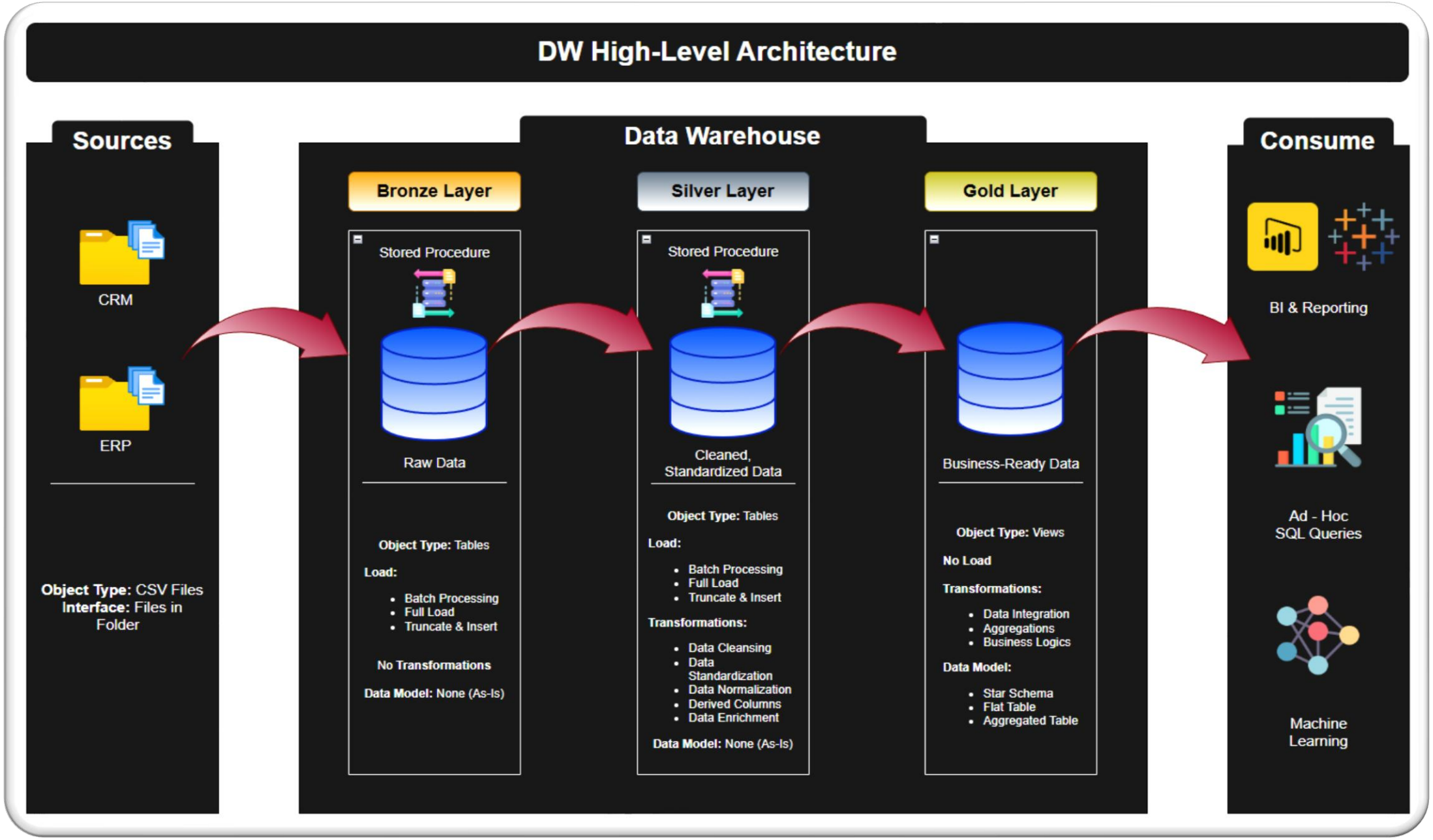
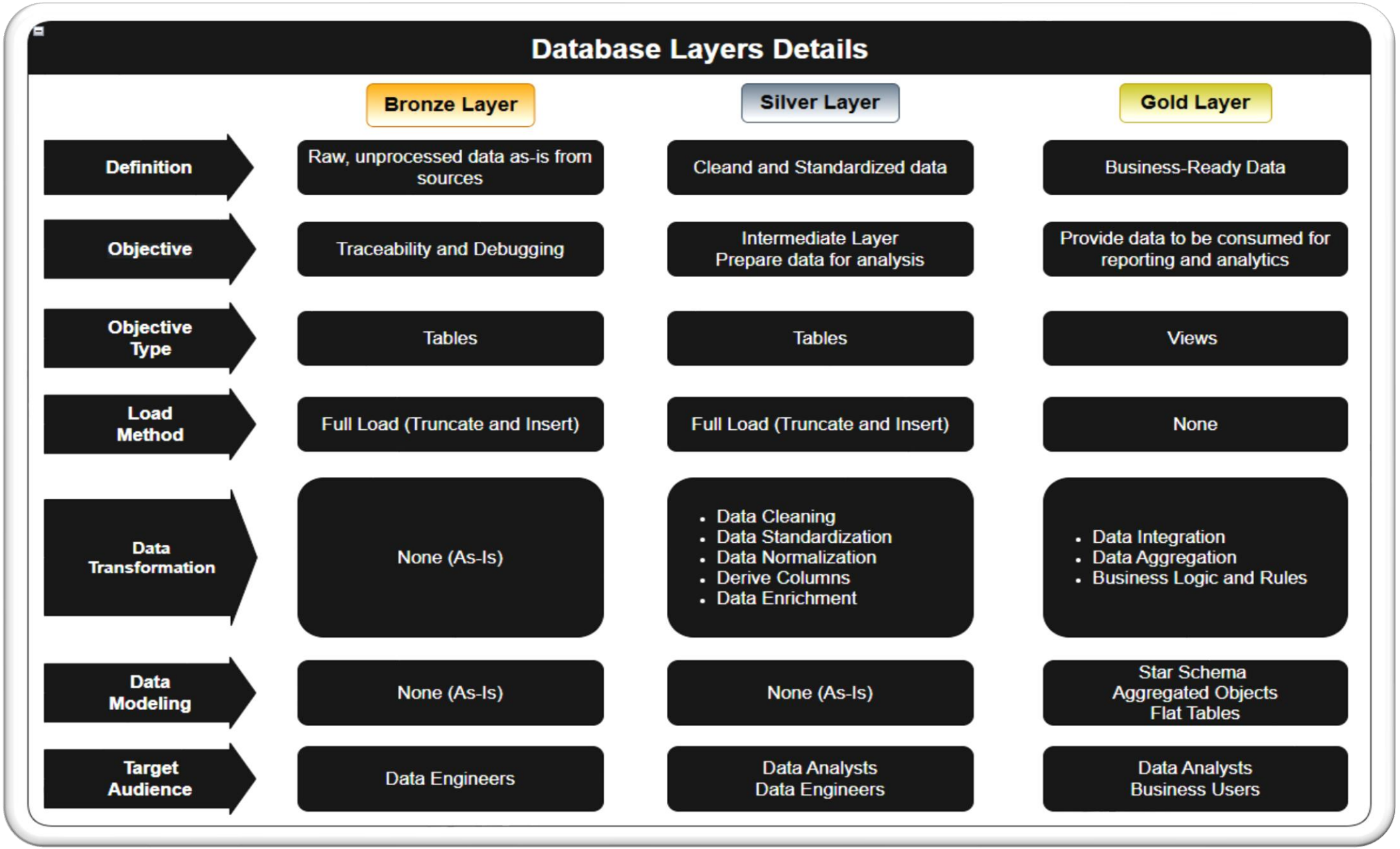
SQL Data Warehouse Project



