

CONSTRUÇÃO DE UMA DATA WAREHOUSE NO POSTGRESQL A PARTIR DO BANCO OLTP ADVENTUREWORKS (SQL SERVER) USANDO PYTHON E AUTOMAÇÕES COM AIRFLOW

Autor: João Victor Berger Aguiar

Instituição: Centro Universitário Salesiano – UNISALES

Curso: Sistemas de Informação

Orientador: James Alves

Data: 25/11/2025

RESUMO

Este artigo apresenta o desenvolvimento de um projeto voltado à construção de um Data Warehouse utilizando a base de dados AdventureWorks, fornecida pela Microsoft. O estudo fez uso de diferentes ferramentas tecnológicas, cada uma aplicada conforme seu propósito específico no processo. Os dados foram extraídos, tratados e posteriormente inseridos em um banco de dados PostgreSQL, seguindo princípios de modelagem multidimensional para análise. O objetivo central foi demonstrar o uso de ferramentas de automação para executar, de forma sistemática e com mínima intervenção humana, as etapas necessárias do processo ETL (Extract, Transform, Load).

Palavras-Chave: Data Warehouse; Airflow; PostgreSQL; SQLServer; ETL

ABSTRACT

This article presents the development of a project focused on building a Data Warehouse using the AdventureWorks database, provided by Microsoft. The study employed different technological tools, each applied according to its specific purpose within the workflow. The data was extracted, processed, and subsequently loaded into a PostgreSQL database, following multidimensional modeling principles for analytical purposes. The main objective was to demonstrate the use of automation tools to systematically execute the ETL (Extract, Transform, Load) process with minimal human intervention.

Keywords: Data Warehouse; Airflow; PostgreSQL; SQLServer; ELT

1. INTRODUÇÃO

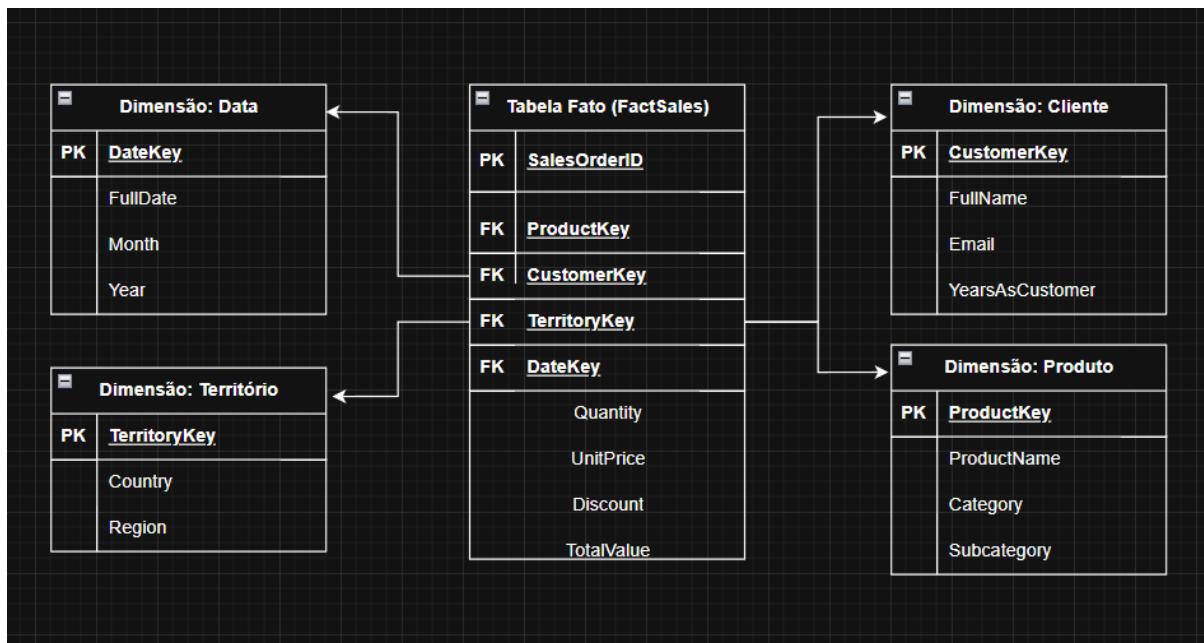
O projeto teve como objetivo demonstrar o processo de criação de uma Data Warehouse utilizando o banco de dados AdventureWorks, disponibilizado pela Microsoft

O projeto abrange a restauração do banco OLTP, desenvolvimento de um modelo multidimensional, construção de ETL, carga dos dados no PostgreSQL e a implementação de indicadores de negócio.

