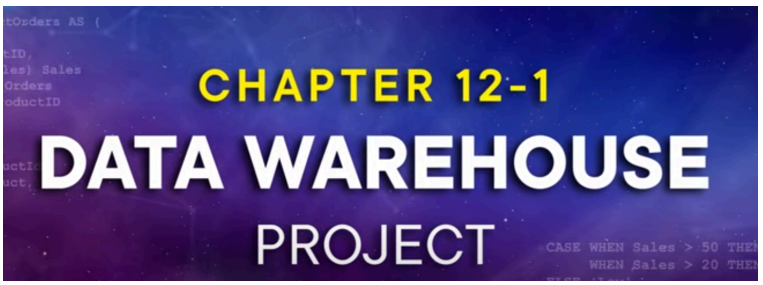
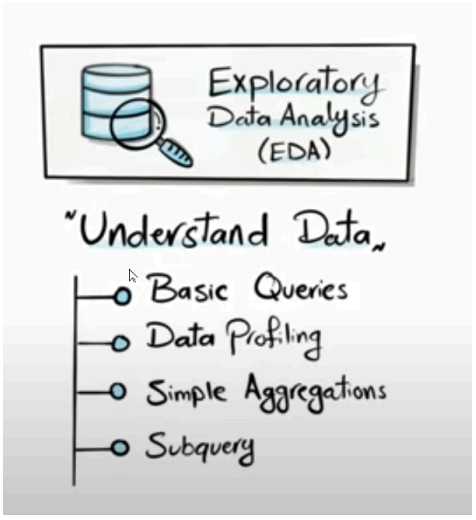
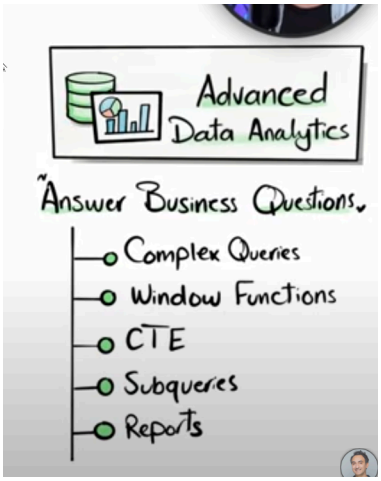
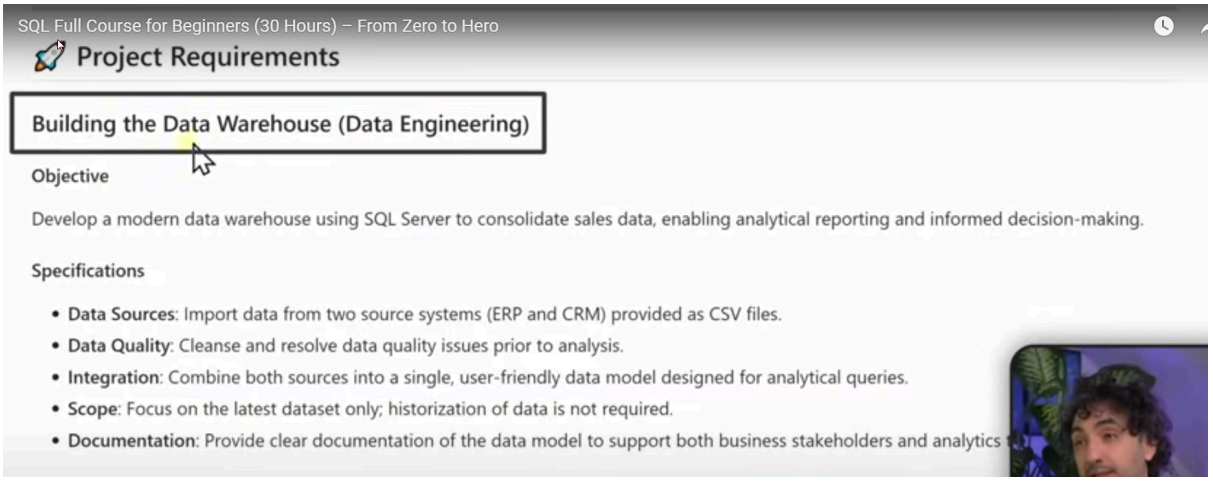
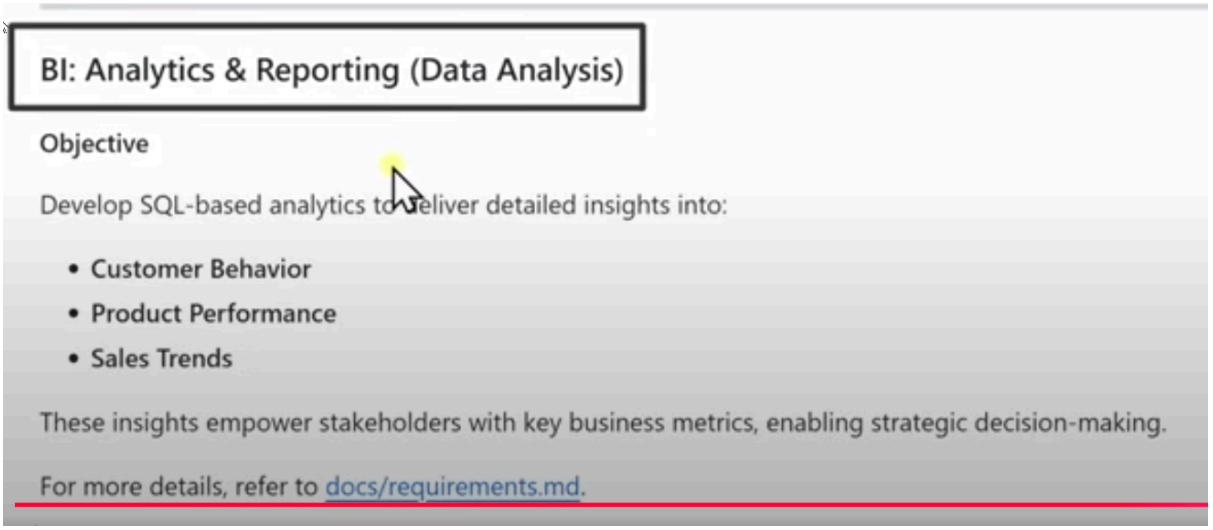






