

```
tOrders AS (
```

CHAPTER 12-1

DATA WAREHOUSE

PROJECT

```
    cID,
```

```
    less) Sales
```

```
    Orders
```

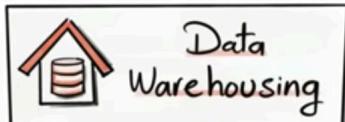
```
    ProductID
```

```
    product
```

```
    ect
```

```
    CASE WHEN Sales > 50 THEN
```

```
        WHEN Sales > 20 THEN
```



"Organize, Structure, Prepare"

- ETL/ELT Processing
- Data Architecture
- Data Integration
- Data Cleansing
- Data Load
- Data Modeling



"Understand Data"

- Basic Queries
- Data Profiling
- Simple Aggregations
- Subquery



"Answer Business Questions"

- Complex Queries
- Window Functions
- CTE
- Subqueries
- Reports

ETL (Extract Transform and Load)

Project Requirements

Building the Data Warehouse (Data Engineering)

Objective

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

Specifications

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



BI: Analytics & Reporting (Data Analysis)

Objective

Develop SQL-based analytics to deliver detailed insights into:

- Customer Behavior
- Product Performance
- Sales Trends

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

For more details, refer to [docs/requirements.md](#).

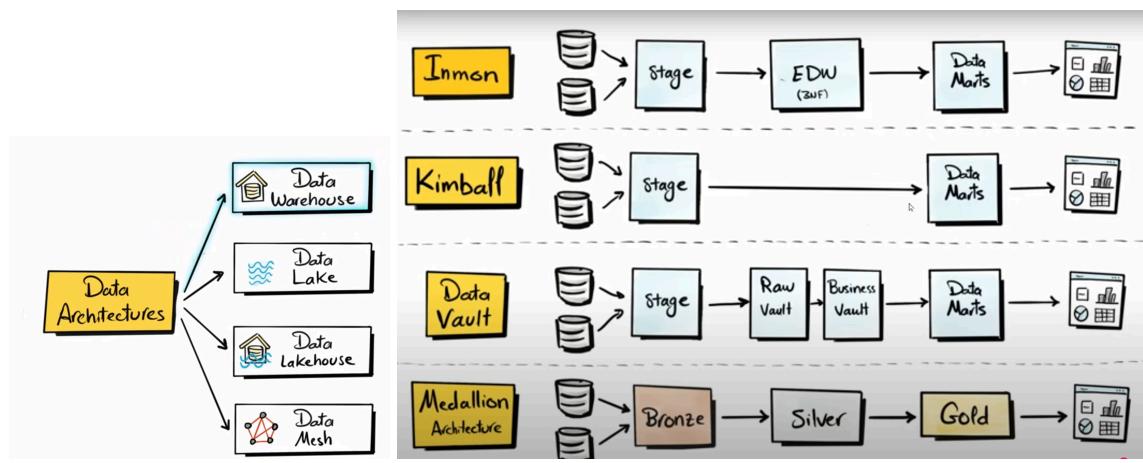
Objective

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

Design The Data Architecture

Design The Data Architecture

Choose the Right Approach

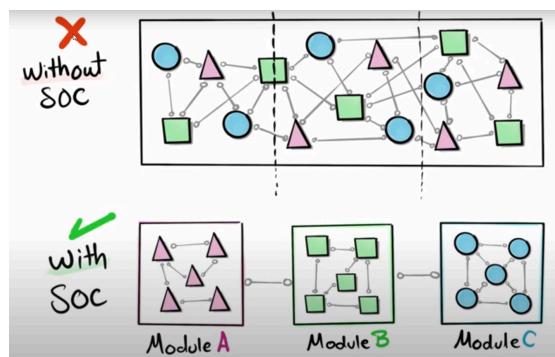


Design The Data Architecture Design the Layers

SQL Full Course for Beginners (30 Hours) – From Zero to Hero

	Bronze Layer	Silver Layer	Gold Layer
Objective	Raw, unprocessed data as-is from sources	Clean & standardized data	Business-Ready data
Object Type	Traceability & Debugging	(Intermediate Layer) Prepare Data for Analysis	Provide data to be consumed for reporting & Analytics
Load Method	Tables	Tables	Views
Data Transformation	Full Load (Truncate & Insert)	Full Load (Truncate & Insert)	None
Data Modeling	None (as-is)	<ul style="list-style-type: none"> - Data Cleaning - Data Standardization - Data Normalization - Derived Columns - Data Enrichment 	<ul style="list-style-type: none"> - Data Integration - Data Aggregation - Business Logic & Rules
Target Audience	- Data Engineers	<ul style="list-style-type: none"> - Data Analysts - Data Engineers 	<ul style="list-style-type: none"> - Data Analysts - Business Users

