleansed, standardized, and structured data derived from the Bronze (raw) layer.

Table: silver.crm_cust_info

Column Name	Data Type	Description
cst_id	INT	Cleaned and validated customer ID
cst_key	NVARCHAR(50)	Standardized customer key
cst_firstname	NVARCHAR(50)	Trimmed and properly cased first name
cst_lastname	NVARCHAR(50)	Trimmed and properly cased last name

cst_marital_status	NVARCHAR(50)	Normalized marital status values
cst_gndr	NVARCHAR(50)	Standardized gender value (e.g., 'M', 'F', 'Other')
cst_create_date	DATE	Converted to proper date format from raw Bronze data
dwh_create_date	DATETIME2	Timestamp of when the record was loaded into Silver layer

Changes applied

- ☑ Added: dwh_create_date
- ☑ Converted: cst_create_date from possible raw string/int to DATE
- ☑ Duplicates resolved using ROW_NUMBER() (keeps latest record per customer).
- ☑ Trimmed whitespace.
- ☑ Coded values for gender and marital status converted to readable text.

Table: silver.crm_prd_info

Column Name	Data Type	Description
prd_id	INT	Cleaned product identifier
cat_id	NVARCHAR(50)	Derived category ID (joined/enriched from ERP category table)
prd_key	NVARCHAR(50)	Standardized product key
prd_nm	NVARCHAR(50)	Normalized product name
prd_cost	INT	Validated product cost
prd_line	NVARCHAR(50)	Product line classification
prd_start_dt	DATE	Standardized start date (converted from DATETIME)
prd_end_dt	DATE	Standardized end date (converted from DATETIME)
dwh_create_date	DATETIME2	ETL load timestamp

Changes

- ☑ Converted: Start/end dates to DATE
- ☑ Added: dwh_create_date
- ☑ `cat_id` and `prd_key` extracted from `prd_key`.
- ☑ Product line codes mapped to full names.
- ☑ `prd_end_dt` dynamically computed.

Table: silver.crm_sales_details

Column Name	Data Type	Description
sls_ord_num	NVARCHAR(50)	Standardized order number
sls_prd_key	NVARCHAR(50)	Product key reference (validated against product master)

sls_cust_id	INT	Validated customer ID
sls_order_dt	DATE	Converted from raw integer (e.g., YYYYMMDD) to proper date
sls_ship_dt	DATE	Converted from raw integer (e.g., YYYYMMDD) to proper date
sls_due_dt	DATE	Converted from raw integer (e.g., YYYYMMDD) to proper date
sls_sales	INT	Validated total sales amount
sls_quantity	INT	Cleaned quantity value
sls_price	INT	Standardized product price
dwh_create_date	DATETIME2	Timestamp when record was inserted into Silver

Changes

- ☑ Converted All date columns (sls_order_dt, sls_ship_dt, sls_due_dt) from INT to DATE
- ☑ Added: dwh_create_date
- ☑ Validated relationships with crm_prd_info and crm_cust_info
- ☑ Invalid date values replaced with `NULL`.
- ☑ Sales and price corrected if inconsistent or negative.

Table: silver.erp_loc_a101

Column Name	Data Type	Description
cid	NVARCHAR(50)	Location or customer ID
cntry	NVARCHAR(50)	Standardized country name (e.g., “United States” instead of “US”)
dwh_create_date	DATETIME2	Timestamp of ETL load into Silver

Changes

- ☑ Added: dwh_create_date
- ☑ Standardized country names or codes for consistency
- ☑ Replaced hyphens in cid.
- ☑ Country codes standardized to full names.

Table: silver.erp_cust_az12

Column Name	Data Type	Description
cid	NVARCHAR(50)	Customer identifier (cleaned)
bdate	DATE	Standardized birth date
gen	NVARCHAR(50)	Normalized gender (M/F/Other)
dwh_create_date	DATETIME2	Timestamp of ETL load into Silver

Changes

- ☑ Added: dwh_create_date
- ☑ Ensured proper DATE format for birthdate
- ☑ Prefix cleanup for cid.
- ☑ Future birthdates nullified.
- ☑ Gender values standardized.

Table: silver.erp_px_cat_g1v2

Column Name	Data Type	Description
id	NVARCHAR(50)	Product or category ID
cat	NVARCHAR(50)	Cleaned category name
subcat	NVARCHAR(50)	Cleaned subcategory name
maintenance	NVARCHAR(50)	Maintenance flag or type (standardized)
dwh_create_date	DATETIME2	ETL load timestamp

Changes

- ☑ Added: dwh_create_date
- ☑ Direct copy (no transformation applied).

1.6 Gold Layer

1.6.1 Data Catalog of the Gold Layer

The Gold Layer serves as the business-level data representation, designed to support analytical and reporting use cases. It comprises dimension tables and fact tables that store data for specific business metrics.

1.6.1.a gold.dim_customers

Contains enriched customer details, combining demographic and geographic attributes.

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 used for customer tracking and referencing.
first_name	NVARCHAR(50)	The customer's first name.
last_name	NVARCHAR(50)	The customer's last or family name.
country	NVARCHAR(50)	The country of residence of the customer (e.g., 'Australia').
marital_status	NVARCHAR(50)	The marital status of the customer (e.g., 'Married', 'Single').
gender	NVARCHAR(50)	The gender of the customer (e.g., 'Male', 'Female', 'n/a').
birthdate	DATE	The date of birth of the customer, formatted as YYYY-MM-DD (e.g., 1971-10-06).
create_date	DATE	The date and time when the customer record was created in the system.