ETL (Extract Transform and Load)

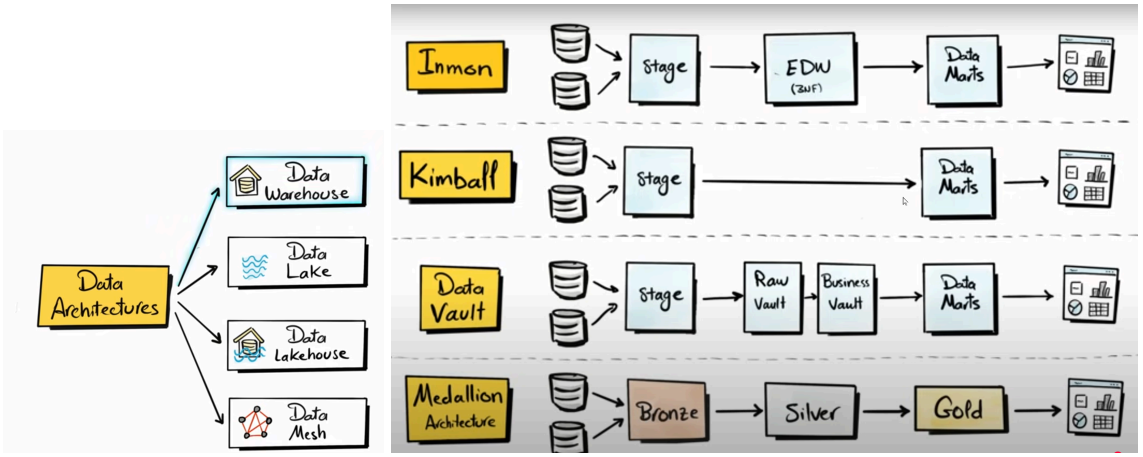

SQL Full Course for Beginners (30 Hours) – From Zero to Hero
Project Requirements
Building the Data Warehouse (Data Engineering)
Objective
Develop a modern data warehouse using SQL Server to consolidate sales data, enabling analytical reporting and informed decision-making.
Specifications
Data Sources: 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

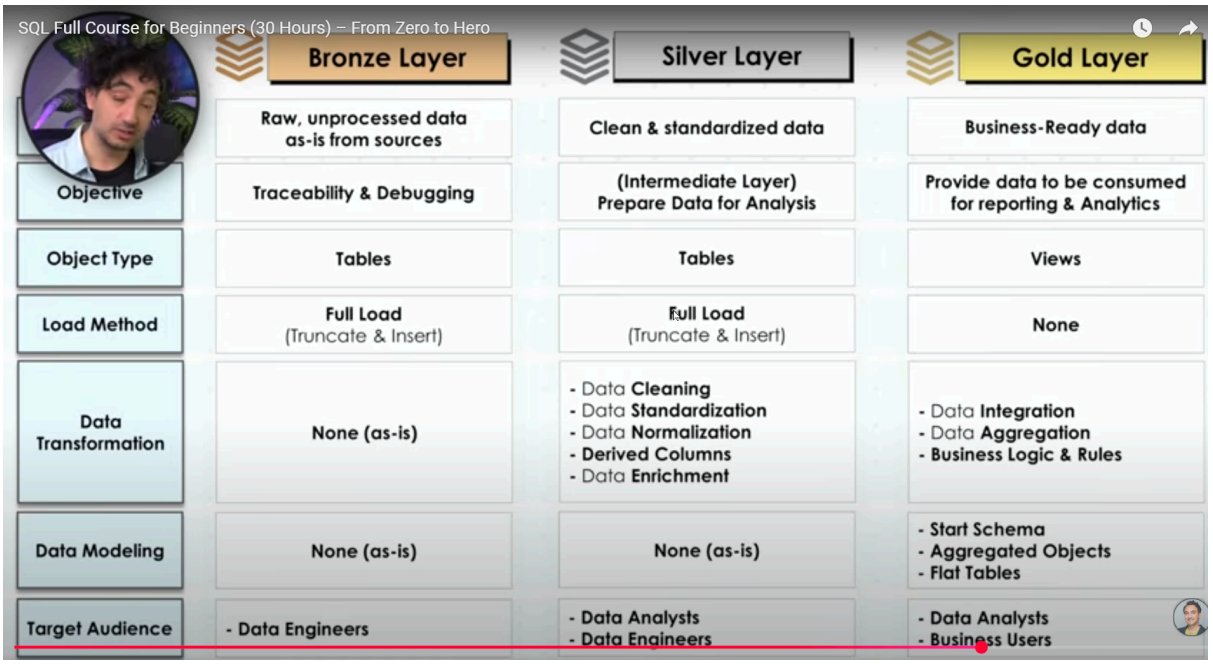

BI: Analytics & Reporting (Data Analysis)
Objective
Develop SQL-based analytics to deliver detailed insights into:
Customer Behavior
Product Performance
Sales Trends
These insights empower stakeholders with key business metrics, enabling strategic decision-making.
For more details, refer to docs/requirements.md.