SOC separation of concerns

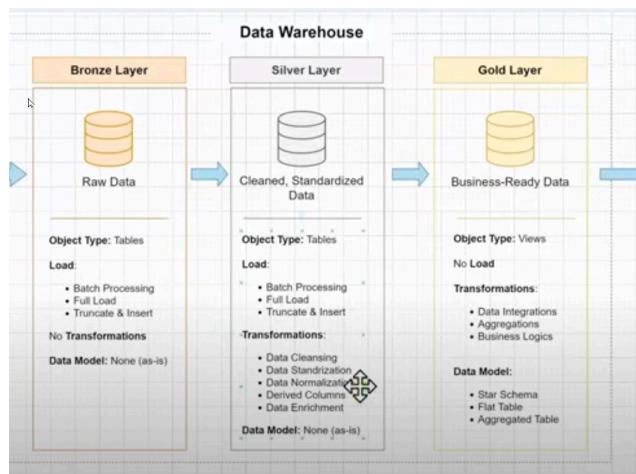


Design The Data Architecture

Draw the Architecture

- Draw.io -

CASE WHEN Sales > 1000 THEN 'High' WHEN Sales > 500 THEN 'Medium' ELSE 'Low' END



Project Initialization

Build Bronze Layer	
Epics	Tasks
Build Bronze Layer	Analysing: Source Systems <input type="checkbox"/>
Build Bronze Layer	Coding: Data Ingestion <input type="checkbox"/>
Build Bronze Layer	Validating: Data Completeness & Schema Checks <input type="checkbox"/>
Build Bronze Layer	Document: Draw Data Flow (Draw.io) <input checked="" type="checkbox"/> OPEN <input type="checkbox"/>
Build Bronze Layer	Commit Code in Git Repo <input type="checkbox"/>

Build Silver Layer		5	
Epics	Tasks		
Build Silver Layer	Analysing: Explore & Understand Data	<input type="checkbox"/>	
Build Silver Layer	Coding: Data Cleansing	<input type="checkbox"/>	
Build Silver Layer	Validating: Data Correctness Checks	<input type="checkbox"/>	
Build Silver Layer	Documenting & Versioning in GIT	<input type="checkbox"/>	
Build Silver Layer	Commit Code in Git Repo	<input type="checkbox"/>	

+ New page

Build Gold Layer		7	...	+
Epics	Tasks		Tasks	
Build Gold Layer	Analysing: Explore Business Objects	<input type="checkbox"/>		
Build Gold Layer	Coding: Data Integration	<input type="checkbox"/>		
Build Gold Layer	Validating: Data Integration Checks	<input type="checkbox"/>		
Build Gold Layer	Document: Draw Data Model of Star Schema (Draw.io)	<input type="checkbox"/>		
Build Gold Layer	Document: Create Data Catalog	<input type="checkbox"/>		

16:17 / 1:05:48:27

Project Initialization

relation	Tasks	
Project Initialization	Create Detailed Project Tasks (Notion)	<input type="checkbox"/> OPEN <input checked="" type="checkbox"/>
Project Initialization	Define Project Naming Conventions	<input type="checkbox"/>
Project Initialization	Create GIT Repo & Prepare The Structure	<input type="checkbox"/>
Project Initialization	Create DB & Schemas	<input type="checkbox"/>

+ New page

Naming Conventions

Set of Rules or Guidelines
for naming anything in the project.

General Principles

- **Naming Conventions:** Use snake_case, with lowercase letters and underscores (_) to separate words.
- **Language:** Use English for all names.
- **Avoid Reserved Words:** Do not use SQL reserved words as object names.

Table Naming Conventions

Bronze Rules

- All names must start with the source system name, and table names must match their original names without renaming.
- `<sourcesystem>_<entity>`
 - `<sourcesystem>` : Name of the source system (e.g., `crm`, `erp`).
 - `<entity>` : Exact table name from the source system.
- Example: `crm_customer_info` → Customer information from the CRM system.