1.6.1.b gold.dim_products

Provides detailed information about products and their related attributes.

Column Name	Data Type	Description
product_key	INT	Surrogate key uniquely identifying each product record in the dimension table.
product_id	INT	Unique identifier assigned to the product for internal tracking and reference.
product_number	NVARCHAR(50)	Alphanumeric code representing the product, used for categorization or inventory.
product_name	NVARCHAR(50)	Descriptive name of the product, including key details such as type, color, or size.
category_id	NVARCHAR(50)	Unique identifier for the product's category, linking it to its classification.
category	NVARCHAR(50)	Broad classification of the product (e.g., Bikes, Components).
subcategory	NVARCHAR(50)	Detailed classification within the product category (e.g., product type).
maintenance_required	NVARCHAR(50)	Indicates whether the product requires maintenance (e.g., 'Yes', 'No').
cost	INT	The base price or cost of the product, measured in monetary units.
product_line	NVARCHAR(50)	The product line or series to which the product belongs (e.g., Road, Mountain).
start_date	DATE	The date the product became available for sale or use.

1.6.1.c gold.fact_sales

Contains transactional sales data used for analytical and performance reporting.

Column Name	Data Type	Description
order_number	NVARCHAR(50)	Unique alphanumeric identifier for each sales order (e.g., 'SO54496').
product_key	INT	Surrogate key linking the sales order to the product dimension.
customer_key	INT	Surrogate key linking the sales order to the customer dimension.
order_date	DATE	The date when the sales order was placed.
shipping_date	DATE	The date when the order was shipped to the customer.
due_date	DATE	The date when the payment for the order was due.
sales_amount	INT	Total monetary value of the sale for the line item, in whole currency units.
quantity	INT	Number of units of the product ordered in the line item.
price	INT	Price per unit of the product for the line item, in whole currency units.

1.6.2 Gold Layer

```
/*
===============================================================================
DDL Script: Create Gold Views
===============================================================================
Script Purpose:
    This script creates views for the Gold layer in the data warehouse.
    The Gold layer represents the final dimension and fact tables (Star Schema)

    Each view performs transformations and combines data from the Silver layer
    to produce a clean, enriched, and business-ready dataset.

Usage:
    - These views can be queried directly for analytics and reporting.
===============================================================================
*/

-- =============================================================================
-- Create Dimension: gold.dim_customers
-- =============================================================================
IF OBJECT_ID('gold.dim_customers', 'V') IS NOT NULL
    DROP VIEW gold.dim_customers;
GO

CREATE VIEW gold.dim_customers AS
SELECT
    ROW_NUMBER() OVER (ORDER BY cst_id) AS customer_key, -- Surrogate key
    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.cntry                           AS country,
    ci.cst_marital_status              AS marital_status,
    CASE
        WHEN ci.cst_gndr != 'n/a' THEN ci.cst_gndr -- CRM is the primary source for gender
        ELSE COALESCE(ca.gen, 'n/a')                  -- Fallback to ERP data
    END                                AS gender,
    ca.bdate                           AS birthdate,
    ci.cst_create_date                 AS create_date
FROM silver.crm_cust_info ci
LEFT JOIN silver.erp_cust_az12 ca
    ON ci.cst_key = ca.cid
LEFT JOIN silver.erp_loc_a101 la
    ON ci.cst_key = la.cid;
```

```sql
GO

-- =============================================================================
-- Create Dimension: gold.dim_products
-- =============================================================================
IF OBJECT_ID('gold.dim_products', 'V') IS NOT NULL
    DROP VIEW gold.dim_products;
GO

CREATE VIEW gold.dim_products AS
SELECT
    ROW_NUMBER() OVER (ORDER BY pn.prd_start_dt, pn.prd_key) AS product_key, -- Surrogate 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  AS 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
LEFT JOIN silver.erp_px_cat_g1v2 pc
    ON pn.cat_id = pc.id
WHERE pn.prd_end_dt IS NULL; -- Filter out all historical data
GO

-- =============================================================================
-- Create Fact Table: gold.fact_sales
-- =============================================================================
IF OBJECT_ID('gold.fact_sales', 'V') IS NOT NULL
    DROP VIEW gold.fact_sales;
GO

CREATE VIEW gold.fact_sales AS
SELECT
    sd.sls_ord_num  AS order_number,
    pr.product_key  AS product_key,
    cu.customer_key AS 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_products pr
    ON sd.sls_prd_key = pr.product_number
LEFT JOIN gold.dim_customers cu
    ON sd.sls_cust_id = cu.customer_id;
GO
```