2. DESENVOLVIMENTO

2.1. MODELO MULTIDIMENSIONAL

Figura 1 – Modelo Multidimensional proposto



2.2. DICIONÁRIO DE DADOS DO MODELO MULTIDIMENSIONAL

Dimensão: Data

1. FullDate: Data completa
2. Month: Mês
3. Year: Ano

Dimensão: Território

1. Country: País da compra
2. Region: Região da compra

Dimensão: Cliente

1. FullName: Nome completo do cliente

2. Email: Email do cliente

3. YearsAsCustomer: Anos como cliente

Dimensão: Produto

1. ProductName: Nome do produto

2. Category: Categoria

3. Subcategory: SubCategoria

Tabela Fato: FactSales

1. SalesOrderID: Identificador de cada compra

2. ProductKey: Chave identificadora do produto

3. CustomerKey: Chave identificadora do cliente

4. TerritoryKey: Chave identificadora do território

5. DataKey: Chave identificadora da data

6. Quantity: Quantidade do produto para determinado ID

7. UnitPrice: Preço relativo ao determinado ID

8. Discount: Desconto aplicado para a venda com determinado ID

9. TotalValue: Valor total para a venda com determinado ID

2.3. DESCRIÇÃO DO PROCESSO ETL

O processo de ETL (Extract, Transform, Load) foi desenvolvido em Python e executado de forma automatizada por meio do Apache Airflow. Para a implementação, foram utilizadas as bibliotecas **pyodbc**, **pandas** e **SQLAlchemy**, cada uma desempenhando uma função específica no fluxo de dados. O objetivo do processo foi extrair informações do banco de dados transacional AdventureWorks, tratá-las e carregá-las em um Data Warehouse estruturado em PostgreSQL, seguindo os princípios da modelagem multidimensional.

O fluxo foi estruturado em três etapas principais:

- Extração:

Na fase de extração, foi estabelecida uma conexão com o banco SQL Server que hospeda o ambiente OLTP do AdventureWorks, utilizando a biblioteca

pyodbc. Por meio dessa conexão, foram executadas consultas SQL para acessar as tabelas relevantes ao modelo multidimensional, como informações de produtos, clientes e transações de venda. Os dados retornados foram convertidos para estruturas manipuláveis em Python.

- Transformação:

Após a extração, os dados passaram pela etapa de transformação, realizada com o apoio da biblioteca pandas. Nessa fase, foram aplicados procedimentos de limpeza, padronização, normalização e integração dos dados. Entre as transformações realizadas estão:

- junções entre tabelas para enriquecer o conjunto analítico;
- tratamento de valores ausentes e inconsistências;
- criação de chaves substitutas (surrogate keys);
- derivação de novos atributos relevantes para o modelo analítico.

Essa etapa assegurou a conformidade dos dados com o modelo dimensional adotado.

- Carga:

Na etapa final, os dados transformados foram gravados no PostgreSQL utilizando a biblioteca SQLAlchemy, que permitiu a interação com o banco no formato ORM e facilitou operações de inserção e atualização. As tabelas de dimensões foram carregadas inicialmente, seguidas pelas tabelas fato. A carga foi projetada para rodar de forma incremental, reduzindo retrabalho e garantindo maior eficiência na manutenção do Data Warehouse.

 *Link para o projeto no GitHub: <https://github.com/Berger212/AirflowETL>*

2.4. DAGS

Figura 2 - DAG “mestre”

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows the project structure under "AIRFLOW-DOCKER". The "mestre.py" file is selected and highlighted in blue.
- Code Editor:** Displays the Python code for the "mestre.py" DAG. The code defines a DAG named "pipeline_master" with a daily schedule, starting at 2025-01-01. It uses a TriggerDagRunOperator to trigger other DAGs sequentially: "run_extract", "run_transform", "run_metrics", "run_indicators", and "run_validation".

```
from airflow import DAG
from datetime import datetime
from airflow.operators.trigger_dagrun import TriggerDagRunOperator

with DAG(
    "pipeline_master",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False
):
    extract = TriggerDagRunOperator(task_id="run_extract", trigger_dag_id="extract_mysql")
    transform = TriggerDagRunOperator(task_id="run_transform", trigger_dag_id="transform_core_dv")
    metrics = TriggerDagRunOperator(task_id="run_metrics", trigger_dag_id="load_metrics")
    indicators = TriggerDagRunOperator(task_id="run_indicators", trigger_dag_id="update_business_indicators")
    validation = TriggerDagRunOperator(task_id="run_validation", trigger_dag_id="validation_reporting")

    extract >> transform >> metrics >> indicators >> validation
```
- Terminal:** Shows a PowerShell terminal window with the command "PS C:\Users\Uberg\Documents\airflow-docker>".
- Status Bar:** Provides information about the current session, including the agent, pick model, and memory usage.