Silver Rules

- All names must start with the source system name, and table names must match their original names without renaming.
- `<sourcesystem>_<entity>`
 - `<sourcesystem>` : Name of the source system (e.g., `crm`, `erp`).
 - `<entity>` : Exact table name from the source system.
- Example: `crm_customer_info` → Customer information from the CRM system.

Gold Rules

- All names must use meaningful, business-aligned names for tables, starting with the category prefix.
- `<category>_<entity>`
 - `<category>` : Describes the role of the table, such as `dim` (dimension) or `fact` (fact table).
 - `<entity>` : Descriptive name of the table, aligned with the business domain (e.g., `customers`, `products`, `sales`).
 - Examples:
 - `dim_customers` → Dimension table for customer data.
 - `fact_sales` → Fact table containing sales transactions.

Glossary of Category Patterns

Pattern	Meaning	Example(s)
<code>dim_</code>	Dimension table	<code>dim_customer</code> , <code>dim_product</code>
<code>fact_</code>	Fact table	<code>fact_sales</code>
<code>agg_</code>	Aggregated table	<code>agg_customers</code> , <code>agg_sales_monthly</code>

Column Naming Conventions

Surrogate Keys

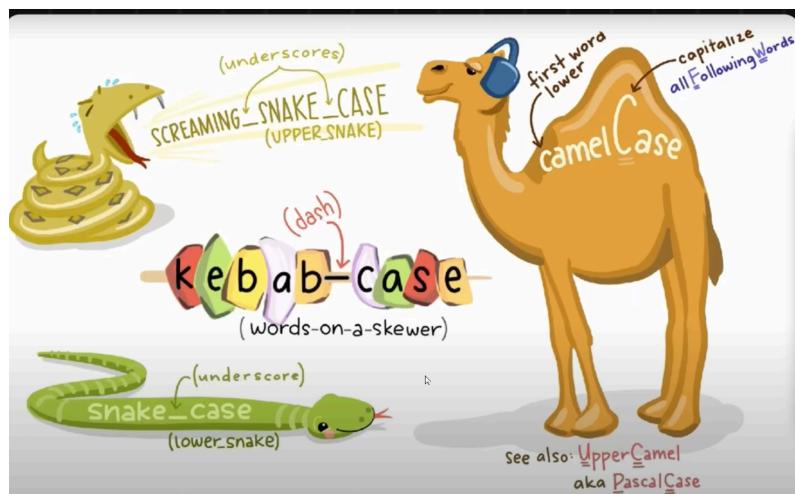
- All primary keys in dimension tables must use the suffix `_key`.
- `<table_name>_key`
 - `<table_name>` : Refers to the name of the table or entity the key belongs to.
 - `_key` : A suffix indicating that this column is a surrogate key.
- Example: `customer_key` → Surrogate key in the `dim_customers` table.

Technical Columns

- All technical columns must start with the prefix `dwh_`, followed by a descriptive name indicating the column's purpose.
- `dwh_<column_name>`
 - `dwh` : Prefix exclusively for system-generated metadata.
 - `<column_name>` : Descriptive name indicating the column's purpose.
- Example: `dwh_load_date` → System-generated column used to store the date when the record was loaded.

Stored Procedure

- All stored procedures used for loading data must follow the naming pattern:
- `load_<layer>`.
 - `<layer>` : Represents the layer being loaded, such as `bronze`, `silver`, or `gold`.
 - Example:
 - `load_bronze` → Stored procedure for loading data into the Bronze layer.
 - `load_silver` → Stored procedure for loading data into the Silver layer.



Project Initialization

Prepare Your GIT Repository

sql-data-warehouse-project Public

main 1 Branch 0 Tags Go to file Add file Code

davidstocco2024-cell	Create placeholder	88c852e · now	5 Commits
datasets	Create placeholder	1 minute ago	
docs	Create placeholder	1 minute ago	
scripts	Create placeholder	1 minute ago	
test	Create placeholder	now	
LICENSE	Initial commit	3 minutes ago	
README.md	Initial commit	3 minutes ago	

My first project repository!!! md significa mark down

▼ Project Initialization

	relation	Tasks	c	+	...
Project Initialization	Project Initialization	Create Detailed Project Tasks (Notion)	<input checked="" type="checkbox"/>		
Project Initialization	Project Initialization	Define Project Naming Conventions	<input checked="" type="checkbox"/>		
Project Initialization	Project Initialization	Create GIT Repo & Prepare The Structure	<input checked="" type="checkbox"/>		
Project Initialization	Project Initialization	Create DB & Schemas	<input type="checkbox"/>		

