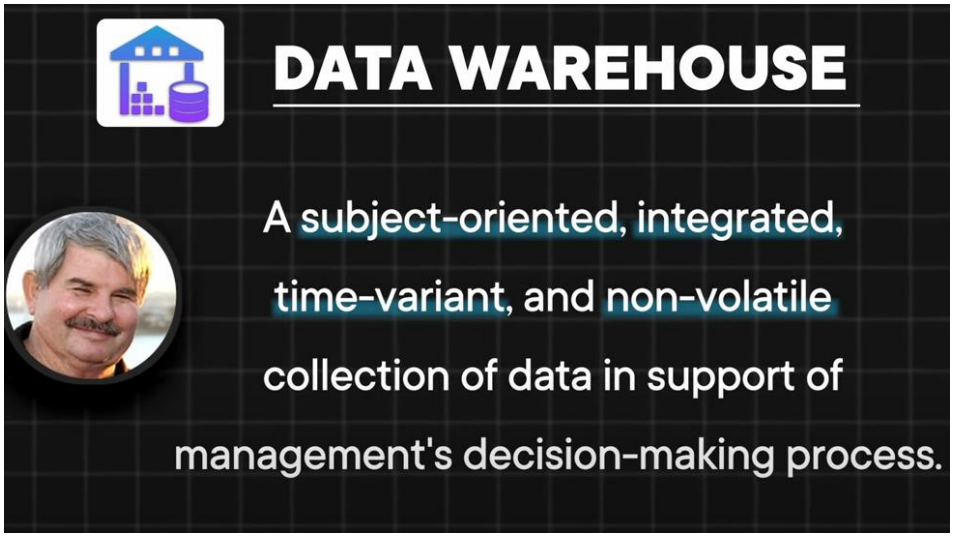
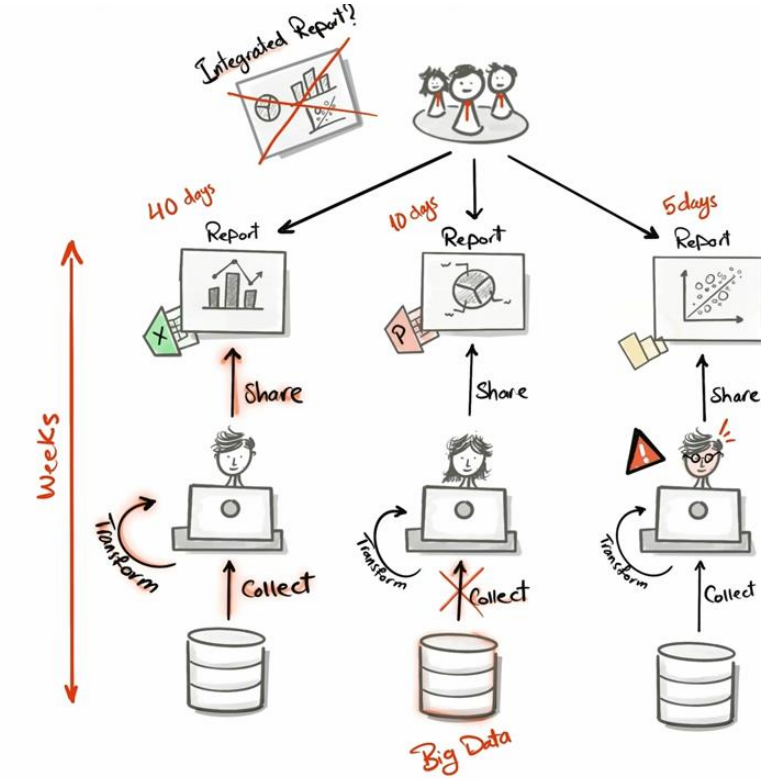


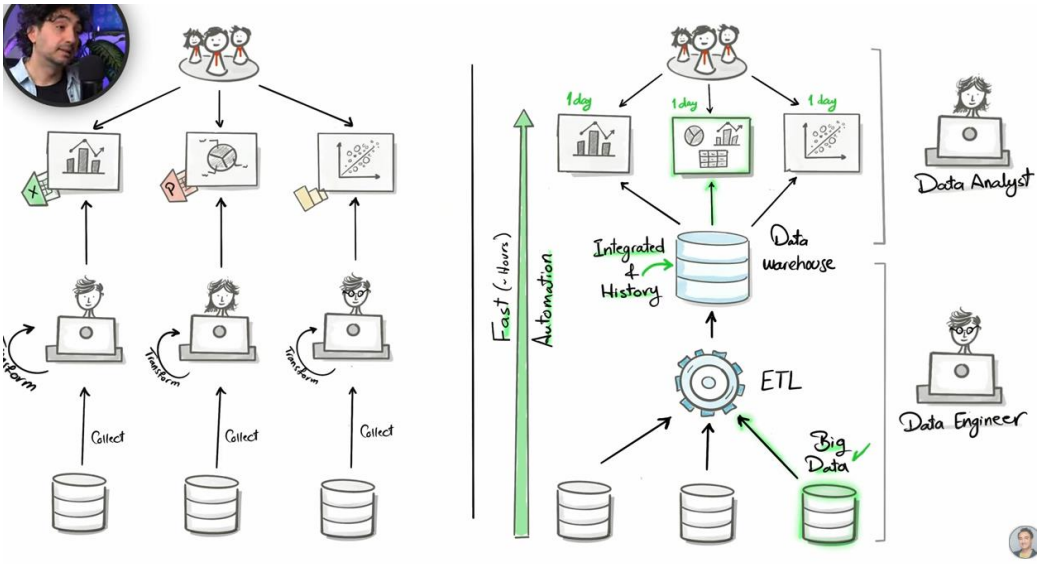
When data is collected from multiple sources, Data Analysts manually compile and generate reports for stakeholders. However, this approach is time-consuming, especially when handling large data sources (Big Data), which can lead to delays in report generation. As a result, this method is inefficient and tedious for large-scale reporting.

Moreover, if we need to create an integrated report that combines insights from multiple reports built on different sources, the process becomes even more complex and inefficient. This setup lacks effective data management, making it challenging to maintain consistency, accuracy, and scalability.



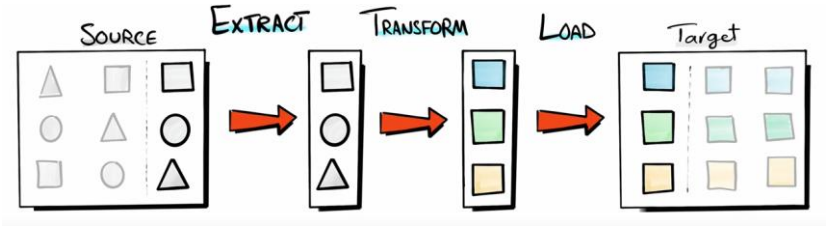
To overcome this challenge, we need a Data Warehouse where a copy of the source data and transformed data is continuously stored in a single database, Delta Lakehouse, or an S3 bucket. Data Engineers perform ETL operations to move data from various sources into the Data Warehouse, ensuring that all relevant information is centralized and up to date.

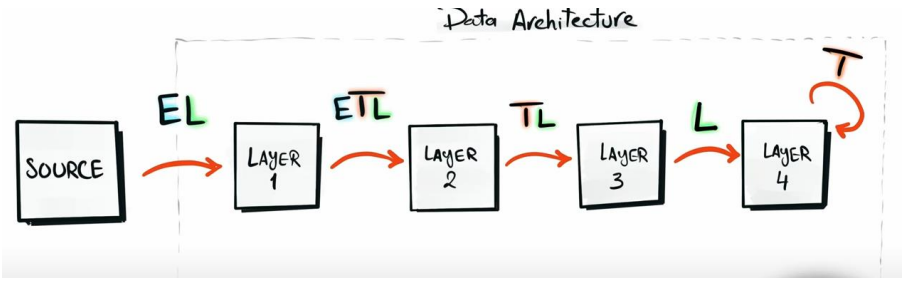
Once the data is available in the Data Warehouse, Data Analysts can efficiently generate reports for stakeholders. Additionally, integrated reports can be built more easily since the Data Warehouse always holds the latest data, thanks to ongoing ETL processes. This approach improves data management, enhances reporting efficiency, and ensures timely insights.

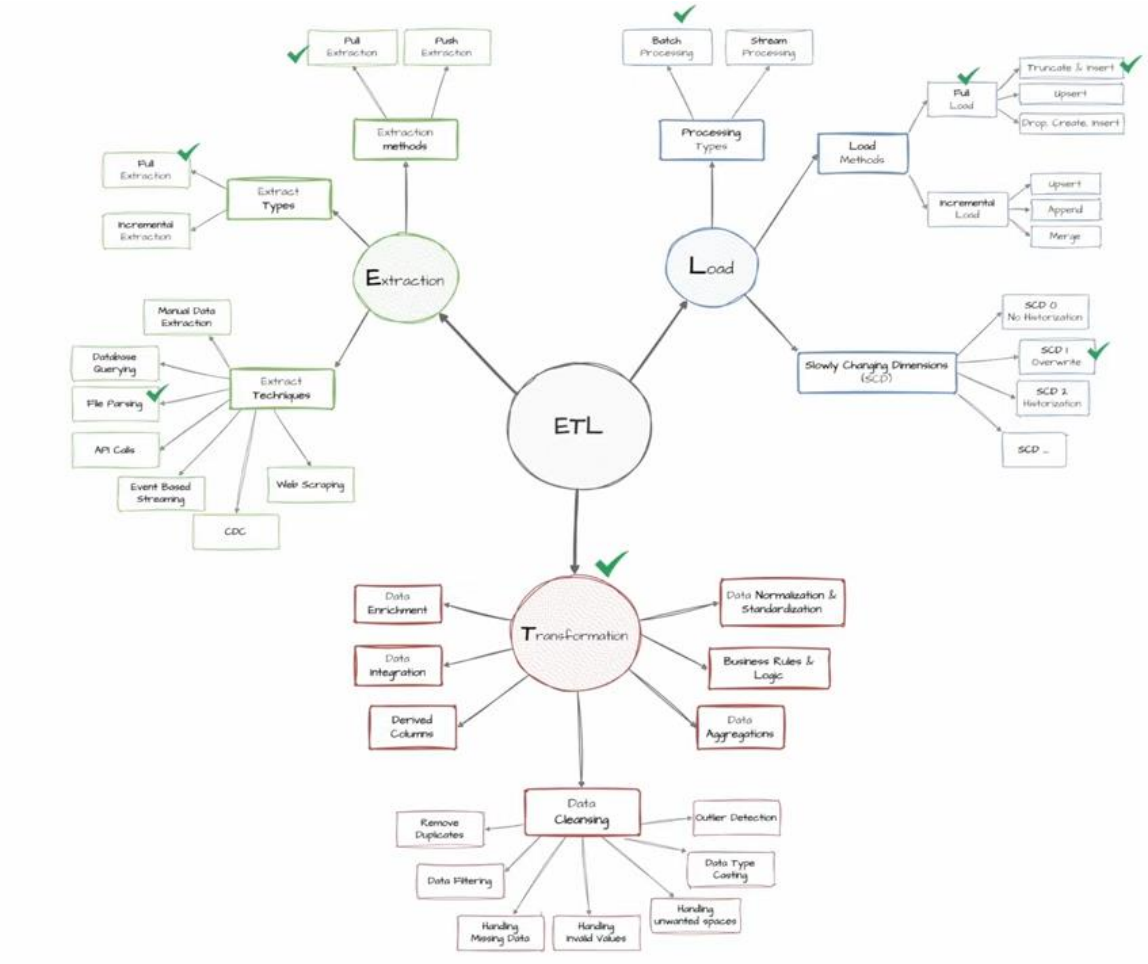


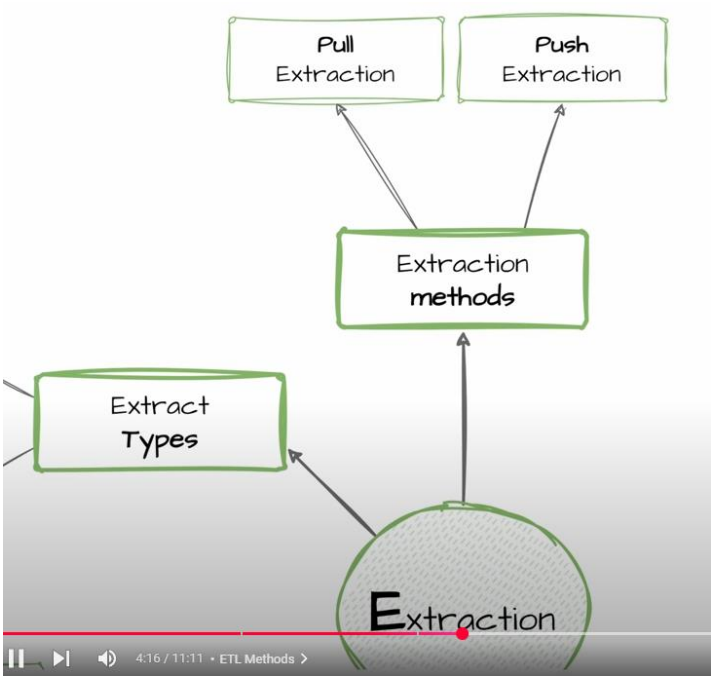
ETL

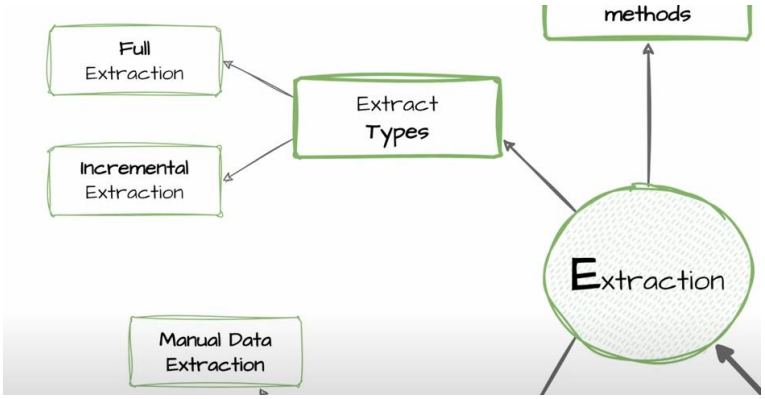
ETL is a fundamental component of any Data Warehouse project. It plays a crucial role in extracting, transforming, and loading data, but the implementation of ETL varies for each project. The approach to ETL is entirely dependent on the architecture of the system, which dictates how data is sourced, processed, and stored to meet specific business requirements.

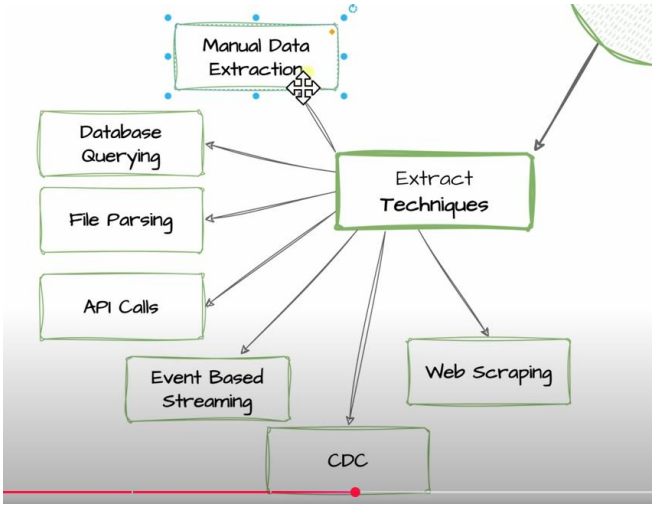


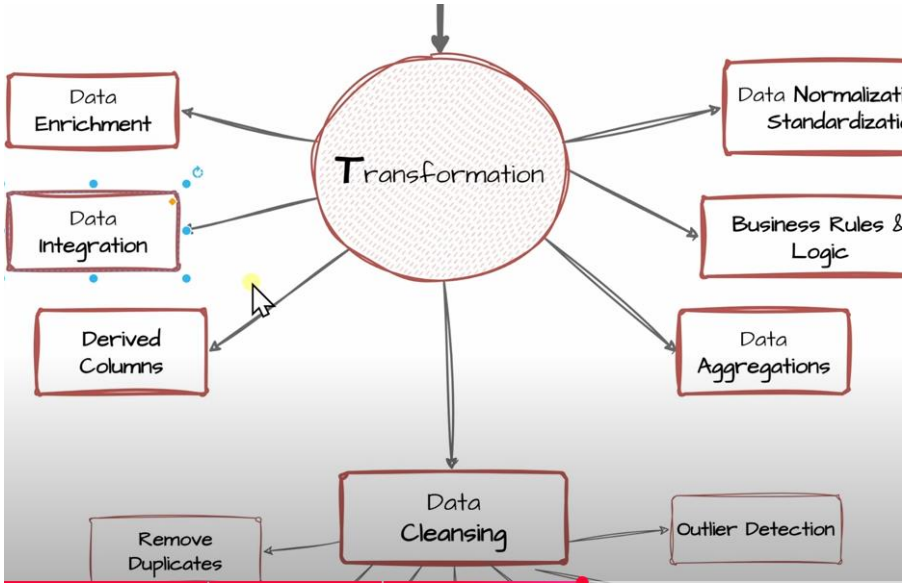


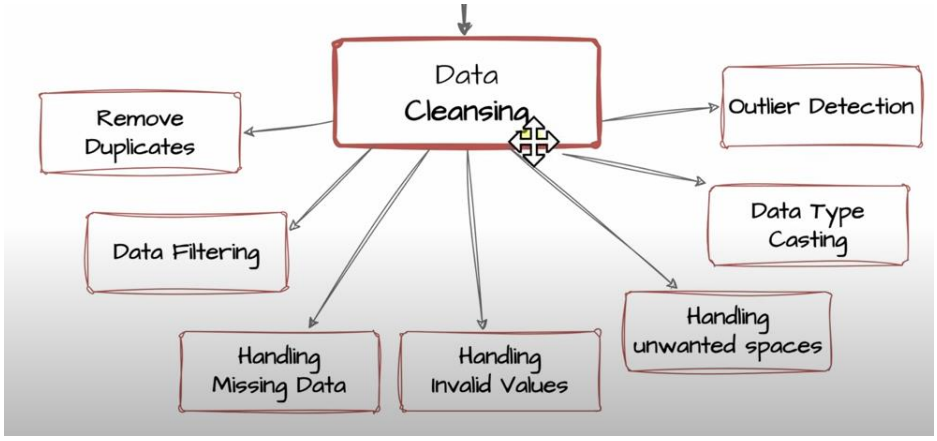


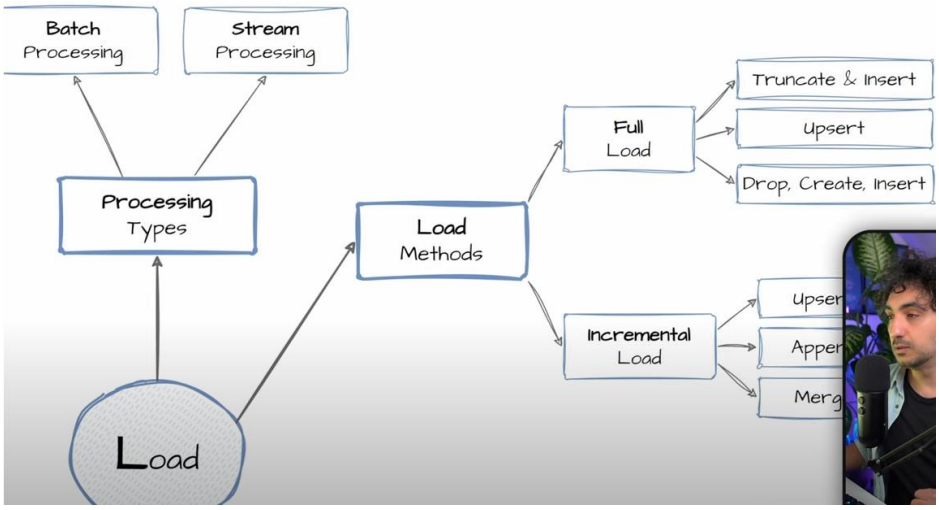

Pull
Extraction
Push
Extraction
Extraction
methods
Extract
Types
Extraction
4:16 / 11:11 • ETL Methods

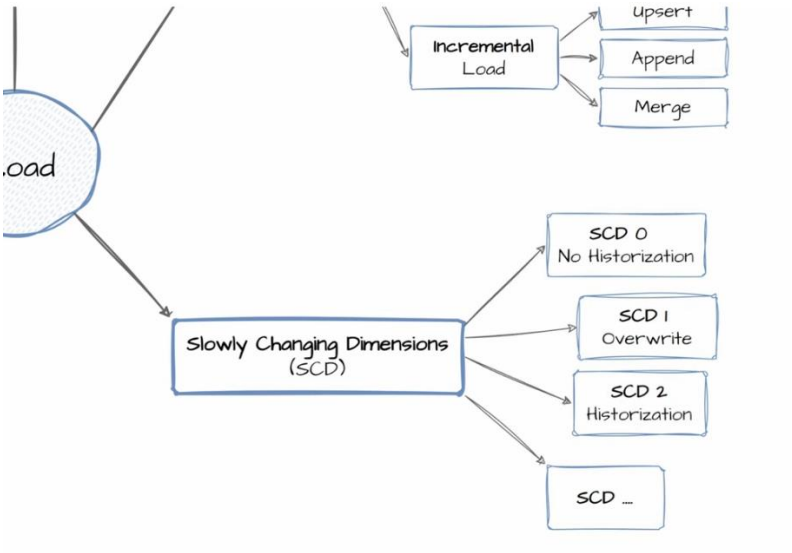

methods
Full
Extraction
Extract
Types
Incremental
Extraction
Extraction
Manual Data
Extraction


Manual Data
Extraction
Database
Querying
Extract
Techniques
File Parsing
API Calls
Event Based
Streaming
Web Scraping
CDC


Transformation
Data Enrichment
Data Integration
Derived Columns
Data Normalizat
Standardizati
Business Rules &
Logic
Data Aggregations
Data Cleansing
Remove Duplicates
Outlier Detection


Data Cleansing
Remove Duplicates
Outlier Detection
Data Filtering
Data Type Casting
Handling unwanted spaces
Handling Missing Data
Handling Invalid Values


Batch Processing
Stream Processing
Processing Types
Load
Load Methods
Full Load
Truncate & Insert
Upsert
Drop, Create, Insert
Incremental Load



Notion

Add icon Add cover Add comment

Data Warehouse Project

Project Epics

Aa Name	Progress
Requirement Analysis	0.00%
Design Data Architecture	0.00%
Project Initialization	0.00%

+ New page

Project Tasks

↑ Tasks | Tasks

- Requirement Analysis
- Design Data Architecture
- Project Initialization

1 hidden group

+ Add a group

Requirement Analysis

🚀 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 t

BI: Analytics & Reporting (Data Analysis)

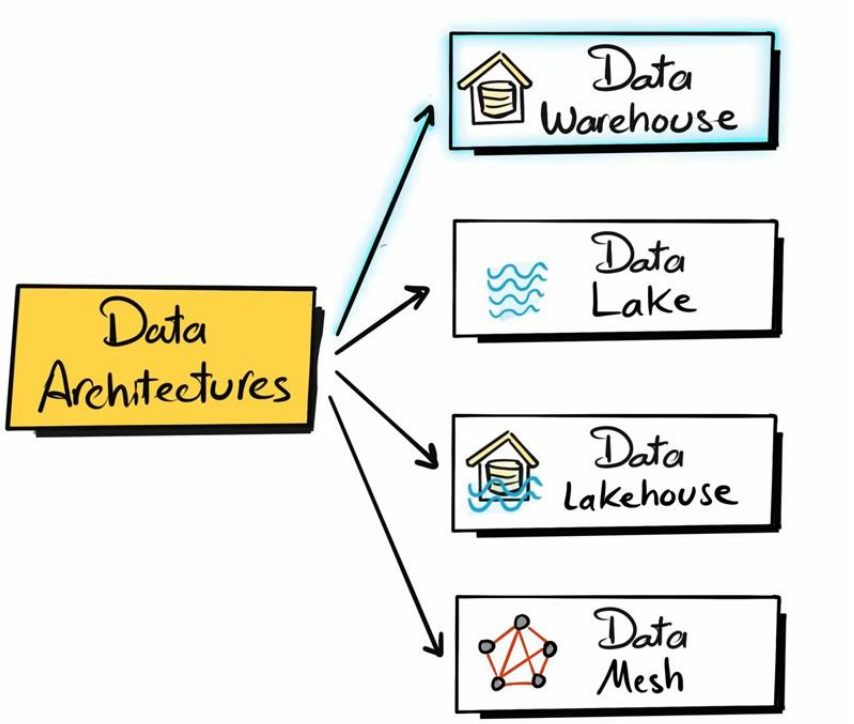
Objective

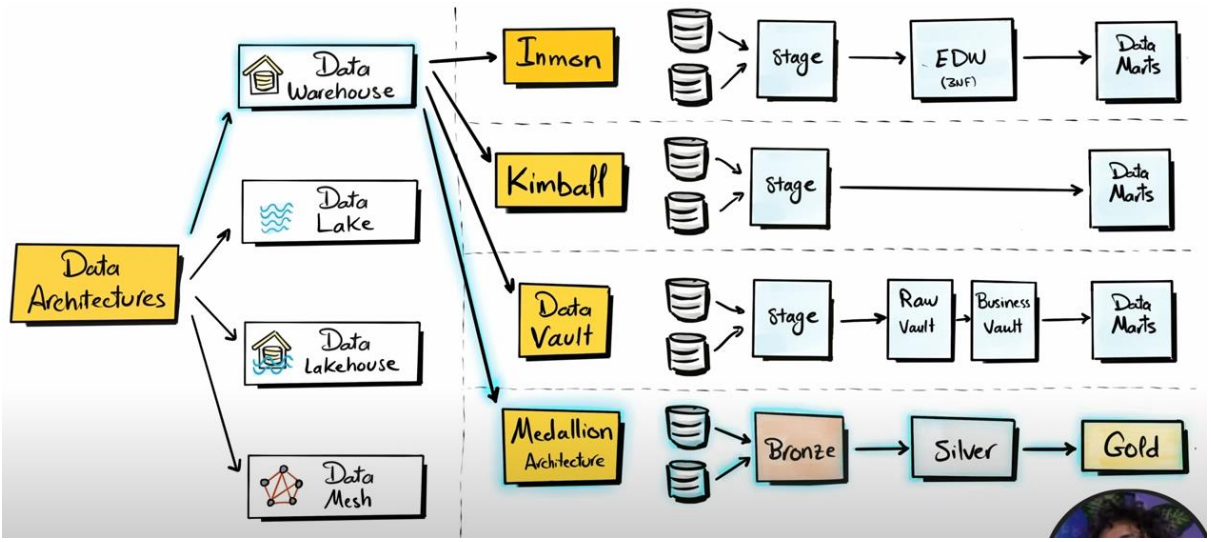
Develop SQL-based analytics to deliver detailed insights into:

- **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.