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

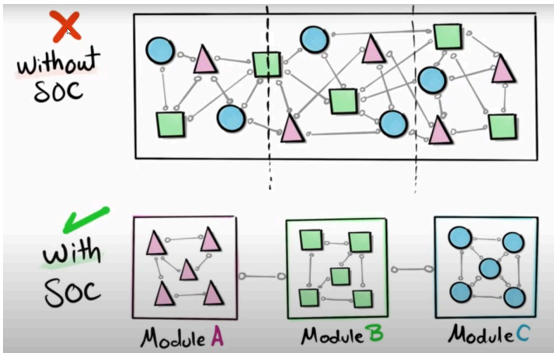

Design The Data Architecture


Design The Data Architecture
Choose the Right Approach

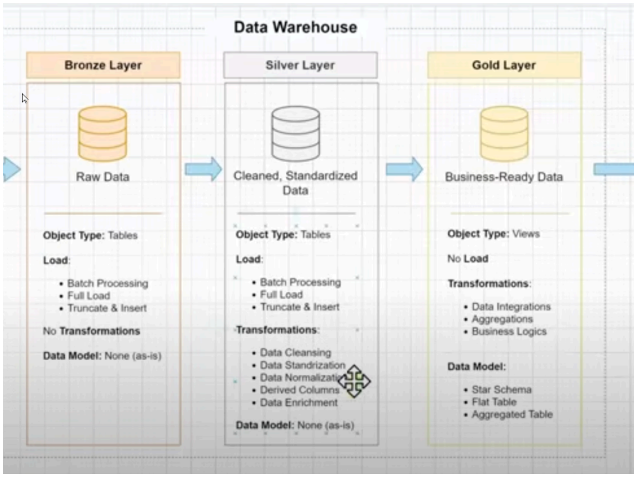




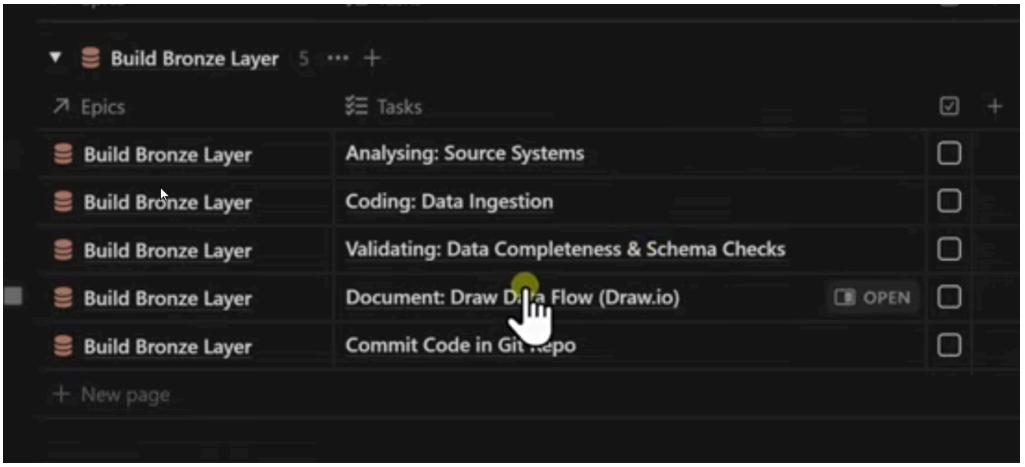
SOC separation of concerns

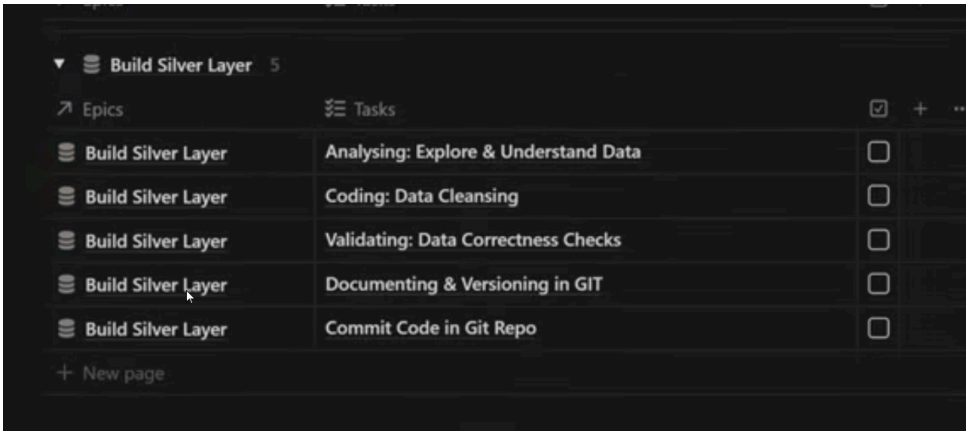

Without SOC
With SOC
Module A
Module B
Module C

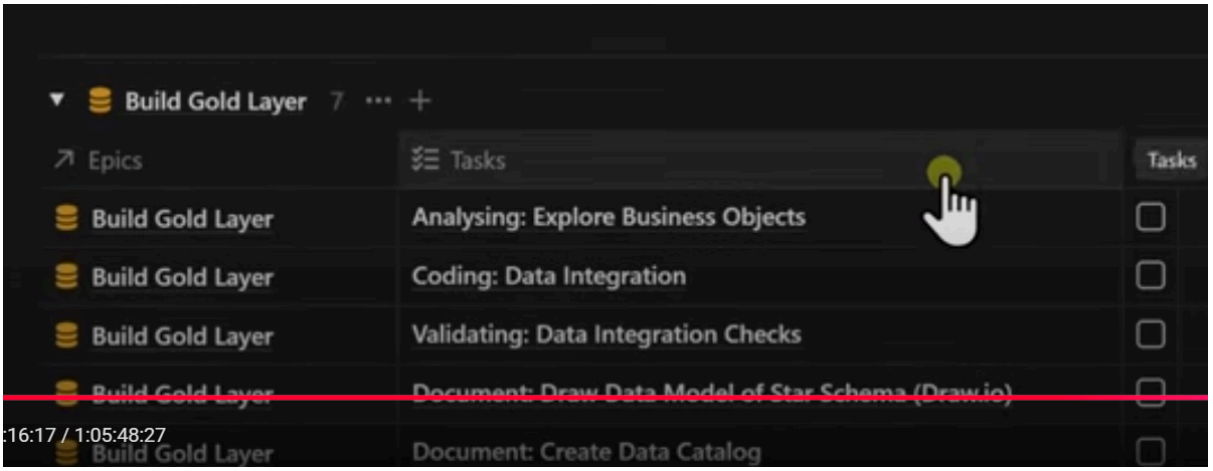

Design The Data Architecture
Draw the Architecture
- Drawi.io -
CASE WHEN Sales >
WHEN Sales >

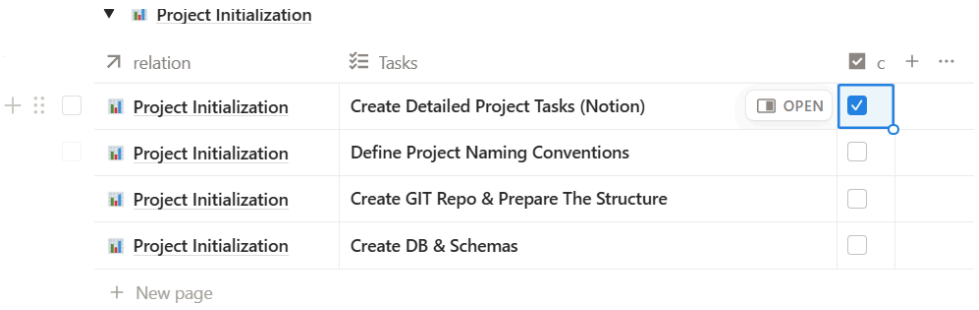

Data Warehouse
Bronze Layer
Silver Layer
Gold Layer
Raw Data
Cleaned, Standardized Data
Business-Ready Data
Object Type: Tables
Load:
• Batch Processing
• Full Load
• Truncate & Insert
No Transformations
Data Model: None (as-is)
Object Type: Tables
Load:
• Batch Processing
• Full Load
• Truncate & Insert
Transformations:
• Data Cleansing
• Data Standrization
• Data Normalization
• Derived Columns
• Data Enrichment
Data Model: None (as-is)
Object Type: Views
No Load
Transformations:
• Data Integrations