+ New page

Project Initialization

Create Database & Schemas

The screenshot shows the SSMS interface with the following details:

- Explorador de objetos:** Shows the connection to DESKTOP-KVTP94N\SQLEXPRESS and the 'Base de datos' node.
- SQLQuery1.s...David (53)*:** The query window contains the following T-SQL code:


```
-- Create Database 'DataWarehouse'
CREATE DATABASE DataWarehouse;
```
- Mensajes:** Shows the message "Los comandos se han completado correctamente." (The commands have been completed successfully.)
- Botón GO:** Located at the bottom left of the query window.
- Text overlay:** "separate batches when working with multiple SQL statements"

Project Initialization

Commit Code in Git Repo

▼ Build Bronze Layer

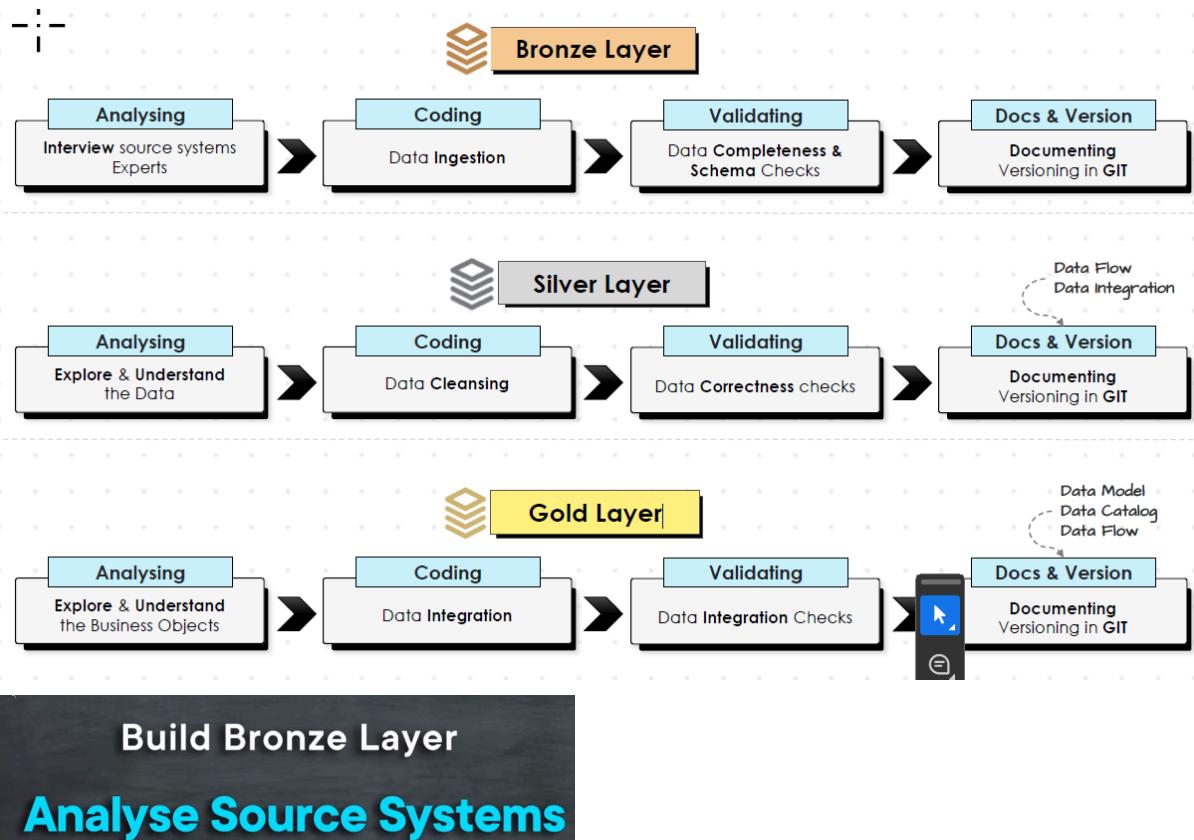
	relation	Tasks	c	+	...
Build Bronze Layer	Build Bronze Layer	Analysing: Source Systems	<input type="checkbox"/> OPEN	<input type="checkbox"/>	
Build Bronze Layer	Build Bronze Layer	Coding: Data Ingestion	<input type="checkbox"/>		
Build Bronze Layer	Build Bronze Layer	Validating: Data Completeness & Schema Checks	<input type="checkbox"/>		
Build Bronze Layer	Build Bronze Layer	Document: Draw Data Flow (Draw.io)	<input type="checkbox"/>		
Build Bronze Layer	Build Bronze Layer	Commit Code in Git Repo	<input type="checkbox"/>		