https://www.linkedin.com/in/johncustodio/



https://www.linkedin.com/in/johncustodio/

Source System Interview

Business Context & Ownership

- Who owns the data?
- What business process it supports?
- System and data documentation
- Data model and data catalog

Architecture & Tech Stack

- How is data stored (SQL Server, Oracle, AWS, Azure...)?
- What are the integration capabilities?

Extract & Load

- Incremental Vs Full Load?
- Data Scope and historical needs
- What are the expected size of the extracts?
- Are there any data volume limitations?
- How to avoid impacting the source system's performance?
- Authentication and authorization (tokens, SSH keys, VPN, IP whitelisting)

https://www.linkedin.com/in/johncustodio/

Query Query History

```sql
-- ===============================================================
-- Stored Procedure: load_bronze_data
-- ===============================================================
-- Purpose:
--   Loads data from CSV files into the bronze layer tables.
--   - Truncates tables before load.
--   - Uses COPY for bulk insert from CSV.
--   - Logs progress and duration.
-- ===============================================================

CREATE OR REPLACE PROCEDURE bronze.load_bronze_data()
LANGUAGE plpgsql
AS $$
DECLARE
    start_time TIMESTAMP;
    end_time TIMESTAMP;
BEGIN
    RAISE NOTICE '================================================';
    RAISE NOTICE 'Starting Bronze Layer Data Load';
    RAISE NOTICE '================================================';

    -- CRM TABLES ------------------------------------------------
    RAISE NOTICE '------------------------------------------------';
    RAISE NOTICE 'Loading CRM Tables';
    RAISE NOTICE '------------------------------------------------';

    -- bronze.crm_cust_info
    start_time := clock_timestamp();
    RAISE NOTICE '>> Truncating bronze.crm_cust_info...';
    TRUNCATE TABLE bronze.crm_cust_info;
    RAISE NOTICE '>> Loading data from /data/csvs/cust_info.csv...';
    COPY bronze.crm_cust_info
    FROM '/data/csvs/cust_info.csv'
    WITH (FORMAT csv, HEADER true);
    end_time := clock_timestamp();
    RAISE NOTICE '>> Load duration: % seconds', EXTRACT(epoch FROM (end_time - start_time));

    -- bronze.crm_prd_info
    start_time := clock_timestamp();
    RAISE NOTICE '>> Truncating bronze.crm_prd_info...';
    TRUNCATE TABLE bronze.crm_prd_info;
    RAISE NOTICE '>> Loading data from /data/csvs/prd_info.csv...';
    COPY bronze.crm_prd_info
    FROM '/data/csvs/prd_info.csv'
    WITH (FORMAT csv, HEADER true);
    end_time := clock_timestamp();
    RAISE NOTICE '>> Load duration: % seconds', EXTRACT(epoch FROM (end_time - start_time));

    -- bronze.crm_sales_details
```

Query Query History

```sql
DROP TABLE IF EXISTS silver.crm_cust_info;

CREATE TABLE silver.crm_cust_info (
    cst_id              INTEGER,
    cst_key             VARCHAR(50),
    cst_firstname       VARCHAR(50),
    cst_lastname        VARCHAR(50),
    cst_marital_status VARCHAR(50),
    cst_gndr            VARCHAR(50),
    cst_create_date     DATE,
    dwh_create_date     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Drop and create silver.crm_prd_info
DROP TABLE IF EXISTS silver.crm_prd_info;

CREATE TABLE silver.crm_prd_info (
    prd_id          INTEGER,
    cat_id          VARCHAR(50),
    prd_key         VARCHAR(50),
    prd_nm          VARCHAR(50),
    prd_cost        INTEGER,
    prd_line        VARCHAR(50),
    prd_start_dt    DATE,
    prd_end_dt      DATE,
    dwh_create_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 3. Drop and create silver.crm_sales_details
DROP TABLE IF EXISTS silver.crm_sales_details;

CREATE TABLE silver.crm_sales_details (
    sls_ord_num     VARCHAR(50),
    sls_prd_key     VARCHAR(50),
    sls_cust_id     INTEGER,
    sls_order_dt    DATE,
    sls_ship_dt     DATE,
    sls_due_dt      DATE,
    sls_sales       INTEGER,
    sls_quantity    INTEGER,
    sls_price       INTEGER,
    dwh_create_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 4. Drop and create silver.erp_loc_a101
DROP TABLE IF EXISTS silver.erp_loc_a101;

CREATE TABLE silver.erp_loc_a101 (
    cid             VARCHAR(50),
```