1.6.3 Gold Layer Quality Checks

```sql
/*
===============================================================================
Quality Checks
===============================================================================
Script Purpose:
    This script performs quality checks to validate the integrity, consistency,
    and accuracy of the Gold Layer. These checks ensure:
    - Uniqueness of surrogate keys in dimension tables.
    - Referential integrity between fact and dimension tables.
    - Validation of relationships in the data model for analytical purposes.

Usage Notes:
    - Investigate and resolve any discrepancies found during the checks.
===============================================================================
*/

-- ====================================================================
-- Checking 'gold.dim_customers'
-- ====================================================================
-- Check for Uniqueness of Customer Key in gold.dim_customers
-- Expectation: No results
SELECT
    customer_key,
    COUNT(*) AS duplicate_count
FROM gold.dim_customers
GROUP BY customer_key
HAVING COUNT(*) > 1;

-- ====================================================================
-- Checking 'gold.product_key'
-- ====================================================================
-- Check for Uniqueness of Product Key in gold.dim_products
-- Expectation: No results
SELECT
    product_key,
    COUNT(*) AS duplicate_count
FROM gold.dim_products
GROUP BY product_key
HAVING COUNT(*) > 1;

-- ====================================================================
-- Checking 'gold.fact_sales'
-- ====================================================================
-- Check the data model connectivity between fact and dimensions
SELECT *
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
ON c.customer_key = f.customer_key
LEFT JOIN gold.dim_products p
ON p.product_key = f.product_key
WHERE p.product_key IS NULL OR c.customer_key IS NULL
```

2. Exploratory Data Analysis & Advanced Analytics

2.1 Database Exploration

```sql
/*
===============================================================================
Database Exploration
===============================================================================
Purpose:
    - To explore the structure of the database, including the list of tables and their
schemas.
    - To inspect the columns and metadata for specific tables.

Table Used:
    - INFORMATION_SCHEMA.TABLES
    - INFORMATION_SCHEMA.COLUMNS
===============================================================================
*/

-- Retrieve a list of all tables in the database
SELECT
    TABLE_CATALOG,
    TABLE_SCHEMA,
    TABLE_NAME,
    TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES;

-- Retrieve all columns for a specific table (dim_customers)
SELECT
    COLUMN_NAME,
    DATA_TYPE,
    IS_NULLABLE,
    CHARACTER_MAXIMUM_LENGTH
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'dim_customers';
```

	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_loc_a101	BASE TABLE
5	DataWarehouse	bronze	erp_cust_az12	BASE TABLE
6	DataWarehouse	bronze	erp_px_cat_g1v2	BASE TABLE
7	DataWarehouse	silver	crm_cust_info	BASE TABLE
8	DataWarehouse	silver	crm_prd_info	BASE TABLE
9	DataWarehouse	silver	crm_sales_details	BASE TABLE
10	DataWarehouse	silver	erp_loc_a101	BASE TABLE
11	DataWarehouse	silver	erp_cust_az12	BASE TABLE
12	DataWarehouse	silver	erp_px_cat_g1v2	BASE TABLE
13	DataWarehouse	gold	dim_customers	VIEW
14	DataWarehouse	gold	dim_products	VIEW
15	DataWarehouse	gold	fact_sales	VIEW

	COLUMN_NAME	DATA_TYPE	IS_NULLABLE	CHARACTER_MAXIMUM_LENGTH
1	customer_key	bigint	YES	NULL
2	customer_id	int	YES	NULL
3	customer_number	nvarchar	YES	50
4	first_name	nvarchar	YES	50
5	last_name	nvarchar	YES	50
6	country	nvarchar	YES	50
7	marital_status	nvarchar	YES	50
8	gender	nvarchar	YES	50
9	birthdate	date	YES	NULL
10	create_date	date	YES	NULL

This focuses on **exploring and validating the database metadata** using the INFORMATION_SCHEMA views a standardized system view set available in SQL Server and most relational databases. These views help data engineers, analysts, and DBAs understand database objects without directly accessing system tables.

The script performs two primary operations

1. **List all tables in the database**
    - Uses INFORMATION_SCHEMA.TABLES to retrieve:
        - Database name (TABLE_CATALOG)
        - Schema (TABLE_SCHEMA)
        - Table name (TABLE_NAME)
        - Type (TABLE_TYPE, e.g., BASE TABLE, VIEW)
          [Verify that all tables (Bronze, Silver, Gold layers) are successfully created and categorized correctly.]

2. **Inspect column details for a specific table (dim_customers)**
    - Uses INFORMATION_SCHEMA.COLUMNS to display:
        - Column names and data types
        - Nullability constraints
        - Character maximum lengths (for string fields)
          [Confirm column definitions, validate schema alignment with design specifications, and prepare for data quality or ETL mappings.]

2.2 Dimensions Explorations

```sql
/*
===============================================================================
Dimensions Exploration
===============================================================================
Purpose:
    - To explore the structure of dimension tables.

SQL Functions Used:
    - DISTINCT
    - ORDER BY
===============================================================================
*/

-- Retrieve a list of unique countries from which customers originate
SELECT DISTINCT
    country
FROM gold.dim_customers
ORDER BY country;

-- Retrieve a list of unique categories, subcategories, and products
SELECT DISTINCT
    category,
    subcategory,
    product_name
FROM gold.dim_products
ORDER BY category, subcategory, product_name;
```

	country
1	Australia
2	Canada
3	France
4	Germany
5	n/a
6	United Kingdom
7	United States

	category	subcategory	product_name
1	NULL	NULL	HL Mountain Pedal
2	NULL	NULL	HL Road Pedal
3	NULL	NULL	LL Mountain Pedal
4	NULL	NULL	LL Road Pedal
5	NULL	NULL	ML Mountain Pedal
6	NULL	NULL	ML Road Pedal
7	NULL	NULL	Touring Pedal
8	Acces...	Bike Racks	Hitch Rack - 4-Bike
9	Acces...	Bike Stands	All-Purpose Bike ...
10	Acces...	Bottles an...	Mountain Bottle C...
11	Acces...	Bottles an...	Road Bottle Cage
12	Acces...	Bottles an...	Water Bottle - 30 ...
13	Acces...	Cleaners	Bike Wash - Diss...
14	Acces...	Fenders	Fender Set - Mou...
15	Acces...	Helmets	Sport-100 Helmet...
16	Acces...	Helmets	Sport-100 Helmet...
17	Acces...	Helmets	Sport-100 Helmet...
18	Acces...	Hydration ...	Hydration Pack - ...
19	Acces...	Lights	Headlights - Dual-...
20	Acces...	Lights	Headlights - Wea...
21	Acces...	Lights	Taillights - Battery...