A DAG denominada pipeline_master tem como finalidade atuar como fluxo de orquestração central do processo de ETL. Diferentemente das demais DAGs, que executam tarefas específicas, esta DAG é responsável pela coordenação da execução sequencial.

A execução ocorre diariamente, conforme definido no parâmetro “schedule”. O parâmetro “catchup=false” assegura que execuções retroativas não sejam disparadas caso o serviço seja reiniciado.

O controle das etapas é feito por meio do operador “TriggerDagRunOperator”, que permite que a DAG principal acione outras DAGs registradas no Airflow.

Figura 3 - DAG atualiza

The screenshot shows the VS Code interface with the following details:

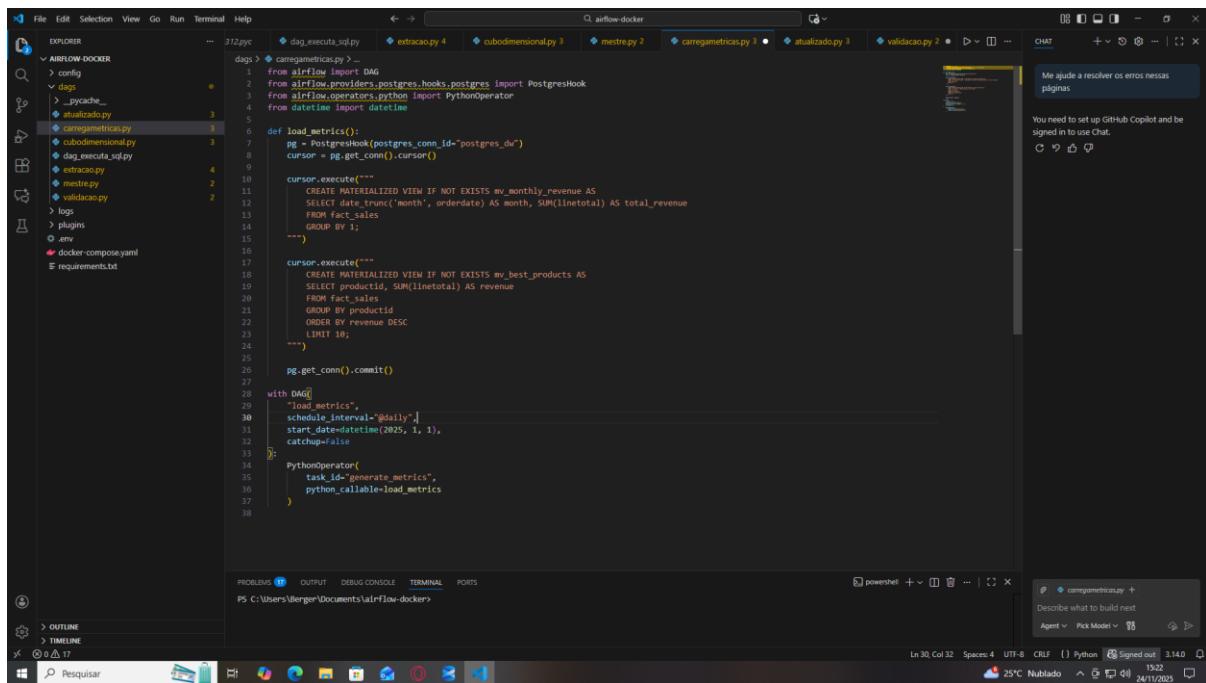
- File Explorer:** Shows files like `atualizado.py`, `dag_executes.sql.py`, `extração.py`, `cubimensional.py`, `metreto.py`, `carregamétricas.py`, `atualizado.py`, `validação.py`, `logs`, `plugins`, `.env`, `docker-compose.yaml`, and `requirements.txt`.
- Code Editor:** Displays the `atualizado.py` file content. The code defines a DAG `atualizado` that runs weekly. It uses `PostgresHook` to create materialized views for `mv_ticket_medio`, `mv_ltv`, and `mv_churn`. It then processes these views using a `PythonOperator` to update business indicators.
- Terminal:** Shows the command `PS C:\Users\Bergen\Documents\airflow-docker>`.
- Status Bar:** Shows the current file path as `PS C:\Users\Bergen\Documents\airflow-docker>`, and other status information like `Line 34, Col 24`, `Spaces 4`, `UTF-8`, `CRLF`, `Python`, and `Signed out`.

A DAG update_business_indicators é responsável pela atualização periódica de indicadores estratégicos, garantindo que as métricas estejam sempre atualizadas.

A execução ocorre semanalmente e utiliza a conexão com o Data Warehouse em PostgreSQL para a criação de materialized views, que servem como estruturas para leitura e análises recorrentes.