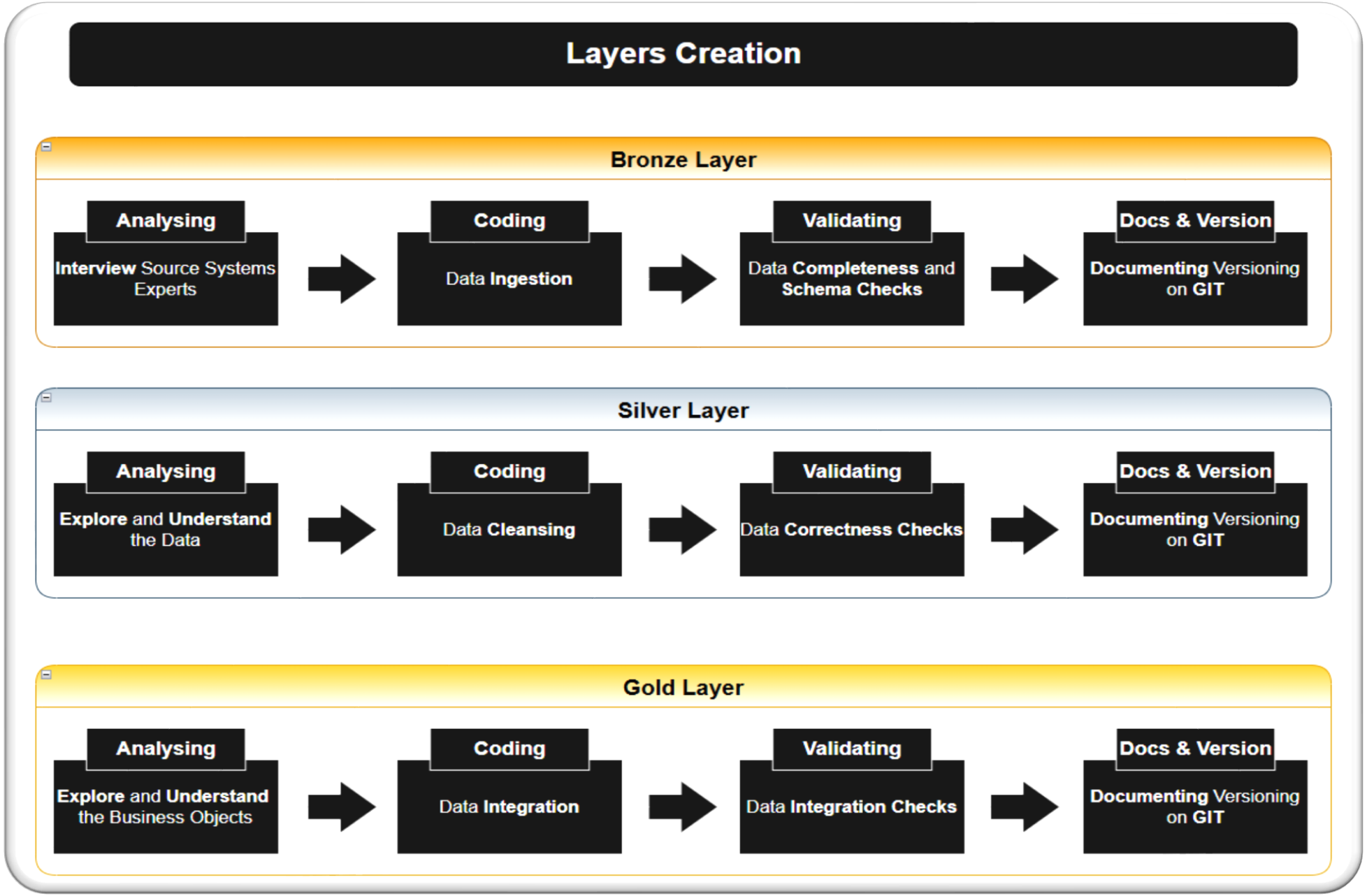
https://www.linkedin.com/in/johncustodio/

Query Query History

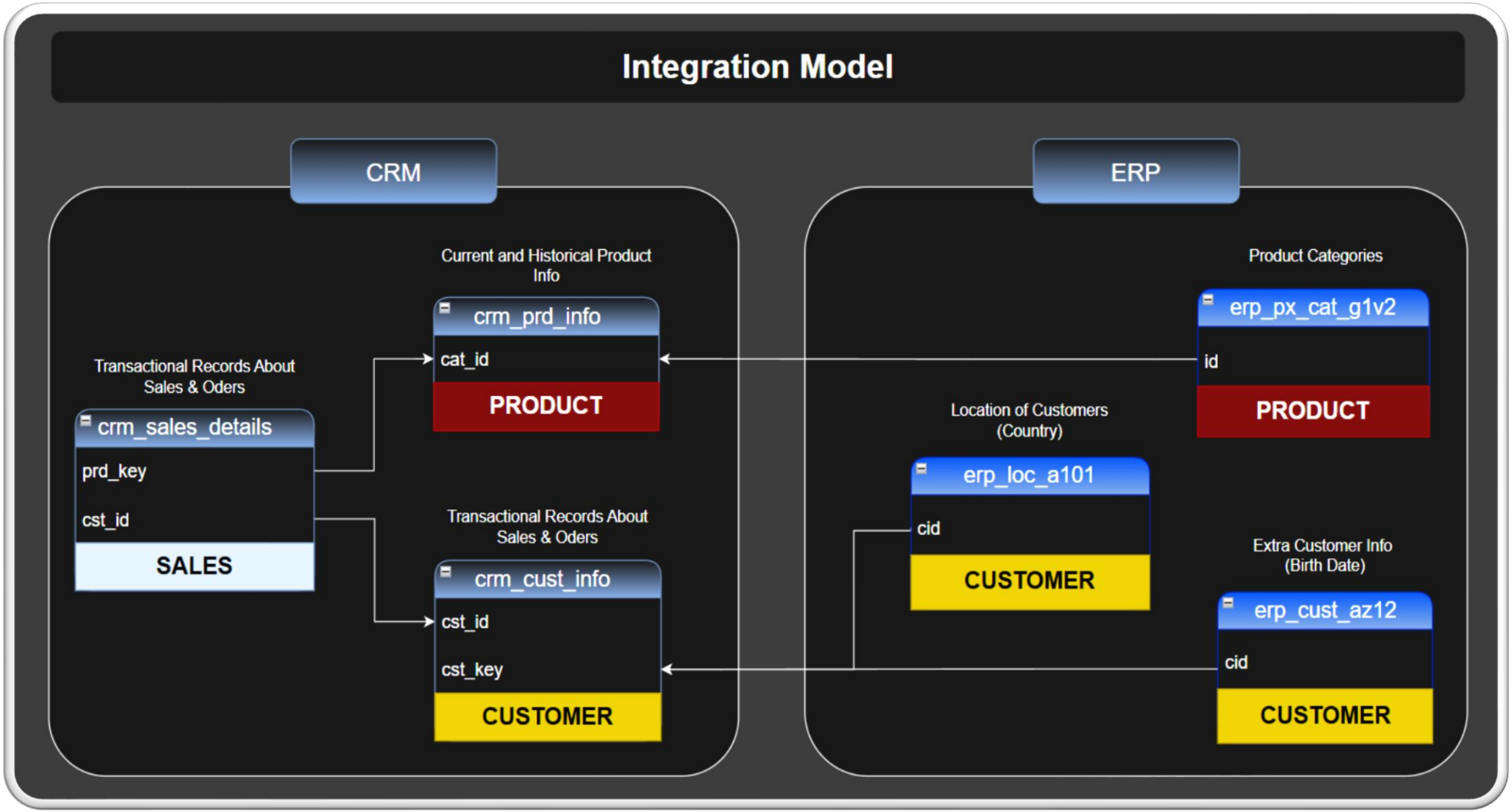
```sql
CREATE OR REPLACE PROCEDURE silver.load_silver()
LANGUAGE plpgsql
AS $$
DECLARE
    start_time TIMESTAMP;
    end_time TIMESTAMP;
    batch_start_time TIMESTAMP;
    batch_end_time TIMESTAMP;
BEGIN
    batch_start_time := NOW();
    RAISE NOTICE '================================================';
    RAISE NOTICE 'Loading Silver Layer';
    RAISE NOTICE '================================================';

    RAISE NOTICE '------------------------------------------------';
    RAISE NOTICE 'Loading CRM Tables';
    RAISE NOTICE '------------------------------------------------';

    -- 1. Loading silver.crm_cust_info
    start_time := NOW();
    RAISE NOTICE '>> Truncating Table: silver.crm_cust_info';
    TRUNCATE TABLE silver.crm_cust_info;
    RAISE NOTICE '>> Inserting Data Into: silver.crm_cust_info';
    INSERT INTO silver.crm_cust_info (
        cst_id,
        cst_key,
        cst_firstname,
        cst_lastname,
        cst_marital_status,
        cst_gndr,
        cst_create_date
    )
    SELECT
        cst_id,
        cst_key,
        TRIM(cst_firstname) AS cst_firstname,
        TRIM(cst_lastname) AS cst_lastname,
        CASE
            WHEN UPPER(TRIM(cst_marital_status)) = 'S' THEN 'Single'
            WHEN UPPER(TRIM(cst_marital_status)) = 'M' THEN 'Married'
            ELSE 'n/a'
        END AS cst_marital_status, -- Normalize marital status values to readable format
        CASE
            WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
            WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
            ELSE 'n/a'
        END AS cst_gndr, -- Normalize gender values to readable format
        cst_create_date
    FROM (
```