This focuses on exploring the structure and diversity of data within the dimension tables in the Gold Layer.
Dimension exploration helps analysts and data engineers understand the available categories, hierarchies, and reference values that define the business context for facts (e.g., sales).

The script performs the following analyses

1. **Customer Geography Analysis**
    - Uses the DISTINCT and ORDER BY clauses on gold.dim_customers to list **all unique countries** where customers are located. [Identify the geographical spread of customers and validate if country data is complete and standardized (e.g., no duplicates or inconsistent spellings like “USA” vs “United States”).]

2. **Product Hierarchy Exploration**

   - Retrieves unique combinations of category, subcategory, and product_name from gold.dim_products.
   - Results are sorted in hierarchical order (category → subcategory → product_name).

     [Understand the structure of the product catalog and ensure that the product categorization is correctly populated (e.g., no missing category mappings or redundant product entries).]

2.3 Date Range Exploration

```sql
/*
===============================================================================
Date Range Exploration
===============================================================================
Purpose:
    - To determine the temporal boundaries of key data points.
    - To understand the range of historical data.

SQL Functions Used:
    - MIN(), MAX(), DATEDIFF()
===============================================================================
*/

-- Determine the first and last order date and the total duration in months
SELECT
    MIN(order_date) AS first_order_date,
    MAX(order_date) AS last_order_date,
    DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS order_range_months
FROM gold.fact_sales;

-- Find the youngest and oldest customer based on birthdate
SELECT
    MIN(birthdate) AS oldest_birthdate,
    DATEDIFF(YEAR, MIN(birthdate), GETDATE()) AS oldest_age,
    MAX(birthdate) AS youngest_birthdate,
    DATEDIFF(YEAR, MAX(birthdate), GETDATE()) AS youngest_age
FROM gold.dim_customers;
```

	first_order_date	last_order_date	order_range_months
1	2010-12-29	2014-01-28	37

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

2.4 Measures Exploration

```sql
/*
===============================================================================
Measures Exploration (Key Metrics)
===============================================================================
Purpose:
    - To calculate aggregated metrics (e.g., totals, averages) for quick insights.
    - To identify overall trends or spot anomalies.

SQL Functions Used:
    - COUNT(), SUM(), AVG()
===============================================================================
*/

-- Find the Total Sales
SELECT SUM(sales_amount) AS total_sales FROM gold.fact_sales

-- Find how many items are sold
SELECT SUM(quantity) AS total_quantity FROM gold.fact_sales

-- Find the average selling price
SELECT AVG(price) AS avg_price FROM gold.fact_sales

-- Find the Total number of Orders
SELECT COUNT(order_number) AS total_orders FROM gold.fact_sales
SELECT COUNT(DISTINCT order_number) AS total_orders FROM gold.fact_sales

-- Find the total number of products
SELECT COUNT(product_name) AS total_products FROM gold.dim_products

-- Find the total number of customers
SELECT COUNT(customer_key) AS total_customers FROM gold.dim_customers;

-- Find the total number of customers that has placed an order
SELECT COUNT(DISTINCT customer_key) AS total_customers FROM gold.fact_sales;

-- Generate a Report that shows all key metrics of the business
SELECT
     'Total Sales' AS measure_name,
     SUM(sales_amount) AS measure_value FROM gold.fact_sales
UNION ALL

SELECT
     'Total Quantity',
     SUM(quantity) FROM gold.fact_sales
UNION ALL

SELECT
     'Average Price',
     AVG(price) FROM gold.fact_sales
UNION ALL

SELECT
     'Total Orders',
     COUNT(DISTINCT order_number) FROM gold.fact_sales
UNION ALL

SELECT
     'Total Products',
     COUNT(DISTINCT product_name) FROM gold.dim_products
UNION ALL

SELECT
```

```
    'Total Customers',
    COUNT(customer_key) FROM gold.dim_customers;
```

	total_sales
1	29356250

	total_quantity
1	60423

	avg_price
1	486

	total_orders
1	60398

	total_orders
1	27659

	total_products
1	295

	total_customers
1	18484

	total_customers
1	18484

	measure_name	measure_value
1	Total Sales	29356250
2	Total Quantity	60423
3	Average Price	486
4	Total Orders	27659
5	Total Products	295
6	Total Custom...	18484