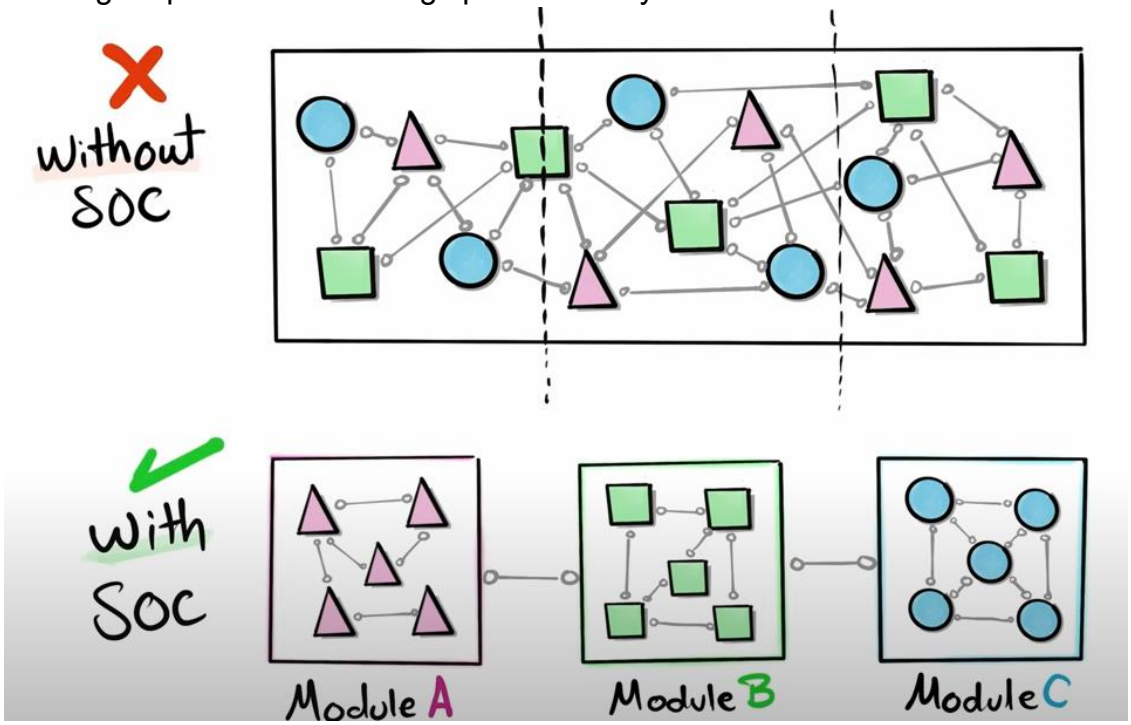
Design The Data Architecture

Design the Layers

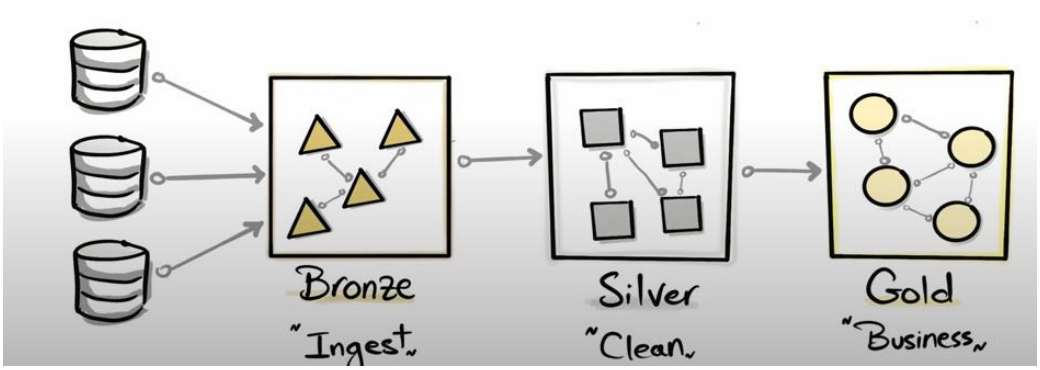
	Bronze Layer	Silver Layer	Gold Layer
Definition	Raw, unprocessed data as-is from sources	Clean & standardized data	Business-Ready data
Objective	Traceability & Debugging	(Intermediate Layer) Prepare Data for Analysis	Provide data to be consumed for reporting & Analytics
Object Type	Tables	Tables	Views
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

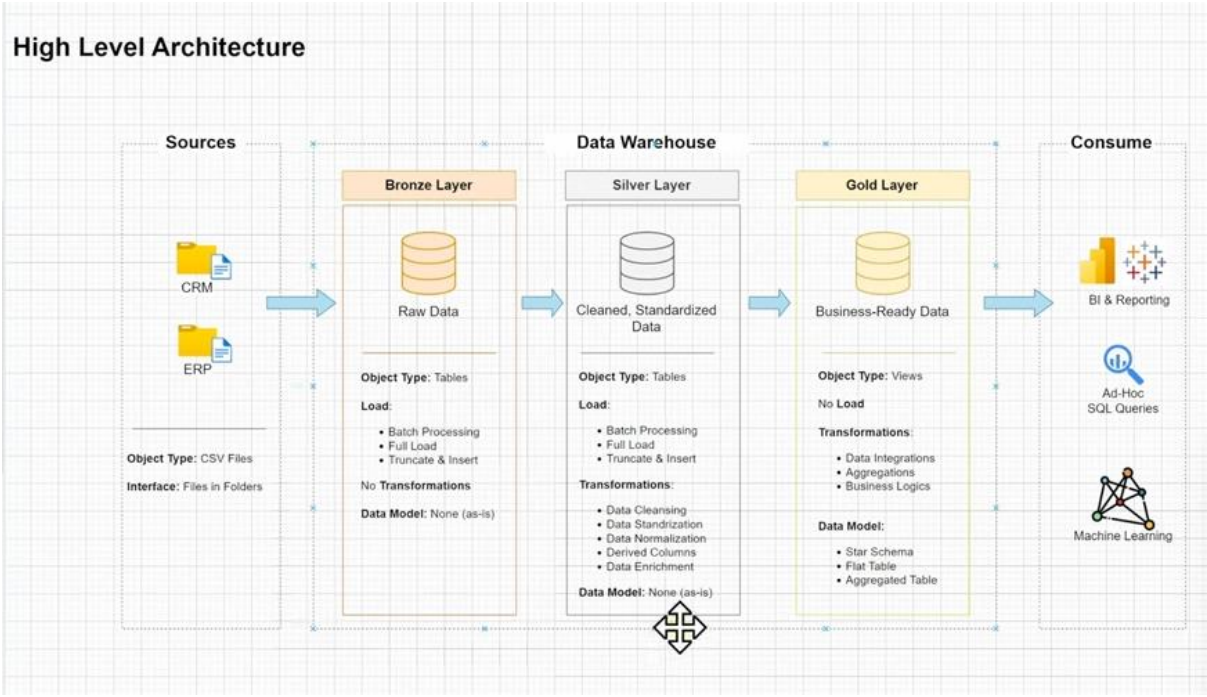
Separation of Concerns in Data Architecture

Separation of concerns (SoC) in data architecture ensures that different components of a data system are designed and managed independently, improving scalability, maintainability, and security. It helps in structuring data systems effectively by dividing responsibilities among specialized layers.



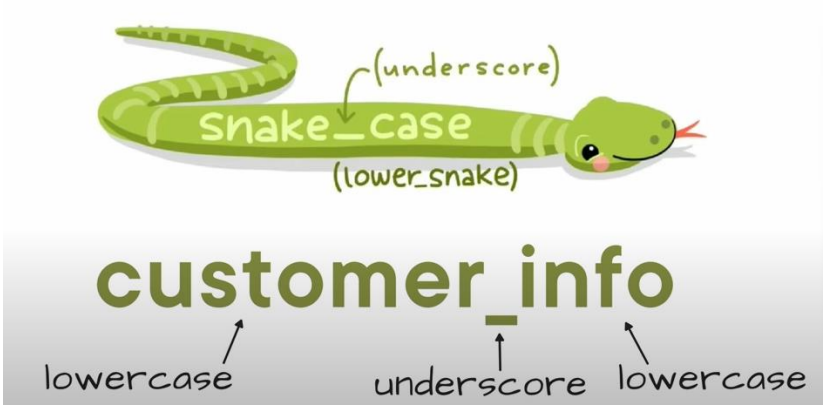
A well-designed data project ensures that models and processes operate independently, allowing for better scalability, maintainability, and flexibility.




High Level Architecture
Sources
CRM
ERP
Object Type: CSV Files
Interface: Files in Folders
Data Warehouse
Bronze Layer
Raw Data
Object Type: Tables
Load:
• Batch Processing
• Full Load
• Truncate & Insert
No Transformations
Data Model: None (as-is)
Silver Layer
Cleaned, Standardized Data
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)
Gold Layer
Business-Ready Data
Object Type: Views
No Load
Transformations:
• Data Integrations
• Aggregations
• Business Logics
Data Model:
• Star Schema
• Flat Table
• Aggregated Table
Consume
BI & Reporting
Ad-Hoc SQL Queries
Machine Learning


Project Initialization
Define Naming Conventions


Naming Conventions
Set of Rules or Guidelines
for naming anything in the project.
- Database
- Schema
- Tables
- Store Procedures ...



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

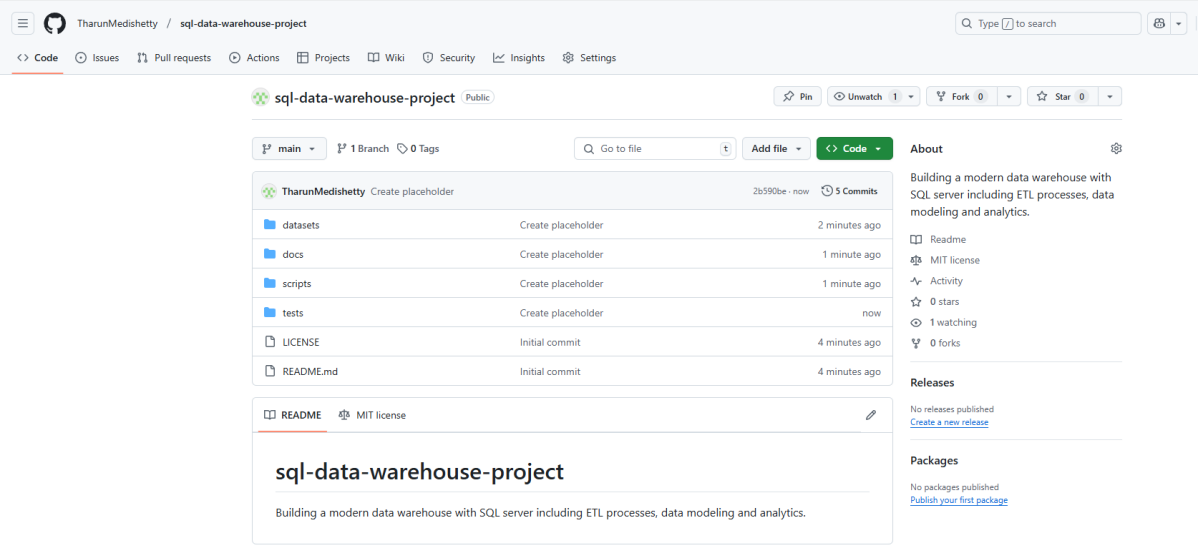
- All primary keys in dimension tables must use the suffix `_key`.
- `table_name>_key`
  - `<table_name>`: Refers to the name of the table or entity the key belongs to.
  - `_key`: A suffix indicating that this column is a surrogate key.
  - Example: `customer_key` → Surrogate key in the `dim_customers`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

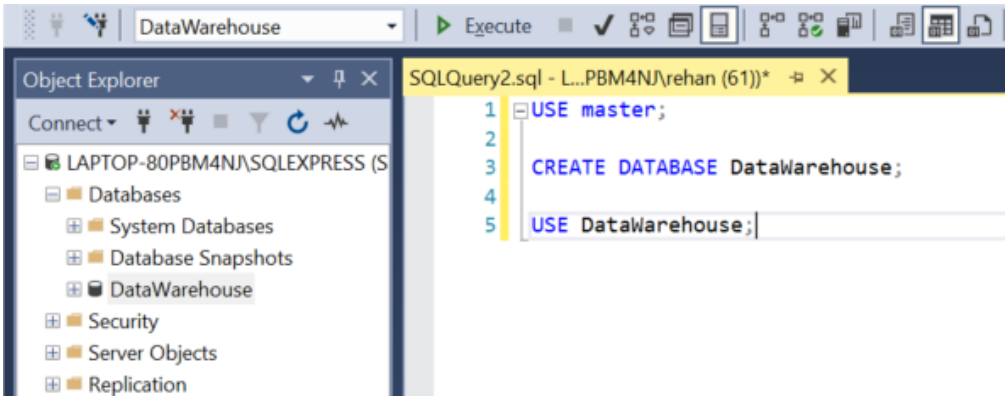

TharunMedishetty / sql-data-warehouse-project
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
sql-data-warehouse-project
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
TharunMedishetty Create placeholder
2b590be · now
5 Commits
datasets Create placeholder 2 minutes ago
docs Create placeholder 1 minute ago
scripts Create placeholder 1 minute ago
tests Create placeholder now
LICENSE Initial commit 4 minutes ago
README.md Initial commit 4 minutes ago
Building a modern data warehouse with SQL server including ETL processes, data modeling and analytics.
Readme
MIT license
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
README
MIT license
sql-data-warehouse-project
Building a modern data warehouse with SQL server including ETL processes, data modeling and analytics.

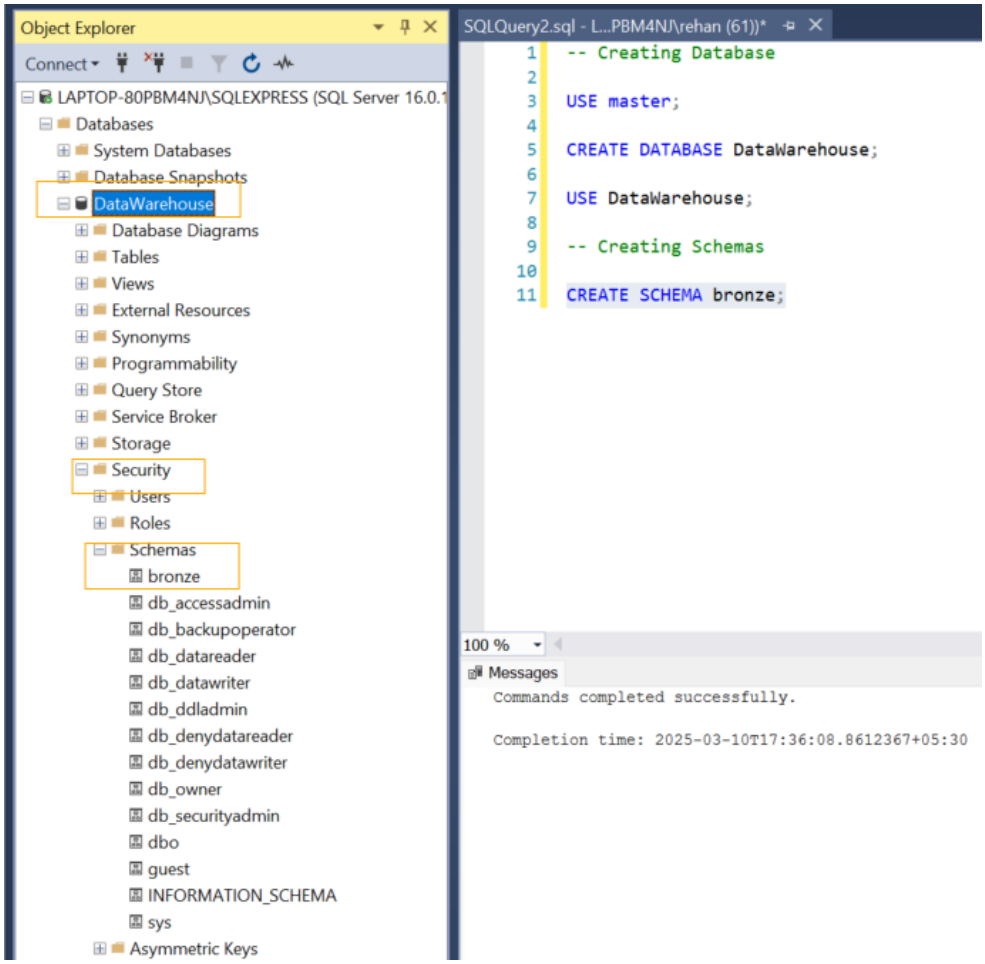

Markdown (.md)
Lightweight markup language that you can use
to add formatting elements to plaintext text documents.

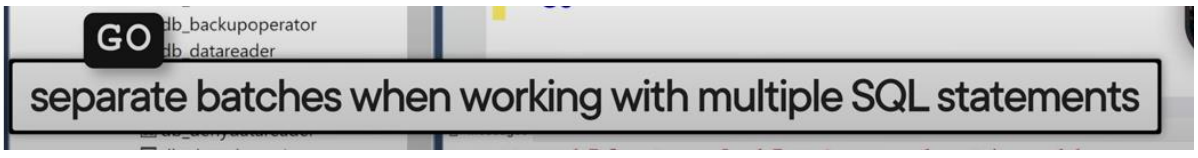

What is Markdown?
see [Markdown](www.markdownlink.com)
> Markdown is a lightweight markup language

List of tips
1. *One asterisk Italicizes*
2. **Two asterisks emphasize**
What is Markdown?
see Markdown
Markdown is a lightweight markup language
List of tips
1. One asterisk Italicizes
2. Two asterisks emphasize

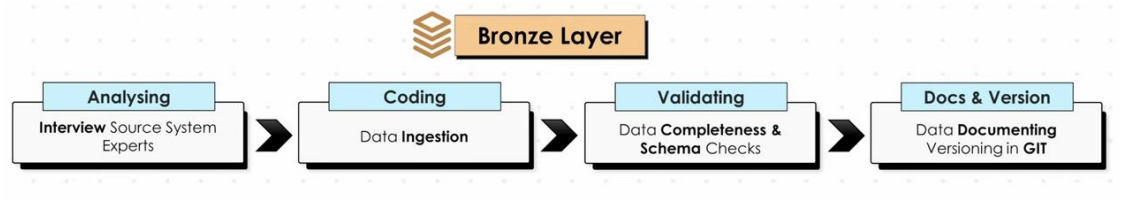

Project Initialization
Create Database & Schemas





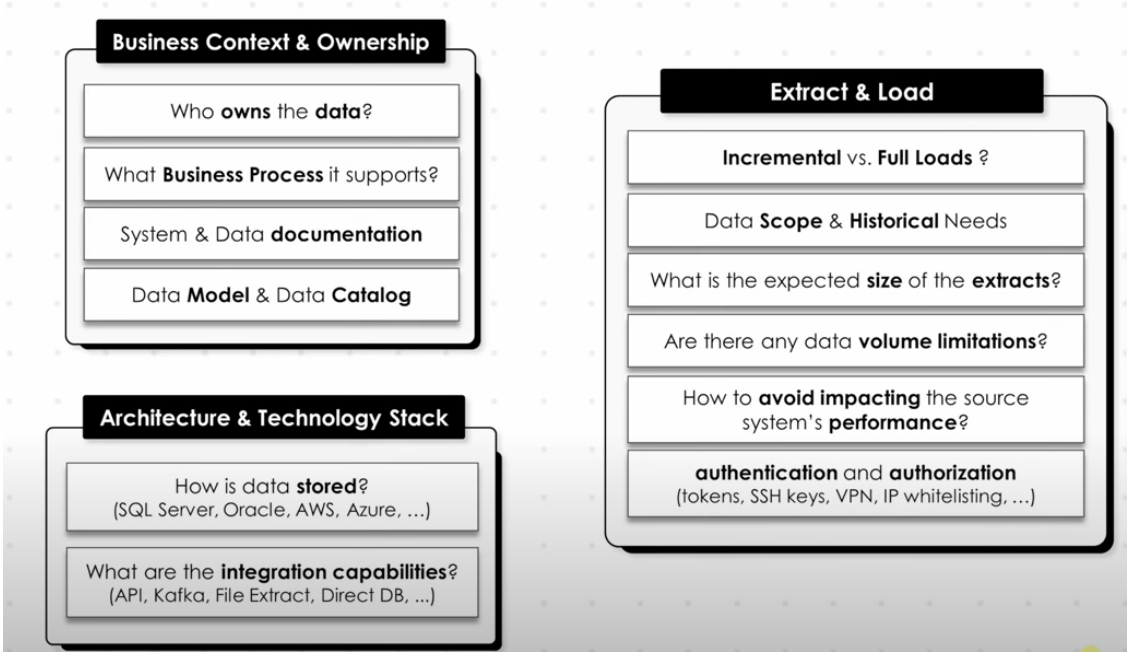


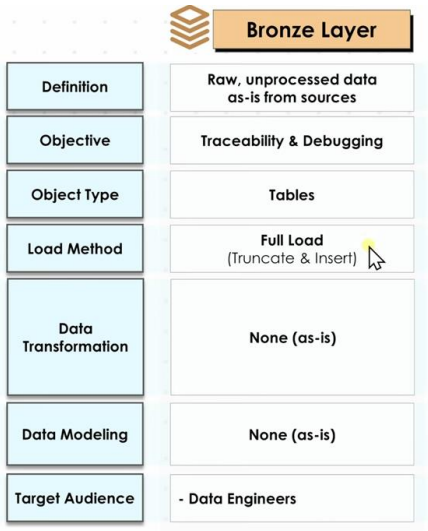
Bronze Layer



Build Bronze Layer

Analyse Source Systems





DDL

Data Definition Language defines the structure of database tables

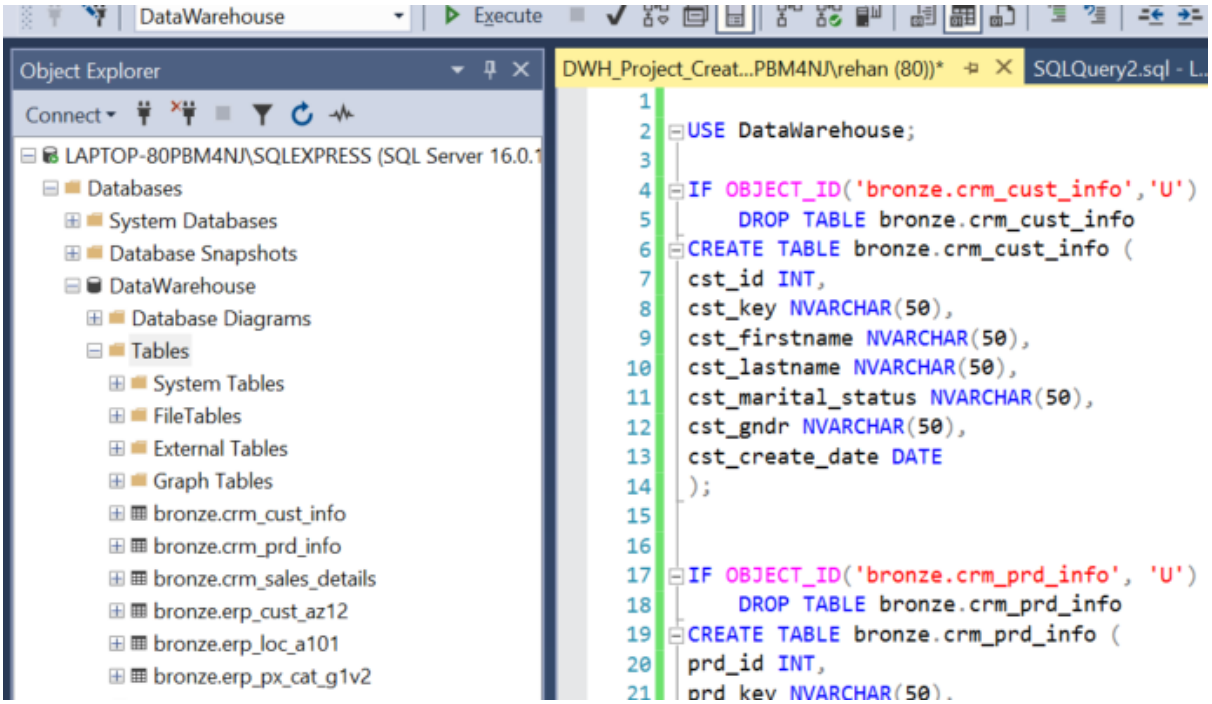
Consult the technical experts of the source system to understand its metadata.

DATA PROFILING

Explore the data to identify column names and data types

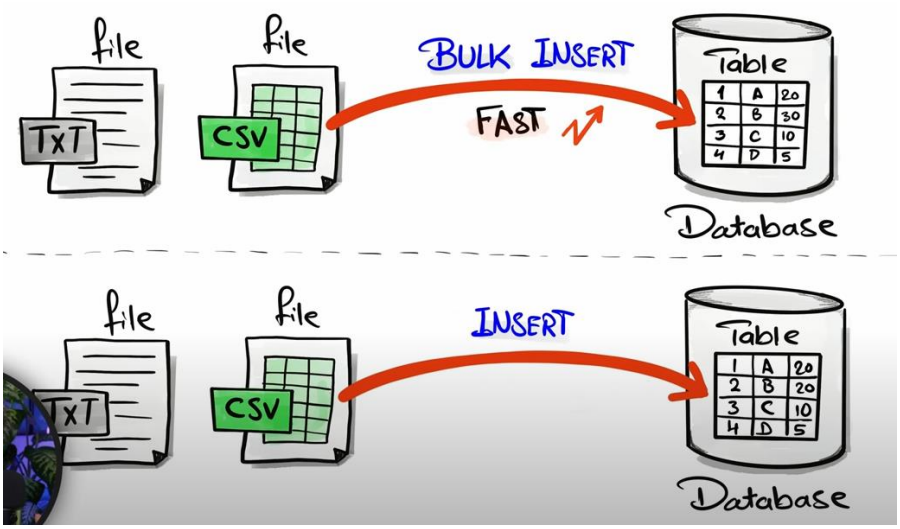
Bronze Rules

- 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.