Query Query History

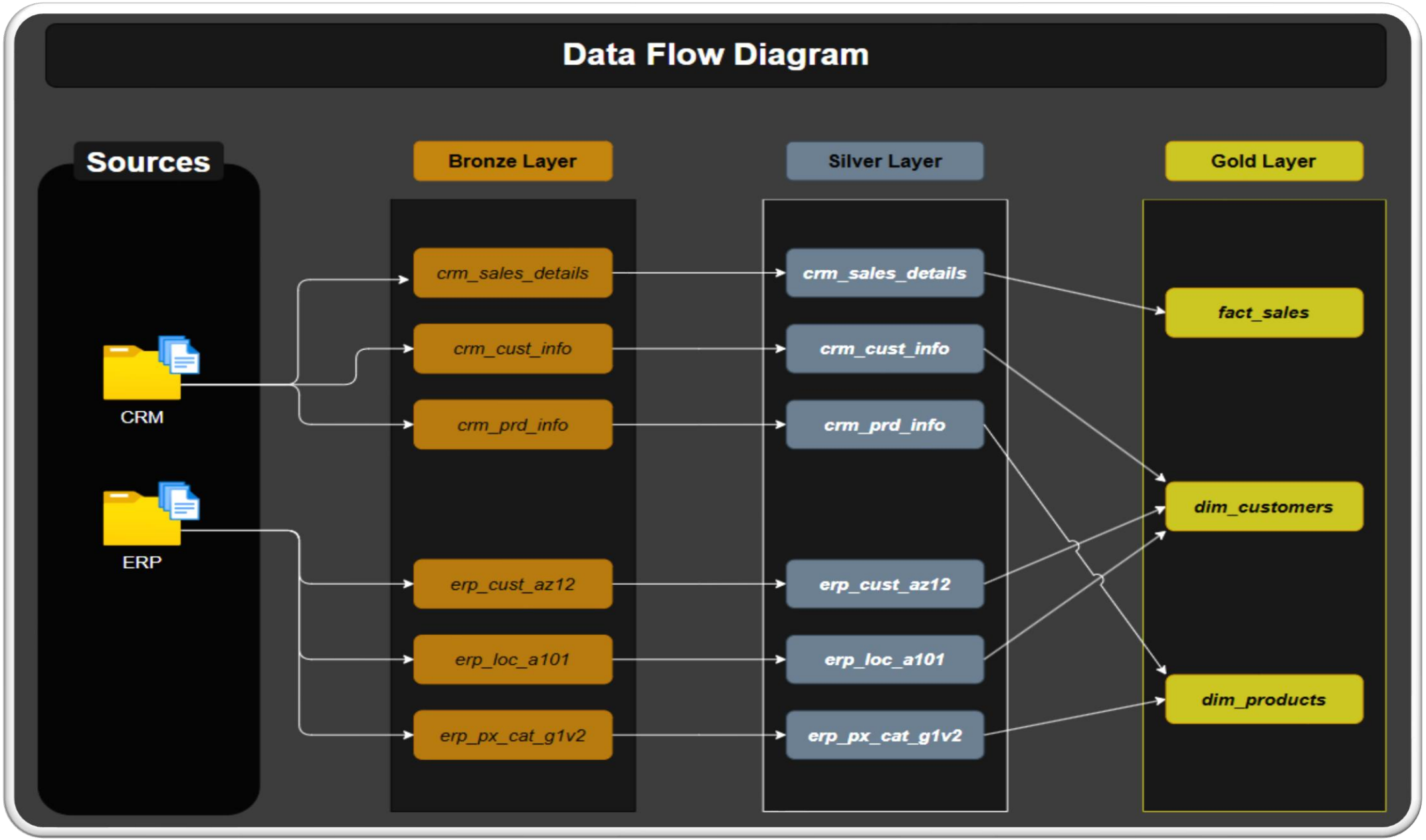
```sql
-- 1. CREATE DIMENSION: gold.dim_customers.
-- ====================================================================

DROP VIEW IF EXISTS gold.dim_customers;

CREATE VIEW gold.dim_customers AS
    SELECT
        ROW_NUMBER() OVER (ORDER BY cst_id) AS customer_key, -- Surrogate key
        ci.cst_id AS customer_id,
        ci.cst_key AS customer_number,
        ci.cst_firstname AS first_name,
        ci.cst_lastname AS last_name,
        la.cntry AS country,
        ci.cst_marital_status AS marital_status,
        CASE
            WHEN ci.cst_gndr != 'n/a' THEN ci.cst_gndr
            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 AS ci
    LEFT JOIN silver.erp_cust_az12 AS ca
    ON ci.cst_key = ca.cid
    LEFT JOIN silver.erp_loc_a101 AS la
    ON ci.cst_key = la.cid;

-- ====================================================================
-- 2. CREATE DIMENSION: gold.dim_products.
-- ====================================================================

DROP VIEW IF EXISTS gold.dim_products;

CREATE VIEW gold.dim_products AS
    SELECT
        ROW_NUMBER() OVER (ORDER BY pn.prd_start_dt, pn.prd_key) AS product_key, -- Surrogate key
        pn.prd_id AS product_id,
        pn.prd_key AS product_number,
        pn.prd_nm AS product_name,
        pn.cat_id AS category_id,
        pc.cat AS category,
        pc.subcat AS subcategory,
        pc.maintenance,
        pn.prd_cost AS cost,
        pn.prd_line AS product_line,
        pn.prd_start_dt AS start_date
    FROM silver.crm_prd_info AS pn
    LEFT JOIN silver.erp_px_cat_g1v2 AS pc
    ON pn.cat_id = pc.id
    WHERE pn.prd_end_dt IS NULL; -- Filter out all historical data
```