Foram utilizados Airflow para orquestração da execução, PostgresHook para conexão com o banco de dados, PythonOperator para execução do processamento em Python e PostgreSQL como destino dos dados.

Figura 4 - DAG carrega



```
File Edit Selection View Go Run Terminal Help airflow-docker
EXPLORER ... 3/2/pyc dag_executa_sql.py extracao.py cubodimensional.py mestre.py carregametrics.py atualizado.py validacao.py
AIRFLOW-DOCKER
> config
> dag
> _pycache_ ...
> carregametrics.py
> cubodimensional.py
> dag_executa_sql.py
> extracao.py
> mestre.py
> validacao.py
logs
plugins
.env
docker-compose.yaml
requirements.txt

dag_executa_sql.py > ...
1 from airflow import DAG
2 from airflow.providers.postgres.hooks.postgres import PostgresHook
3 from airflow.operators.python import PythonOperator
4 from datetime import datetime
5
6 def load_metrics():
7     pg = PostgresHook(postgres_conn_id="postgres_dv")
8     cursor = pg.get_conn().cursor()
9
10    cursor.execute("""
11        CREATE MATERIALIZED VIEW IF NOT EXISTS mv_monthly_revenue AS
12            SELECT date_trunc('month', orderdate) AS month, SUM(linetotal) AS total_revenue
13            FROM fact_sales
14            GROUP BY 1;
15    """)
16
17    cursor.execute("""
18        CREATE MATERIALIZED VIEW IF NOT EXISTS mv_best_products AS
19            SELECT productid, SUM(linetotal) AS revenue
20            FROM fact_sales
21            GROUP BY productid
22            ORDER BY revenue DESC
23            LIMIT 10;
24    """)
25
26    pg.get_conn().commit()
27
28    with DAG(
29        "load_metrics",
30        schedule_interval="@daily",
31        start_date=datetime(2025, 1, 1),
32        catchup=False
33    ):
34        PythonOperator(
35            task_id="generate_metrics",
36            python_callable=load_metrics
37        )
38

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\lbergen\Documents\airflow-docker>
powerhell + v ... x
+ compagmetrcs.py +
Describe what to build next
Agent v Pick Model v
Ln 30 Col 32 Spaces:4 UTF-8 CRLF Python Signed out 3.14.0 Q
25°C Nublado ⌂ 19:22 24/11/2025
```

A DAG `load_metrics` tem por objetivo calcular as métricas operacionais essenciais para o monitoramento diário da performance. Ela utiliza consultas SQL para garantir desempenho adequado.

A execução ocorre diariamente, garantindo métricas atualizadas.

Figura 5 - DAG cubodimensional

```

File Edit Selection View Go Run Terminal Help Q airflow_docker
EXPLORER
  AIRFLOW_DOCKER
    > config
    > logs
    > pycache_
    > utilsairflow.py
    > cubodimensional.py
    > dag_executa_sql.py
    > extracto.py
    > mestre.py
    > validacao.py
    > logs
    > plugins
    > .env
    docker-compose.yaml
  requirements.txt

372.pyc dag_executa_sql.py extracto.py cubodimensional.py mestre.py 2 carregemetrcas.py 3 atualizado.py 3 validacao.py 2
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from datetime import datetime

def transform_load_dw():
    pg = PostgresHook(postgres_conn_id="postgres_dw")
    conn = pg.get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS dim_customer AS
        SELECT customerid, personid, territoryid
        FROM customer_raw
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS fact_sales AS
        SELECT
            h.salesorderid,
            h.customerid,
            d.productid,
            h.orderdate,
            d.quantity,
            d.unitprice,
            d.linetotal
        FROM salesorderdetail_raw d
        JOIN salesorderheader_raw h ON d.salesorderid = h.salesorderid;
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS dim_product AS
        SELECT
            p.productid,
            p.name AS category
        FROM product_raw p
        LEFT JOIN productsubcategory_raw s ON p.productssubcategoryid = s.productssubcategoryid
        LEFT JOIN productcategory_raw c ON s.productcategoryid = c.productcategoryid;
    """)

    conn.commit()

with DAG(
    "transform_core_dw",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False
):
    task = PythonOperator(
        task_id="transform_dw",
        python_callable=transform_load_dw
)

```

A DAG `transform_core_dw` é responsável pela etapa de transformação dos dados e pela carga final no Data Warehouse seguindo a modelagem multidimensional proposta. Ela atua como o núcleo do pipeline.

O objetivo principal desta DAG é criar e atualizar a camada dimensional no PostgreSQL.