Build Bronze Layer

Develop SQL Load Scripts



In a **bulk insert**, all the data from the files is loaded into the database at once, making the process faster and more efficient. In contrast, a standard **INSERT** statement inserts data **row by row**, which can be significantly slower, especially for large datasets.

```
USE DataWarehouse;

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

HINT

Save frequently used SQL code in stored procedures in database

Add PRINTS

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

Add TRY...CATCH

Ensures error handling, data integrity, and issue logging for easier debugging

TRY...CATCH

SQL runs the TRY block, and if it fails, it runs the CATCH block to handle the error

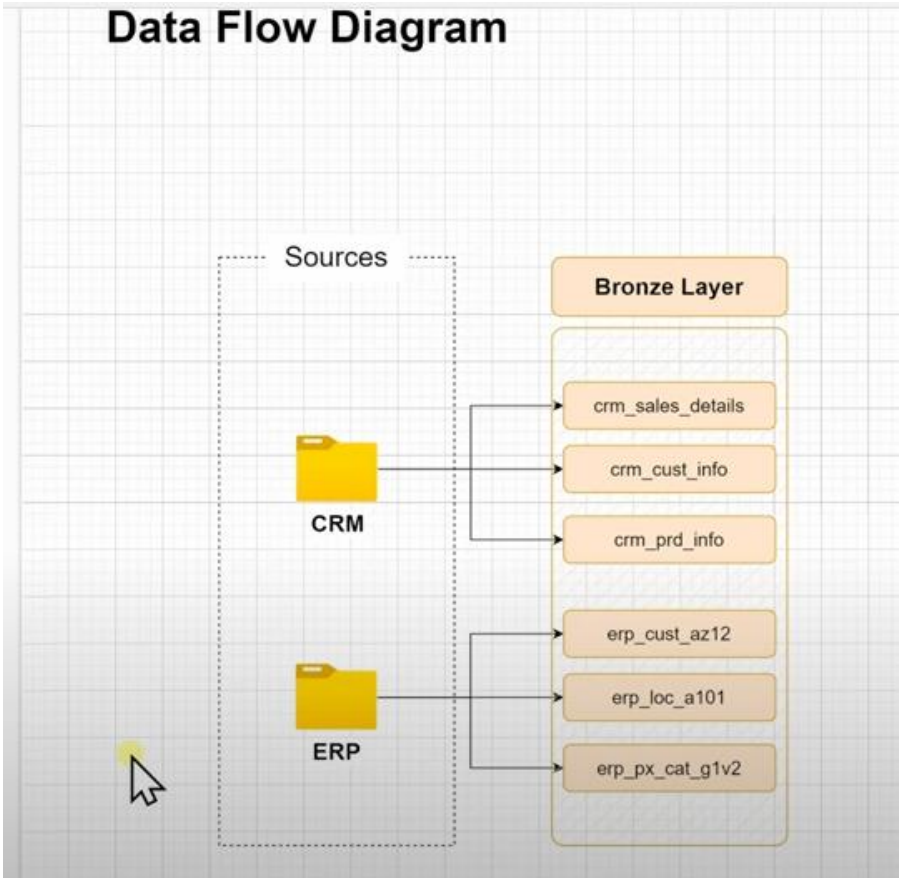
Track ETL Duration

Helps to identify bottlenecks, optimize performance, monitor trends, detect issues

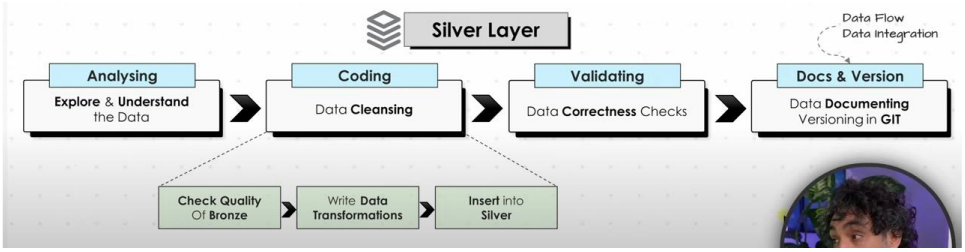
DATEDIFF()

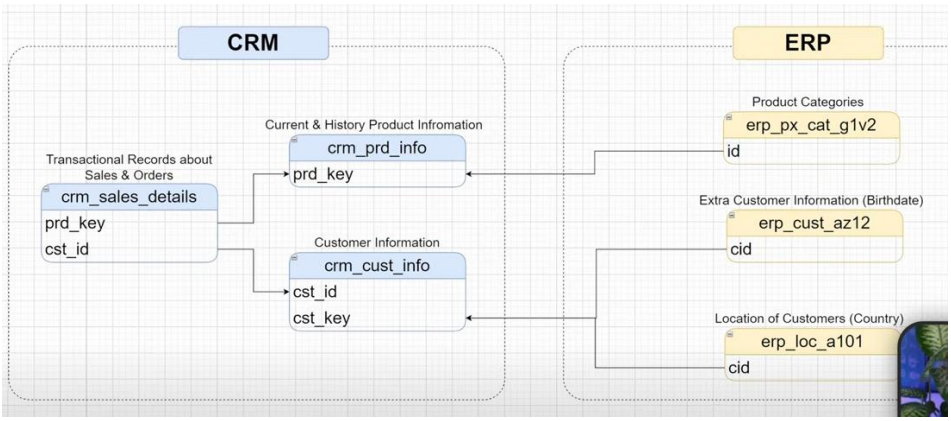
calculates the difference between two dates, returns days, months, or years

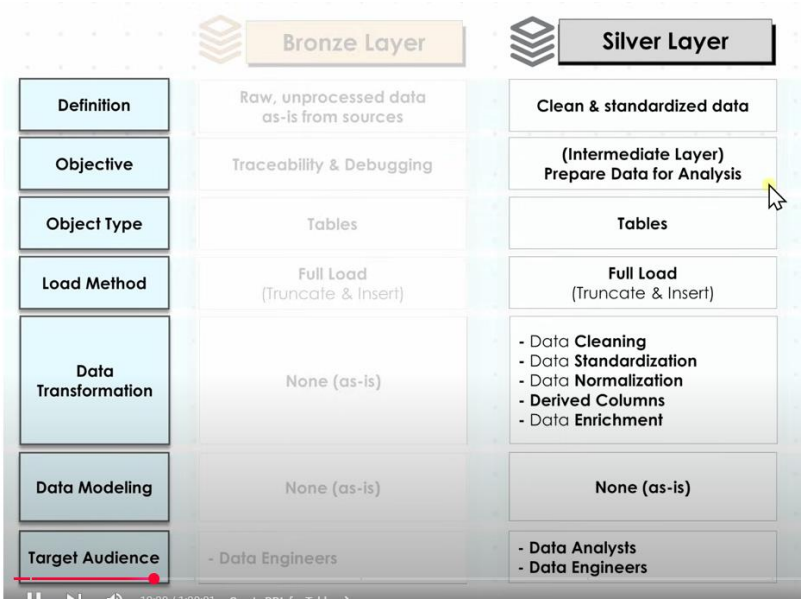
Calculate the Duration of Loading Bronze Layer "Whole Batch"

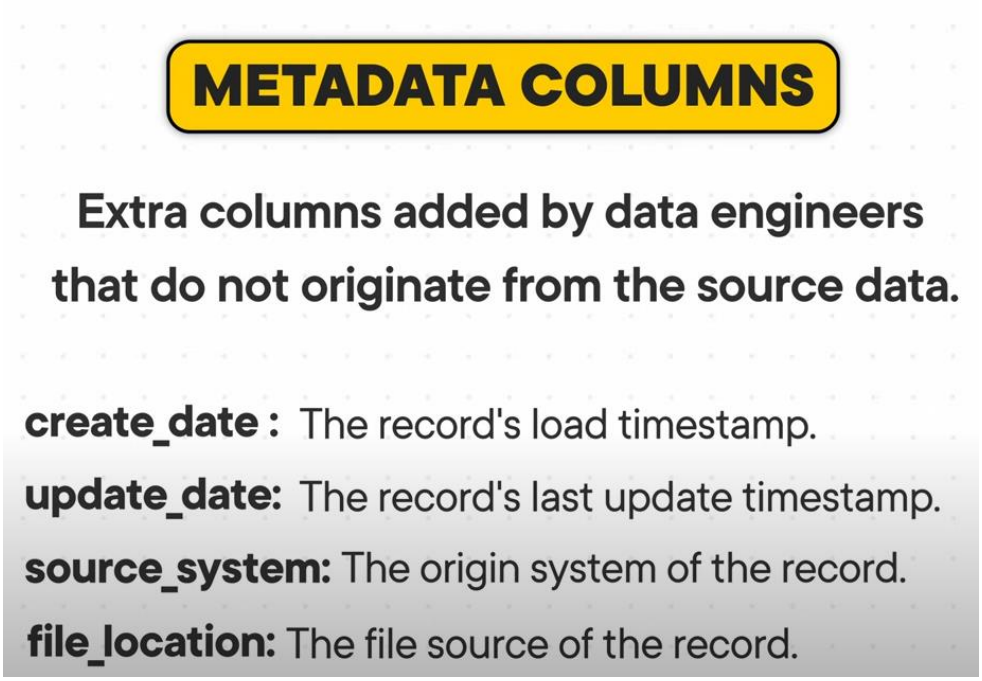


Silver Layer









Initially, we create silver tables with the same structure as bronze tables. After reviewing the data across different tables, we determine whether additional metadata columns are needed and incorporate them accordingly. Similarly, in our project, we follow this approach by first replicating the bronze tables in Silver and later enhancing them as required.

Table : silver.load_silver

Results | Messages

	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

We have duplicates in the **primary key** and **NULL values** in the cst_id column. Upon closer inspection, the duplicates have different **cst_create_date** values. To clean the data:

1. **Remove records where cst_id is NULL**
2. **For duplicates, retain only the latest record** based on cst_create_date

quality_checks_sil...0PBM4NJ\rehan (52))* | SQLQuery5.sql - L...PBM4NJ\rehan (71))* | DWH_Proj

```sql
-----silver.crm_cust_info-----
-- Check for Nulls or Duplicates in Primary Key
-- Expectation: No Result

SELECT
cst_id,
COUNT(*)
FROM bronze.crm_cust_info
GROUP BY cst_id
HAVING COUNT(*)>1 or cst_id IS NULL;

SELECT * FROM bronze.crm_cust_info WHERE cst_id = 29466;
```

100 %

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

	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

```sql

--query to remove the duplicates and Nulls in primary key
SELECT
*
FROM
    (SELECT
    *,
    ROW_NUMBER() OVER(PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_latest
    FROM
    bronze.crm_cust_info
    WHERE cst_id IS NOT NULL) t
WHERE flag_latest = 1
```

Now, we need to check for the string columns (cst_firstname,cst_lastname)

```sql
-- Check for unwanted Spaces
SELECT cst_firstname
FROM bronze.crm_cust_info
WHERE cst_firstname != TRIM(cst_firstname)
```

TRIM()

Removes leading and trailing spaces from a string

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

```sql
--query to remove the unwanted spaces from the string columns

SELECT
cst_id,
cst_key,
TRIM(cst_firstname) AS cst_firstname,
TRIM(cst_lastname) AS cst_lastname,
cst_marital_status,
cst_gndr,
cst_create_date
FROM
bronze.crm_cust_info
```

Quality Check

Check the consistency of values in low cardinality columns

Results Messages

	cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date
97	11096	AW00011096	Andr┼®s	Anand	M	M	2025-10-07
98	11097	AW00011097	Edwin	Nara	M	M	2025-10-07
99	11098	AW00011098	Mallory	Rubio	S	F	2025-10-07
100	11099	AW00011099	Adam	Ross	M	M	2025-10-07
101	11100	AW00011100	Latasha	Navarro	S	F	2025-10-07
102	11101	AW00011101	Abby	Sai	S	F	2025-10-07
103	11102	AW00011102	Julia	Nelson	S	F	2025-10-07
104	11103	AW00011103	Cassie	Chande	S	F	2025-10-07
105	11104	AW00011104	Edgar	Sara	M	M	2025-10-07
106	11105	AW00011105	Candace	Fernandez	S	F	2025-10-07
107	11106	AW00011106	Jessie	Liu	S	M	2025-10-07

24:23 / 1:38:01 · Clean & Load crm_cust_info

```
52  -- Data standardization and Consistency
53  SELECT DISTINCT
54  cst_marital_status,cst_gndr
55  FROM
56  bronze.crm_cust_info
```

100 %

Results | Messages

	cst_marital_status	cst_gndr
1	M	M
2	S	M
3	S	F
4	S	NULL
5	M	F
6	M	NULL
7	NULL	NULL

In our project, we need to use **abbreviations** for values instead of single letters. Additionally, we must determine the appropriate way to handle **NULL values**, which will be finalized after discussions with the **source system manager**.

```
--query to standardize the low cardinality columns

SELECT
cst_id,
cst_key,
TRIM(cst_firstname) AS cst_firstname,
TRIM(cst_lastname) AS cst_lastname,
CASE
    WHEN UPPER(cst_marital_status) = 'M' THEN 'Married'
    WHEN UPPER(cst_marital_status) = 'S' THEN 'Single'
    ELSE 'n/a'
END AS cst_marital_status,
CASE
    WHEN UPPER(cst_gndr) = 'F' THEN 'Female'
    WHEN UPPER(cst_gndr) = 'M' THEN  'Male'
    ELSE 'n/a'
END AS cst_gndr
FROM
(SELECT
    *,
    ROW_NUMBER() OVER(PARTITION BY cst_id ORDER BY cst_create_date DESC) AS flag_latest
    FROM
    bronze.crm_cust_info
    WHERE cst_id IS NOT NULL) t
WHERE flag_latest = 1
```

From the above code, we've completed cleaning the data for the table **bronze.crm_cust_info** and we can store it into **silver.load_silver**

Table : silver.crm_prd_info

```
6  SELECT TOP(20) * FROM [bronze].[crm_prd_info];
```

100 %

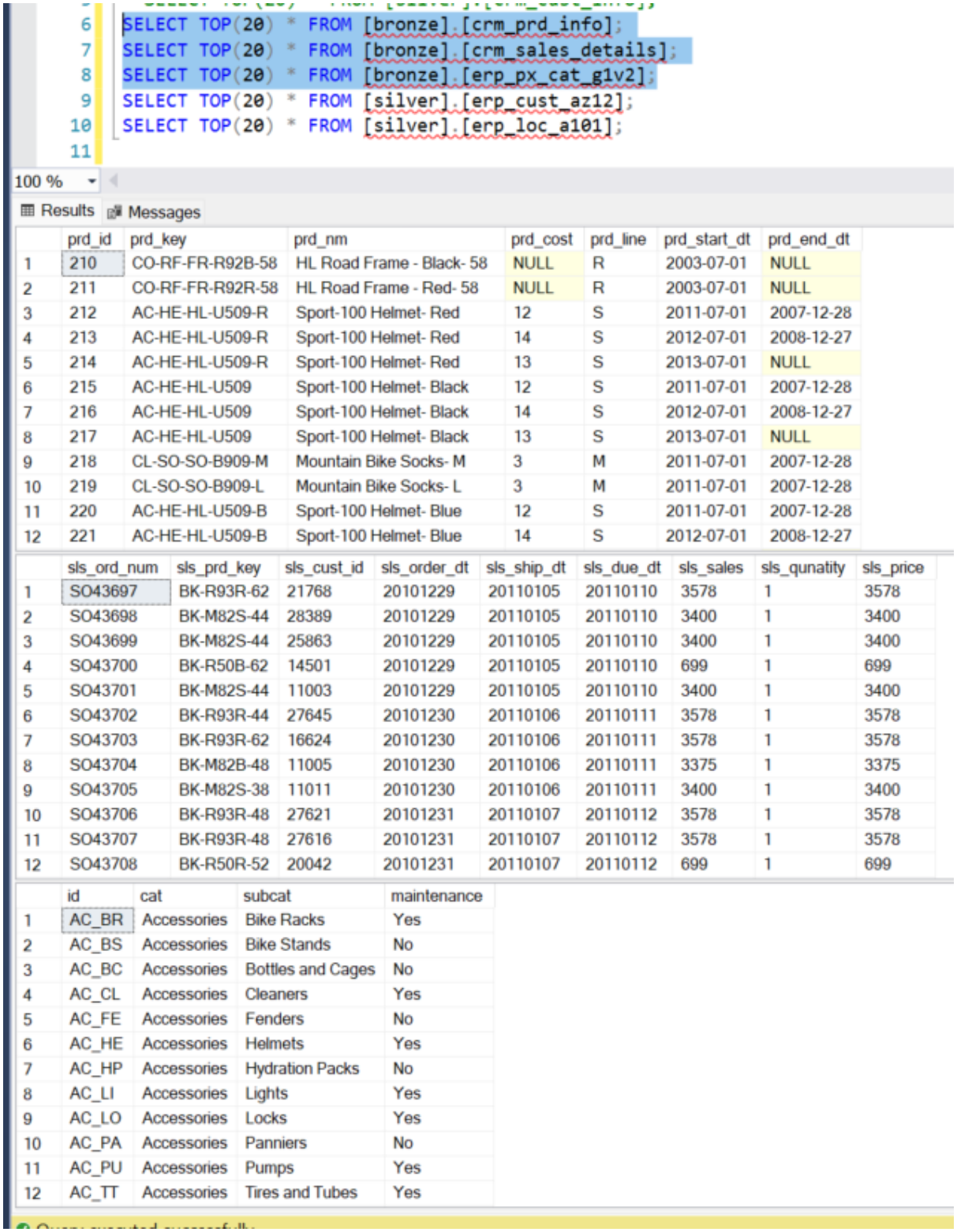
Results | Messages

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01	NULL
2	211	CO-RF-FR-R92R-58	HL Road Frame - Red- 58	NULL	R	2003-07-01	NULL
3	212	AC-HE-HL-U509-R	Sport-100 Helmet- Red	12	S	2011-07-01	2007-12-28
4	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01	2008-12-27
5	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01	NULL
6	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01	2007-12-28
7	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01	2008-12-27
8	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01	NULL
9	218	CL-SO-SO-B909-M	Mountain Bike Socks- M	3	M	2011-07-01	2007-12-28
10	219	CL-SO-SO-B909-L	Mountain Bike Socks- L	3	M	2011-07-01	2007-12-28
11	220	AC-HE-HL-U509-B	Sport-100 Helmet- Blue	12	S	2011-07-01	2007-12-28
12	221	AC-HE-HL-U509-B	Sport-100 Helmet- Blue	14	S	2012-07-01	2008-12-27
13	222	AC-HE-HL-U509-B	Sport-100 Helmet- Blue	13	S	2013-07-01	NULL
14	223	CL-CA-CA-1098	AWC Logo Cap	6	S	2011-07-01	2007-12-28
15	224	CL-CA-CA-1098	AWC Logo Cap	5	S	2012-07-01	2008-12-27
16	225	CL-CA-CA-1098	AWC Logo Cap	7	S	2013-07-01	NULL
17	226	CL-JE-LJ-0192-S	Long-Sleeve Logo Jersey- S	32	S	2011-07-01	2007-12-28
18	227	CL-JE-LJ-0192-S	Long-Sleeve Logo Jersey- S	29	S	2012-07-01	2008-12-27
19	228	CL-JE-LJ-0192-S	Long-Sleeve Logo Jersey- S	38	S	2013-07-01	NULL
20	229	CL-JE-LJ-0192-M	Long-Sleeve Logo Jersey- M	32	S	2011-07-01	2007-12-28

prd_id: It’s primary key and we don’t have any nulls or duplicates values in it.

```
95  -- Check for Nulls or Duplicates in Primary Key
96  -- Expectation: No Result
97
98
99  SELECT
100 prd_id,
101 COUNT(*)
102 FROM
103 bronze.crm_prd_info
104 GROUP BY prd_id
105 HAVING COUNT(*)>1 OR prd_id IS NULL
106
```

prd_key: The prd_key column contains metadata from another table, specifically **sales_details (sls_prd_key)** and **px_cat_g1v2 (id)**. We need to split this column into two separate columns to correctly store and manage the respective data.



```
SELECT TOP(20) * FROM [bronze].[crm_prd_info];
SELECT TOP(20) * FROM [bronze].[crm_sales_details];
SELECT TOP(20) * FROM [bronze].[erp_px_cat_g1v2];
SELECT TOP(20) * FROM [silver].[erp_cust_az12];
SELECT TOP(20) * FROM [silver].[erp_loc_a101];
```

100 %

Results Messages

	prd_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt
1	210	CO-RF-FR-R92B-58	HL Road Frame - Black- 58	NULL	R	2003-07-01	NULL
2	211	CO-RF-FR-R92R-58	HL Road Frame - Red- 58	NULL	R	2003-07-01	NULL
3	212	AC-HE-HL-U509-R	Sport-100 Helmet- Red	12	S	2011-07-01	2007-12-28
4	213	AC-HE-HL-U509-R	Sport-100 Helmet- Red	14	S	2012-07-01	2008-12-27
5	214	AC-HE-HL-U509-R	Sport-100 Helmet- Red	13	S	2013-07-01	NULL
6	215	AC-HE-HL-U509	Sport-100 Helmet- Black	12	S	2011-07-01	2007-12-28
7	216	AC-HE-HL-U509	Sport-100 Helmet- Black	14	S	2012-07-01	2008-12-27
8	217	AC-HE-HL-U509	Sport-100 Helmet- Black	13	S	2013-07-01	NULL
9	218	CL-SO-SO-B909-M	Mountain Bike Socks- M	3	M	2011-07-01	2007-12-28
10	219	CL-SO-SO-B909-L	Mountain Bike Socks- L	3	M	2011-07-01	2007-12-28
11	220	AC-HE-HL-U509-B	Sport-100 Helmet- Blue	12	S	2011-07-01	2007-12-28
12	221	AC-HE-HL-U509-B	Sport-100 Helmet- Blue	14	S	2012-07-01	2008-12-27

	sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_qunatity	sls_price
1	SO43697	BK-R93R-62	21768	20101229	20110105	20110110	3578	1	3578
2	SO43698	BK-M82S-44	28389	20101229	20110105	20110110	3400	1	3400
3	SO43699	BK-M82S-44	25863	20101229	20110105	20110110	3400	1	3400
4	SO43700	BK-R50B-62	14501	20101229	20110105	20110110	699	1	699
5	SO43701	BK-M82S-44	11003	20101229	20110105	20110110	3400	1	3400
6	SO43702	BK-R93R-44	27645	20101230	20110106	20110111	3578	1	3578
7	SO43703	BK-R93R-62	16624	20101230	20110106	20110111	3578	1	3578
8	SO43704	BK-M82B-48	11005	20101230	20110106	20110111	3375	1	3375
9	SO43705	BK-M82S-38	11011	20101230	20110106	20110111	3400	1	3400
10	SO43706	BK-R93R-48	27621	20101231	20110107	20110112	3578	1	3578
11	SO43707	BK-R93R-48	27616	20101231	20110107	20110112	3578	1	3578
12	SO43708	BK-R50R-52	20042	20101231	20110107	20110112	699	1	699

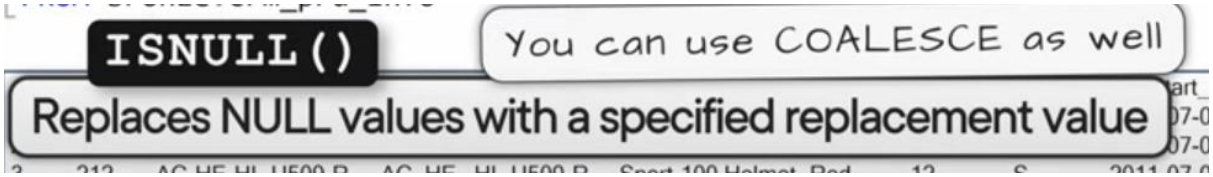
	id	cat	subcat	maintenance
1	AC_BR	Accessories	Bike Racks	Yes
2	AC_BS	Accessories	Bike Stands	No
3	AC_BC	Accessories	Bottles and Cages	No
4	AC_CL	Accessories	Cleaners	Yes
5	AC_FE	Accessories	Fenders	No
6	AC_HE	Accessories	Helmets	Yes
7	AC_HP	Accessories	Hydration Packs	No
8	AC_LI	Accessories	Lights	Yes
9	AC_LO	Accessories	Locks	Yes
10	AC_PA	Accessories	Panniers	No
11	AC_PU	Accessories	Pumps	Yes
12	AC_TT	Accessories	Tires and Tubes	Yes

prd_nm: It’s a string column, so we’ve checked for unwanted spaces and we’ve found nothing.

```sql
-- Check for Nulls
-- Expectation: No Result

SELECT
prd_key
FROM bronze.crm_prd_info
WHERE prd_key IS NULL

-- splitting prd_key into two columns

SELECT
prd_id,
SUBSTRING(prd_key,7,LEN(prd_key)) AS prd_key,
REPLACE(SUBSTRING(prd_key,1,5) ,'-','_') AS cat_id,
prd_nm,
prd_cost,
prd_line,
prd_start_dt,
prd_end_dt
FROM bronze.crm_prd_info

-- Check for Unwanted Spaces for String Columns
-- Expectation: No Result

SELECT
prd_key
FROM  bronze.crm_prd_info
WHERE prd_key != TRIM(prd_key)

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

```

prd_cost: It's a numeric column, so we need to check for negative values and nulls. We've found few null values here. We need to handle them after discussing with the team. Here, in our case we are making them zero.

```sql
-- Check for Nulls or Negative Numbers
-- Expectation: No Result

SELECT
prd_cost
FROM bronze.crm_prd_info
WHERE prd_cost IS NULL OR prd_cost < 0
```



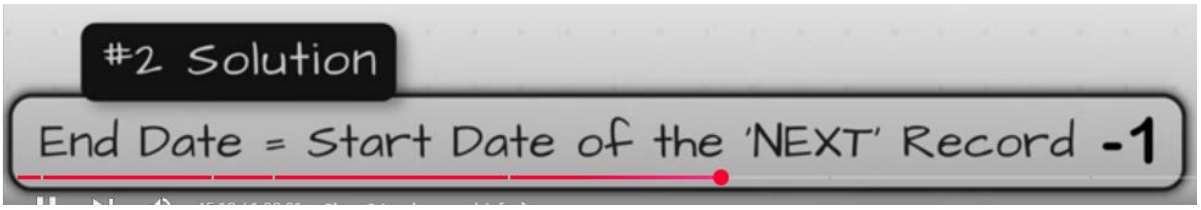
```sql
--Replacing Nulls with 0

SELECT
prd_id,
SUBSTRING(prd_key,7,LEN(prd_key)) AS prd_key,
REPLACE(SUBSTRING(prd_key,1,5) ,'-','_') AS cat_id,
prd_nm,
ISNULL(prd_cost,0) AS prd_cost,
prd_line,
prd_start_dt,
prd_end_dt
FROM bronze.crm_prd_info
```

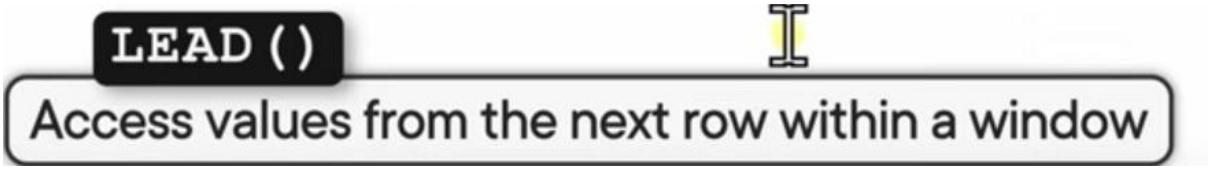
prd_line: It is low cardinality column. So, we need to fill with the abbreviation instead of single letter.