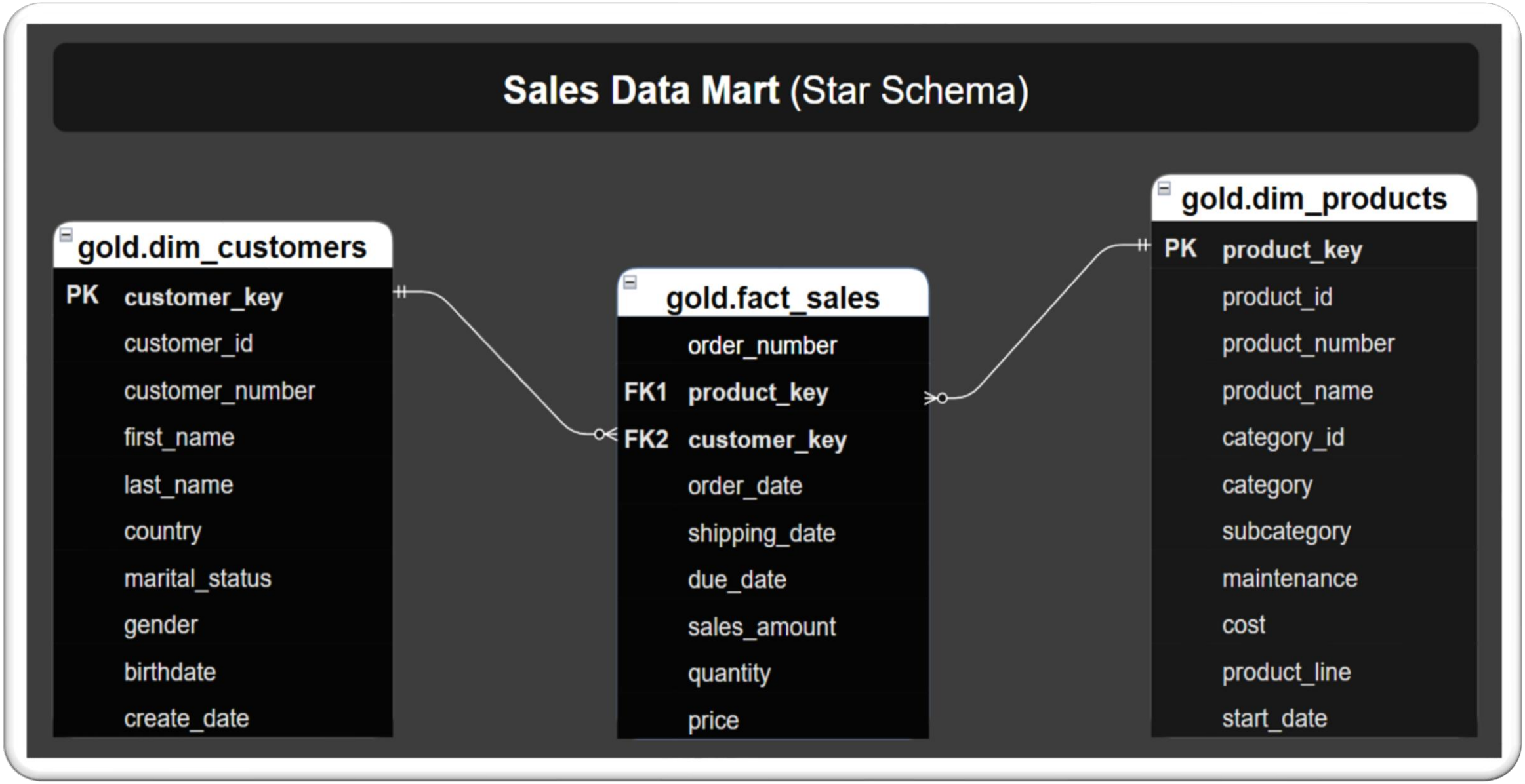
https://www.linkedin.com/in/johncustodio/



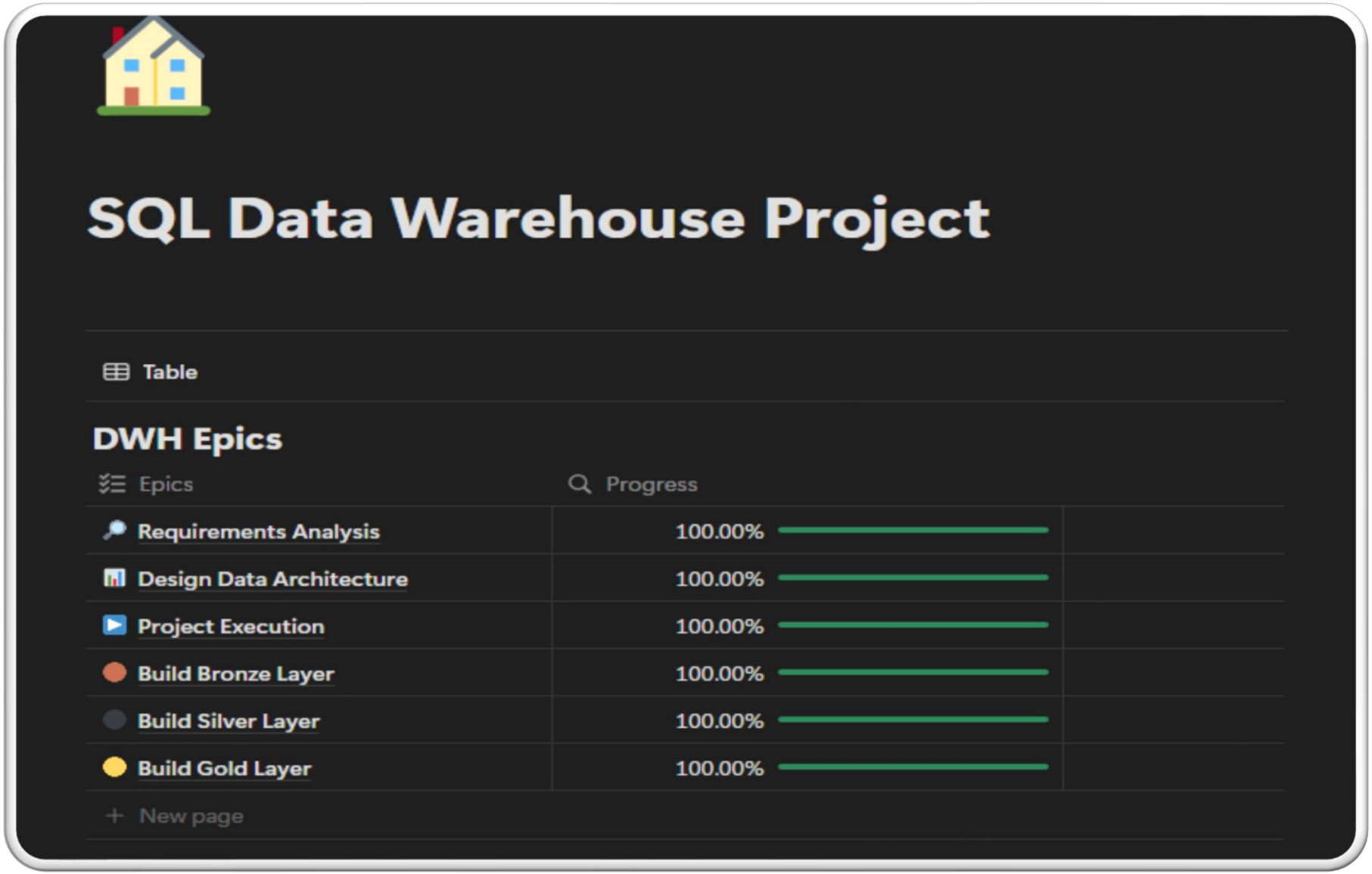
https://www.linkedin.com/in/johncustodio/