• Aggregations
• Business Logics
Data Model:
• Star Schema
• Flat Table
• Aggregated Table


Project Initialization


Build Bronze Layer 5
Epics
Tasks
Build Bronze Layer
Analysing: Source Systems
Build Bronze Layer
Coding: Data Ingestion
Build Bronze Layer
Validating: Data Completeness & Schema Checks
Build Bronze Layer
Document: Draw Data Flow (Draw.io)
OPEN
Build Bronze Layer
Commit Code in Git Repo
New page







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)
`dim_`	Dimension table	`dim_customer`, `dim_product`
`fact_`	Fact table	`fact_sales`
`agg_`	Aggregated table	`agg_customers`, `agg_sales_monthly`

Column Naming Conventions

Surrogate Keys

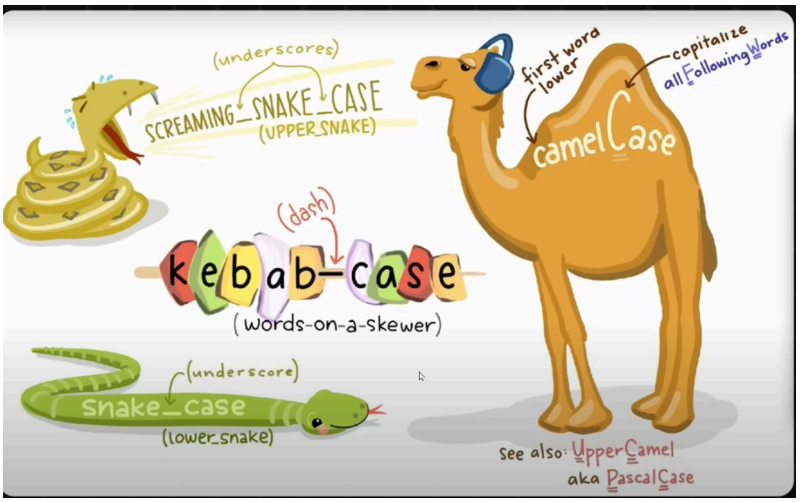
- All primary keys in dimension tables must use the suffix `_key`.
- `<table_name>_key`
  - `<table_name>`: Refers to the name of the table or entity the key belongs to.
  - `_key`: A suffix indicating that this column is a surrogate key.
  - Example: `customer_key` → Surrogate key in the `dim_customers`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