2.5 Magnitude Analysis

```sql
/*
===============================================================================
Magnitude Analysis
===============================================================================
Purpose:
    - To quantify data and group results by specific dimensions.
    - For understanding data distribution across categories.

SQL Functions Used:
    - Aggregate Functions: SUM(), COUNT(), AVG()
    - GROUP BY, ORDER BY
===============================================================================
*/

-- Find total customers by countries
SELECT
    country,
    COUNT(customer_key) AS total_customers
FROM gold.dim_customers
GROUP BY country
ORDER BY total_customers DESC;

-- Find total customers by gender
SELECT
    gender,
    COUNT(customer_key) AS total_customers
FROM gold.dim_customers
GROUP BY gender
ORDER BY total_customers DESC;

-- Find total products by category
SELECT
    category,
    COUNT(product_key) AS total_products
FROM gold.dim_products
GROUP BY category
ORDER BY total_products DESC;

-- What is the average costs in each category?
SELECT
    category,
    AVG(cost) AS avg_cost
FROM gold.dim_products
GROUP BY category
ORDER BY avg_cost DESC;

-- What is the total revenue generated for each category?
SELECT
    p.category,
    SUM(f.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;


-- What is the total revenue generated by each customer?
SELECT
    c.customer_key,
    c.first_name,
    c.last_name,
```

```sql
    SUM(f.sales_amount) AS total_revenue
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
    ON c.customer_key = f.customer_key
GROUP BY
    c.customer_key,
    c.first_name,
    c.last_name
ORDER BY total_revenue DESC;

-- What is the distribution of sold items across countries?
SELECT
    c.country,
    SUM(f.quantity) AS total_sold_items
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
    ON c.customer_key = f.customer_key
GROUP BY c.country
ORDER BY total_sold_items DESC;
```

	country	total_customers
1	United States	7482
2	Australia	3591
3	United Kingdom	1913
4	France	1810
5	Germany	1780
6	Canada	1571
7	n/a	337

	gender	total_customers
1	Male	9341
2	Female	9128
3	n/a	15

	category	total_products
1	Components	127
2	Bikes	97
3	Clothing	35
4	Accessories	29
5	NULL	7

	category	avg_cost
1	Bikes	949
2	Components	264
3	NULL	28
4	Clothing	24
5	Accessories	13

	category	total_revenue
1	Bikes	28316272
2	Accessories	700262
3	Clothing	339716

	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
8	1308	Brad	She	13172

2.6 Ranking Analysis

```sql
/*
===============================================================================
Ranking Analysis
===============================================================================
Purpose:
    - To rank items (e.g., products, customers) based on performance or other metrics.
    - To identify top performers or laggards.

SQL Functions Used:
    - Window Ranking Functions: RANK(), DENSE_RANK(), ROW_NUMBER(), TOP
    - Clauses: GROUP BY, ORDER BY
===============================================================================
*/

-- Which 5 products Generating the Highest Revenue?
-- Simple Ranking
SELECT TOP 5
    p.product_name,
    SUM(f.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;

-- Complex but Flexibly Ranking Using Window Functions
SELECT *
FROM (
    SELECT
        p.product_name,
        SUM(f.sales_amount) AS total_revenue,
        RANK() OVER (ORDER BY SUM(f.sales_amount) DESC) AS rank_products
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON p.product_key = f.product_key
    GROUP BY p.product_name
) AS ranked_products
WHERE rank_products <= 5;

-- What are the 5 worst-performing products in terms of sales?
SELECT TOP 5
    p.product_name,
    SUM(f.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;

-- Find the top 10 customers who have generated the highest revenue
SELECT TOP 10
    c.customer_key,
    c.first_name,
    c.last_name,
    SUM(f.sales_amount) AS total_revenue
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
    ON c.customer_key = f.customer_key
GROUP BY
    c.customer_key,
    c.first_name,
    c.last_name
```

```sql
ORDER BY total_revenue DESC;

-- The 3 customers with the fewest orders placed
SELECT TOP 3
    c.customer_key,
    c.first_name,
    c.last_name,
    COUNT(DISTINCT order_number) AS total_orders
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
    ON c.customer_key = f.customer_key
GROUP BY
    c.customer_key,
    c.first_name,
    c.last_name
ORDER BY total_orders ;
```

	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

	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

	product_name	total_revenue
1	Racing Socks- L	2430
2	Racing Socks- M	2682
3	Patch Kit/8 Patches	6382
4	Bike Wash - Dissol...	7272
5	Touring Tire Tube	7440

	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
8	1308	Brad	She	13172

	customer_key	first_name	last_name	total_orders
1	16	Chloe	Young	1
2	17	Wyatt	Hill	1
3	21	Jordan	King	1

2.7 Change Over Time

```sql
/*
===============================================================================
Change Over Time Analysis
===============================================================================
Purpose:
    - To track trends, growth, and changes in key metrics over time.
    - For time-series analysis and identifying seasonality.
    - To measure growth or decline over specific periods.

SQL Functions Used:
    - Date Functions: DATEPART(), DATETRUNC(), FORMAT()
    - Aggregate Functions: SUM(), COUNT(), AVG()
===============================================================================
*/

-- Analyse sales performance over time
-- Quick Date Functions
SELECT
    YEAR(order_date) AS order_year,
    MONTH(order_date) AS order_month,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY YEAR(order_date), MONTH(order_date);

-- DATETRUNC()
SELECT
    DATETRUNC(month, order_date) AS order_date,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY DATETRUNC(month, order_date)
ORDER BY DATETRUNC(month, order_date);

-- FORMAT()
SELECT
    FORMAT(order_date, 'yyyy-MMM') AS order_date,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MMM')
ORDER BY FORMAT(order_date, 'yyyy-MMM');
```