https://www.linkedin.com/in/johncustodio/



https://www.linkedin.com/in/johncustodio/



https://www.linkedin.com/in/johncustodio/

SQL Data Warehouse Project

---

1. Introduction

This project aims to build a modern Data Warehouse using SQL, incorporating ETL processes, data modeling, and analytics. It serves as a portfolio project, showcasing industry best practices in data engineering and analytics. The main goal is to centralize, standardize, and integrate data from multiple systems into a single repository, enabling reliable analysis and management reporting. Clear objectives and structured methodologies were defined to guide the work, ensuring organization and traceability.

- **Data Warehouse**
    - What is a Data Warehouse?
      A centralized and structured repository that integrates data from multiple sources. Its main purpose is to provide a single, consistent view to support business analysis and decision-making.

- **ETL**
    - What is ETL (Extract, Transform, Load)?
      A process that extracts data from different systems, transforms it into a consistent format, and loads it into the Data Warehouse.

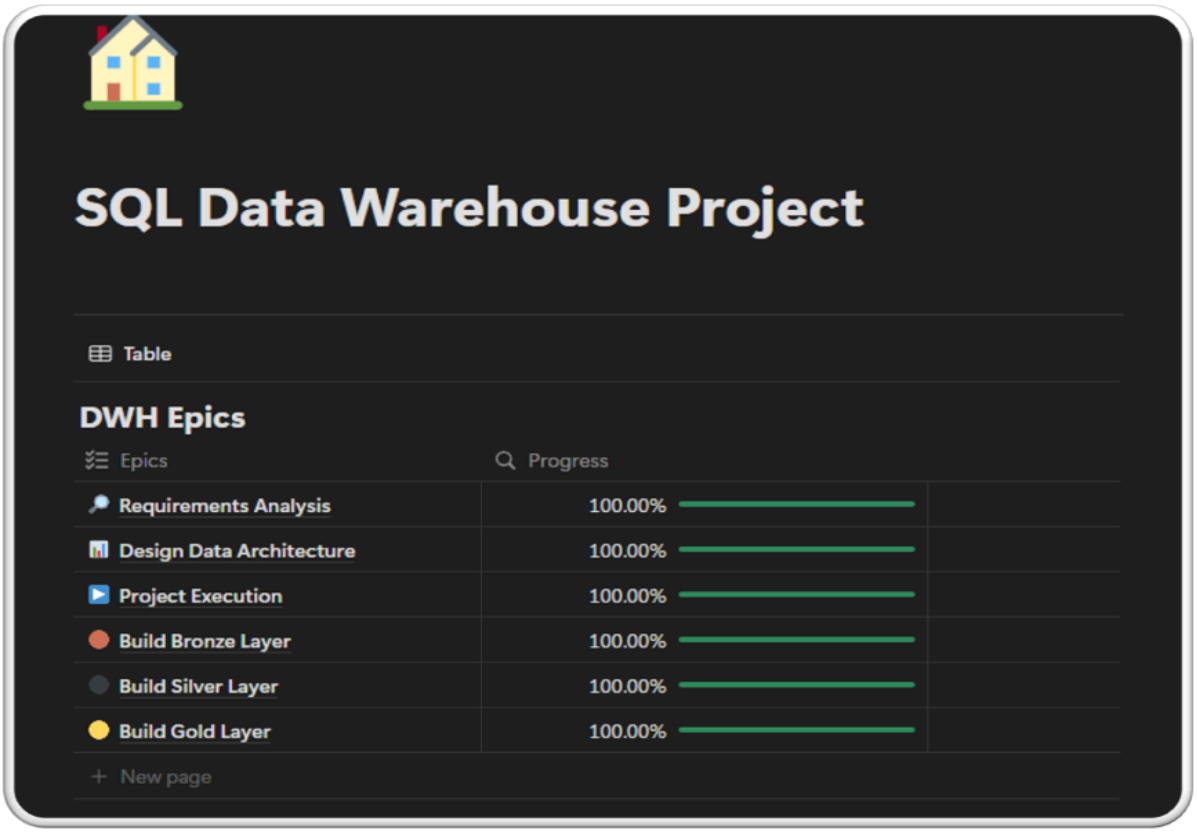
---

2. Initiation and Planning

We started by gathering all project requirements and defining the best approach to handle the complexities involved. We used Notion as a planning tool, documenting each step and creating a clear roadmap for the next phases. We also set up a GitHub repository for version control and documentation, and we aligned the architecture and essential conventions for building the Data Warehouse.

- **Requirements Gathering**
  Defined the project scope and business needs.

- **Project Plan (Notion)**
  Centralized all tasks, timelines, and responsibilities in a single platform.



- **Layered Approach**
  Defined the layers of the DW: Bronze (raw data), Silver (cleaned and standardized data), and Gold (business model).

- **GIT Repository**
  Created for version control and documentation:
  - https://github.com/JohnCustodio/sql-data-warehouse-project

- **Architecture and Data Modeling**
  We designed the high-level architecture of the DW in Draw.io and defined the data model for each layer.