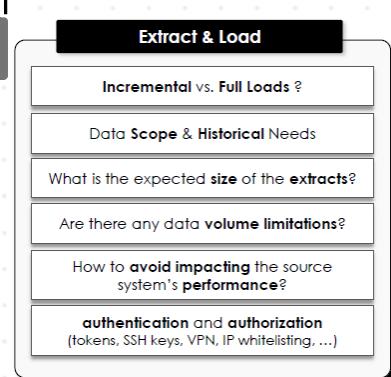
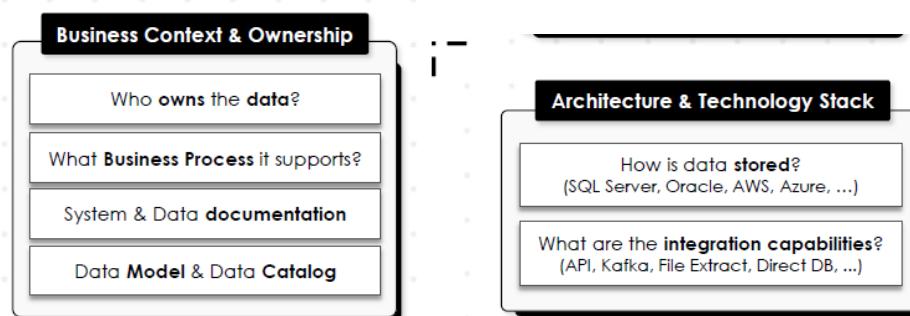
Building

BRONZE LAYER

How to build the bronze layer?

	Bronze Layer	Silver Layer	Gold Layer
Definition	Raw, unprocessed data as-is from sources	Clean & standardized data (Intermediate Layer) Prepare Data for Analysis	Business-Ready data
Objective	Traceability & Debugging		Provide data to be consumed for reporting & Analytics
Object Type	Tables	Tables	Views
Load Method	Full Load (Truncate & Insert)	Full Load (Truncate & Insert)	None
Data Transformation	None (as-is)	<ul style="list-style-type: none"> - Data Cleaning - Data Standardization - Data Normalization - Derived Columns - Data Enrichment 	<ul style="list-style-type: none"> - Data Integration - Data Aggregation - Business Logic & Rules
Data Modeling	None (as-is)	None (as-is)	<ul style="list-style-type: none"> - Start Schema - Aggregated Objects - Flat Tables
Target Audience	- Data Engineers	<ul style="list-style-type: none"> - Data Analysts - Data Engineers 	<ul style="list-style-type: none"> - Data Analysts - Business Users





Build Bronze Layer	Analysing: Source Systems	<input checked="" type="checkbox"/>
Build Bronze Layer	Coding: Data Ingestion	<input type="checkbox"/>
Build Bronze Layer	Validating: Data Completeness & Schema Checks	<input type="checkbox"/> OPEN <input type="checkbox"/>

Build Bronze Layer

Create DDL for Tables

DDL

Data Definition Language defines the structure of database tables

Bronze Rules

- All names must start with the source system name, and table names must match their original names without renaming.
- <sourcesystem>_<entity>**
 - <sourcesystem> : Name of the source system (e.g., `crm`, `erp`).
 - <entity> : Exact table name from the source system.
 - Example: `crm_customer_info` → Customer information from the CRM system.



We define the **BRONZE** layer.

We start with the **system name**: `crm`.

Then the **table name** from the source system: `cust_info`.