	order_year	order_month	total_sales	total_customers	total_quantity
1	2010	12	43419	14	14
2	2011	1	469795	144	144
3	2011	2	466307	144	144
4	2011	3	485165	150	150
5	2011	4	502042	157	157
6	2011	5	561647	174	174
7	2011	6	737793	230	230
8	2011	7	596710	188	188

	order_date	total_sales	total_customers	total_quantity
1	2010-12-01	43419	14	14
2	2011-01-01	469795	144	144
3	2011-02-01	466307	144	144
4	2011-03-01	485165	150	150
5	2011-04-01	502042	157	157
6	2011-05-01	561647	174	174
7	2011-06-01	737793	230	230
8	2011-07-01	596710	188	188

	order_date	total_sales	total_customers	total_quantity
1	2010-Dec	43419	14	14
2	2011-Apr	502042	157	157
3	2011-Aug	614516	193	193
4	2011-Dec	669395	222	222
5	2011-Feb	466307	144	144
6	2011-Jan	469795	144	144
7	2011-Jul	596710	188	188
8	2011-Jun	737793	230	230
9	2011-Mar	485165	150	150
10	2011-May	561647	174	174
11	2011-Nov	660507	208	208
12	2011-Oct	708164	221	221
13	2011-Sep	603047	185	185
14	2012-Apr	400324	219	219
15	2012-Aug	523887	294	294

2.8 Cumulative Analysis

```sql
/*
===============================================================================
Cumulative Analysis
===============================================================================
Purpose:
    - To calculate running totals or moving averages for key metrics.
    - To track performance over time cumulatively.
    - Useful for growth analysis or identifying long-term trends.

SQL Functions Used:
    - Window Functions: SUM() OVER(), AVG() OVER()
===============================================================================
*/

-- Calculate the total sales per month
-- and the running total of sales over time
SELECT
       order_date,
       total_sales,
       SUM(total_sales) OVER (ORDER BY order_date) AS running_total_sales,
       AVG(avg_price) OVER (ORDER BY order_date) AS moving_average_price
FROM
(
    SELECT
        DATETRUNC(year, order_date) AS order_date,
        SUM(sales_amount) AS total_sales,
        AVG(price) AS avg_price
    FROM gold.fact_sales
    WHERE order_date IS NOT NULL
    GROUP BY DATETRUNC(year, order_date)
) t
```

	order_date	total_sales	running_total_sales	moving_average_price
1	2010-01-01	43419	43419	3101
2	2011-01-01	7075088	7118507	3146
3	2012-01-01	5842231	12960738	2670
4	2013-01-01	16344878	29305616	2080
5	2014-01-01	45642	29351258	1668

2.9 Performance Analysys

```sql
/*
===============================================================================
Performance Analysis (Year-over-Year, Month-over-Month)
===============================================================================
Purpose:
    - To measure the performance of products, customers, or regions over time.
    - For benchmarking and identifying high-performing entities.
    - To track yearly trends and growth.

SQL Functions Used:
    - LAG(): Accesses data from previous rows.
    - AVG() OVER(): Computes average values within partitions.
    - CASE: Defines conditional logic for trend analysis.
===============================================================================
*/

/* Analyze the yearly performance of products by comparing their sales
to both the average sales performance of the product and the previous year's sales */
WITH yearly_product_sales AS (
    SELECT
        YEAR(f.order_date) AS order_year,
        p.product_name,
        SUM(f.sales_amount) AS current_sales
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON f.product_key = p.product_key
    WHERE f.order_date IS NOT NULL
    GROUP BY
        YEAR(f.order_date),
        p.product_name
)
SELECT
    order_year,
    product_name,
    current_sales,
    AVG(current_sales) OVER (PARTITION BY product_name) AS avg_sales,
    current_sales - AVG(current_sales) OVER (PARTITION BY product_name) AS diff_avg,
    CASE
        WHEN current_sales - AVG(current_sales) OVER (PARTITION BY product_name) > 0 THEN 'Above Avg'
        WHEN current_sales - AVG(current_sales) OVER (PARTITION BY product_name) < 0 THEN 'Below Avg'
        ELSE 'Avg'
    END AS avg_change,
    -- Year-over-Year Analysis
    LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) AS py_sales,
    current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) AS diff_py,
    CASE
        WHEN current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) > 0 THEN 'Increase'
        WHEN current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) < 0 THEN 'Decrease'
        ELSE 'No Change'
    END AS py_change
FROM yearly_product_sales
ORDER BY product_name, order_year;
```

	order_year	product_name	current_sales	avg_sales	diff_avg	avg_change	py_sales	diff_py	py_change
1	2012	All-Purpose Bike Stand	159	13197	-13038	Below Avg	NULL	NULL	No Change
2	2013	All-Purpose Bike Stand	37683	13197	24486	Above Avg	159	37524	Increase
3	2014	All-Purpose Bike Stand	1749	13197	-11448	Below Avg	37683	-35934	Decrease
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	Increase
6	2014	AWC Logo Cap	747	6570	-5823	Below Avg	18891	-18144	Decrease
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	Decrease
9	2013	Classic Vest- L	11968	6240	5728	Above Avg	NULL	NULL	No Change
10	2014	Classic Vest- L	512	6240	-5728	Below Avg	11968	-11456	Decrease
11	2013	Classic Vest- M	11840	6368	5472	Above Avg	NULL	NULL	No Change
12	2014	Classic Vest- M	896	6368	-5472	Below Avg	11840	-10944	Decrease
13	2012	Classic Vest- S	64	3648	-3584	Below Avg	NULL	NULL	No Change
14	2013	Classic Vest- S	10368	3648	6720	Above Avg	64	10304	Increase
15	2014	Classic Vest- S	512	3648	-3136	Below Avg	10368	-9856	Decrease