- **Naming Conventions**
  Established consistent naming rules for schemas, tables, views, and columns.

- **Database and Schemas**
  It was structured the PostgreSQL environment with separated schemas for each layer,

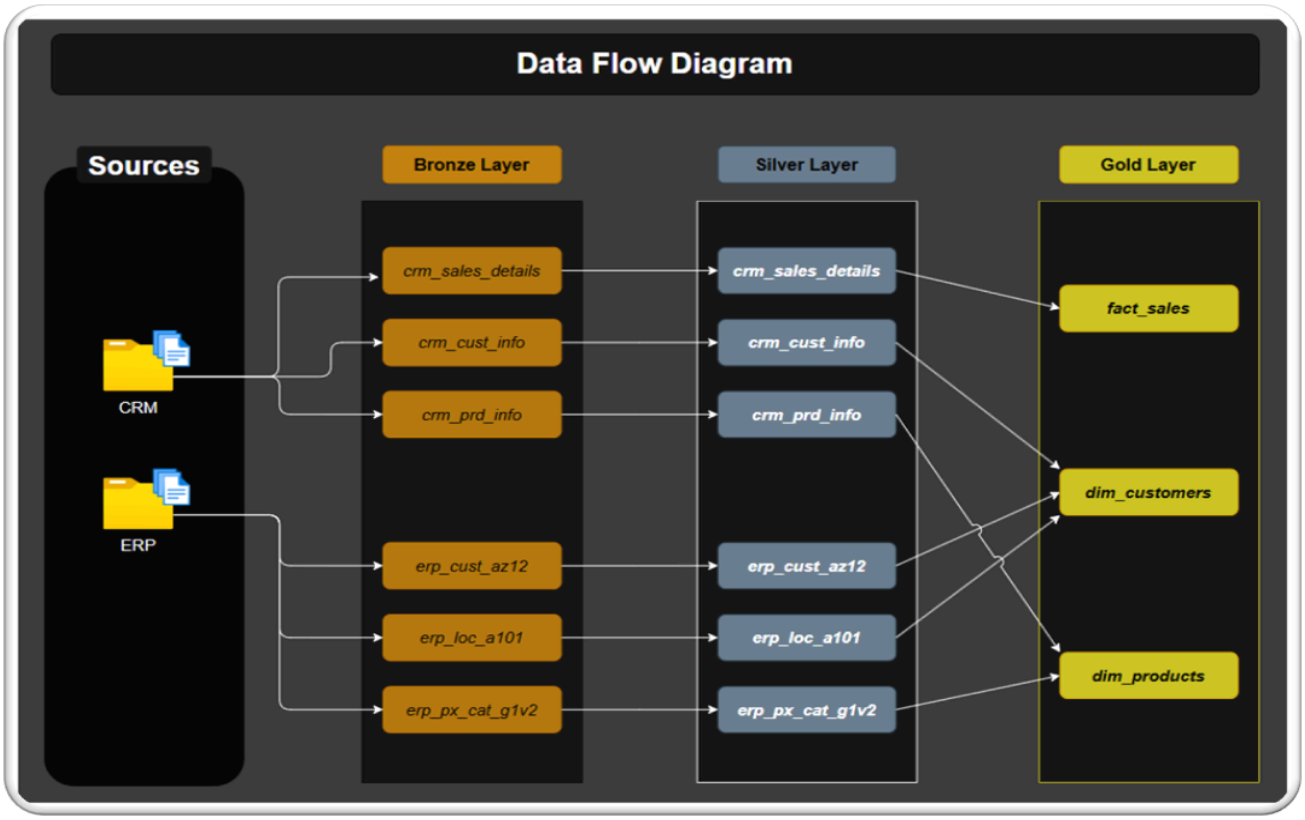
running the server in a Docker container, and accessing/manipulating data via PgAdmin.

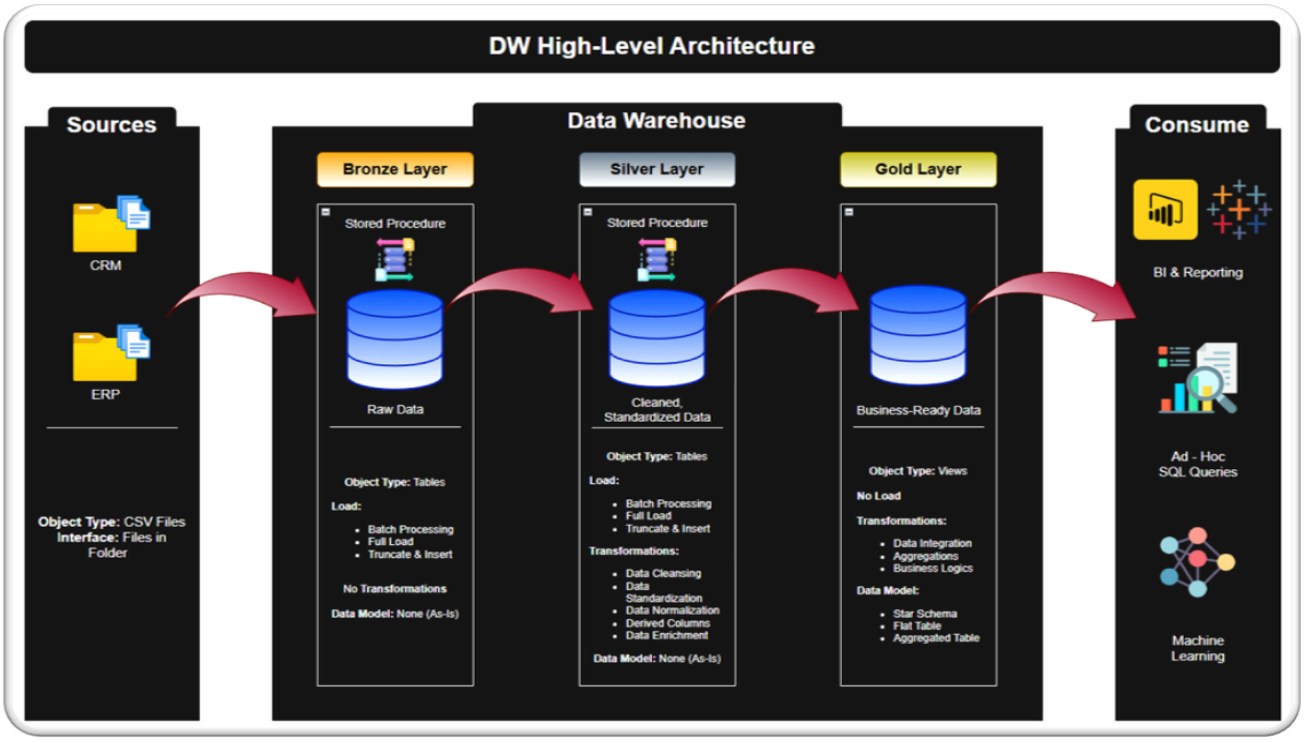
---

3. Execution

The execution phase was organized into three main steps, each representing a DW layer. We followed an iterative and incremental approach, allowing for quick adjustments and partial deliveries that were continuously validated.