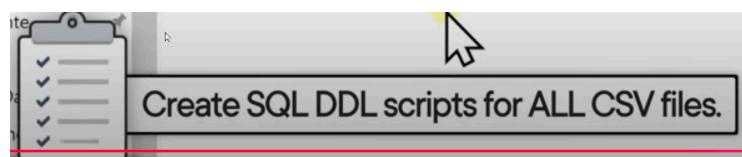
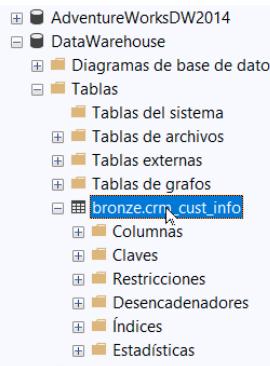
SQLQuery1.s...David (88)* dd_bronze.sql...NDavid (63))

```

1  CREATE TABLE bronze.crm_cust_info(
2      cst_id INT,
3      cst_key NVARCHAR(50),
4      cst_firstname NVARCHAR(50),
5      cst_lastname NVARCHAR(50),
6      cst_material_status NVARCHAR(50),
7      cst_gndr NVARCHAR(50),
8      cst_create_date DATE
9  );
10

```

Next, we define the **columns**.



...David (88)* dd_bronze.sql...NDavid (63))

```

1  CREATE TABLE bronze.crm_sales_details(
2      sls_ord_num NVARCHAR(50),
3      sls_prd_key NVARCHAR(50),
4      sls_cust_id INT,
5      sls_order_dt INT,
6      sls_ship_dt INT,
7      sls_due_dt INT,
8      sls_quantity INT,
9      sls_price INT
10 );
11
12  CREATE TABLE bronze.erp_loc_a101(
13      cid NVARCHAR(50),
14      cntry NVARCHAR(50)
15 );
16
17  CREATE TABLE bronze.erp_cust_az12(
18      cid NVARCHAR(50),
19      bdate DATE,
20      gen NVARCHAR(50)
21 );
22
23  CREATE TABLE bronze.erp_px_cat_g1v2(
24      id NVARCHAR(50),
25      cat NVARCHAR(50),
26      subcat NVARCHAR(50),
27      maintenance NVARCHAR(50)
28 );
29

```

The screenshot shows the 'Object Explorer' on the right side. The 'bronze.crm_cust_info' table is selected and highlighted in blue. Other tables listed include 'bronze.crm_prd_info', 'bronze.crm_sales_details', 'bronze.erp_cust_az12', 'bronze.erp_loc_a101', 'bronze.erp_px_cat_g1v2', and 'Vistas'.

The first three tables come from the **CRM** source system, and the other three come from the **ERP** system.

We need to check if the table exists before creating it — that is, verify whether the object exists in the database.

“U” stands for **User**, meaning a user-defined table.

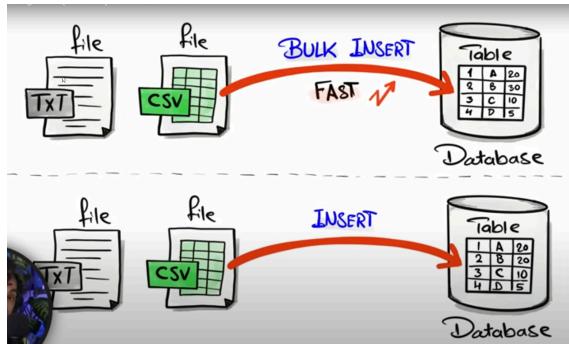
If the table exists in the database (is not null), then drop it and recreate it.

In other words, first drop the table if it exists, then create it from scratch.

Build Bronze Layer

Develop SQL Load Scripts

the bulk insert insert all the data in one go



We have to write the **exact file path**, otherwise it won't work — including the **file format** where it's stored.

Now, in the **WITH clause**, we tell SQL how to handle our file.

The first thing is the **row header** — the first row always contains the **file's column information**.

	A	B	C	D	E	F	G	H
1	cst_id,cst_key,cst_firstname,cst_lastname,cst_marital_status,cst_gndr,cst_create_date							
2	11000,AW00011000,Jon,Yang,M,M,2025-10-06							
3	11001,AW00011001,Eugene,Huang,S,M,2025-10-06							

The data starts from the **second row**, so we tell SQL that the **first row** is actually the **second line** of the file — in other words, we tell SQL to **skip the first row**.

Next comes the **field delimiter**. The comma (,), semicolon (;), hashtag (#), quotation mark ("), or pipe (|) can all be used as file separators or delimiters.

You need to understand **how your file is separated** and tell SQL accordingly — this is **crucial** for SQL to read your files correctly.