```sql
-- Data standardization and Consistency
SELECT DISTINCT
prd_line
FROM
bronze.crm_prd_info

--Replacing with Abbrevations
SELECT
prd_id,
SUBSTRING(prd_key,7,LEN(prd_key)) AS prd_key,
REPLACE(SUBSTRING(prd_key,1,5) ,'-','_') AS cat_id,
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,
CASE UPPER(TRIM(prd_line))
    WHEN 'M' THEN 'Mountain'
    WHEN 'R' THEN 'Road'
    WHEN 'S' THEN 'Other Sales'
    WHEN 'T' THEN 'Touring'
    ELSE 'n/a'
END AS prd_line,
prd_start_dt,
prd_end_dt
FROM bronze.crm_prd_info

```

prd_start_dt and prd_end_dt: We need to validate the **prd_end_dt** column to ensure that the **end dates are not earlier than the corresponding prd_start_dt**. Upon review, we found some records with this issue. To correct them, we will **update the prd_end_dt** by setting it to **one day before the prd_start_dt of the same product**. Before using it, we need to discuss with the team.

```sql
-- Check for Invalid Date Orders
-- End date must not be earlier than start date
SELECT
*
FROM bronze.crm_prd_info
WHERE prd_end_dt < prd_start_dt


-- check for nulls in prd_start_dt

SELECT
*
FROM bronze.crm_prd_info
WHERE prd_start_dt IS NULL
```






```sql
---Replacing End date with day before of start date for each prd_key

SELECT
    prd_id,
    REPLACE(SUBSTRING(prd_key,1,5), ',','-') AS cat_id,
    SUBSTRING(prd_key,7,LEN(prd_key)) AS prd_key,
    prd_nm,
    ISNULL(prd_cost,0) AS prd_cost,
    CASE UPPER(TRIM(prd_line))
        WHEN 'M' THEN 'Mountain'
        WHEN 'R' THEN 'Road'
        WHEN 'S' THEN 'Other Sales'
        WHEN 'T' THEN 'Touring'
        ELSE 'n/a'
    END AS prd_line,
    prd_start_dt,
    DATEADD(DAY, -1, LEAD(prd_start_dt) OVER (PARTITION BY prd_key ORDER BY prd_start_dt)) AS prd_end_dt
FROM bronze.crm_prd_info;
```

Now, we need to add cat_id column to silver.crm_prd_info as we've derived this column from bronze.crm.prd_info.

Table : 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_qunatity	sls_price
10	SO43706	BK-R93R-48	27621	20101231	20110107	20110112	3578	1	3578
11	SO43707	BK-R93R-48	27616	20101231	20110107	20110112	3578	1	3578
12	SO43708	BK-R50R-52	20042	20101231	20110107	20110112	699	1	699
13	SO43709	BK-R93R-52	16351	20101231	20110107	20110112	3578	1	3578
14	SO43710	BK-R93R-56	16517	20101231	20110107	20110112	3578	1	3578
15	SO43711	BK-R93R-56	27606	20110101	20110108	20110113	3578	1	3578
16	SO43712	BK-R93R-44	13513	20110101	20110108	20110113	3578	1	3578
17	SO43713	BK-R93R-62	27601	20110102	20110109	20110114	3578	1	3578

This table does not have a primary key, and some of the first three columns contain no null values. The next three columns are of type **INT** but represent **dates**, so we need to convert them into the **DATE** format.

```
247  -- checking for NULL
248  SELECT
249  *
250  FROM bronze.crm_sales_details
251  WHERE sls_ord_num IS NULL
252
253  SELECT
254  *
255  FROM bronze.crm_sales_details
256  WHERE sls_prd_key IS NULL
257
258  SELECT
259  *
260  FROM bronze.crm_sales_details
261  WHERE sls_cust_id IS NULL
262
```

Negative numbers or zeros can't be cast to a date

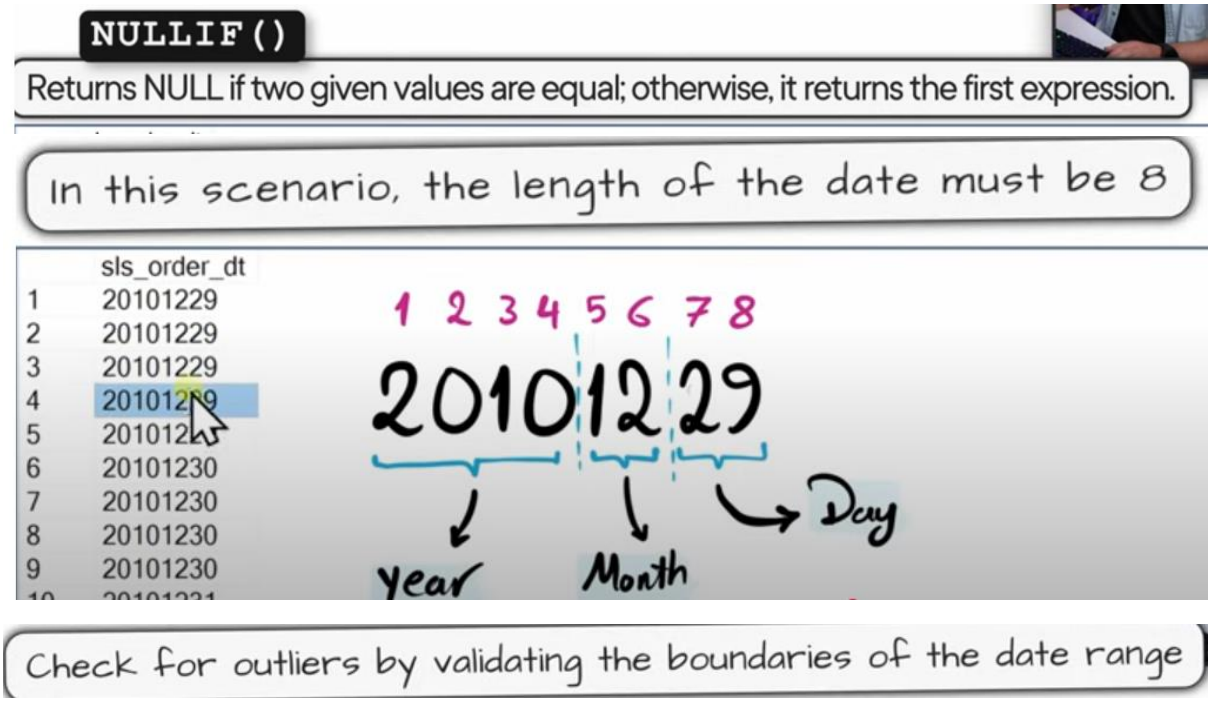
```
263  -- check for Invalid Dates
264
265  SELECT
266  sls_order_dt
267  FROM
268  bronze.crm_sales_details
269  WHERE sls_order_dt <= 0
270
271
```

100 %

Results | Messages

	sls_order_dt
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

We've found some zero values and we need to replace them with NULL



Check for outliers by validating the boundaries of the date range

We can't convert INT to DATE directly. So first we need to cast it to VARCHAR then DATE.

```
--Replacing with NULls when dt <=0 or len(dt) != 8

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,
sls_sales,
sls_qunatity,
sls_price
FROM bronze.crm_sales_details
```

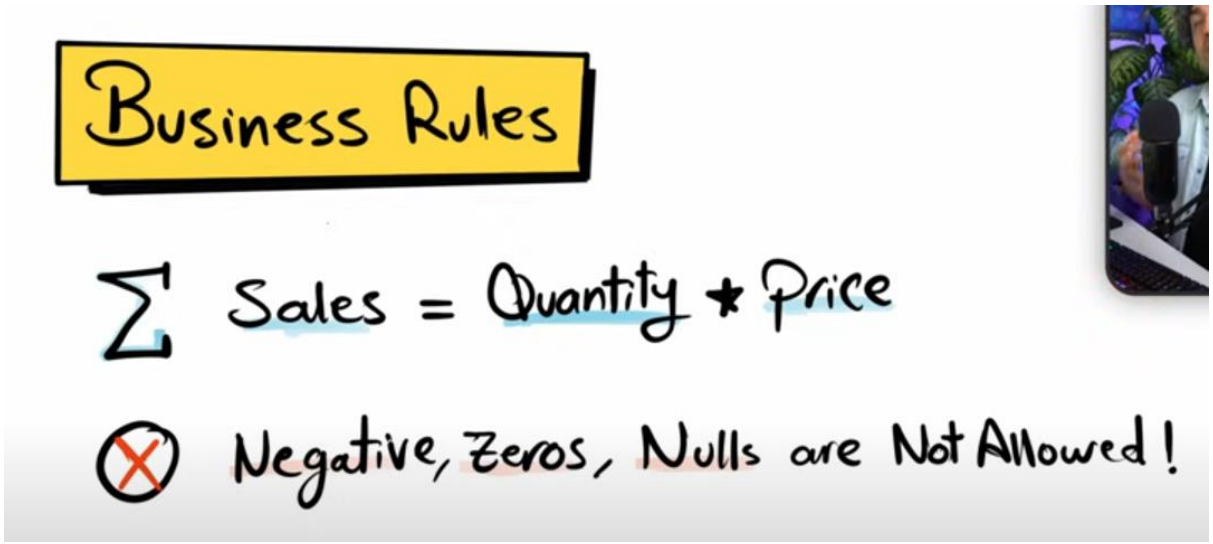

```sql
---Invalid order dates

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

100 %

Results | Messages

sls_ord_num	sls_prd_key	sls_cust_id	sls_order_dt	sls_ship_dt	sls_due_dt	sls_sales	sls_qunatity	sls_price

We don’t have any invalid order dates.

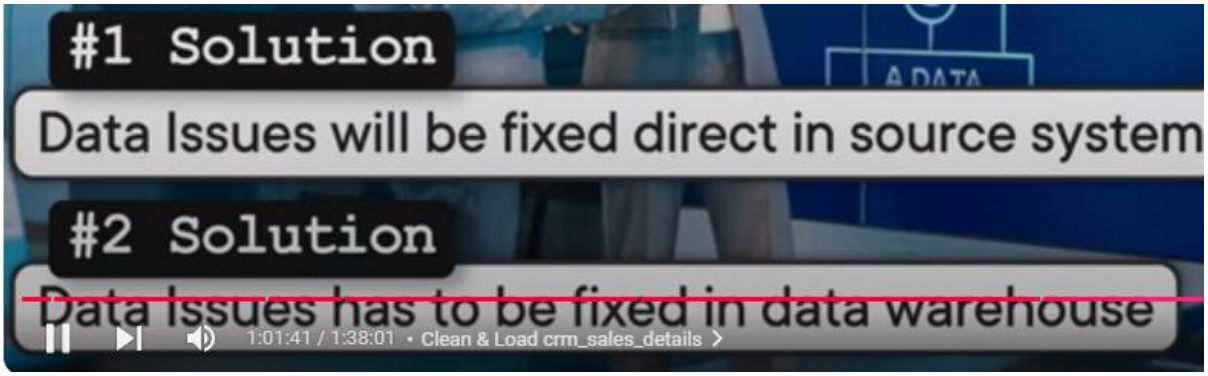


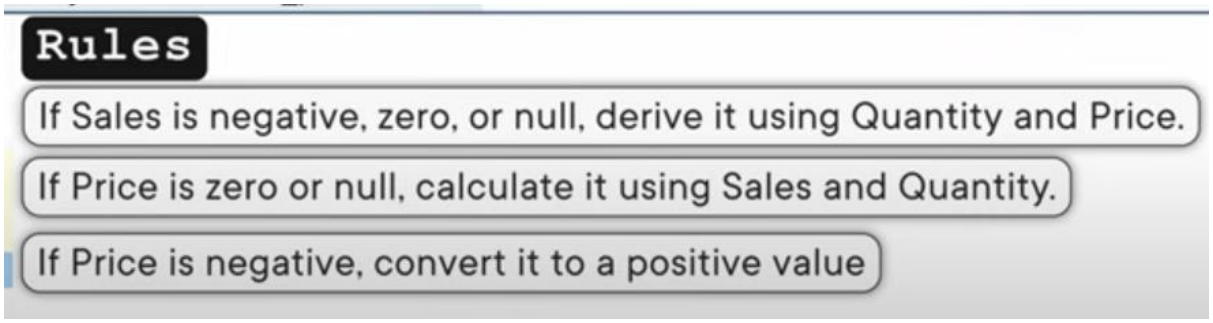
```sql
-- Data consistency with sales, quantity and price columns
-- sales = qunatity * price
-- Values must not be zero, negative or null


SELECT
sls_sales,
sls_qunatity,
sls_price
FROM bronze.crm_sales_details
WHERE sls_sales != sls_qunatity * sls_price
OR sls_sales IS NULL OR sls_qunatity IS NULL OR sls_price IS NULL
OR sls_sales <= 0 OR sls_qunatity <= 0 OR sls_price <= 0
```

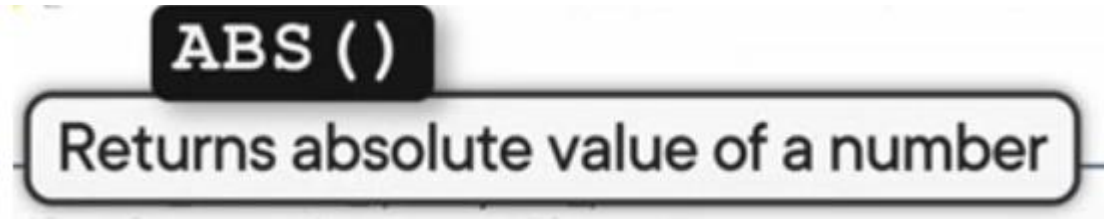
100 %

Results | Messages

	sls_sales	sls_qunatity	sls_price
1	10	2	NULL
2	25	5	NULL
3	70	2	NULL
4	9	1	NULL
5	35	1	NULL
6	100	10	NULL
7	16	2	NULL
8	769	1	-769
9	30	1	-30
10	22	1	-22
11	35	2	35
12	1701	1	-1701
13	21	1	-21
14	50	2	50
15	NULL	1	9
16	NULL	1	35
17	NULL	1	8
18	NULL	1	22
19	-18	1	9
20	0	1	10
21	50	2	50
22	NULL	1	10
23	5	2	5
24	64	4	64
25	25	3	25
26	40	1	2
27	2	1	50
28	33	1	24


#1 Solution
Data Issues will be fixed direct in source system
#2 Solution
Data Issues has to be fixed in data warehouse
1:01:41 / 1:38:01 • Clean & Load crm_sales_details


Rules
If Sales is negative, zero, or null, derive it using Quantity and Price.
If Price is zero or null, calculate it using Sales and Quantity.
If Price is negative, convert it to a positive value


ABS ()
Returns absolute value of a number

```sql
----fix
SELECT
sls_sales AS old_sales,
sls_qunatity,
sls_price AS old_prices,
CASE
    WHEN sls_sales IS NULL
        OR sls_sales <= 0
        OR sls_sales != sls_qunatity * ABS(sls_price)
        THEN sls_qunatity * ABS(sls_price)
    ELSE sls_sales
END AS sls_sales,
CASE
    WHEN sls_price IS NULL OR sls_price = 0
        THEN sls_sales/NULLIF(sls_qunatity,0)
    WHEN sls_price < 0 THEN ABS(sls_price)
    ELSE sls_price
END AS sls_price
FROM bronze.crm_sales_details
WHERE sls_sales != sls_qunatity * sls_price
OR sls_sales IS NULL OR sls_qunatity IS NULL OR sls_price IS NULL
OR sls_sales <= 0 OR sls_qunatity <= 0 OR sls_price <= 0

```

100 %

Results Messages

	old_sales	sls_qunatity	old_prices	sls_sales	sls_price
8	769	1	-769	769	769
9	30	1	-30	30	30
10	22	1	-22	22	22
11	35	2	35	70	35
12	1701	1	-1701	1701	1701
13	21	1	-21	21	21
14	50	2	50	100	50
15	NULL	1	9	9	9
16	NULL	1	35	35	35
17	NULL	1	8	8	8
18	NULL	1	22	22	22
19	-18	1	9	9	9
20	0	1	10	10	10
21	50	2	50	100	50
22	NULL	1	10	10	10
23	5	2	5	10	5
24	64	4	64	256	64
25	25	3	25	75	25

```sql
-----Corrected Query for the whole table

SELECT
sls_ord_num,
sls_prd_key,
sls_cust_id,
CASE
    WHEN sls_order_dt <= 0 OR LEN(sls_order_dt) != 8 THEN NUL
    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_qunatity * ABS(sls_price)
        THEN sls_qunatity * ABS(sls_price)
    ELSE sls_sales
END AS sls_sales,
sls_qunatity,
CASE
    WHEN sls_price IS NULL OR sls_price = 0
        THEN sls_sales/NULLIF(sls_qunatity,0)
    WHEN sls_price < 0 THEN ABS(sls_price)
    ELSE sls_price
END AS sls_price
FROM bronze.crm_sales_details
```

Table : silver.erp_cust_az12

100 %

Results | Messages

	cid	bdate	gen
1	NASAW00011000	1971-10-06	Male
2	NASAW00011001	1976-05-10	Male
3	NASAW00011002	1971-02-09	Male
4	NASAW00011003	1973-08-14	Female
5	NASAW00011004	1979-08-05	Female
6	NASAW00011005	1976-08-01	Male
7	NASAW00011006	1976-12-02	Female
8	NASAW00011007	1969-11-06	Male
9	NASAW00011008	1975-07-04	Female
10	NASAW00011009	1969-09-29	Male
11	NASAW00011010	1969-08-05	Female
12	NASAW00011011	1969-05-03	Male
13	NASAW00011012	1979-01-14	Female
14	NASAW00011013	1979-08-03	Male
15	NASAW00011014	1973-11-06	Female
16	NASAW00011015	1984-08-26	Female
17	NASAW00011016	1984-10-25	Male
18	NASAW00011017	1949-12-24	Female
19	NASAW00011018	1955-10-06	Male
20	NASAW00011019	1983-09-04	Male

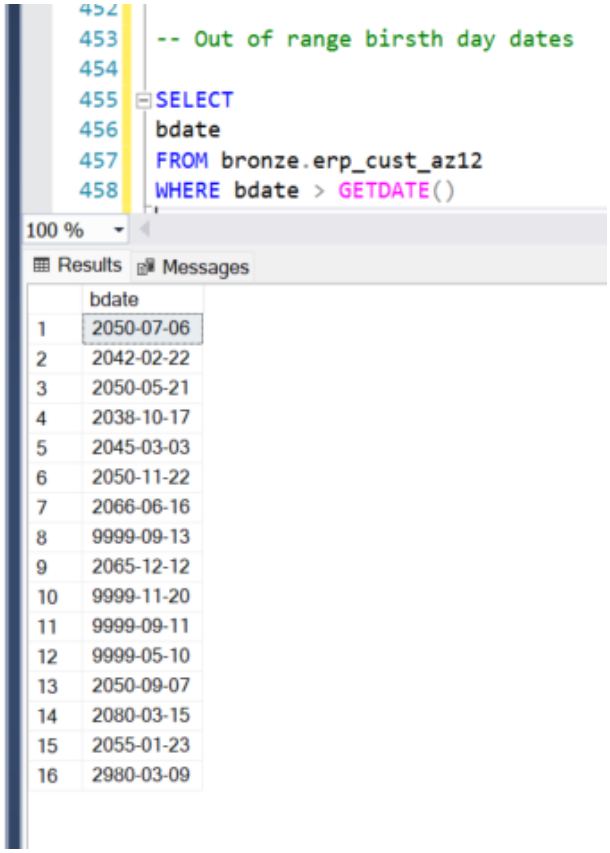
We have a **cid** column like the **cst_key** in **bronze.crm_cust_info**, but in the current table, it includes a **"NAS"** prefix. We need to remove this prefix using the **SUBSTRING** function.

100 %

Results | Messages

	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

We've found few birth days those are ahead of current time, and we can replace this values with Nulls.

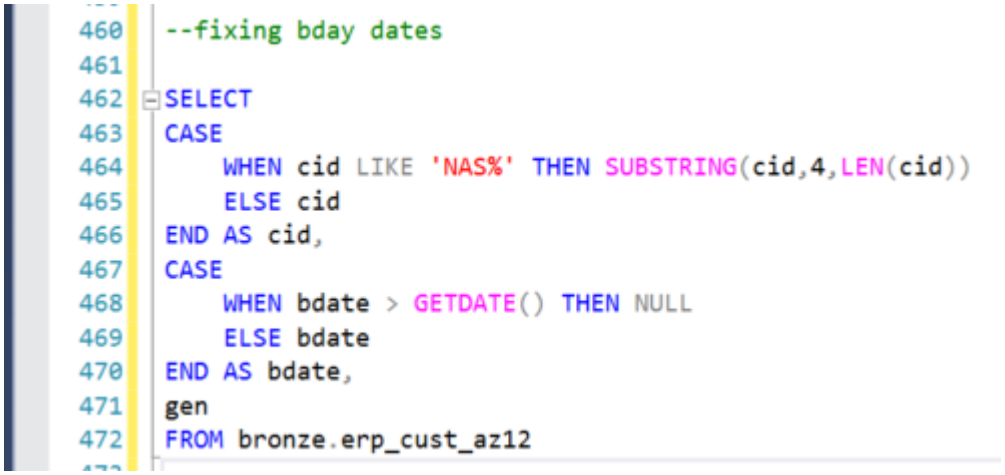


```
453  -- Out of range birsth day dates
454
455  SELECT
456  bdate
457  FROM bronze.erp_cust_az12
458  WHERE bdate > GETDATE()
```

100 %

Results | Messages

	bdate
1	2050-07-06
2	2042-02-22
3	2050-05-21
4	2038-10-17
5	2045-03-03
6	2050-11-22
7	2066-06-16
8	9999-09-13
9	2065-12-12
10	9999-11-20
11	9999-09-11
12	9999-05-10
13	2050-09-07
14	2080-03-15
15	2055-01-23
16	2980-03-09



```
460  --fixing bday dates
461
462  SELECT
463  CASE
464      WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid,4,LEN(cid))
465      ELSE cid
466  END AS cid,
467  CASE
468      WHEN bdate > GETDATE() THEN NULL
469      ELSE bdate
470  END AS bdate,
471  gen
472  FROM bronze.erp_cust_az12
```

```sql
----Data Standardization and consistency

SELECT
DISTINCT gen
FROM  bronze.erp_cust_az12
```

100 %

Results | Messages

	gen
1	NULL
2	F
3	
4	Male
5	Female
6	M

In gender column, we've found in consistencies in the data, so we need to standardize it.

```sql
--fixing standardization and whole query

SELECT
CASE
    WHEN cid LIKE 'NAS%' THEN SUBSTRING(cid,4,LEN(cid))
    ELSE cid
END AS cid,
CASE
    WHEN TRY_CAST(bdate AS DATE) > GETDATE() THEN NULL -- WE don't get bdate always as DATE
    ELSE bdate
END AS bdate,
CASE
    WHEN UPPER(TRIM(gen)) IN ('M','Male') THEN 'Male'
    WHEN UPPER(TRIM(gen)) IN ('F','Female') THEN 'Female'
    ELSE 'n/a'
END gen
FROM bronze.erp_cust_az12
```

Table : silver.erp_loc_a101

```sql
SELECT TOP(20) * FROM [bronze].[erp_loc_a101];
SELECT TOP(20) * FROM [bronze].[erp_px_cat_g1v2];
```

100 %

Results | Messages

	cid	cntry
1	AW-00011000	Australia
2	AW-00011001	Australia
3	AW-00011002	Australia
4	AW-00011003	Australia
5	AW-00011004	Australia
6	AW-00011005	Australia
7	AW-00011006	Australia
8	AW-00011007	Australia
9	AW-00011008	Australia
10	AW-00011009	Australia
11	AW-00011010	Australia
12	AW-00011011	Australia
13	AW-00011012	US
14	AW-00011013	US
15	AW-00011014	US
16	AW-00011015	US
17	AW-00011016	US
18	AW-00011017	Australia
19	AW-00011018	Australia
20	AW-00011019	Canada

```sql
SELECT TOP(5) * FROM [bronze].[crm_cust_info];
SELECT TOP(5) * FROM [bronze].[erp_loc_a101];
```

100 %

Results | Messages

	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

	cid	cntry
1	AW-00011000	Australia
2	AW-00011001	Australia
3	AW-00011002	Australia
4	AW-00011003	Australia
5	AW-00011004	Australia

In the **cid** column of the **bronze.erp_loc_a101** table, we need to remove the hyphen (-) to ensure it matches the **cst_key** column in the **bronze.crm_cust_info** table.