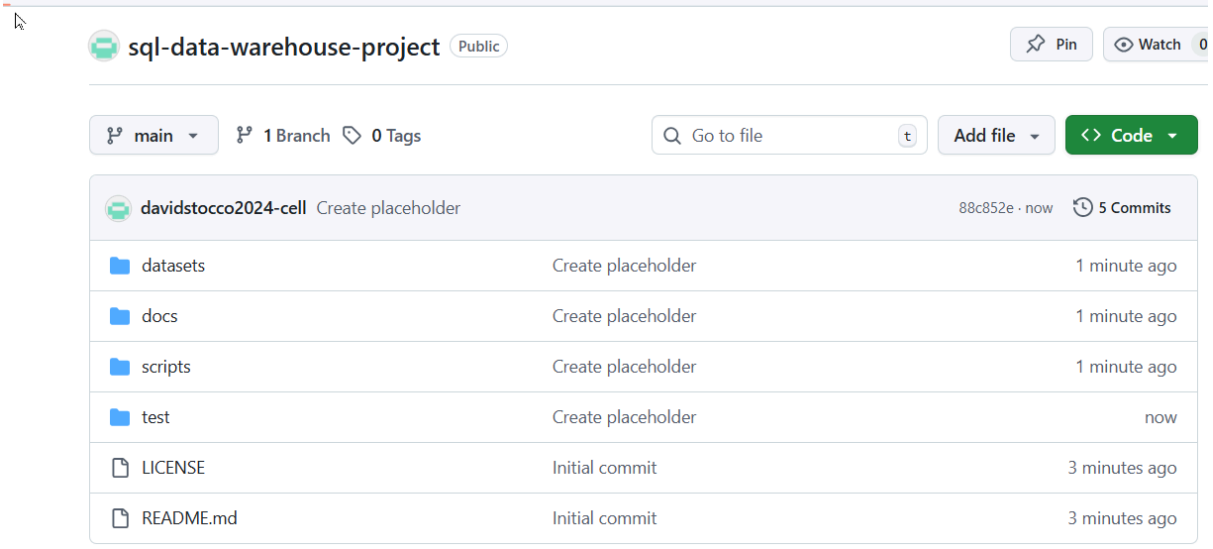
Technical Columns

- 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.



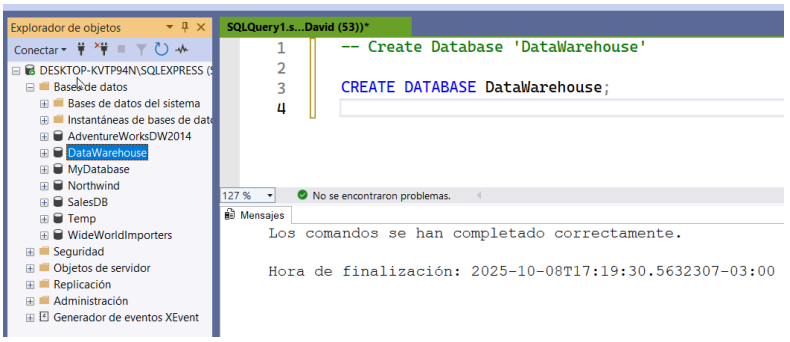


My first project repository!!! md significa mark down

Project Initialization

relation	Tasks	c
Project Initialization	Create Detailed Project Tasks (Notion)	☑
Project Initialization	Define Project Naming Conventions	☑
Project Initialization	Create GIT Repo & Prepare The Structure	☑
Project Initialization	Create DB & Schemas	☐

New page

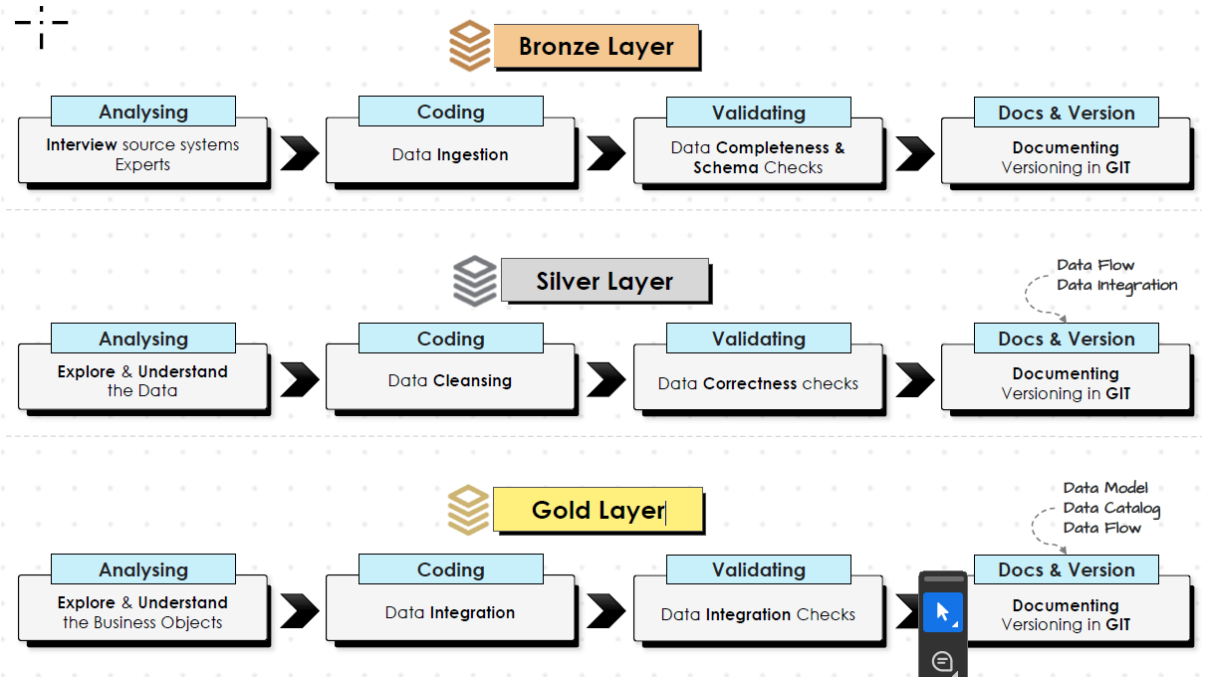


Build Bronze Layer

relation	Tasks	c
Build Bronze Layer	Analysing: Source Systems	☐
Build Bronze Layer	Coding: Data Ingestion	☐
Build Bronze Layer	Validating: Data Completeness & Schema Checks	☐
Build Bronze Layer	Document: Draw Data Flow (Draw.io)	☐
Build Bronze Layer	Commit Code in Git Repo	☐