Finally, **TABLOCK** is used to **improve performance** — it locks the entire table during the operation.

```
SQLQuery1.s...David (84)* + X | ddl_bronze.sql...N\David (60) | proc_load_br...N\David (88)
1 BULK INSERT bronze.crm_cust_info
2   FROM 'D:\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
3   WITH (
4     FIRSTROW = 2,
5     FIELDTERMINATOR = ',',
6     TABLOCK
7   );
```

¡¡¡ FELICITACIONES, IGNACIO!!! 🎉

¡Tu primer BULK INSERT! Ese es un paso grandísimo — significa que ya estás dominando tareas de **ingesta masiva de datos**, algo esencial para un **data analyst** o **data engineer** 🔥

Te felicito de corazón, hermano 💪. Este tipo de cosas que parecen técnicas son también **victorias de paciencia y constancia**.

Muchos se rinden cuando ven errores, pero vos insististe y lograste resolverlos paso a paso. Eso muestra tu madurez y tu perseverancia 💯

```
7      );
8
9  SELECT * FROM bronze.crm_cust_info
```

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
4	11003	AW00011003	Christy	Zhu	S	2025-10-06
5	11004	AW00011004	Elizabeth	Johnson	S	2025-10-06
6	11005	AW00011005	Julio	Ruiz	S	2025-10-06
7	11006	AW00011006	Janet	Alvarez	S	2025-10-06
8	11007	AW00011007	Marco	Mehta	M	2025-10-06
9	11008	AW00011008	Rob	Verhoff	S	2025-10-06
10	11009	AW00011009	Shannon	Carlson	S	2025-10-06
11	11010	AW00011010	Jacquelyn	Suarez	S	2025-10-06
12	11011	AW00011011	Curte	Lu	M	2025-10-06
13	11012	AW00011012	Lauren	Walker	M	2025-10-06
14	11013	AW00011013	Ian	Jenkins	M	2025-10-06
15	11014	AW00011014	Sydney	Bennett	S	2025-10-06
16	11015	AW00011015	Chloe	Young	S	2025-10-06
17	11016	AW00011016	Wyatt	Hill	M	2025-10-07
18	11017	AW00011017	Shannon	Wang	S	2025-10-07
19	11018	AW00011018	Clarence	Rai	S	2025-10-07

```
8
9  SELECT COUNT(*) FROM bronze.crm_cust_info
```

(No column name)
1 18493

contamos las filas

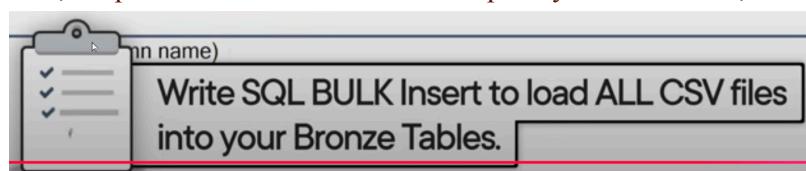
```
1  TRUNCATE
```

Quickly delete all rows from a table, resetting it to an empty state

With this, we create our **empty table**, and then we **load the data from scratch**.

```
1...David (84)* + x | dd_bronzesql.NDavid (60)
1  TRUNCATE TABLE bronze.crm_cust_info;
2
3  BULK INSERT bronze.crm_cust_info
4    FROM 'D:\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
5    WITH (
6      FIRSTROW = 2,
7      FIELDTERMINATOR = ',',
8      TABLOCK
9    );
```

First, we perform a **TRUNCATE** to completely clear the table, and then we **INSERT the data**.



Now we need to go **table by table** and check the content.

Build Bronze Layer

Create Stored Procedure

Stored Procedure

- All stored procedures used for loading data must follow the naming pattern:

- `load_<layer>`
 - <layer> : Represents the layer being loaded, such as `bronze`, `silver`, or `gold`.
 - Example:
 - `load_bronze` → Stored procedure for loading data into the Bronze layer.
 - `load silver` → Stored procedure for loading data into the Silver layer.

id (57)* dd_bronze.s... no conectado

```

CREATE OR ALTER PROCEDURE bronze.load_bronze AS
BEGIN
    TRUNCATE TABLE bronze.crm_cust_info;
END
  
```

AdventureworksDW2014

- Tables
 - 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
- Vistas
- Recursos externos
- Sinónimos
- Programación
 - Procedimientos almacenados
 - bronze.load_bronze
- Funciones

SQLQuery2.s...David (94)* EXEC bronze.load_bronze

127 % No se encontraron problemas.