Figura 6 - DAG extração

The screenshot shows the Airflow UI with the DAG 'extract_mysql' selected. The top navigation bar includes 'Search Dogs', 'Trigger', 'Run Backfill', and 'Repose Dog'. The DAG details show it was last run on 2025-11-24, 21:00:00, and the next run is scheduled for 2025-11-25, 21:00:00. The DAG is owned by 'airflow' and has the tag 'v1'. The 'Code' tab is active, displaying the following Python code:

```

from airflow import DAG
from airflow.providers.mysql.hooks.mysql import MySqlHook
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from datetime import datetime

TABLES = [
    "SalesOrderHeader",
    "SalesOrderDetail",
    "Customer",
    "Product",
    "ProductCategory",
    "ProductSubCategory"
]

def extract_table(table):
    mysql = MySqlHook(mysql_conn_id="mysql_adventureworks")
    df = mysql.get_pandas_df(f"SELECT * FROM {table}")
    df.to_sql(table.lower() + "_raw", postgres.get_sqlalchemy_engine(), if_exists="replace", index=False)

def extract_all():
    for table in TABLES:
        extract_table(table)

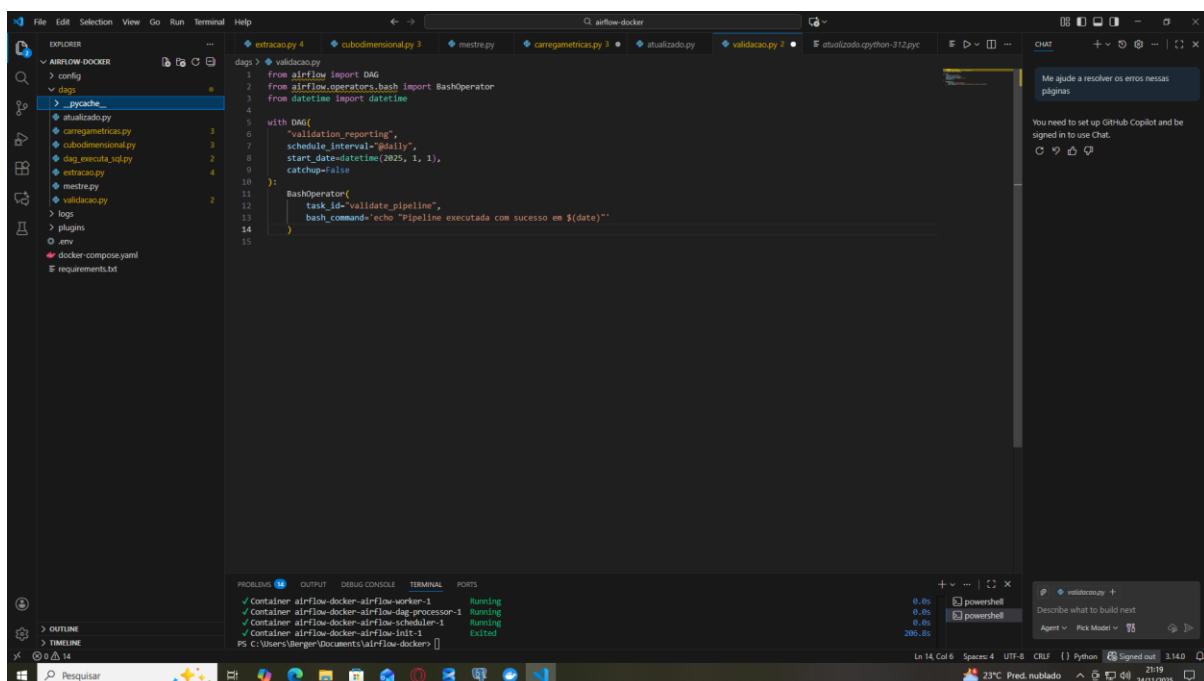
with DAG(
    "extract_mysql",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False
):
    task = PythonOperator(
        task_id="extract_data",
        python_callable=extract_all
)

```

A DAG extract_mysql corresponde à primeira etapa do pipeline ETL, sendo responsável para extração dos dados brutos diretamente do banco AdventureWorks. O objetivo é coletar as tabelas necessárias para o processo analítico, padronizá-las e armazená-las.

Foram usadas o Apache Airflow para controlar a execução, MySqlHook e PostgresHook para comunicação entre os bancos, Pandas para manipulação tabular e SQLAlchemy para gravar dados no PostgreSQL.