2.10 Data Segmentation

```sql
/* ===============================================================================
Data Segmentation Analysis
===============================================================================
Purpose:
    - To group data into meaningful categories for targeted insights.
    - For customer segmentation, product categorization, or regional analysis.

SQL Functions Used:
    - CASE: Defines custom segmentation logic.
    - GROUP BY: Groups data into segments.
=============================================================================== */
/*Segment products into cost ranges and
count how many products fall into each segment*/
WITH product_segments AS (
    SELECT
        product_key,
        product_name,
        cost,
        CASE
            WHEN cost < 100 THEN 'Below 100'
            WHEN cost BETWEEN 100 AND 500 THEN '100-500'
            WHEN cost BETWEEN 500 AND 1000 THEN '500-1000'
            ELSE 'Above 1000'
        END AS cost_range
    FROM gold.dim_products
)
SELECT
    cost_range,
    COUNT(product_key) AS total_products
FROM product_segments
GROUP BY cost_range
ORDER BY total_products DESC;

/* Group customers into three segments based on their spending behavior:
        - VIP: Customers with at least 12 months of history and spending more than €5,000.
        - Regular: Customers with at least 12 months of history but spending €5,000 or less.
        - New: Customers with a lifespan less than 12 months.
And find the total number of customers by each group */
WITH customer_spending AS (
    SELECT
        c.customer_key,
        SUM(f.sales_amount) AS total_spending,
        MIN(order_date) AS first_order,
        MAX(order_date) AS last_order,
        DATEDIFF(month, MIN(order_date), MAX(order_date)) AS lifespan
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_customers c
        ON f.customer_key = c.customer_key
    GROUP BY c.customer_key
)
SELECT
    customer_segment,
    COUNT(customer_key) AS total_customers
FROM (
    SELECT
        customer_key,
        CASE
            WHEN lifespan >= 12 AND total_spending > 5000 THEN 'VIP'
            WHEN lifespan >= 12 AND total_spending <= 5000 THEN 'Regular'
            ELSE 'New'
        END AS customer_segment
    FROM customer_spending
) AS segmented_customers
GROUP BY customer_segment
ORDER BY total_customers DESC;
```

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

	customer_segment	total_customers
1	New	14631
2	Regular	2198
3	VIP	1655

2.11 Whole Analysis

```
/*
===============================================================================
Part-to-Whole Analysis
===============================================================================
Purpose:
    - To compare performance or metrics across dimensions or time periods.
    - To evaluate differences between categories.
    - Useful for A/B testing or regional comparisons.

SQL Functions Used:
    - SUM(), AVG(): Aggregates values for comparison.
    - Window Functions: SUM() OVER() for total calculations.
===============================================================================
*/
-- Which categories contribute the most to overall sales?
WITH category_sales AS (
    SELECT
        p.category,
        SUM(f.sales_amount) AS total_sales
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON p.product_key = f.product_key
    GROUP BY p.category
)
SELECT
    category,
    total_sales,
    SUM(total_sales) OVER () AS overall_sales,
    ROUND((CAST(total_sales AS FLOAT) / SUM(total_sales) OVER ()) * 100, 2) AS
percentage_of_total
FROM category_sales
ORDER BY total_sales DESC;
```

	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

2.12 Customers Report

```
/*
===============================================================================
Customer Report
===============================================================================
Purpose:
    - This report consolidates key customer metrics and behaviors

Highlights:
    1. Gathers essential fields such as names, ages, and transaction details.
       2. Segments customers into categories (VIP, Regular, New) and age groups.
    3. Aggregates customer-level metrics:
          - total orders
          - total sales
          - total quantity purchased
          - total products
          - lifespan (in months)
    4. Calculates valuable KPIs:
           - recency (months since last order)
             - average order value
             - average monthly spend
===============================================================================
*/

-- =============================================================================
-- Create Report: gold.report_customers
-- =============================================================================
IF OBJECT_ID('gold.report_customers', 'V') IS NOT NULL
    DROP VIEW gold.report_customers;
GO

CREATE VIEW gold.report_customers AS

WITH base_query AS(
/*---------------------------------------------------------------------------
1) Base Query: Retrieves 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()) age
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers 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,
```

```sql
        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
FROM base_query
GROUP BY
        customer_key,
        customer_number,
        customer_name,
        age
)
SELECT
customer_key,
customer_number,
customer_name,
age,
CASE
         WHEN age < 20 THEN 'Under 20'
         WHEN age between 20 and 29 THEN '20-29'
         WHEN age between 30 and 39 THEN '30-39'
         WHEN age between 40 and 49 THEN '40-49'
         ELSE '50 and above'
END AS age_group,
CASE
    WHEN lifespan >= 12 AND total_sales > 5000 THEN 'VIP'
    WHEN lifespan >= 12 AND total_sales <= 5000 THEN 'Regular'
    ELSE 'New'
END AS customer_segment,
last_order_date,
DATEDIFF(month, last_order_date, GETDATE()) AS recency,
total_orders,
total_sales,
total_quantity,
total_products
lifespan,
-- Compuate average order value (AVO)
CASE WHEN total_sales = 0 THEN 0
        ELSE total_sales / total_orders
END AS avg_order_value,
-- Compuate average monthly spend
CASE WHEN lifespan = 0 THEN total_sales
     ELSE total_sales / lifespan
END AS avg_monthly_spend
FROM customer_aggregation
```