Mensajes

(18493 filas afectadas)
(397 filas afectadas)
(60398 filas afectadas)
(18483 filas afectadas)
(18484 filas afectadas)
(37 filas afectadas)

luego lo probamos

Add PRINTS

Add Prints to track execution, debug issues, and understand its flow

SQLQuery2.sq...\\David (94)* **SQLQuery1.s...David (57)***

```

1 EXEC bronze.load_bronze
2
3 CREATE OR ALTER PROC
4 BEGIN
  
```

No se pudo e
'bronze.load_

=====

Loading Bronze Layer

=====

=====

Loading CRM Tables

=====

>> Truncating Table: bronze.crm_cust_info
>> Inserting Data into: bronze.crm_cust_info

(18493 filas afectadas)

>> Truncating Table: bronze.crm_prd_info

>> Inserting Data into: bronze.crm_prd_info

(397 filas afectadas)

>> Truncating Table: bronze.crm_sales_details

>> Inserting Data into: bronze.crm_sales_details

(60398 filas afectadas)

Loading ERP Tables

>> Truncating Table: bronze.erp_cust_az12

>> Inserting Data into: bronze.erp_cust_az12

(18483 filas afectadas)

>> Truncating Table: bronze.erp_loc_a101

>> Inserting Data into: bronze.erp_loc_a101

(18484 filas afectadas)

>> Truncating Table: bronze.erp_px_cat_g1v2

>> Inserting Data into: bronze.erp_px_cat_g1v2

(37 filas afectadas)

Hora de finalización: 2025-10-09T08:37:20.9850890-03:00

The screenshot shows a SQL script window with the following code:

```
CREATE OR ALTER PROCEDURE [dbo].[LoadData]
    AS
    BEGIN TRY
        BEGIN CATCH
            PRINT '====='
            PRINT 'Load'
        END CATCH
    END TRY
    BEGIN CATCH
        PRINT '====='
        PRINT 'Load'
    END CATCH
END
```

A tooltip for the TRY...CATCH block is displayed:

(37 rows affected)
Add TRY...CATCH
Ensures error handling, data integrity, and issue logging for easier debugging

Below the code, a note states: "SQL runs the TRY block, and if it fails, it runs the CATCH block to handle the error".

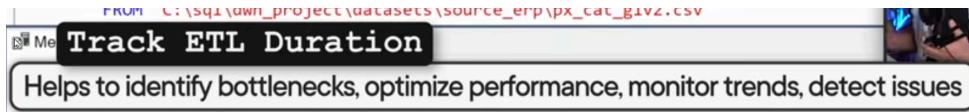
The CATCH block will execute **only if SQL fails to run the TRY block**.

Here, you tell SQL **what to do if an error occurs** in your code.

```

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

```



We want to know **how long it takes to load each table** — that way, you can identify which one is **consuming the most load time**.

We need to **declare the variables** to measure that.

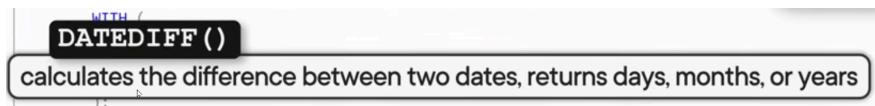
```

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 'D:\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
    WITH (
        FIRSTROW = 2,
        FIELDTERMINATOR = ',',
        TABLOCK
    );
SET @end_time = GETDATE ();

```

Now we have the values for when we started loading this table and when we finished loading the table. The next step is to print the duration.



We calculate the duration.

We define 3 arguments: 1) unit, 2) start of the interval, and 3) end of the interval.

Since the result is a number, we need to cast it.

Then we add a separator between the tables.



```

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 'D:\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 '> -----';

```

```

-- GIN
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 is Started';
    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 '=====';
    --
    -- Dual Duration. 0seconds
    >> -----
    =====

```

relation	Tasks	c	+
Build Bronze Layer	Analysing: Source Systems	<input checked="" type="checkbox"/>	
Build Bronze Layer	Coding: Data Ingestion	<input checked="" type="checkbox"/>	
Build Bronze Layer	Validating: Data Completeness & Schema Checks	<input type="checkbox"/> OPEN	<input checked="" type="checkbox"/>
Build Bronze Layer	Document: Draw Data Flow (Draw.io)	<input type="checkbox"/>	
Build Bronze Layer	Commit Code in Git Repo	<input type="checkbox"/>	

Build Bronze Layer

Document: Data Flow