Figura 7 - DAG extração



The screenshot shows a code editor interface with several tabs open. The active tab is 'validacao.py' containing the following Python code:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

with DAG(
    'validacao',
    validation_interval='@daily',
    start_date=datetime(2025, 1, 1),
    catchup=False
):
    BashOperator(
        task_id='validate_pipeline',
        bash_command='echo "Pipeline executada com sucesso em $(date)"'
    )
```

The code editor has a dark theme. On the left, there's an Explorer sidebar showing a directory structure for 'AIRFLOW DOCKER' with files like 'atualizado.py', 'carregametricas.py', 'cubodimensional.py', 'dag_executa_sql.py', 'extractao.py', 'mestre.py', and 'validacao.py'. Below the code editor is a terminal window showing Docker container status:

```
PS C:\Users\lberger\Documents\airflow-docker> [REDACTED]
```

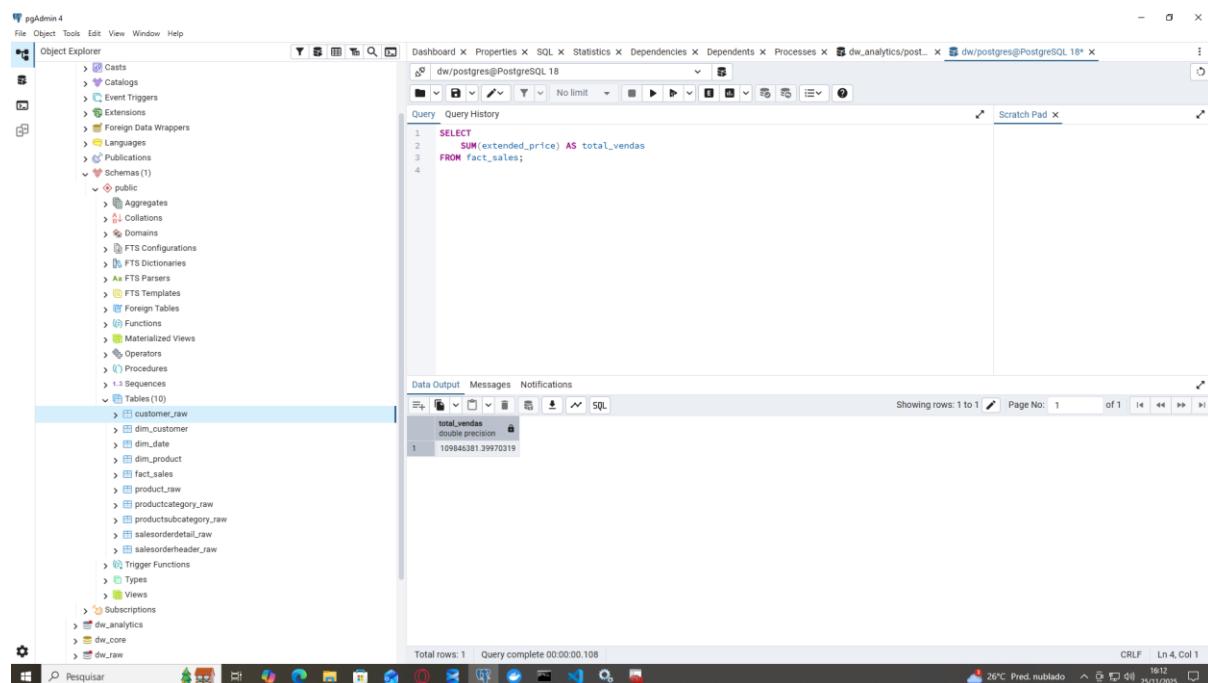
At the bottom, there's a navigation bar with icons for file operations, search, and other common functions.

A DAG validation_reporting executa a etapa final do pipeline de processamento de dados, tendo como objetivo validar a execução das fases anteriores do processo ETL.

Apache Airflow foi utilizado como orquestrador de fluxo e BashOperator para execução de comandos de sistema operacional

2.3. DESCRIÇÃO DO PROCESSO ETL

Figura 8 - SQL Total de vendas



The screenshot shows the pgAdmin 4 interface. On the left is the Object Explorer pane, which lists various database objects like Schemas, Tables, and Views. The main area is the Query Editor, where a simple SQL SELECT statement is written to calculate the total sales:

```
1 SELECT
2     SUM(extended_price) AS total_vendas
3 FROM fact_sales;
4
```

The Data Output tab shows the result of the query:

total_vendas
109846381.39970319

Below the table, it says "Showing rows: 1 to 1" and "Page No: 1 of 1". At the bottom of the pgAdmin window, there's a status bar with system information like "CRLF Ln 4, Col 1", "26°C Pred. nublado", and a date/time stamp "29/1/2025".

SQL demonstrando a query para obter o total de vendas.

Figura 9 - SQL Ticket médio

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with various database objects like Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, and Schemas. The Schemas section shows a public schema with tables such as customer_raw, dim_customer, dim_date, dim_product, fact_sales, product_raw, productcategory_raw, productsubcategory_raw, salesorderdetail_raw, salesorderheader_raw, Trigger Functions, Types, Views, Subscriptions, dw_analytics, dw_core, and dw_raw. The main window shows a query editor with the following SQL code:

```
1 SELECT
2     AVG(extended_price) AS ticket_medio
3 FROM fact_sales;
```

The Data Output tab shows the result of the query:

ticket_medio
905.4492066215221

Below the table, it says "Showing rows: 1 to 1" and "Page No: 1". The status bar at the bottom right indicates "Successfully run. Total CRLF Ln 4, Col 1" and "alerta de tempestade 16:13 25/11/2025".

SQL demonstrando a query para obter o ticket médio.

Figura 10 - SQL Faturamento mensal

The screenshot shows the pgAdmin 4 interface, similar to Figure 9. The left sidebar shows the Object Explorer with the same database objects. The main window shows a query editor with the following SQL code:

```
1 SELECT
2     DATE_TRUNC('month', order_date) AS mes,
3     SUM(extended_price) AS total_mensal
4 FROM fact_sales
5 GROUP BY DATE_TRUNC('month', order_date)
6 ORDER BY mes;
```

The Data Output tab shows the results of the query:

mes	total_mensal
2011-05-01 00:00:00	502805.9149000003
2011-06-01 00:00:00	459910.8248000003
2011-07-01 00:00:00	2044600.0033379968
2011-08-01 00:00:00	249516.73346001
2011-09-01 00:00:00	502073.8458000007
2011-10-01 00:00:00	4588761.81613
2011-11-01 00:00:00	737839.8214000015
2011-12-01 00:00:00	1309863.2511400015
2012-01-01 00:00:00	3970627.2789579835
2012-02-01 00:00:00	1475426.9099799981
2012-03-01 00:00:00	2973748.3384279976
2012-04-01 00:00:00	1634600.798319974
2012-05-01 00:00:00	3074600.815380037

Below the table, it says "Showing rows: 1 to 38" and "Page No: 1". The status bar at the bottom right indicates "NVD34 0,77% 16:15 25/11/2025".

SQL demonstrando a query para obter o faturamento mensal.

Figura 11 - SQL Quantidade total de itens vendidos

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with various database objects like Schemas, Tables, and Views. The main pane shows a query editor with the following SQL code:

```
1 SELECT
2     SUM(order_quantity) AS items_vendidos
3 FROM fact_sales;
```

The Data Output tab shows the result of the query:

items_vendidos
274914

Below the table, the status bar indicates "Total rows: 1 Query complete 00:00:00.165". A message bar at the bottom right says "Successfully run. Total query runtime: 165 msec. 1 rows affected."

SQL demonstrando a query para obter a quantidade total de itens vendidos.

Figura 12 - SQL Número total de pedidos

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with various database objects like Schemas, Tables, and Views. The main pane shows a query editor with the following SQL code:

```
1 SELECT
2     COUNT(DISTINCT sales_key) AS total_pedidos
3 FROM fact_sales;
```

The Data Output tab shows the result of the query:

total_pedidos
31465

Below the table, the status bar indicates "Total rows: 1 Query complete 00:00:00.141". A message bar at the bottom right says "Successfully run. Total query runtime: 141 msec. 1 rows affected."

SQL demonstrando a query para obter o número total de pedidos.

Figura 13 - SQL Média de itens por pedido

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled 'Object Explorer' and lists various database objects under 'Tables (10)', with 'customer_raw' selected. The main area is a query editor with the following SQL code:

```

SELECT
    AVG(order_quantity) AS media_itens_por_pedido
FROM fact_sales;

```

The results pane shows a single row of data:

media_itens_por_pedido
2.2660797744751354

Below the results, a message bar indicates: "Successfully run. Total query runtime: 98 msec. 1 rows affected." The status bar at the bottom right shows "Chuva esta noite" and the date "23/11/2025".

SQL demonstrando a query para obter a média de itens por pedido.

Figura 14 - SQL Top 10 clientes por valor

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled 'Object Explorer' and lists various database objects under 'Tables (10)', with 'customer_raw' selected. The main area is a query editor with the following SQL code:

```

SELECT
    customer_key,
    SUM(exended_price) AS valor_total
FROM fact_sales
GROUP BY customer_key
ORDER BY valor_total DESC
LIMIT 10;

```

The results pane shows the top 10 clients with their total values:

customer_key	valor_total
29818	877107.192220999
29715	853849.1795239995
29722	841908.770709998
30117	816755.5742759995
29614	799277.8950619998
29639	787777.0437679994
29790	748317.5292599995
29617	740985.833741999
29798	730798.7139139994
272722	6493670004

Below the results, a message bar indicates: "Successfully run. Total query runtime: 00:00:00.172." The status bar at the bottom right shows "IBOVESPA +0,33%" and the date "23/11/2025".

SQL demonstrando a query para obter o top 10 clientes por valor gasto.

Figura 15 - SQL Cliente com maior ticket médio

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with various database objects like Schemas, Tables, and Views. The main area shows a SQL query in the Query Editor:

```

1 SELECT
2     customer_key,
3     AVG(ticket_price) AS ticket_medio
4 FROM fact_sales
5 GROUP BY customer_key
6 ORDER BY ticket_medio DESC
7 LIMIT 1;

```

The Data Output tab shows the results of the query:

customer_key	ticket_medio
30111	7277.010606578947

Total rows: 1 Query complete 00:00:00.128

SQL demonstrando a query para obter o cliente com o maior ticket médio.

Figura 16 - SQL Top 10 produtos mais vendidos

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with various database objects like Schemas, Tables, and Views. The main area shows a SQL query in the Query Editor:

```

1 SELECT
2     product_key,
3     SUM(order_quantity) AS quantidade_vendida
4 FROM fact_sales
5 GROUP BY product_key
6 ORDER BY quantidade_vendida DESC
7 LIMIT 10;

```

The Data Output tab shows the results of the query:

product_key	quantidade_vendida
712	8311
670	6815
711	6743
715	6592
708	6532
707	6266
864	4247
873	3865
884	3864
714	3636

Total rows: 10 Query complete 00:00:00.112

SQL demonstrando a query para obter os 10 produtos mais vendidos.

Figura 17 - SQL Top 10 produtos mais vendidos (por quantidade)

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Tables (10)', the 'customer_raw' table is selected. The main pane displays a SQL query:

```

1 SELECT
2     product_key,
3     sum(quantity) AS quantidade_vendida
4 FROM fact_sales
5 GROUP BY product_key
6 ORDER BY quantidade_vendida DESC
7 LIMIT 10;
8

```

The results are shown in a table:

product_key	quantidade_vendida
712	8311
870	6815
711	6743
715	6592
708	6532
707	6266
864	4247
873	3865
884	3864
714	3636

Total rows: 10 Query complete 00:00:00.119

SQL demonstrando a query para obter os 10 produtos mais vendidos por quantidade.

Figura 18 - SQL Top 10 produtos por faturamento

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Tables (10)', the 'customer_raw' table is selected. The main pane displays a SQL query:

```

1 SELECT
2     product_key,
3     SUM(extended_price) AS valor_vendas
4 FROM fact_sales
5 GROUP BY product_key
6 ORDER BY valor_vendas DESC
7 LIMIT 10;
8

```

The results are shown in a table:

product_key	valor_vendas
782	440592.800400055
783	400494.761841063
779	369576.0252720583
780	343478.8604230457
781	3434256.941912805
784	3309673.216908046
793	2515857.314918006
794	234765.953454014
795	2013447.7750000176
753	1847818.6280000093

Total rows: 10 Query complete 00:00:00.116

Successfully run. Total query runtime: 116 msec. 10 rows affected.

SQL demonstrando a query para obter os 10 produtos com maior faturamento.

Figura 19 - SQL Preço médio por produto

```

SELECT
    product_key,
    AVG(sale_value) AS preco_medio
FROM fact_sales
GROUP BY product_key;

```

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree view is expanded to show various database objects like Schemas, Tables, and Views. In the center, the Query History tab displays the executed SQL query. Below it, the Data Output tab shows the results of the query, which consists of 266 rows. The results table has two columns: 'product_key' (bigint) and 'preco_medio' (double precision). The 'Messages' tab at the bottom indicates the query was successfully run and completed quickly.

product_key	preco_medio
1	898
2	200.05199999999996
3	790
4	1886.6291379310238
5	828
6	215.3227043478261
7	938
8	24.39399999999915
9	753
10	3035.8796000000216
11	765
12	486.93648738317347
13	970
14	695.3830848056571
15	781
16	1821.1371182163068
17	951
18	242.9202922374554
19	839
20	824.0123872340432
21	732
22	356.98900000000054
23	887
24	602.3460000000002
25	867
26	54.6526851818521

SQL demonstrando a query para obter o preço médio por produto.

3. CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto permitiu demonstrar, na prática, a aplicação dos principais conceitos de Business Intelligence, desde a extração de dados de um ambiente transacional até a modelagem e construção de um Data Warehouse segundo a abordagem dimensional. A criação das tabelas dimensionais e da tabela fato possibilitou a análise histórica de vendas com múltiplas perspectivas. Além disso, a construção dos indicadores, demonstrou o valor agregado que um Data Warehouse oferece ao transformar dados dispersos em insights relevantes para negócios.

Por fim, o projeto reforça a importância de boas práticas no gerenciamento de dados, incluindo limpeza, padronização, documentação e governança, garantindo não apenas o funcionamento técnico da solução, mas também sua utilidade no contexto organizacional.