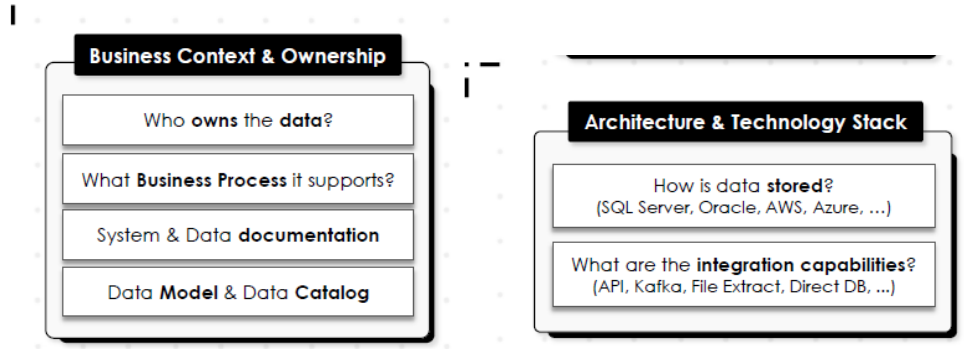
How to build the bronze layer?

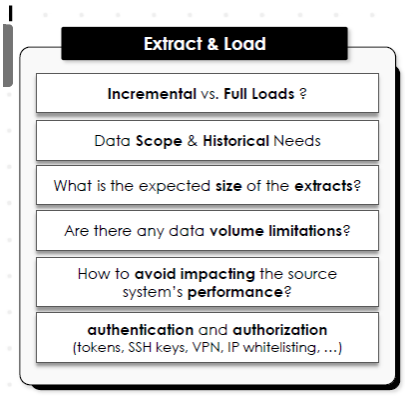
	Bronze Layer	Silver Layer	Gold Layer
Definition	Raw, unprocessed data as-is from sources	Clean & standardized data	Business-Ready data
Objective	Traceability & Debugging	(Intermediate Layer) Prepare Data for Analysis	Provide data to be consumed for reporting & Analytics
Object Type	Tables	Tables	Views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

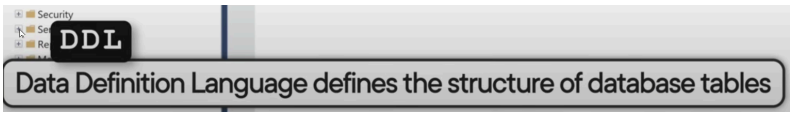


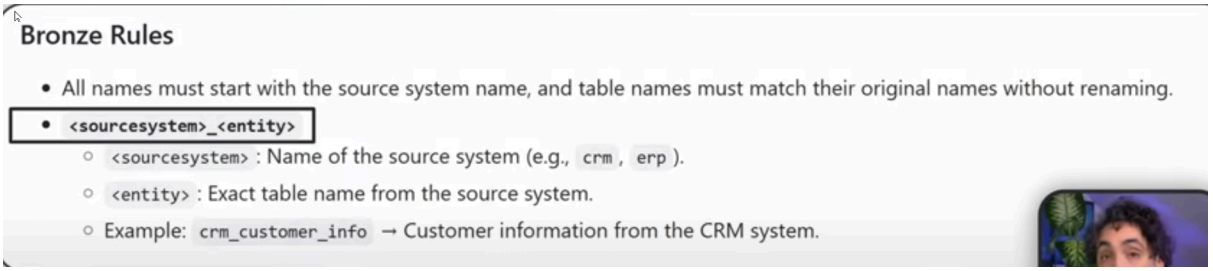
Build Bronze Layer

Analyse Source Systems









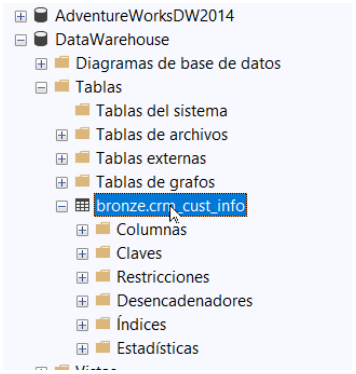
We define the **BRONZE layer**.

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

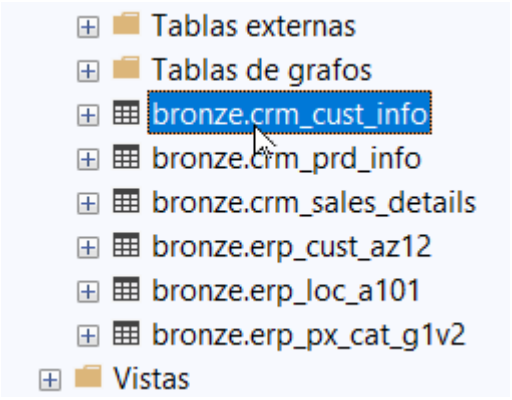
Then the **table name** from the source system: `cust_info`.