We don’t have any NULL values for cid column

```sql
-- checking for NULL or Duplicates

SELECT
cid,
--cntry
COUNT(*)
FROM
bronze.erp_loc_a101
GROUP BY cid
HAVING COUNT(*) > 1 OR cid IS NULL
```

We’ve found in consistencies in the cntry column

```sql
--Data Standardization
SELECT
DISTINCT cntry
FROM
bronze.erp_loc_a101
```

100 %

Results | Messages

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

```sql
-- fixing standardization and query for whole table

SELECT
REPLACE(cid,'-','') AS cid,
CASE
    WHEN UPPER(TRIM(cntry)) = 'DE' THEN 'Germany'
    WHEN UPPER(TRIM(cntry)) IN ('US','USA') THEN 'United States'
    WHEN cntry IS NULL OR cntry = '' THEN 'n/a'
    ELSE TRIM(cntry)
END AS cntry
FROM
bronze.erp_loc_a101
```

Table : silver.erp_px_cat_g1v2

```sql
SELECT TOP(20) * FROM [bronze].[erp_px_cat_g1v2];
```

100 %

Results | Messages

	id	cat	subcat	maintenance
1	AC_BR	Accessories	Bike Racks	Yes
2	AC_BS	Accessories	Bike Stands	No
3	AC_BC	Accessories	Bottles and Cages	No
4	AC_CL	Accessories	Cleaners	Yes
5	AC_FE	Accessories	Fenders	No
6	AC_HE	Accessories	Helmets	Yes
7	AC_HP	Accessories	Hydration Packs	No
8	AC_LI	Accessories	Lights	Yes
9	AC_LO	Accessories	Locks	Yes
10	AC_PA	Accessories	Panniers	No
11	AC_PU	Accessories	Pumps	Yes
12	AC_TT	Accessories	Tires and Tubes	Yes
13	BI_MB	Bikes	Mountain Bikes	Yes
14	BI_RB	Bikes	Road Bikes	Yes
15	BI_TB	Bikes	Touring Bikes	Yes
16	CL_BS	Clothing	Bib-Shorts	No
17	CL_CA	Clothing	Caps	No
18	CL_GL	Clothing	Gloves	No
19	CL_JE	Clothing	Jerseys	No
20	CL_SH	Clothing	Shorts	No

```sql
--- check for unwanted spaces

SELECT
*
FROM bronze.erp_px_cat_g1v2
WHERE cat != TRIM(cat) OR subcat != TRIM(subcat) OR maintenance != TRIM(maintenance)
```

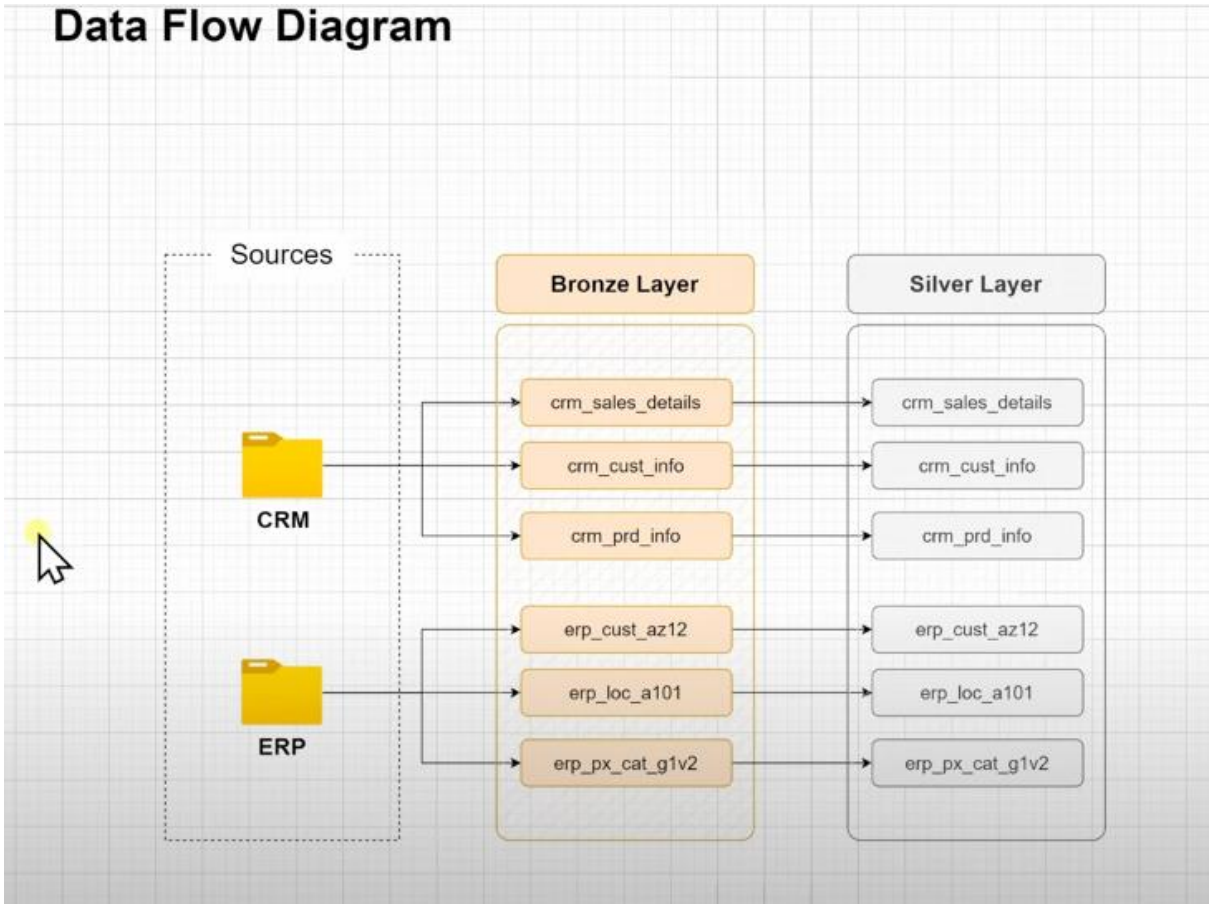
100 %

Results | Messages

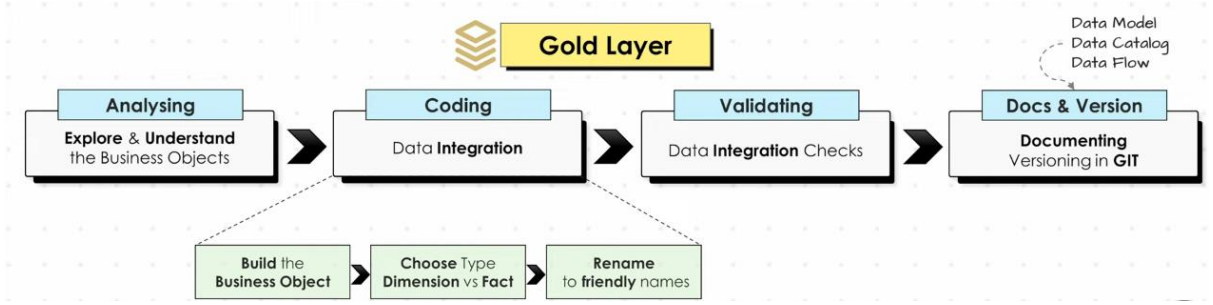
id	cat	subcat	maintenance

```sql
-- Data Standardization and Consistency

SELECT
DISTINCT
cat
FROM
bronze.erp_px_cat_g1v2;

SELECT
DISTINCT
subcat
FROM bronze.erp_px_cat_g1v2;
```

We don’t need to do anything on this table as we’ve nice data quality for this table


Data Flow Diagram
Sources
CRM
ERP
Bronze Layer
crm_sales_details
crm_cust_info
crm_prd_info
erp_cust_az12
erp_loc_a101
erp_px_cat_g1v2
Silver Layer
crm_sales_details
crm_cust_info
crm_prd_info
erp_cust_az12
erp_loc_a101
erp_px_cat_g1v2

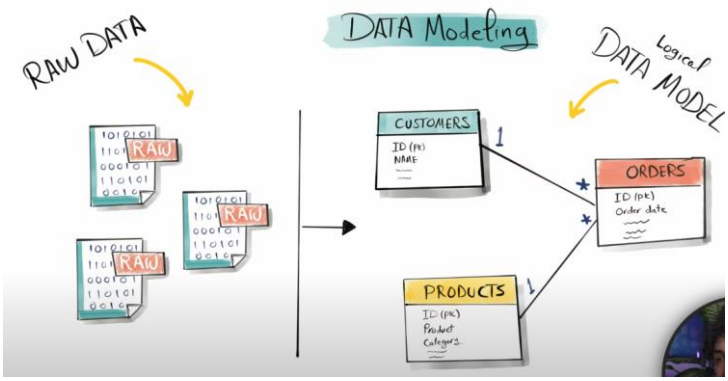
Gold Layer


Gold Layer
Data Model
Data Catalog
Data Flow
Analysing
Explore & Understand the Business Objects
Coding
Data Integration
Validating
Data Integration Checks
Docs & Version
Documenting Versioning in GIT
Build the Business Object
Choose Type Dimension vs Fact
Rename to friendly names

Data Modelling

Data Modelling:

Data modelling is the process of creating a conceptual or logical representation of data structures, their relationships, and the rules governing them. It is widely used in designing databases, systems, or applications to organize and manage data effectively.



A data model consists of 3 components

Entities:

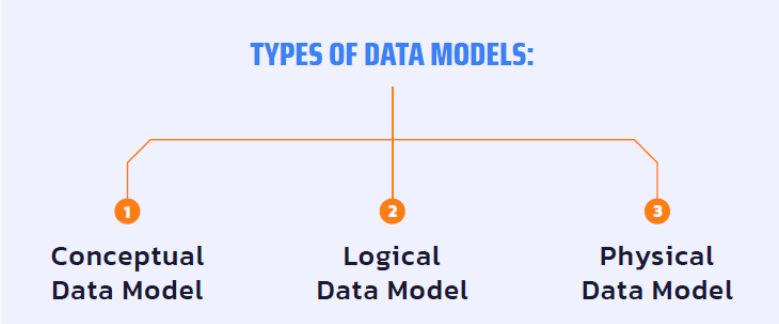
These are the main things we want to store information about. For instance, in a business, an entity could be "Customer" or "Product."

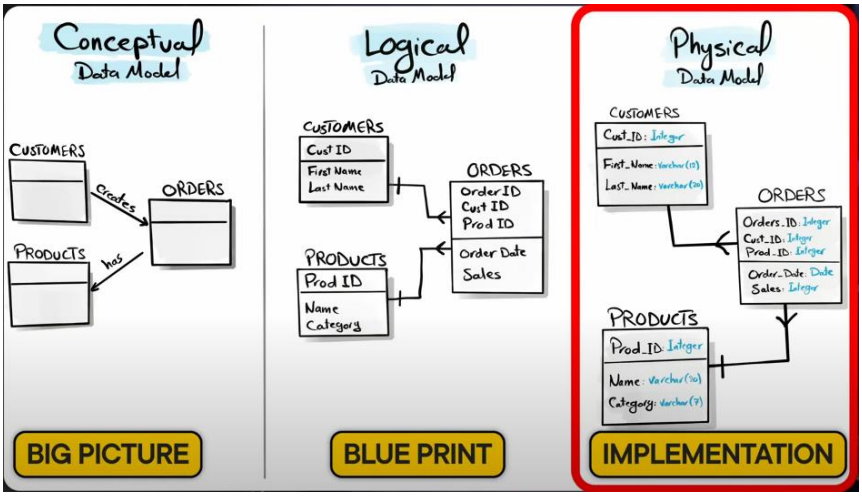
Attributes:

These are the specific pieces of information about an entity. For a "Customer," attributes could include name, address, and phone number.

Relationships:

These define how entities are connected or related to each other. For example, a "Customer" can have a relationship with an "Order."





1 CONCEPTUAL DATA MODEL

A high-level view of what needs to be stored and how different entities relate to each other. It's like a bird's eye view.

2 LOGICAL DATA MODEL

More detailed than the conceptual model, specifying attributes and relationships. It's like a floor plan of a house.

3 PHYSICAL DATA MODEL

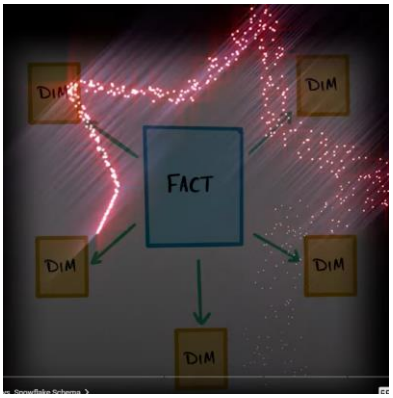
It specifies how the data will be stored, considering database technologies and constraints. It's like the actual construction of the house.

Note: In most cases, we work with conceptual or logical data models, as building a physical model requires significant effort. Additionally, tools like Databricks and Power BI allow us to create and manage physical data models efficiently. In our project, we primarily use a logical data model.

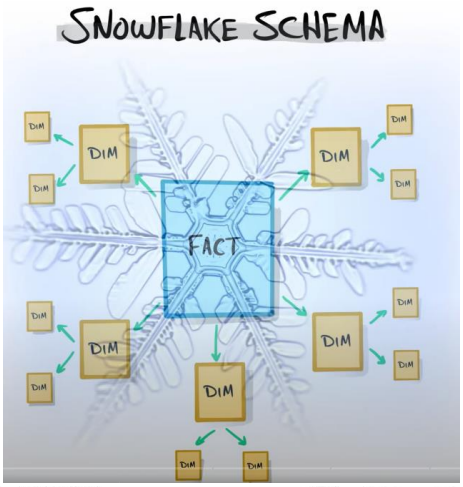
We've two special data models for,

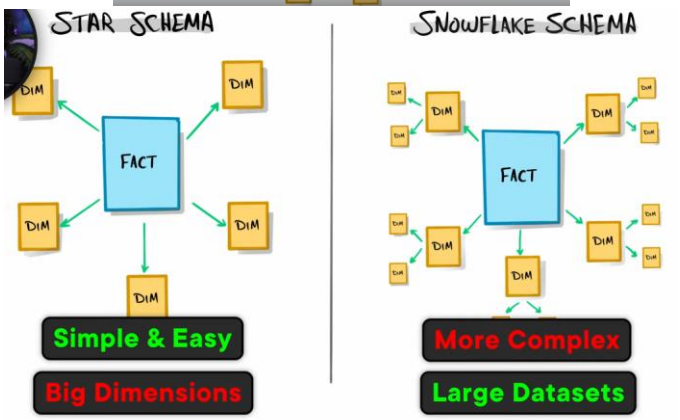
- 1)optimized for reporting
- 2)Flexible
- Easy to understand

1) **Start schema:** In a star schema, the FACT table is centrally positioned and surrounded by DIMENSION (DIM) tables. The DIM tables store descriptive information, providing context to the data, while the FACT table contains measurable values such as quantity and sales.

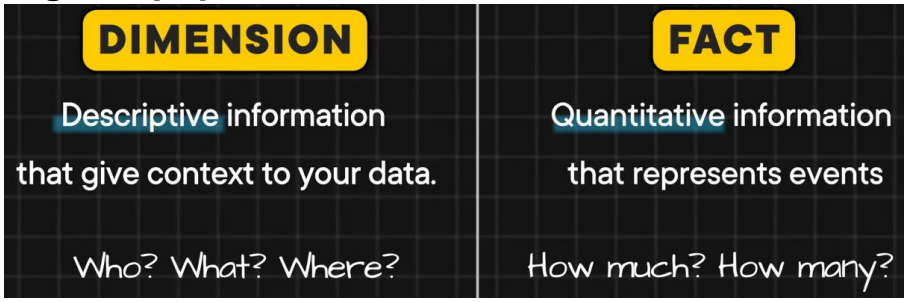


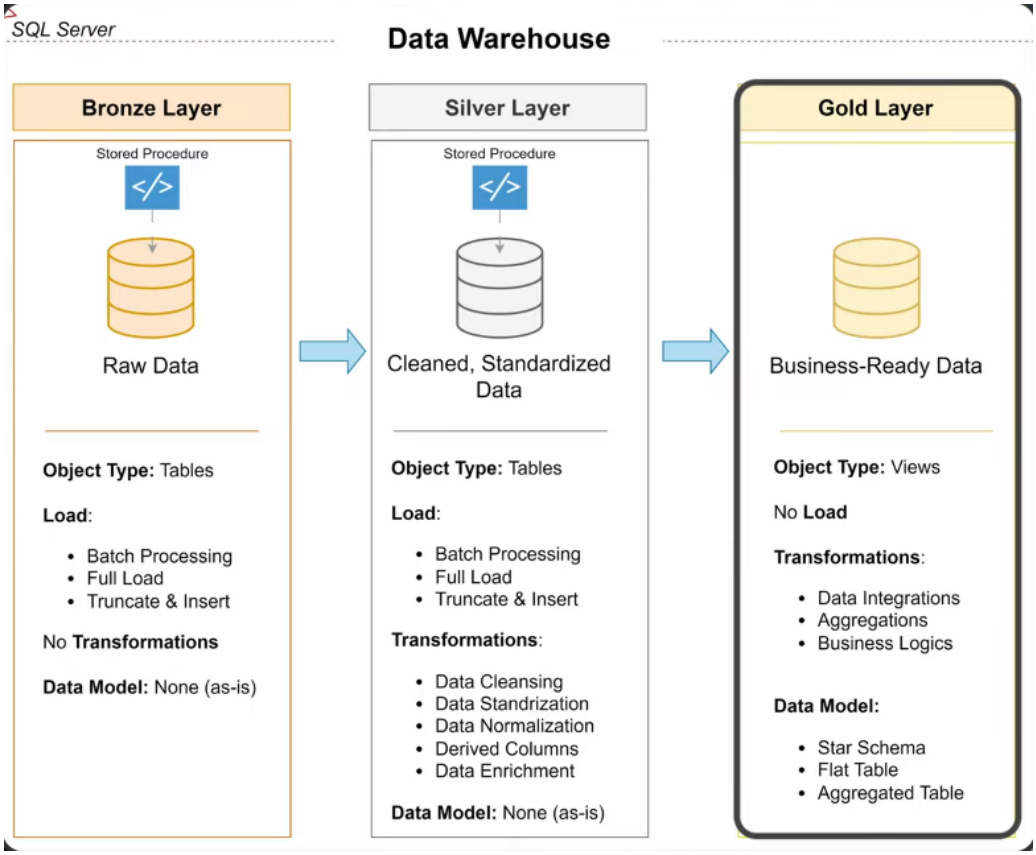
2) **Snowflake Schema:** This is like a star schema, but the DIMENSION (DIM) tables are further normalized to reduce data redundancy and optimize storage efficiency.

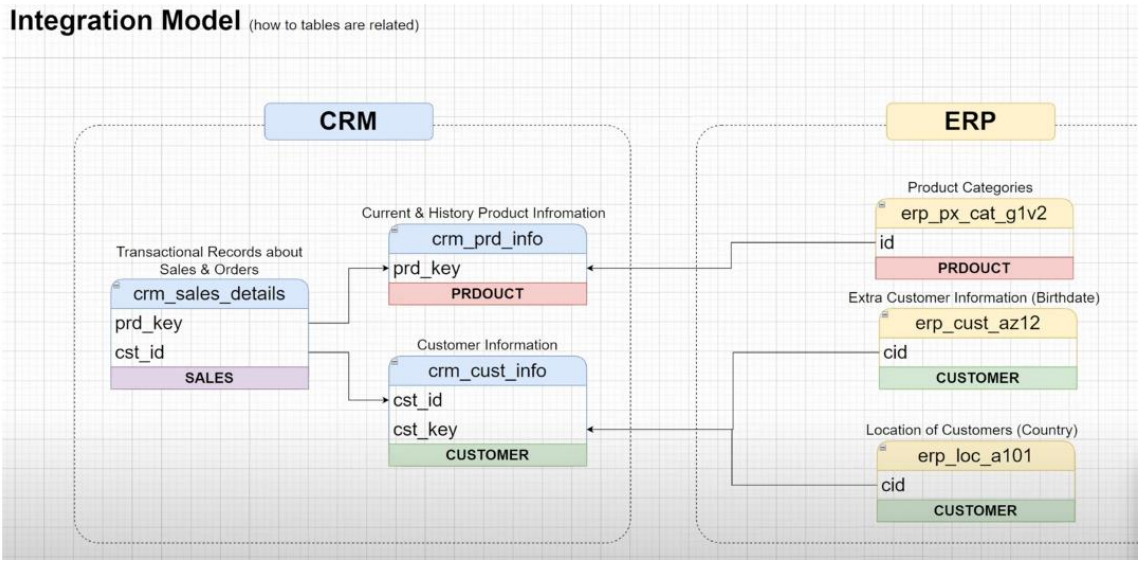


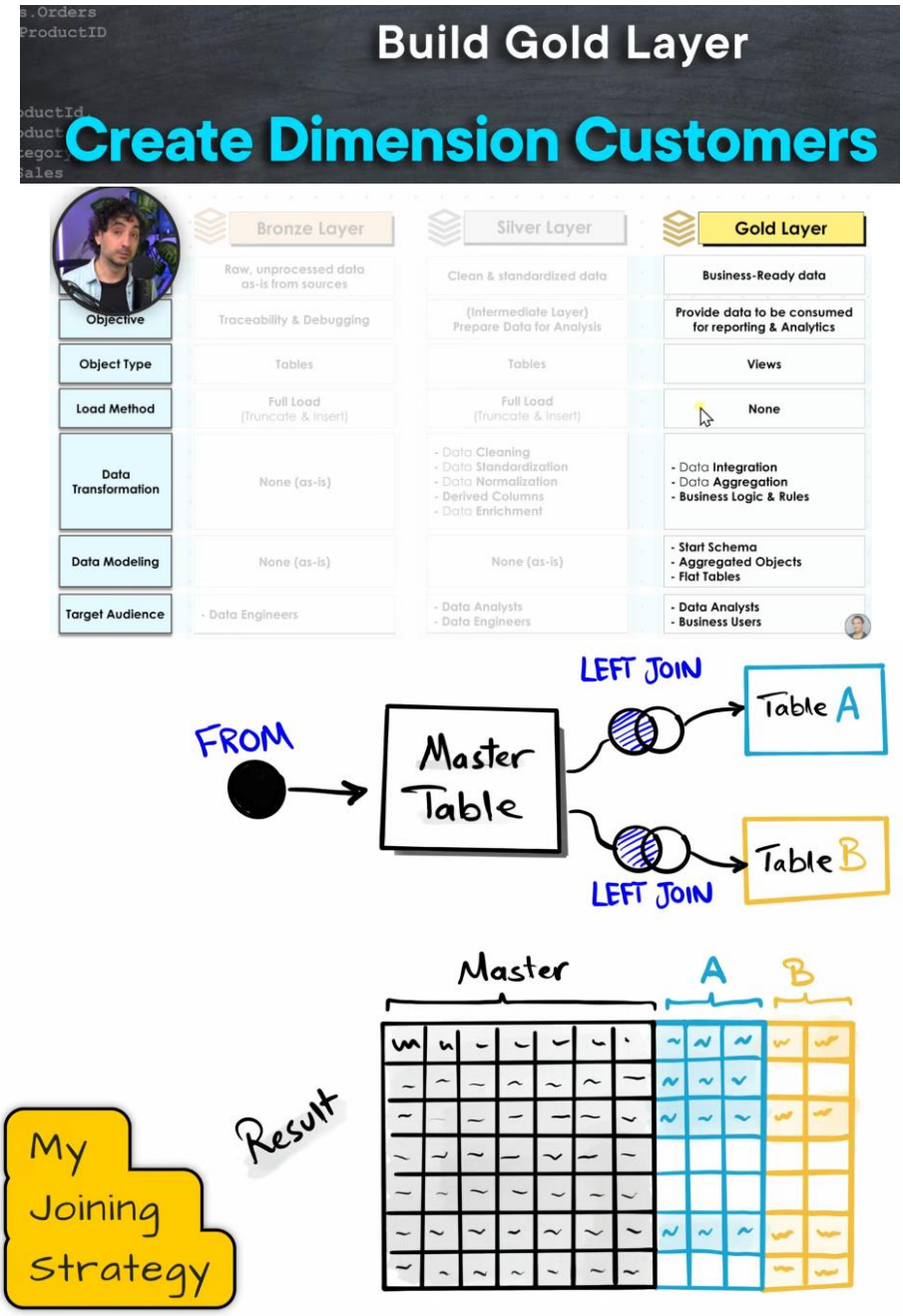


In our project, we've chosen star schema as it is easy for reporting and popular




SQL Server
Data Warehouse
Bronze Layer
Stored Procedure
Raw Data
Object Type: Tables
Load:
Batch Processing
Full Load
Truncate & Insert
No Transformations
Data Model: None (as-is)
Silver Layer
Stored Procedure
Cleaned, Standardized Data
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)
Gold Layer
Business-Ready Data
Object Type: Views
No Load
Transformations:
Data Integrations
Aggregations
Business Logics
Data Model:
Star Schema
Flat Table
Aggregated Table


Integration Model
(how to tables are related)
CRM
ERP
Transactional Records about Sales & Orders
crm_sales_details
prd_key
cst_id
SALES
Current & History Product Infromation
crm_prd_info
prd_key
PRDOUCT
Customer Information
crm_cust_info
cst_id
cst_key
CUSTOMER
Product Categories
erp_px_cat_g1v2
id
PRDOUCT
Extra Customer Information (Birthdate)
erp_cust_az12
cid
CUSTOMER
Location of Customers (Country)
erp_loc_a101
cid
CUSTOMER



NOTE: It is preferable to use a LEFT JOIN instead of an INNER JOIN to ensure that we do not lose any data from the primary table, even if there are no matching records in the joined table.

```sql
SELECT
ci.cst_id,
ci.cst_key,
ci.cst_firstname,
ci.cst_lastname,
ci.cst_marital_status,
ci.cst_gndr,
ci.cst_create_date,
ca.bdate,
ca.gen,
la.cntry
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
```

100 %

Results | Messages

	cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	bdate	gen	cntry
1	11000	AW00011000	Jon	Yang	Married	Male	2025-10-06	1971-10-06	Male	Australia
2	11001	AW00011001	Eugene	Huang	Single	Male	2025-10-06	1976-05-10	Male	Australia
3	11002	AW00011002	Ruben	Torres	Married	Male	2025-10-06	1971-02-09	Male	Australia
4	11003	AW00011003	Christy	Zhu	Single	Female	2025-10-06	1973-08-14	Female	Australia
5	11004	AW00011004	Elizabeth	Johnson	Single	Female	2025-10-06	1979-08-05	Female	Australia
6	11005	AW00011005	Julio	Ruiz	Single	Male	2025-10-06	1976-08-01	Male	Australia

```sql
SELECT cst_id,COUNT(*) FROM
(
SELECT
ci.cst_id,
ci.cst_key,
ci.cst_firstname,
ci.cst_lastname,
ci.cst_marital_status,
ci.cst_gndr,
ci.cst_create_date,
ca.bdate,
ca.gen,
la.cntry
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
)t
GROUP BY cst_id
HAVING COUNT(*)>1
```

100 %

Results | Messages

cst_id	(No column name)

We don’t have any duplicates after joining multiple tables.

cst_id	cst_key	cst_firstname	cst_lastname	cst_marital_status	cst_gndr	cst_create_date	bdate	gen	cntry
11000	AW00011000	Jon	Yang	Married	Male	2025-10-06	1971-10-06	Male	Australia
11001	AW00011001	Eugene	Huang	Single	Male	2025-10-06	1976-05-10	Male	Australia

Currently, we have multiple gender columns from different tables. To ensure consistency and avoid redundancy, we need to integrate the data and consolidate it into a single gender column.

```sql
SELECT
DISTINCT
ci.cst_gndr,
ca.gen
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
ORDER BY 1,2
```

100 %

Results | Messages

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

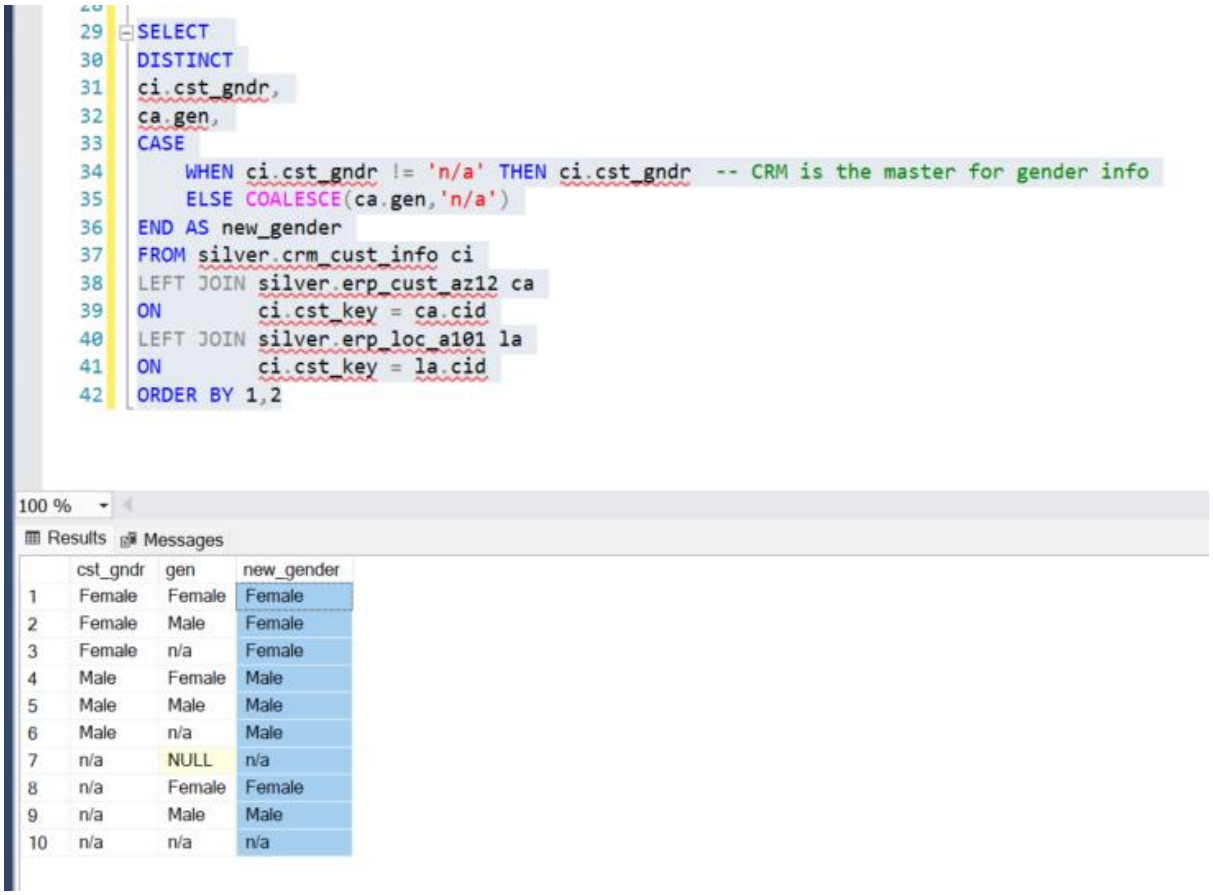
	cst_gndr	gen
1	Female	Female
2	Female	Male
3	Female	n/a
4	Male	Female
5	Male	Male
6	Male	n/a
7	n/a	NULL

NULLs often come from joined tables!
NULL will appear if SQL finds no match

	cst_gndr	gen
1	Female	Female
2	Female	Male
3	Female	n/a
4	Male	Female
5	Male	Male

1)We found NULL values after joining the tables, even though NULLs were already handled in the silver table. These NULL values are not originating from the silver table but are instead a result of the join operation. When there are non-matching records between the tables, NULL values appear in the output.