2.13 Product Report

```sql
/*
===============================================================================
Product Report
===============================================================================
Purpose:
    - This report consolidates key product metrics and behaviors.

Highlights:
    1. Gathers essential fields such as product name, category, subcategory, and cost.
    2. Segments products by revenue to identify High-Performers, Mid-Range, or Low-Performers.
    3. Aggregates product-level metrics:
       - total orders
       - total sales
       - total quantity sold
       - total customers (unique)
       - lifespan (in months)
    4. Calculates valuable KPIs:
       - recency (months since last sale)
       - average order revenue (AOR)
       - average monthly revenue
===============================================================================
*/
-- =============================================================================
-- Create Report: gold.report_products
-- =============================================================================
IF OBJECT_ID('gold.report_products', 'V') IS NOT NULL
    DROP VIEW gold.report_products;
GO

CREATE VIEW gold.report_products AS

WITH base_query AS (
/*---------------------------------------------------------------------------
1) Base Query: Retrieves core columns from fact_sales and dim_products
---------------------------------------------------------------------------*/
    SELECT
          f.order_number,
        f.order_date,
            f.customer_key,
        f.sales_amount,
        f.quantity,
        p.product_key,
        p.product_name,
        p.category,
        p.subcategory,
        p.cost
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON f.product_key = p.product_key
    WHERE order_date IS NOT NULL  -- only consider valid sales dates
),

product_aggregations AS (
/*---------------------------------------------------------------------------
2) Product Aggregations: Summarizes key metrics at the product level
---------------------------------------------------------------------------*/
SELECT
    product_key,
    product_name,
    category,
    subcategory,
    cost,
```

```sql
    DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS lifespan,
    MAX(order_date) AS last_sale_date,
    COUNT(DISTINCT order_number) AS total_orders,
      COUNT(DISTINCT customer_key) AS total_customers,
    SUM(sales_amount) AS total_sales,
    SUM(quantity) AS total_quantity,
      ROUND(AVG(CAST(sales_amount AS FLOAT) / NULLIF(quantity, 0)),1) AS avg_selling_price
FROM base_query

GROUP BY
    product_key,
    product_name,
    category,
    subcategory,
    cost
)

/*---------------------------------------------------------------------------
  3) Final Query: Combines all product results into one output
---------------------------------------------------------------------------*/
SELECT
      product_key,
      product_name,
      category,
      subcategory,
      cost,
      last_sale_date,
      DATEDIFF(MONTH, last_sale_date, GETDATE()) AS recency_in_months,
      CASE
            WHEN total_sales > 50000 THEN 'High-Performer'
            WHEN total_sales >= 10000 THEN 'Mid-Range'
            ELSE 'Low-Performer'
      END AS product_segment,
      lifespan,
      total_orders,
      total_sales,
      total_quantity,
      total_customers,
      avg_selling_price,
      -- Average Order Revenue (AOR)
      CASE
            WHEN total_orders = 0 THEN 0
            ELSE total_sales / total_orders
      END AS avg_order_revenue,

      -- Average Monthly Revenue
      CASE
            WHEN lifespan = 0 THEN total_sales
            ELSE total_sales / lifespan
      END AS avg_monthly_revenue

FROM product_aggregations
```

Notes :

Aspect	Dimension Table	Fact Table
Purpose	Stores **descriptive / contextual information** about business entities (who, what, where, when, how).	Stores **quantitative or measurable data** — the actual business transactions or events.
Data Type	Mostly **textual or categorical** (names, types, categories).	Mostly **numeric / measurable** (sales amount, quantity, revenue).
Granularity	Low granularity (one record per entity, e.g., one row per customer or product).	High granularity (one record per transaction or event).
Primary Key	**Surrogate key** (e.g., customer_key, product_key) uniquely identifies each dimension record.	**Composite foreign keys** referencing related dimension keys.
Foreign Keys	Usually **not present** or minimal (may link to hierarchies like region → country).	Contains **foreign keys** referencing dimension tables.
Examples of Columns	customer_name, gender, city, product_category, date_desc	order_id, product_key, customer_key, sales_amount, units_sold
Nature of Data	**Static or slowly changing** (SCD) — changes occasionally.	**Dynamic and ever-growing** — new facts are added regularly.
Update Frequency	Rare updates (e.g., when a customer changes address).	Frequent inserts (new transactions) and fewer updates.
Use in Queries	Used for **filtering**, **grouping**, and **labeling** (e.g., "Total Sales by Region").	Used for **aggregation and computation** (e.g., "Sum of Sales").
Size	Typically **smaller** in volume compared to facts.	Typically **larger** in volume (millions or billions of rows).
ETL Processing	Loaded **before** fact tables (since facts depend on dimensions).	Loaded **after** dimensions are available.
Example Tables	dim_customer, dim_product, dim_date, dim_region	fact_sales, fact_orders, fact_inventory
Relationships	Acts as the **lookup** or **reference** table.	Acts as the **transaction / measurement** table.