- **Bronze Layer**
    - Analysis of the data sources and creation of corresponding tables in PostgreSQL.
    - SQL scripts for initial data load, without transformations.
    - Creation of stored procedures to automate data loads.
    - Documented the data flow for this layer.
- **Silver Layer**
    - Understanding the data and creating the Silver tables.
    - Cleaning and standardizing data from CRM and ERP sources, including normalization, derivations, and enrichment.
    - Refined load scripts to remove inconsistencies.
    - Stored procedures for repetitive processes.
    - Documented the data flow for the Silver layer.
- **Gold Layer**
    - Applied data modeling concepts to meet business needs.
    - Created dimension tables (dim_customers, dim_products) and fact table (fact_sales).
    - Built the Star Schema model to facilitate analysis and reporting.
    - Created a Data Catalog and documented the final data flow.


Data Flow Diagram
Sources
Bronze Layer
Silver Layer
Gold Layer
CRM
ERP
crm_sales_details
crm_cust_info
crm_prd_info
erp_cust_az12
erp_loc_a101
erp_px_cat_g1v2
fact_sales
dim_customers
dim_products


DW High-Level Architecture
Sources
CRM
ERP
Object Type: CSV Files
Interface: Files in Folder
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 Standardization
Data Normalization
Derived Columns
Data Enrichment
Data Model: None (As-Is)
Gold Layer
Business-Ready Data
Object Type: Views
No Load
Transformations:
Data Integration
Aggregations
Business Logics
Data Model:
Star Schema
Flat Table
Aggregated Table
Consume
BI & Reporting
Ad - Hoc SQL Queries
Machine Learning

---

4. Project Closure

We completed the project with thorough validations and reviews to ensure data quality and usability of the DW. All materials were documented in Notion and versioned in GitHub, forming a complete guide for future maintenance and enhancements.

- **Final Validation**
  Checked data integrity and consistency by running SQL scripts for validation at each layer.

- **Complete Documentation**
  All steps and deliverables were documented in Notion, and all scripts and documents were versioned in Git.

- **Next Steps**
  This Data Warehouse will serve as a foundation for future exploratory and advanced data analysis projects.

Portuguese Version

Projeto de Data Warehouse SQL

---

1. Introdução

Este projeto tem como objetivo construir um Data Warehouse moderno utilizando SQL, incorporando processos de ETL, modelagem de dados e análises. Ele serve como um projeto de portfólio, demonstrando as melhores práticas de engenharia e análise de dados. O foco está em centralizar, padronizar e integrar dados de múltiplos sistemas em um único repositório, facilitando análises confiáveis e relatórios gerenciais. Foram definidos objetivos claros e metodologias que guiaram o trabalho, garantindo organização e rastreabilidade.

- **Data Warehouse**

    - O que é um Data Warehouse?
      Um repositório centralizado e estruturado de dados integrados de várias fontes. Seu objetivo é oferecer uma visão única e consistente para apoiar análises e decisões de negócio.

- **ETL**

    - O que é ETL (Extract, Transform, Load)?
      Um processo que extrai dados de diferentes sistemas, transforma-os em formatos consistentes e carrega-os no Data Warehouse.

---

2. Iniciação e Planejamento

Iniciamos com um levantamento completo dos requisitos do projeto e definimos a abordagem ideal para lidar com as complexidades envolvidas. Utilizamos o Notion como ferramenta de planejamento, registrando cada etapa e garantindo um caminho claro para as próximas fases. Também criamos um repositório no GitHub para controle de versões e documentação, e alinhamos a arquitetura e convenções essenciais para a criação do Data Warehouse.

- **Levantamento de Requisitos**
  Definição do escopo e das necessidades do negócio.

- **Plano de Projeto (Notion)**
  Organização de tarefas, cronogramas e responsáveis em um só lugar.

- **Abordagem de Camadas**
  Definição das camadas do DW: Bronze (dados brutos), Silver (dados limpos e ajustados) e Gold (modelo de negócio).

- **Repositório GIT**
  Criado para versionamento e documentação:
    - https://github.com/JohnCustodio/sql-data-warehouse-project

- **Arquitetura e Modelagem**
  Desenhamos a arquitetura de alto nível do DW no Draw.io e definimos a modelagem de dados para cada camada.

- **Naming Conventions**
  Regras de nomenclatura para schemas, tabelas, views e colunas.

- **Banco de Dados e Schemas**
  Estruturamos o PostgreSQL com schemas separados para cada camada, rodando o servidor em um container Docker, e acessando/manipulando os dados via PgAdmin.

---

3. Execução

A fase de execução foi dividida em três grandes etapas, cada uma representando uma camada do DW. Adotamos um desenvolvimento iterativo e incremental, permitindo ajustes rápidos e entregas parciais que foram validadas continuamente.

- **Camada Bronze**
    - Análise das fontes de dados e criação de tabelas correspondentes no PostgreSQL.
    - Scripts SQL para carga inicial, sem transformações.
    - Criação de Stored Procedures para automatizar as cargas.
    - Documentação do fluxo de dados dessa camada.
- **Camada Silver**
    - Entendimento dos dados e criação de tabelas Silver.
    - Limpeza e padronização dos dados das fontes CRM e ERP, incluindo normalização, derivações e enriquecimento.
    - Scripts de carga refinados para remover inconsistências.
    - Stored Procedures para processos repetitivos.
    - Documentação do fluxo de dados Silver.
- **Camada Gold**

- Aplicação de conceitos de Data Modeling para atender às necessidades de negócio.
- Criação de tabelas de dimensão (dim_customers, dim_products) e de fato (fact_sales).
- Construção do modelo Star Schema para facilitar análises e relatórios.
- Criação de um Catálogo de Dados e documentação do fluxo final.

---

4. Encerramento

Finalizamos com validações e revisões para garantir a qualidade dos dados e a usabilidade do DW. Todo o material gerado foi documentado via Notion e versionado no GitHub, formando um guia completo para futuras manutenções e evoluções.

- **Validação Final**
  Conferimos a integridade e consistência dos dados, executando scripts SQL para validação em cada camada.

- **Documentação Completa**
  Tudo registrado no Notion e scripts versionados no GIT.

- **Próximos Passos**
  Este Data Warehouse servirá como base para futuros projetos de Análise Exploratória e Avançada de Dados.