2) In some cases, one column contains "Female" while another contains "Male," making it unclear which value to use. To resolve this, we need to confirm with the source administrator and choose the appropriate column. In our project, we are using **cst_gndr** from **silver.crm_cust_info**.



```sql
SELECT
DISTINCT
ci.cst_gndr,
ca.gen,
CASE
    WHEN ci.cst_gndr != 'n/a' THEN ci.cst_gndr  -- CRM is the master for gender info
    ELSE COALESCE(ca.gen,'n/a')
END AS new_gender
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
ORDER BY 1,2
```

100 %

Results | Messages

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

General Principles

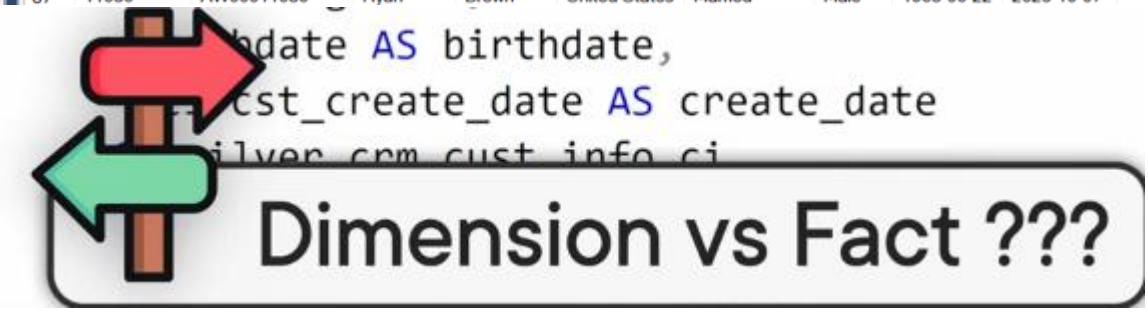
- **Naming Conventions**: Use snake_case, with lowercase letters and underscores (_) to separate words.
- **Language**: Use English for all names.
- **Avoid Reserved Words**: Do not use SQL reserved words as object names.

```sql
SELECT
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 master for gender info
    ELSE COALESCE(ca.gen,'n/a')
END AS gender,
ca.bdate AS birthdate,
ci.cst_create_date AS create_date
FROM silver.crm_cust_info ci
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
```

100 %

Results | Messages

	customer_id	customer_number	first_name	last_name	country	marital_status	gender	birthdate	create_date
79	11078	AW00011078	Gina	Martin	Canada	Single	Female	1985-01-06	2025-10-07
80	11079	AW00011079	Donald	Gonzalez	Australia	Single	Male	1970-03-07	2025-10-07
81	11080	AW00011080	Damien	Chander	Australia	Married	Male	1965-01-13	2025-10-07
82	11081	AW00011081	Savannah	Baker	United States	Married	Female	1972-01-21	2025-10-07
83	11082	AW00011082	Angela	Butler	United States	Single	Female	1972-02-01	2025-10-07
84	11083	AW00011083	Alyssa	Cox	United States	Married	Female	1977-03-11	2025-10-07
85	11084	AW00011084	Lucas	Phillips	United States	Single	Male	1963-03-12	2025-10-07
86	11085	AW00011085	Emily	Johnson	United States	Single	Female	1963-01-16	2025-10-07
87	11086	AW00011086	Ryan	Brown	United States	Married	Male	1963-06-22	2025-10-07

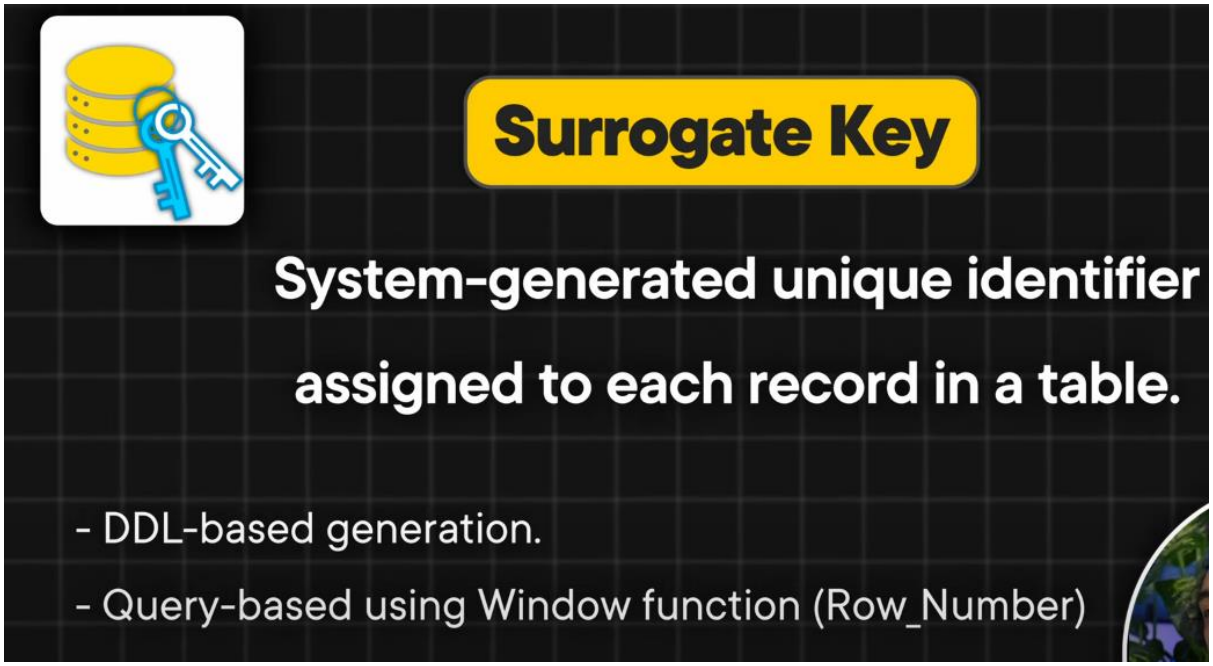


We need to determine the object type—whether it is a Dimension or a Fact table. Since it contains descriptive data, our object type (customers) is classified as a Dimension Table.

When creating a new Dimension table, it is essential to have a primary key. While we can use the primary key from the source system, there are instances where a reliable primary key is unavailable. In such cases, we generate a primary key within the data warehouse, known as a **Surrogate Key**.

A **Surrogate Key** is not a business key; it holds no intrinsic meaning, and no one in the business refers to it. It is solely used to establish relationships within the data model, giving us more control over data integration without relying entirely on the source system.

And there are different ways on how to generate surrogate keys



We've decided to create views as the objects in the gold layer.

```sql
CREATE VIEW gold.dim_customers AS
SELECT
ROW_NUMBER() OVER (ORDER BY cst_id) AS customer_key, ---we can use any column for creating the 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 master for gender info
    ELSE COALESCE(ca.gen,'n/a')
END AS gender,
ca.bdate AS birthdate,
ci.cst_create_date AS create_date
FROM silver.crm_cust_info ci
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

SELECT * FROM gold.dim_customers
```

100 %

Results | Messages

	customer_key	customer_id	customer_number	first_name	last_name	country	marital_status	gender	birthdate	create_date
1	1	11000	AW00011000	Jon	Yang	Australia	Married	Male	1971-10-06	2025-10-06
2	2	11001	AW00011001	Eugene	Huang	Australia	Single	Male	1976-05-10	2025-10-06
3	3	11002	AW00011002	Ruben	Torres	Australia	Married	Male	1971-02-09	2025-10-06
4	4	11003	AW00011003	Christy	Zhu	Australia	Single	Female	1973-08-14	2025-10-06
5	5	11004	AW00011004	Elizabeth	Johnson	Australia	Single	Female	1979-08-05	2025-10-06
6	6	11005	AW00011005	Julio	Ruiz	Australia	Single	Male	1976-08-01	2025-10-06

Build Gold Layer

Create Dimension Products

```sql
SELECT
*
FROM
silver.crm_prd_info pn
LEFT JOIN silver.erp_px_cat_g1v2 pc
ON pn.cat_id = pc.id
```

100 %

Results | Messages

	prd_id	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	dwh_create_date	id	cat	subcat	maintenance	dwh_create_date
1	478	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
2	479	AC_BC	BC-R205	Road Bottle Cage	3	Road	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
3	477	AC_BC	WB-H098	Water Bottle - 30 oz.	2	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
4	483	AC_BR	RA-H123	Hitch Rack - 4-Bike	45	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BR	Accessories	Bike Racks	Yes	2025-03-15 13:28:02.6900000
5	486	AC_BS	ST-1401	All-Purpose Bike Stand	59	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BS	Accessories	Bike Stands	No	2025-03-15 13:28:02.6900000
6	484	AC_CL	CL-9009	Bike Wash - Dissolver	3	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_CL	Accessories	Cleaners	Yes	2025-03-15 13:28:02.6900000
7	485	AC_FE	FE-6654	Fender Set - Mountain	8	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_FE	Accessories	Fenders	No	2025-03-15 13:28:02.6900000
8	215	AC_HE	HL-U509	Sport-100 Helmet, Black	12	Other Sales	2011-07-01	2012-06-30	2025-03-15 13:28:02.4033333	AC_HE	Accessories	Helmets	Yes	2025-03-15 13:28:02.6900000

We want to retain only currently available products and filter out historical products. We can identify active products by checking if **prd_end_dt is NULL**, as this indicates they are still available.

```sql
SELECT
*
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 historical data
```

100 %

Results | Messages

	prd_id	cat_id	prd_key	prd_nm	prd_cost	prd_line	prd_start_dt	prd_end_dt	dwh_create_date	id	cat	subcat	maintenance	dwh_create_date
1	478	AC_BC	BC-M005	Mountain Bottle Cage	4	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
2	479	AC_BC	BC-R205	Road Bottle Cage	3	Road	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
3	477	AC_BC	WB-H098	Water Bottle - 30 oz.	2	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BC	Accessories	Bottles and Cages	No	2025-03-15 13:28:02.6900000
4	483	AC_BR	RA-H123	Hitch Rack - 4-Bike	45	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BR	Accessories	Bike Racks	Yes	2025-03-15 13:28:02.6900000
5	486	AC_BS	ST-1401	All-Purpose Bike Stand	59	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_BS	Accessories	Bike Stands	No	2025-03-15 13:28:02.6900000
6	484	AC_CL	CL-9009	Bike Wash - Dissolver	3	Other Sales	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_CL	Accessories	Cleaners	Yes	2025-03-15 13:28:02.6900000
7	485	AC_FE	FE-6654	Fender Set - Mountain	8	Mountain	2013-07-01	NULL	2025-03-15 13:28:02.4033333	AC_FE	Accessories	Fenders	No	2025-03-15 13:28:02.6900000

No duplicates found for prd_id

```sql
SELECT prd_id, COUNT(*)
FROM
(SELECT
    pn.prd_id,
    pn.cat_id,
    pn.prd_key,
    pn.prd_nm,
    pn.prd_cost,
    pn.prd_line,
    pn.prd_start_dt,
    pc.cat,
    pc.subcat,
    pc.maintenance
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)t --- Filter out historical data
GROUP BY prd_id
HAVING COUNT(*)>1
```

100 %

Results | Messages

prd_id	(No column name)

We don’t have duplicate columns and no need to do any data integration

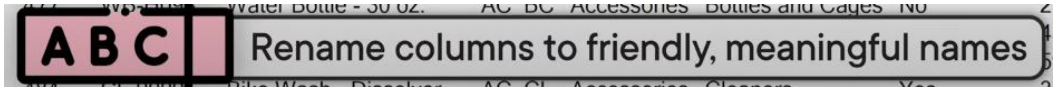
Now, grouping similar data together

```sql
----- grouping similar data together
SELECT
    pn.prd_id,
    pn.prd_key,
    pn.prd_nm,
    pn.cat_id,
    pc.cat,
    pc.subcat,
    pn.prd_cost,
    pn.prd_line,
    pn.prd_start_dt,
    pc.maintenance
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
```

100 %

Results | Messages

	prd_id	prd_key	prd_nm	cat_id	cat	subcat	prd_cost	prd_line	prd_start_dt	maintenance
1	478	BC-M005	Mountain Bottle Cage	AC_BC	Accessories	Bottles and Cages	4	Mountain	2013-07-01	No
2	479	BC-R205	Road Bottle Cage	AC_BC	Accessories	Bottles and Cages	3	Road	2013-07-01	No
3	477	WB-H098	Water Bottle - 30 oz.	AC_BC	Accessories	Bottles and Cages	2	Other Sales	2013-07-01	No




```sql
----- grouping similar data together and renaming the columns to appropriate name

SELECT
    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,
    pn.prd_cost AS cost,
    pn.prd_line AS product_line,
    pn.prd_start_dt AS start_date,
    pc.maintenance AS maintenance
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
```

100 %

Results | Messages

	product_id	product_number	product_name	category_id	category	subcategory	cost	product_line	start_date	maintenance
1	478	BC-M005	Mountain Bottle Cage	AC_BC	Accessories	Bottles and Cages	4	Mountain	2013-07-01	No
2	479	BC-R205	Road Bottle Cage	AC_BC	Accessories	Bottles and Cages	3	Road	2013-07-01	No
3	477	WB-H098	Water Bottle - 30 oz.	AC_BC	Accessories	Bottles and Cages	2	Other Sales	2013-07-01	No
4	483	RA-H123	Hitch Rack - 4-Bike	AC_BR	Accessories	Bike Racks	45	Other Sales	2013-07-01	Yes
5	486	ST-1401	All-Purpose Bike Stand	AC_BS	Accessories	Bike Stands	59	Mountain	2013-07-01	No
6	484	CL-9009	Bike Wash - Dissolver	AC_CL	Accessories	Cleaners	3	Other Sales	2013-07-01	Yes

We need to determine the object type—whether it is a Dimension or a Fact table. Since it contains descriptive data, our object type (products) is classified as a Dimension Table.

Declaring product_key as the surrogate key

```sql
----- grouping similar data together and renaming the columns to appropriate name

SELECT
    ROw_NUMBER() OVER(ORDER BY  pn.prd_start_dt,pn.prd_key) AS product_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,
    pn.prd_cost AS cost,
    pn.prd_line AS product_line,
    pn.prd_start_dt AS start_date,
    pc.maintenance AS maintenance
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
```

100 %

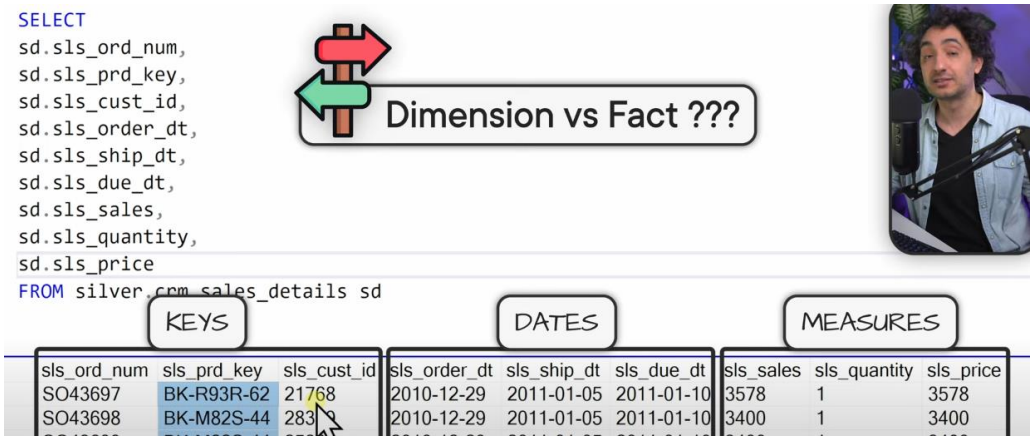
Results | Messages

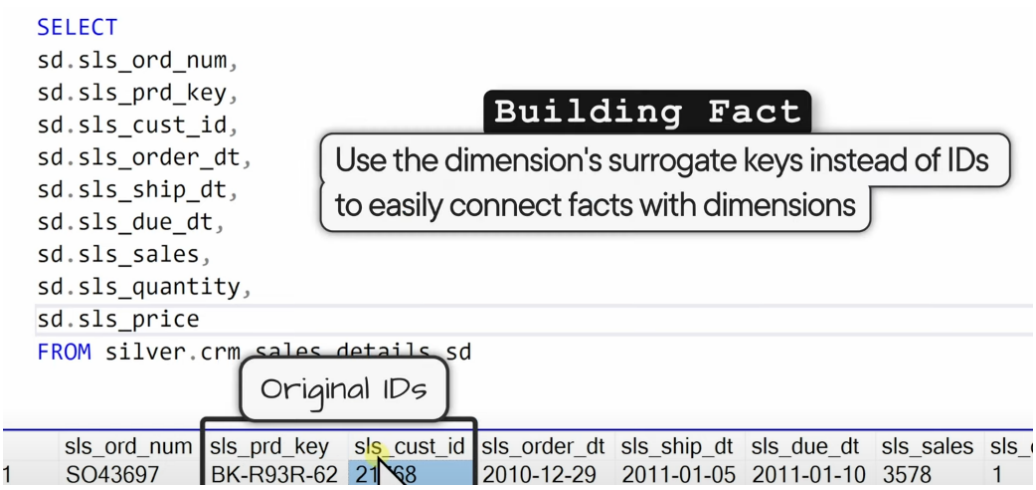
	product_key	product_id	product_number	product_name	category_id	category	subcategory	cost	product_line	start_date	maintenance
1	1	210	FR-R92B-58	HL Road Frame - Black- 58	CO_RF	Components	Road Frames	0	Road	2003-07-01	Yes
2	2	211	FR-R92R-58	HL Road Frame - Red- 58	CO_RF	Components	Road Frames	0	Road	2003-07-01	Yes
3	3	348	BK-M82B-38	Mountain-100 Black- 38	BI_MB	Bikes	Mountain Bikes	1898	Mountain	2011-07-01	Yes

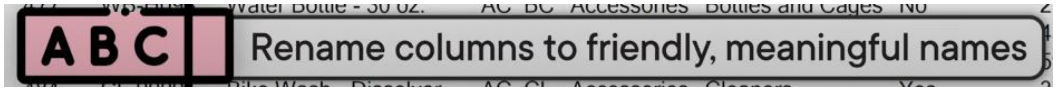
Creating VIEW

```sql
CREATE VIEW gold.dim_products AS
    SELECT
        ROw_NUMBER() OVER(ORDER BY  pn.prd_start_dt,pn.prd_key) AS product_key,--if we use one column also it will work
        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,
        pn.prd_cost AS cost,
        pn.prd_line AS product_line,
        pn.prd_start_dt AS start_date,
        pc.maintenance AS maintenance
    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
```

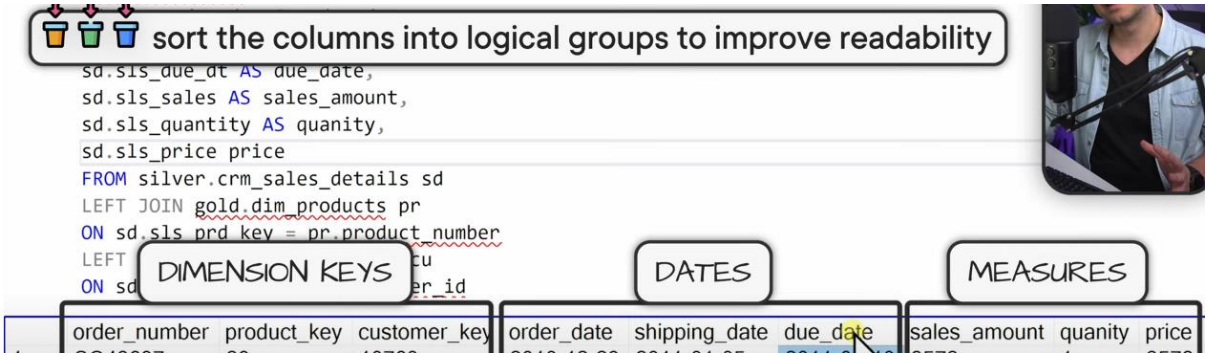
Since it contains sales and quantity metrics, it qualifies as a **Fact Table**.








```sql
-- Renaming the columns

SELECT
sd.sls_ord_num AS order_number,
--sd.sls_prd_key,
pr.product_key,      -- as we want use this surrogate key instead of sls_prd_key
--sd.sls_cust_id,
cu.customer_key,
sd.sls_order_dt AS order_date,
sd.sls_ship_dt AS shipping_date,
sd.sls_due_dt AS due_date,
sd.sls_sales AS sales_amount,
sd.sls_quantity AS quantity,
sd.sls_price AS price
FROM silver.crm_sales_details sd
LEFT JOIN gold.dim_products pr
ON        sd.sls_prd_key = pr.product_number
LEFT JOIN gold.dim_customers cu
ON        sd.sls_cust_id = cu.customer_id
```



We don't need to order the columns here as we've them in correct order.