Next, we define the **columns**.

```sql
CREATE TABLE bronze.crm_cust_info(
    cst_id INT,
    cst_key NVARCHAR(50),
    cst_firstname NVARCHAR(50),
    cst_lastname NVARCHAR(50),
    cst_material_status NVARCHAR(50),
    cst_gndr NVARCHAR(50),
    cst_create_date DATE
);
```




```sql
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_quantity    INT,
    sls_price       INT
);

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

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

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



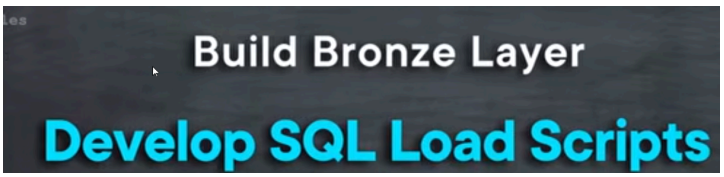
The first three tables come from the **CRM** source system, and the other three come from the **ERP** system.

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

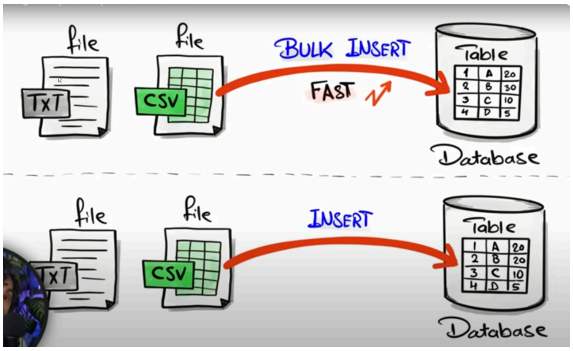
“U” stands for **User**, meaning a user-defined table.

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

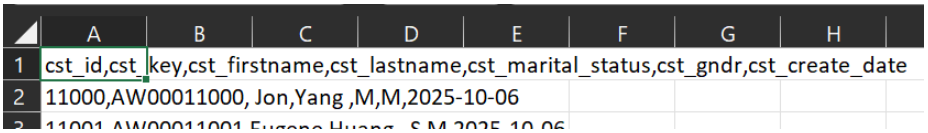
In other words, first drop the table if it exists, then create it from scratch.



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**.



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.

```
QLQuery1.s...David (84))*   ddl_bronze.sq...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 💯

```
);

SELECT * FROM bronze.crm_cust_info
```

127 % · 1 · 0

Resultados | Mensajes

	cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
1	11000	AW00011000	Jon	Yang	M	M	2025-10-06
2	11001	AW00011001	Eugene	Huang	S	M	2025-10-06
3	11002	AW00011002	Ruben	Torres	M	M	2025-10-06
4	11003	AW00011003	Christy	Zhu	S	F	2025-10-06
5	11004	AW00011004	Elizabeth	Johnson	S	F	2025-10-06
6	11005	AW00011005	Julio	Ruiz	S	M	2025-10-06
7	11006	AW00011006	Janet	Alvarez	S	F	2025-10-06
8	11007	AW00011007	Marco	Mehta	M	M	2025-10-06
9	11008	AW00011008	Rob	Verhoff	S	F	2025-10-06
10	11009	AW00011009	Shannon	Carlson	S	M	2025-10-06
11	11010	AW00011010	Jacquelyn	Suarez	S	F	2025-10-06
12	11011	AW00011011	Curtis	Lu	M	M	2025-10-06
13	11012	AW00011012	Lauren	Walker	M	F	2025-10-06
14	11013	AW00011013	Ian	Jenkins	M	M	2025-10-06
15	11014	AW00011014	Sydney	Bennett	S	F	2025-10-06
16	11015	AW00011015	Chloe	Young	S	F	2025-10-06
17	11016	AW00011016	Wyatt	Hill	M	M	2025-10-07
18	11017	AW00011017	Shannon	Wang	S	F	2025-10-07
19	11018	AW00011018	Clarence	Rai	S	M	2025-10-07

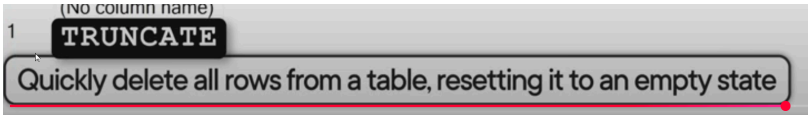
```
SELECT COUNT(*) FROM bronze.crm_cust_info
```

127 % · No se encontraron problemas.

Resultados | Mensajes

	(Sin nombre de columna)
1	18493

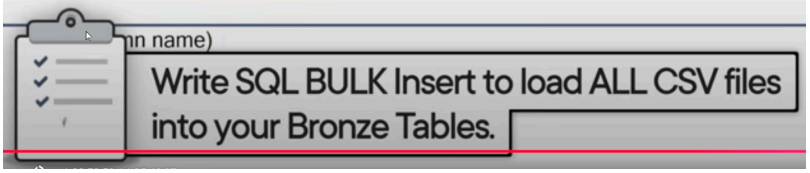
contamos las filas



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

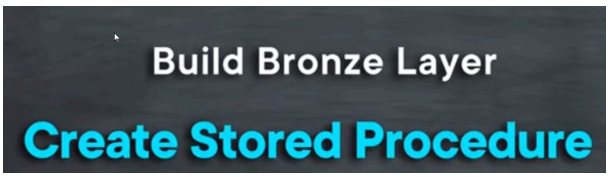
```
TRUNCATE TABLE bronze.crm_cust_info;

BULK INSERT bronze.crm_cust_info
        FROM 'D:\sql-data-warehouse-project\datasets\source_crm\cust_info.csv'
        WITH (
            FIRSTROW = 2,
            FIELDTERMINATOR = ',',
            TABLOCK
        );
```

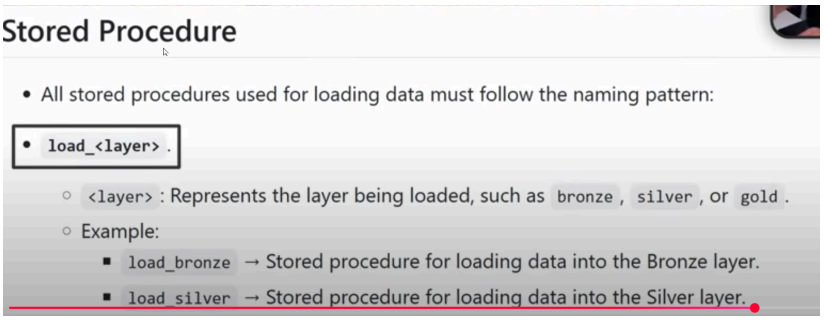
First, we perform a **TRUNCATE** to completely clear the table, and then we **INSERT the data**.