Creating VIEW

```sql
CREATE VIEW gold.fact_sales AS
    SELECT
    sd.sls_ord_num AS order_number,
    --sd.sls_prd_key,
    pr.product_key,       -- as we want use this surrogate key instead of sls_prd_key
    --sd.sls_cust_id,
    cu.customer_key,
    sd.sls_order_dt AS order_date,
    sd.sls_ship_dt AS shipping_date,
    sd.sls_due_dt AS due_date,
    sd.sls_sales AS sales_amount,
    sd.sls_quantity AS quantity,
    sd.sls_price AS price
    FROM silver.crm_sales_details sd
    LEFT JOIN gold.dim_products pr
    ON        sd.sls_prd_key = pr.product_number
    LEFT JOIN gold.dim_customers cu
    ON        sd.sls_cust_id = cu.customer_id


SELECT * FROM [gold].[fact_sales]
```

100 %

Results | Messages

	order_number	product_key	customer_key	order_date	shipping_date	due_date	sales_amount	quantity	price
1	SO43697	20	10769	2010-12-29	2011-01-05	2011-01-10	3578	1	3578
2	SO43698	9	17390	2010-12-29	2011-01-05	2011-01-10	3400	1	3400
3	SO43699	9	14864	2010-12-29	2011-01-05	2011-01-10	3400	1	3400

Fact Check

Check if all dimension tables can successfully join to the fact table

```
----Data Integrity check

SELECT
*
FROM [gold].[fact_sales] f
LEFT JOIN [gold].[dim_customers] c
ON f.customer_key = c.customer_key
LEFT JOIN [gold].[dim_products] p
ON f.product_key = p.product_key
WHERE c.customer_key IS NULL
```

100 %

Results | Messages

order_number	product_key	customer_key	order_date	shipping_date	due_date	sales_amount	quantity	price	customer_key	customer_id	customer_number	first_name	last_name	country	marital_status	gender	birthdate	create_date	produ

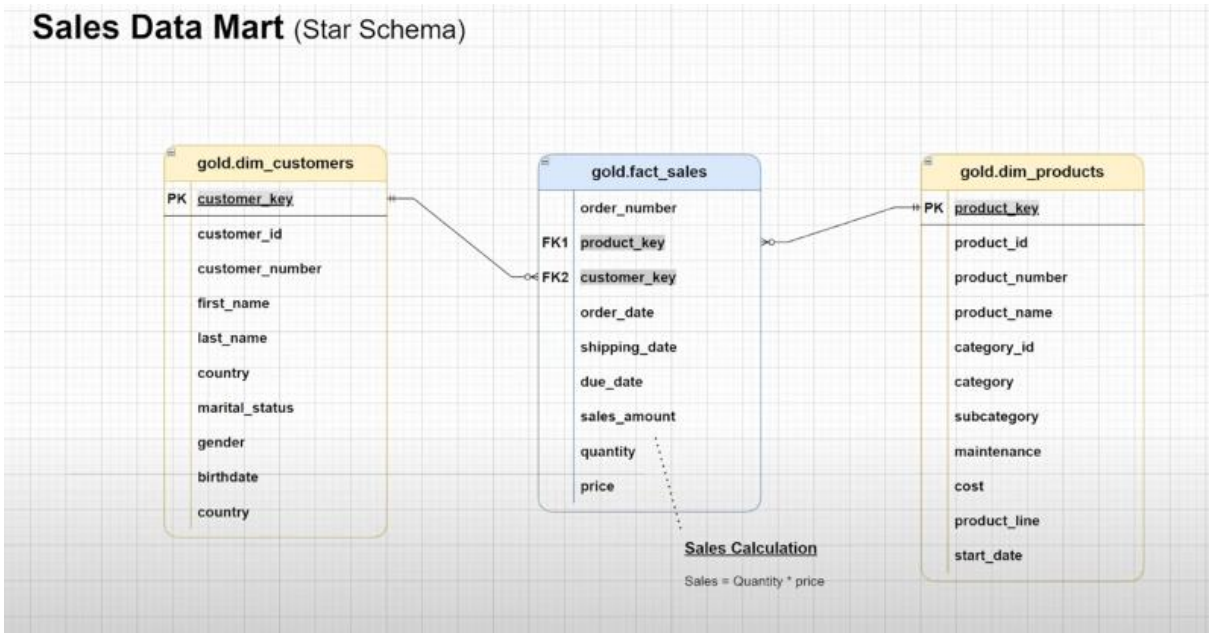
STAR SCHEMA

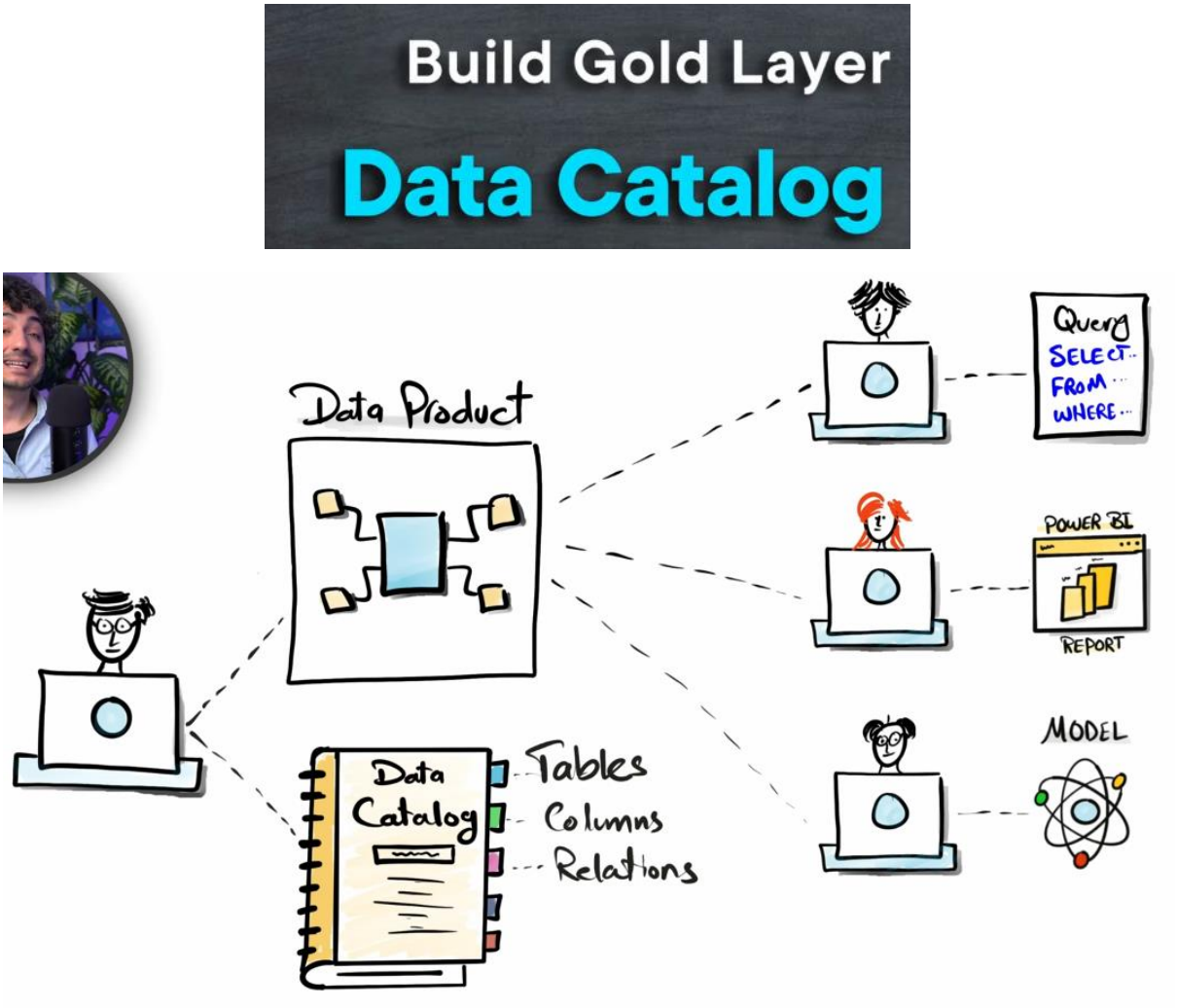
Relationship

in a star schema, the relationship between fact and dimensions is 1-to-many (1:N)

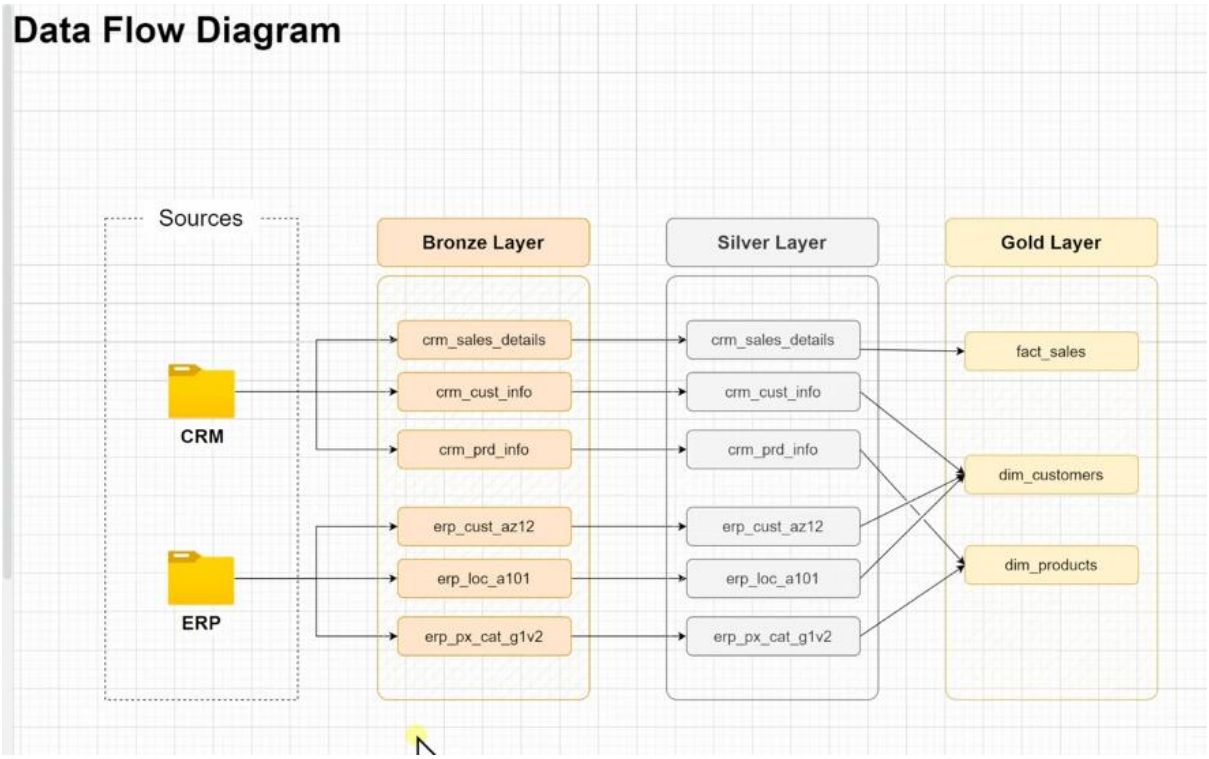
Many (Optional)

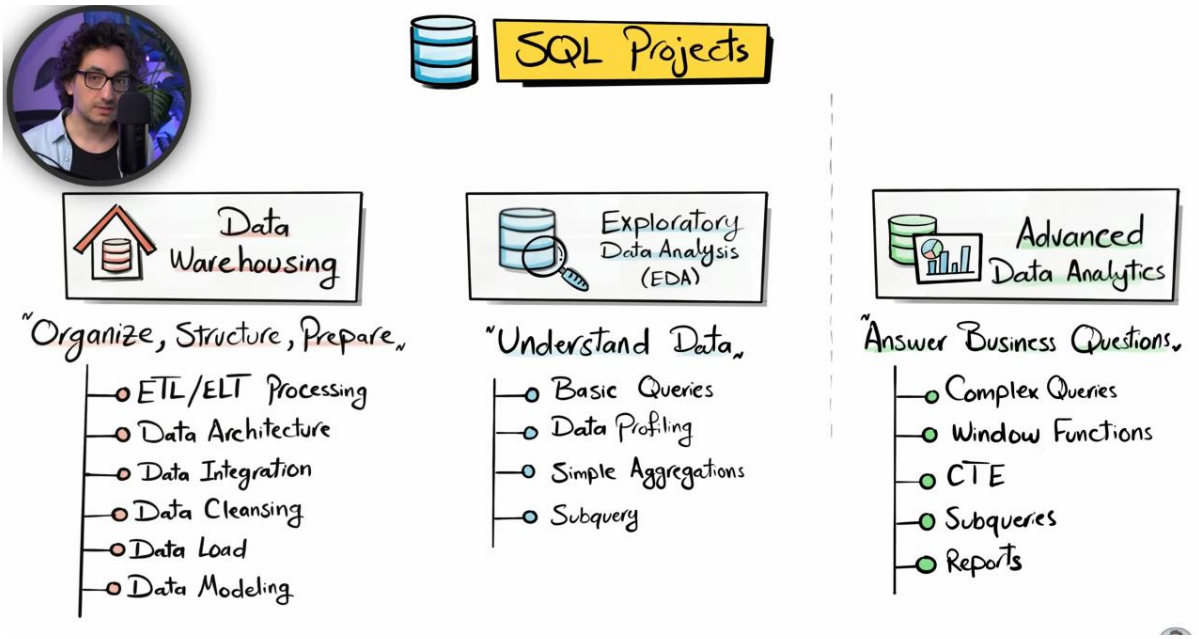
1. Customers who haven't placed any orders yet.
2. Customers who have placed only one order.
3. Customers who have placed multiple orders.





A Data Catalog is a centralized metadata repository that helps organizations efficiently discover, manage, and govern their data assets. It provides a structured inventory of data across various sources, enabling users to search, understand, and utilize data effectively.





Exploratory Data Analysis (EDA)

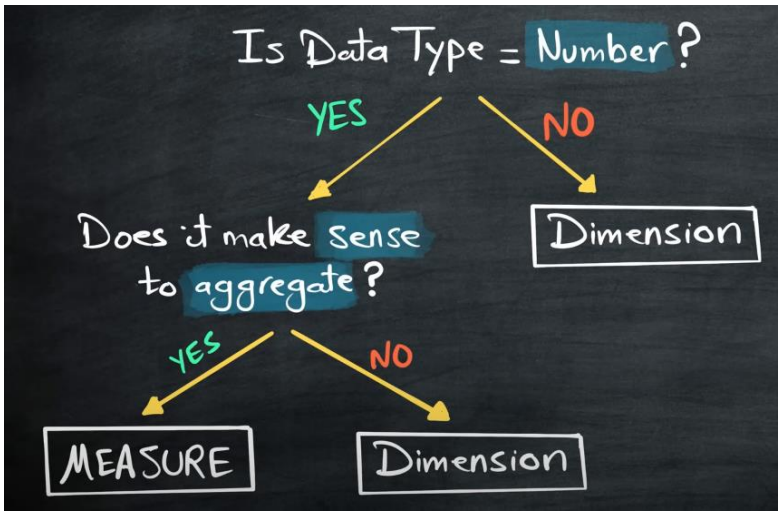
Exploratory Data Analysis (EDA) is the process of analysing and summarizing datasets to uncover patterns, detect anomalies, and extract insights using statistical and visualization techniques. It helps in understanding data distributions, relationships, and potential data quality issues before applying machine learning or further analysis.

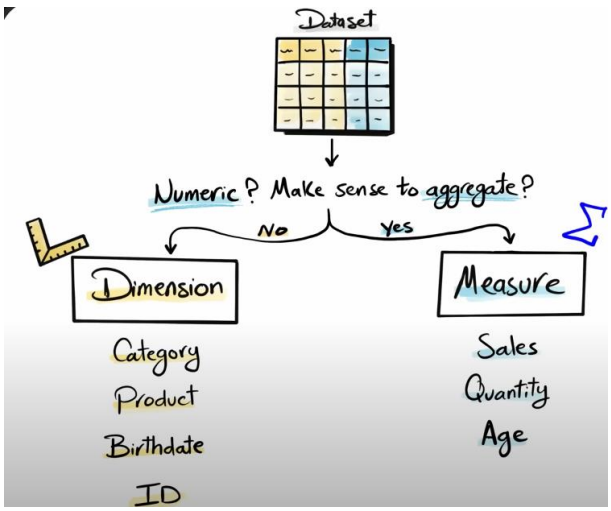
Dimensions & Measures

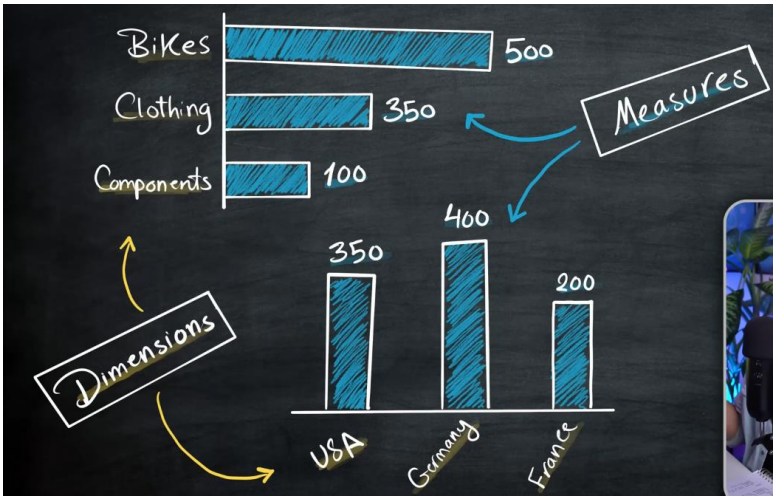
Any dataset is typically categorized into **dimensions** and **measures**:

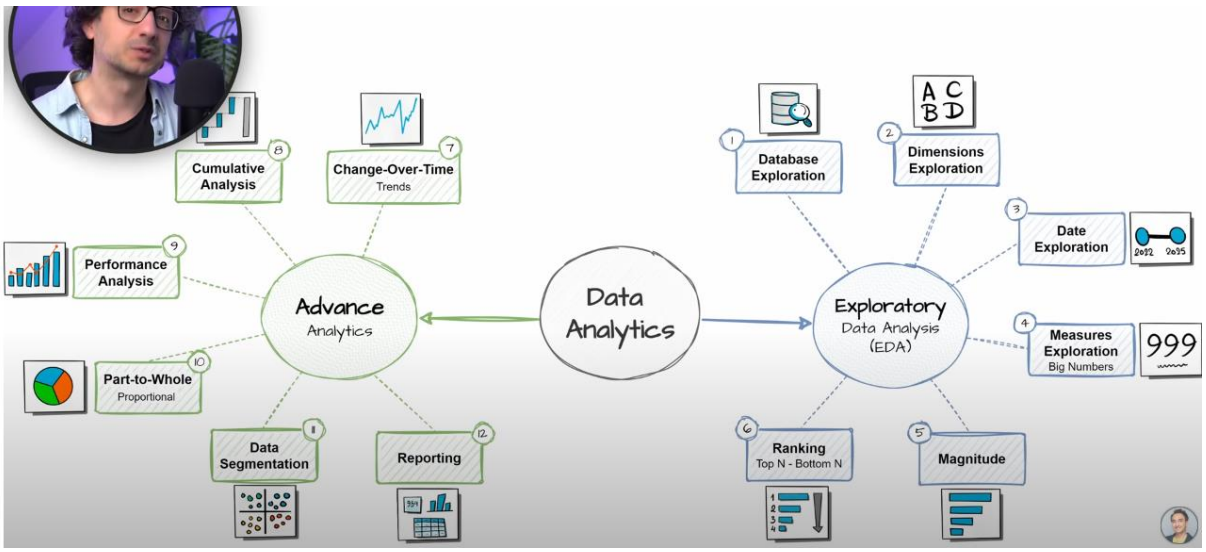
- **Dimensions**: Categorical attributes that provide context to the data (e.g., Product Name, Customer ID, Region, Date). They help in grouping, filtering, and segmenting data.
- **Measures**: Numeric values that can be aggregated and analyzed (e.g., Sales, Quantity, Revenue, Profit). These represent the actual metrics used for calculations.

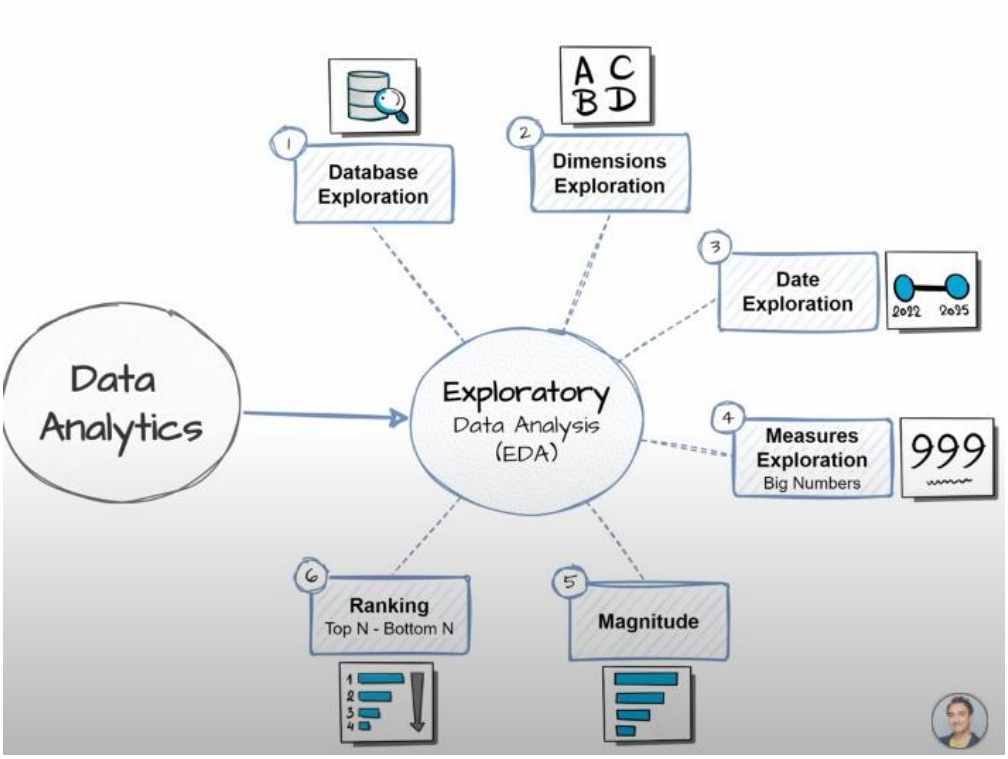
In a **star schema**, dimensions are stored in **dimension tables**, and measures are stored in **fact tables**.






Bikes
500
Clothing
350
Components
100
Measures
400
350
200
Dimensions
USA
Germany
France


Cumulative Analysis
Change-Over-Time
Trends
Performance Analysis
Advance Analytics
Data Analytics
Exploratory Data Analysis (EDA)
Database Exploration
Dimensions Exploration
Date Exploration
Measures Exploration
Big Numbers
999
Part-to-Whole
Proportional
Data Segmentation
Reporting
Ranking
Top N - Bottom N
Magnitude


Database Exploration
Dimensions Exploration
Date Exploration
Data Analytics
Exploratory Data Analysis (EDA)
Measures Exploration
Big Numbers
999
Ranking
Top N - Bottom N
Magnitude

Exploratory Data Analysis

Database Exploration

In **Database Exploration**, we analyze the structure of the database by identifying the number of tables, their relationships, and the types of columns present in each table. This helps in understanding the data model and preparing for further analysis.

```
USE DataWarehouse

-- Explore All objects in the Database

SELECT * FROM INFORMATION_SCHEMA.TABLES
```

100 %

Results | Messages

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

```
-- Explore All columns in the table

SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'fact_sales'
```

100 %

Results | Messages

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUM
1	DataWarehouse	gold	fact_sales	order_number	1	NULL
2	DataWarehouse	gold	fact_sales	product_key	2	NULL
3	DataWarehouse	gold	fact_sales	customer_key	3	NULL
4	DataWarehouse	gold	fact_sales	order_date	4	NULL
5	DataWarehouse	gold	fact_sales	shipping_date	5	NULL
6	DataWarehouse	gold	fact_sales	due_date	6	NULL
7	DataWarehouse	gold	fact_sales	sales_amount	7	NULL
8	DataWarehouse	gold	fact_sales	quantity	8	NULL
9	DataWarehouse	gold	fact_sales	price	9	NULL

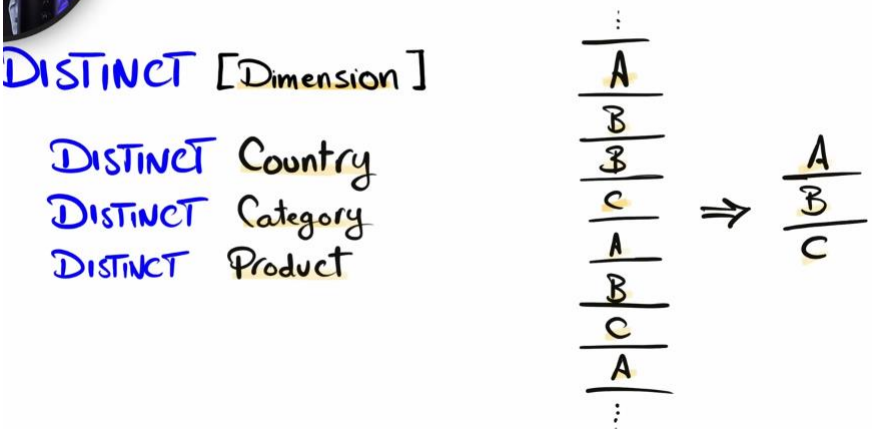
```
USE DataWarehouse

-- Explore All objects in the Database

SELECT * FROM INFORMATION_SCHEMA.TABLES

-- Explore All columns in the table

SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'fact_sales'
```

Exploratory Data Analysis

Dimensions Exploration

Identifying the unique values (or categories) in each dimension.

Recognizing how data might be grouped or segmented, which is useful for later analysis.




```sql
-- Explore All Countries our customers come from.

SELECT DISTINCT country FROM gold.dim_customers
ORDER BY 1
```

100 %

Results | Messages

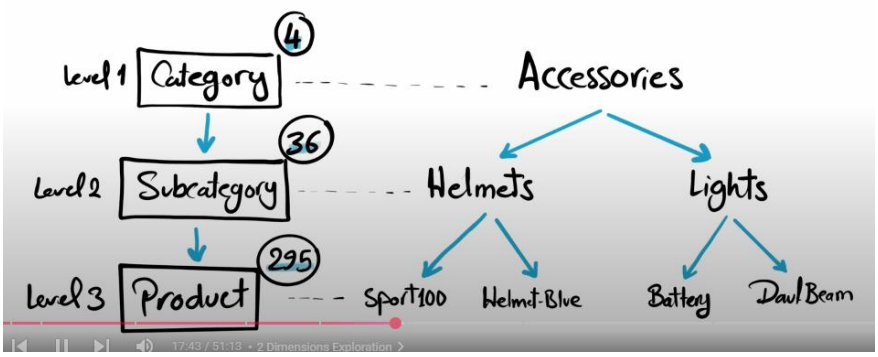
	country
1	Australia
2	Canada
3	France
4	Germany
5	n/a
6	United Kingdom
7	United States

```sql
-- Explore All Categories "The Major Divisions"

SELECT DISTINCT category, subcategory, product_name FROM gold.dim_products
ORDER BY 1,2,3
```

100 %

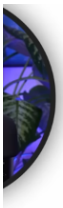
Results | Messages

	category	subcategory	product_name
28	Accessories	Tires and Tubes	LL Mountain Tire
29	Accessories	Tires and Tubes	LL Road Tire
30	Accessories	Tires and Tubes	ML Mountain Tire
31	Accessories	Tires and Tubes	ML Road Tire
32	Accessories	Tires and Tubes	Mountain Tire Tube
33	Accessories	Tires and Tubes	Patch Kit/8 Patches
34	Accessories	Tires and Tubes	Road Tire Tube
35	Accessories	Tires and Tubes	Touring Tire
36	Accessories	Tires and Tubes	Touring Tire Tube
37	Bikes	Mountain Bikes	Mountain-100 Black- 38
38	Bikes	Mountain Bikes	Mountain-100 Black- 42
39	Bikes	Mountain Bikes	Mountain-100 Black- 44
40	Bikes	Mountain Bikes	Mountain-100 Black- 48
41	Bikes	Mountain Bikes	Mountain-100 Silver- 38
42	Bikes	Mountain Bikes	Mountain-100 Silver- 42



Exploratory Data Analysis

Date Exploration



Identify the earliest and latest dates (boundaries).

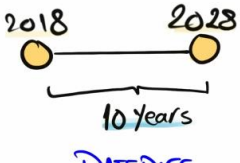
Understand the scope of data and the timespan.