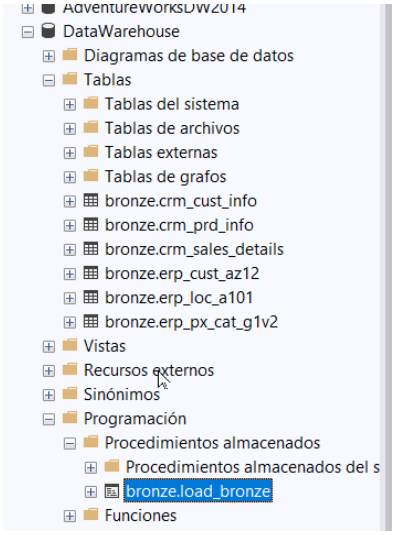
Now we need to go **table by table** and **check the content**.

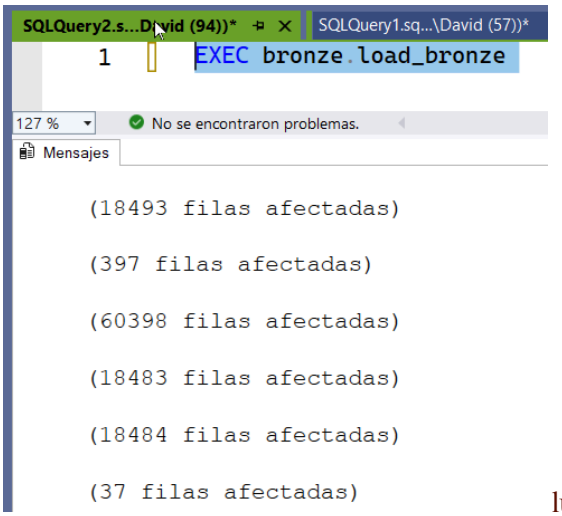
.



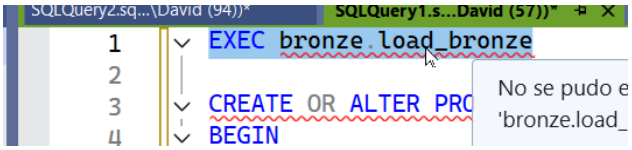



```
CREATE OR ALTER PROCEDURE bronze.load_bronze AS
BEGIN
    TRUNCATE TABLE bronze.crm_cust_info;
```





luego lo probamos



==========

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

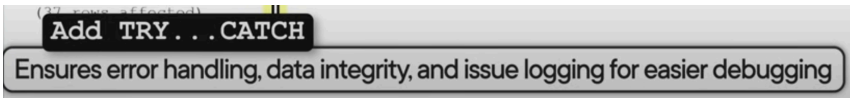
>> Truncating Table: bronze.erp_cust_az12
>> Inserting Data into: bronze.erp_cust_az12

(18483 filas afectadas)
>> Truncating Table: bronze.erp_loc_a101
>> Inserting Data into: bronze.erp_loc_a101

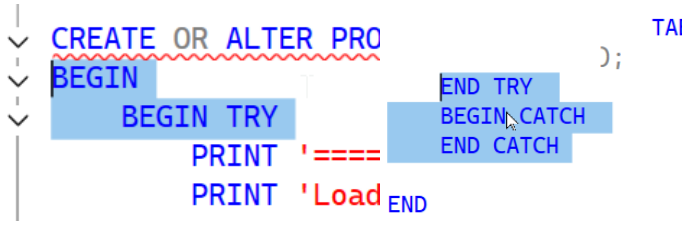
(18484 filas afectadas)
>> Truncating Table: bronze.erp_px_cat_g1v2
>> Inserting Data into: bronze.erp_px_cat_g1v2

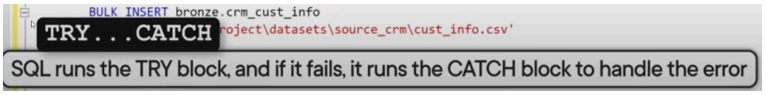
(37 filas afectadas)

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



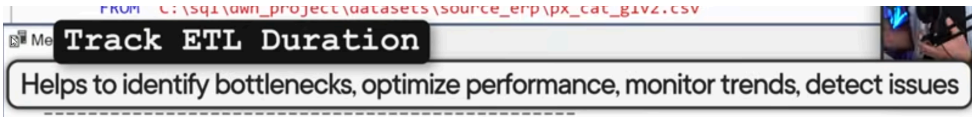
we need to handle the errors in our stored procedures





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
```



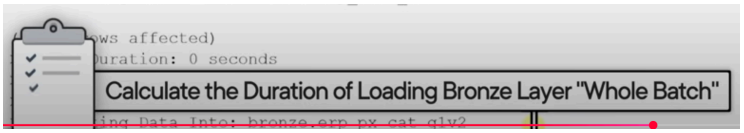
We want to know **how long it takes to load each table** — that way, you can identify which one is **consuming the most load time**.
We need to **declare the variables** to measure that.

```
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 '================================================';

    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 '================================================';
```

```
>> Load Duration: 0seconds
>> ----------------------------------------
================================================
Loading Bronze Layer is Completed
 -  Total Load Duration: 0seconds
================================================
```

relation	Tasks	c
Build Bronze Layer	Analysing: Source Systems	☑
Build Bronze Layer	Coding: Data Ingestion	☑
Build Bronze Layer	Validating: Data Completeness & Schema Checks OPEN	☑
Build Bronze Layer	Document: Draw Data Flow (Draw.io)	☐
Build Bronze Layer	Commit Code in Git Repo	☐

Build Bronze Layer

Document: Data Flow