Date Exploration

MIN/MAX [Date Dimension]

- MIN Order_date
- MAX Create_date
- MIN Birthdate

2019
2020
2018
2018
2022
2023
2023
2023
2022



DATEDIFF

```sql
-- Find the date of the first and last order

SELECT
MIN(order_date) AS first_order_date ,
MAX(order_date) AS last_order_date
FROM gold.fact_sales_tb
```

100 %

Results | Messages

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

```sql
-- How many years of sales are available

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_tb
```

100 %

Results | Messages

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

```sql
-- Find the youngest and oldest customer

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_tb
```

100 %

Results | Messages

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

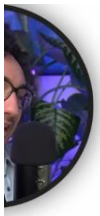
Exploratory Data Analysis

Measures Exploration

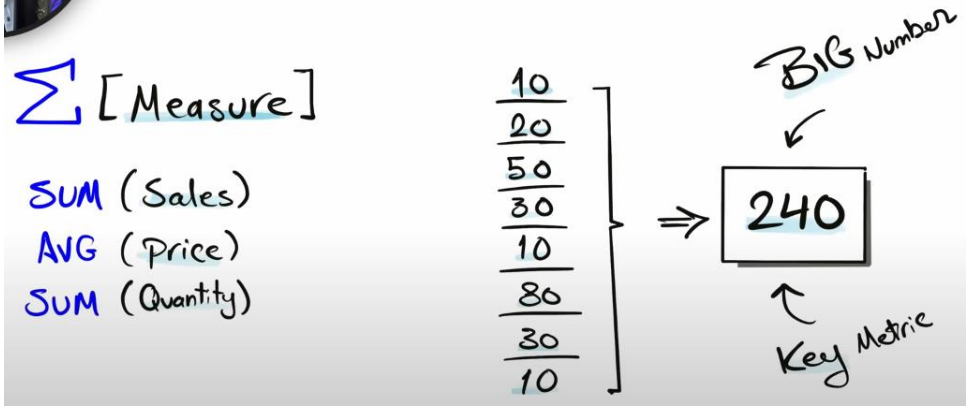
999 Measures Exploration

Calculate the key metric of the business (Big Numbers)

- Highest Level of Aggregation | Lowest Level of Details -



999 Measures Exploration




```sql
-- Find the total sales
SELECT SUM(sales_amount) AS total_sales FROM gold.fact_sales_tb

-- Find how many iterms are sold
SELECT SUM(quantity) AS toatl_quantity FROM gold.fact_sales_tb

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

-- Find the total no of orders
SELECT COUNT( DISTINCT order_number) AS total_orders FROM gold.fact_sales_tb

-- Find the total no of products
SELECT COUNT(product_key) AS total_products FROM gold.dim_products_tb

-- Find the total no of customers
SELECT COUNT(customer_key) AS total_customers FROM gold.dim_customers_tb

-- Find the total no of customers that has placed an order

SELECT COUNT( DISTINCT customer_key) FROM gold.fact_sales_tb
```

```sql
-- 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_tb
UNION ALL
SELECT 'toatl_quantity' AS measure_name, SUM(quantity) AS measure_value FROM gold.fact_sales_tb
UNION ALL
SELECT 'avg_price' AS measure_name, AVG(price) AS measure_value FROM gold.fact_sales_tb
UNION ALL
SELECT 'total_no_of_orders' AS measure_name, COUNT( DISTINCT order_number) AS measure_value FROM gold.fact_sales_tb
UNION ALL
SELECT 'total_no_of_products' AS measure_name, COUNT(product_key) AS measure_value FROM gold.dim_products_tb
UNION ALL
SELECT 'total_no_of_customers' AS measure_name, COUNT(customer_key) AS measure_value FROM gold.dim_customers_tb
```

100 %

Results | Messages

	measure_name	measure_value
1	total_sales	29356250
2	toatl_quantity	60423
3	avg_price	486
4	total_no_of_orders	27659
5	total_no_of_products	295
6	total_no_of_customers	18484

Exploratory Data Analysis

Magnitude Analysis

Magnitude

Compare the measure values by categries.

It helps us understand the importance of different categories.

Magnitude

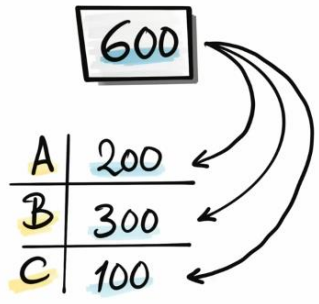
$\sum$ [Measure] By [Dimension]

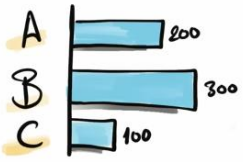
Total Sales By Country

Total Quantity By Category

Average Price By Product

Total Orders By Customer





```sql
-- Find total customers by countries
-- Find total customers by gender
-- Find total products by category
-- What is the average costs in each category?
-- What is the total revenue generated for each category?
-- Find total revenue is generated by each customer
-- What is the distribution of sold items across countries?
```

Exploratory Data Analysis

Ranking Analysis

Order the values of dimensions by measure.

Top N performers | Bottom N Performers

Rank [Dimension] By $\sum$ [Measure]

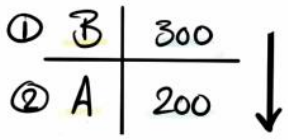
Rank Countries By Total Sales

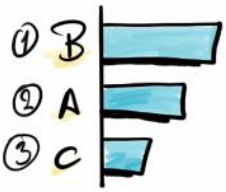
Top 5 Products By Quantity

Bottom 3 Customers By Total Orders

Top RANK() DENSE_RANK()

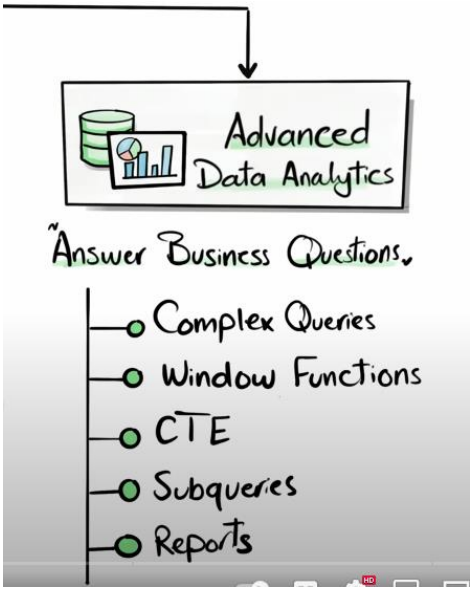
ROW_NUMBER()

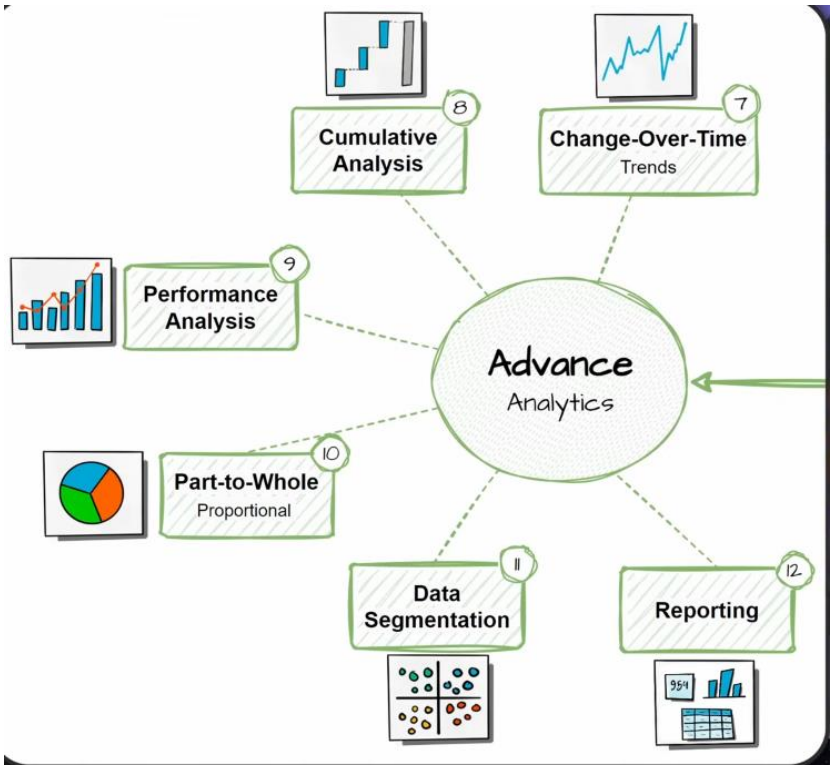





```sql
-- Which 5 products generate the highest revenue?


-- What are the 5 worst-performing products in terms of sales?
```

Advance Data Analytics (EDA)



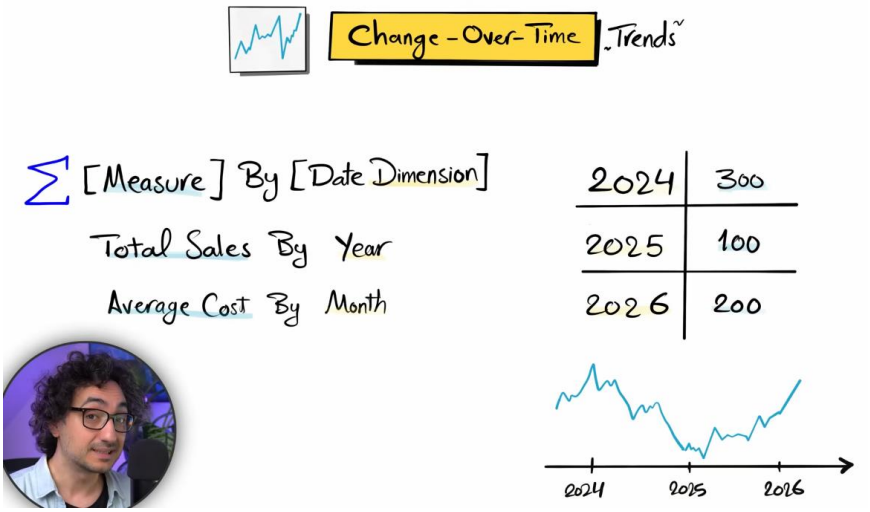


Advanced Analytics Project

Changes Over Time Analysis

Analyze how a measure evolves over time.

Helps track trends and identify seasonality in your data.




```sql
-- 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,
SUM(quantity) AS total_quantity,
COUNT(DISTINCT customer_key) AS total_customers,
COUNT(DISTINCT order_number) AS total_orders
FROM
gold.fact_sales_tb
WHERE order_date IS NOT NULL
GROUP BY YEAR(order_date),
         MONTH(order_date)
ORDER BY YEAR(order_date),
         MONTH(order_date)
```

Cumulative Analysis

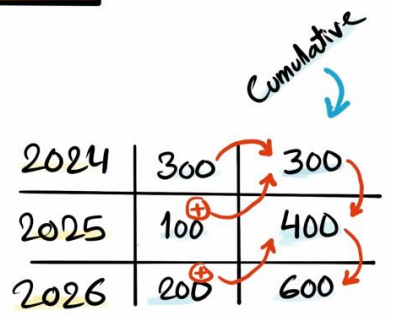
Aggregate the data progressively over time.

Helps to understand whether our business is growing or declining.

Σ [Cumulative Measure] By [Date Dimension]

Running Total Sales By Year

Moving Average of Sales By Month



WINDOW FUNCTIONS

```sql
-- Calculate the total sales per year
-- and the running total of sales over time
SELECT
order_year,
total_sales,
SUM(total_sales)    OVER (ORDER BY order_year) AS running_total_sales,
AVG(average_sales)  OVER (ORDER BY order_year) AS moving_average_sales
FROM
    ( SELECT
        DATETRUNC(YEAR,order_date) AS order_year,
        SUM(sales_amount) AS total_sales,
        AVG(sales_amount) AS average_sales
    FROM gold.fact_sales_tb
    WHERE order_date IS NOT NULL
    GROUP BY DATETRUNC(YEAR,order_date) ) t
```

Performance Analysis

Performance Analysis

Comparing the current value to a target value.

Helps measure success and compare performance.

Performance Analysis

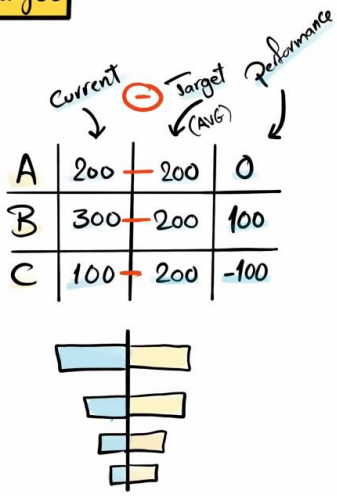
Current [Measure] – Target [Measure]

Current Sales – Average Sales

Current Year Sales – Previous Year Sales

Current Sales – lowest Sales

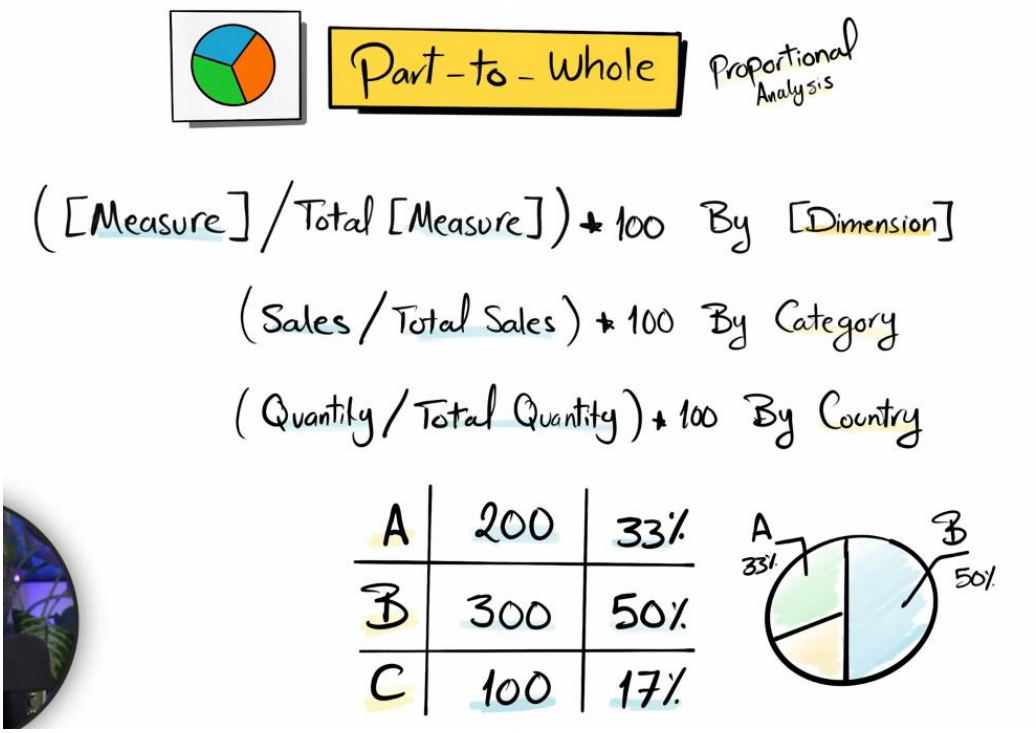
WINDOW FUNCTIONS




```sql
/* 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               AS product_name,
        SUM(f.sales_amount)          AS current_year_sales,
        COUNT(DISTINCT order_number) AS current_year_orders
    FROM
            gold.fact_sales_tb f
    LEFT JOIN gold.dim_products_tb 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_year_sales,
    current_year_orders,
    AVG(current_year_sales) OVER (PARTITION BY product_name) AS avg_sale_price,
    current_year_sales - AVG(current_year_sales) OVER(PARTITION BY product_name) AS diff_avg_price,
    CASE
        WHEN current_year_sales - AVG(current_year_sales) OVER(PARTITION BY product_name) > 0 THEN 'Above Avg'
        WHEN current_year_sales - AVG(current_year_sales) OVER(PARTITION BY product_name) < 0 THEN 'Below Avg'
        ELSE 'Avg'
    END AS avg_change,
    -- Year-over-Year Analysis
    LAG(current_year_sales) OVER (PARTITION BY product_name ORDER BY order_year) AS prev_yr_sales,
    current_year_sales - LAG(current_year_sales) OVER (PARTITION BY product_name ORDER BY order_year) AS diff_prev_yr_price,
    CASE
        WHEN current_year_sales - LAG(current_year_sales) OVER (PARTITION BY product_name ORDER BY order_year) > 0 THEN 'Increase'
        WHEN current_year_sales - LAG(current_year_sales) OVER (PARTITION BY product_name ORDER BY order_year) < 0 THEN 'Decrease'
        ELSE 'No Change'
    END AS prev_yr_change
FROM yearly_product_sales
ORDER BY product_name,
         order_year ;
```

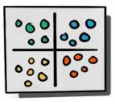
Part-To-Whole Analysis

Analyze how an individual part is performing compared to the overall, allowing us to understand which category has the greatest impact on the business.




```sql
-- Which categories contribute the most to overall sales?
WITH category_sales AS (
    SELECT
        p.category AS category,
        SUM(f.sales_amount) AS sales_amount
    FROM      gold.fact_sales_tb f
    LEFT JOIN gold.dim_products_tb p
    ON        f.product_key = p.product_key
    WHERE f.order_date IS NOT NULL
    GROUP BY p.category )
SELECT
    category,
    sales_amount,
    SUM(sales_amount) OVER() AS overall_sales,
    ROUND((CAST(sales_amount AS FLOAT)/ SUM(sales_amount) OVER())*100, 2 ) AS percentage_of_total
FROM category_sales
ORDER BY percentage_of_total DESC;
```

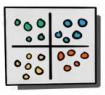
Data Segmentation



Data Segmentation

Group the data based on a specific range.

Helps understand the correlation between two measures.

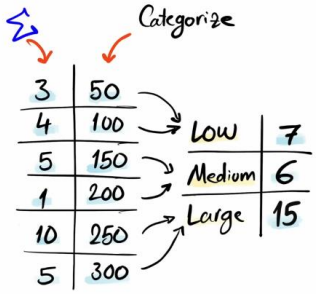


Data Segmentation

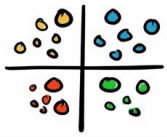
[Measure] By [Measure]

Total Products By Sales Range

Total Customers By Age

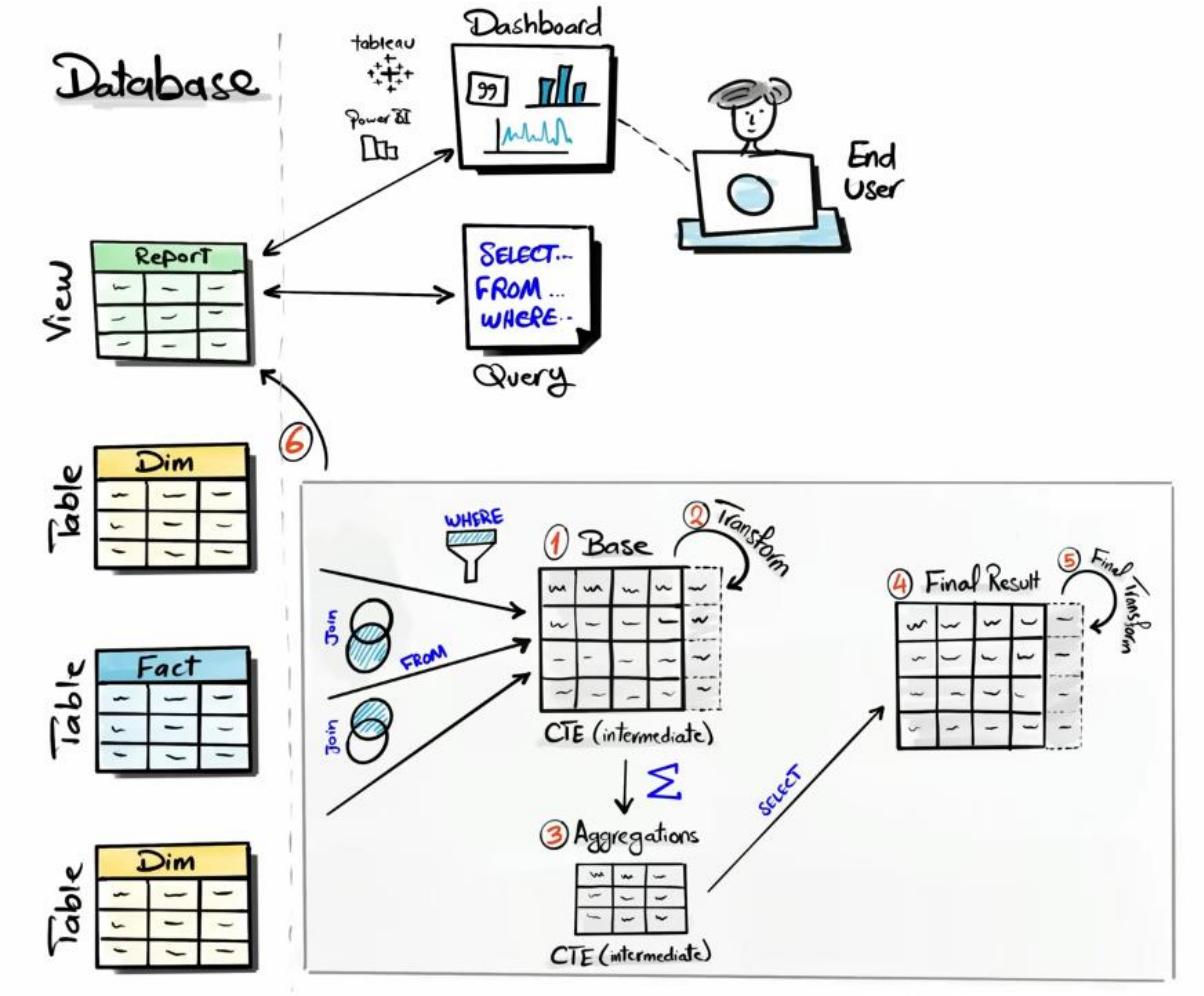


CASE WHEN STATEMENT



```sql
/*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
        customer_key,
        SUM(sales_amount)                                   AS total_spending,
        MIN(order_date)                                     AS min_order_date,
        MAX(order_date)                                     AS max_order_date,
        DATEDIFF(MONTH, MIN(order_date),MAX(order_date)) AS lifespan
    FROM
    gold.fact_sales_tb
    GROUP BY customer_key )
SELECT
customer_segment,
COUNT(DISTINCT customer_key) AS total_customers
FROM
    (SELECT
        customer_key,
        total_spending,
        min_order_date,
        max_order_date,
        lifespan,
        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 ) t
GROUP BY customer_segment
ORDER BY total_customers DESC ;
```

Build Customer 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
===============================================================================
*/
```

```sql
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,
        c.birthdate,
        DATEDIFF(YEAR, c.birthdate, GETDATE())          AS age
    FROM
              gold.fact_sales_tb f
    LEFT JOIN gold.dim_customers_tb c
    ON        f.customer_key = c.customer_key
    ),
    /*-----------------------------------------------------------------------------
    2) Customer Aggregations: Summarizes key metrics at the customer level
    -----------------------------------------------------------------------------*/
    /*-----------------------------------------------------------------------------
    2) Customer Aggregations: Summarizes key metrics at the customer level
    -----------------------------------------------------------------------------*/
        customer_aggregation AS
        (
            SELECT
                customer_key,
                customer_number,
                customer_name,
                age,
                COUNT(DISTINCT order_number)                    AS total_orders,
                SUM(sales_amount)                               AS total_sales,
                SUM(quantity)                                   AS total_quantity,
                COUNT(DISTINCT product_key)                     AS total_products,
                DATEDIFF(MONTH, MIN(order_date),MAX(order_date)) AS lifespan,
                MAX(order_date)                                 AS last_order_date
            FROM
            base_query
            GROUP BY
                customer_key,
                customer_number,
                customer_name,
                age
        )
    /*-----------------------------------------------------------------------------
    3) Calculations: Calculating valuable KPI's
    -----------------------------------------------------------------------------*/
```

```sql
/*-----------------------------------------------------------------------------
3) Calculations: Calculating valuable KPI's
-----------------------------------------------------------------------------*/
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,
    total_orders,
    total_sales,
    total_quantity,
    total_products
    lifespan,
    -- recency
    DATEDIFF(month, last_order_date, GETDATE()) AS recency,
    -- 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;
```

Build Product Report

```
/*
================================================================================
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
================================================================================
*/
```

```sql
USE DataWarehouse

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
            p.product_key,
            p.product_number,
            p.category,
            p.product_name,
            p.subcategory,
            p.cost,
            f.order_number,
            f.customer_key,
            f.sales_amount,
            f.quantity,
            f.order_date
        FROM
                  gold.fact_sales_tb f
        LEFT JOIN gold.dim_products_tb p
        ON        f.product_key = p.product_key
        WHERE f.order_date IS NOT NULL),
    /*-----------------------------------------------------------------------------
    2) Product Aggregations: Summarizes key metrics at the product level
    -----------------------------------------------------------------------------*/
```

```sql
/*---------------------------------------------------------------------------
2) Product Aggregations: Summarizes key metrics at the product level
---------------------------------------------------------------------------*/
        product_aggregations AS(
        SELECT
            product_key,
            product_number,
            category,
            product_name,
            subcategory,
            cost,
            COUNT(DISTINCT order_number)                                    AS total_orders,
            SUM(sales_amount)                                               AS total_sales,
            SUM(quantity)                                                   AS total_quantity
            COUNT(customer_key)                                             AS total_customers,
            MAX(order_date)                                                 AS last_order_date,
            DATEDIFF(MONTH, MIN(order_date), MAX(order_date))               AS lifespan,
            ROUND(AVG(CAST(sales_amount AS FLOAT) / NULLIF(quantity, 0)),1) AS avg_selling_price
        FROM base_query
        GROUP BY
            product_key,
            product_number,
            category,
            product_name,
            subcategory,
            cost)
/*---------------------------------------------------------------------------
3) Calculations: Calculating valuable KPI's
---------------------------------------------------------------------------*/
```

```sql
    /*---------------------------------------------------------------------------
    3) Calculations: Calculating valuable KPI's
    ---------------------------------------------------------------------------*/
    SELECT
        product_key,
        product_number,
        category,
        product_name,
        subcategory,
        cost
        total_orders,
        total_sales,
        avg_selling_price,
        total_quantity      ,
        total_customers,
        last_order_date,
        lifespan,
        -- recency
        DATEDIFF(MONTH, last_order_date, GETDATE() )  AS recency_in_months,
        -- product segment
        CASE
            WHEN total_sales > 50000  THEN 'High-Performer'
            WHEN total_sales >= 10000 THEN 'Mid-Range'
            ELSE 'Low-Performer'
        END AS product_segment,
        -- 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;

--Execution
SELECT * FROM gold.report_products
```