UNIVERSIDADE FEDERAL DO PARANÁ

ALBANO ROBERTO DRESCHER VON MAYWITZ
ARUNI VAN AMSTEL
GIOVANI TRIERWEILER ALVES
LUCAS SOUZA DE OLIVEIRA

PROJETO DE DATA WAREHOUSE

CURITIBA

2024

UNIVERSIDADE FEDERAL DO PARANÁ

PROJETO DE DATA WAREHOUSE

Trabalho será apresentado à disciplina de Banco de Dados 3, do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, para obtenção de nota semestral parcial, ao professor Dr. João Eugênio Marynowski.

CURITIBA

2024

LISTA DE FIGURAS

SUMÁRIO

1) INTRODUÇÃO

No cenário altamente competitivo do varejo, a capacidade de tomar decisões embasadas e realista, baseadas em dados e cruzamento de dados é essencial na realidade atual. Sendo a gestão eficaz dos dados um ativo estratégico para o sucesso de qualquer loja de varejo.

O varejo é um setor essencial da economia, responsável por conectar fabricantes e distribuidores diretamente aos consumidores finais, vendendo produtos em pequenas quantidade, obtendo muitas vezes, as informações fundamentais para o ciclo do negócio, bem como, a fonte dispersa em vários locais diferentes, tornando a análise complexa e a tomada de decisões um desafio. Este projeto de Data Warehouse (DW) surge em resposta a essa necessidade proeminente, visando aprimorar as capacidades analíticas e de tomada de decisões de nossa loja de varejo.

Esse segmento enfrenta diversos desafios, como a necessidade de competir em um mercado altamente dinâmico e globalizado, além de garantir a fidelização dos clientes, que possuem cada vez mais opções de compra. A busca por preços competitivos, o controle de estoque eficiente e a capacidade de responder rapidamente às mudanças nas preferências dos consumidores também são questões cruciais.

Diante desses desafios, a empresa Loja de Varejo identificou a importância de utilizar dados para otimizar suas operações e se manter competitiva. Atualmente, a empresa conta com um banco de dados relacional que armazena informações detalhadas sobre suas operações, como dados dos funcionários, vendas, cargos, lojas, clientes, itens, logins, endereços das lojas, fornecedores, compras e alimentos, sendo esses dados são fundamentais para a gestão eficiente do negócio.

No entanto, para lidar com o crescente volume de informações e extrair insights valiosos de forma mais rápida e eficaz, a empresa identificou a necessidade de evoluir para uma estrutura de dados mais adequada para análise gerencial, identificando a necessidade de cruzamento dos dados para alcançar melhores resultados. Assim, o presente estudo tem como objetivo a implantação de um banco de dados dimensional (Data Warehouse), que permitirá a consulta ágil de indicadores chave de desempenho (*KPI*s) e outras informações relevantes. Com

isso, a empresa poderá tomar decisões mais embasadas e melhorar sua capacidade de resposta às demandas do mercado.

O proposto para a empresa é centralizar o cruzamento dos dados implantando um ambiente de Data Warehouse (DW) aos seus dados, formando um novo conjunto de dados, que antes de serem carregados, os dados seram filtrados e limpos “gerando informação”, gerando os resultados esperados. Após esta etapa, esses dados sofrem somente operações de consulta e exclusão, sem que possam ser alterados, e esta característica representa a não-volatilidade dos dados. A variação em relação ao tempo consiste na manutenção de um histórico de dados em relação ao período de tempo maior que dos sistemas comuns, isto significa que as técnicas de mineração de dados não são aplicadas em tempo real, de forma a não comprometer o desempenho operacional do negócio, bem como, dos bancos transacionais de OLTP.

Ao se analisar um dado de um Data Warehouse (DW), o mesmo estará vinculado a um período determinado de tempo, pois terá uma chave de tempo que irá indicar o dia no qual esses dados foram extraídos.

Portanto, com a implementação do Data Warehouse, o negócio Loja de Varejo, obterá uma ampliação na visão de negócio possibilitando novos insights, transformando estes resultados em uma melhoria continua para a empresa.

2) SITUAÇÃO E MODELO ATUAL

O banco de dados relacional existente segue uma estrutura tradicional, usando um Banco de Dados relacional (SQL Server) para seu uso e como forma de coleta de dados. Embora o banco de dados atenda às necessidades operacionais, há limitações quando se trata de análise de dados e geração de insights gerenciais.

As fontes de dados incluem informações críticas, como transações de vendas, clientes, fornecedores e produtos, organizadas em tabelas que suportam a operação do dia a dia. No entanto, à medida que o volume de dados cresce, a performance do sistema pode se degradar em consultas mais complexas, especialmente devido à natureza transacional e altamente normalizada do banco de dados relacional.

O modelo de banco de dados atual é um modelo relacional, baseado no SGBD SQL Server, que representa seu modelo de negório através das tabelas e suas relações para uma loja de varejo. Foi projetado para suportar consultas analíticas e transacionais.

As chaves compostas e as estrangeiras estabelecem relacionamentos entre as tabelas, permitindo consultas complexas e agregações de dados, sendo a extração de dados feita por consultas complexas, gerando problemas de performance na escalabilidade do negócio. As tabelas de dimensão contêm informações estáticas, enquanto as tabelas de fato registram eventos de negócios que podem ser analisados ao longo do tempo.

Este conjunto de tabelas constitui a estrutura do banco de dados para o projeto de uma Loja de Varejo:

tb001_uf: Armazena dados sobre os estados e suas siglas.
tb002_cidades: Armazena dados sobre as cidades, relacionadas aos estados por meio da chave estrangeira tb001_sigla_uf.
tb003_enderecos: Armazena os endereços, agregando informações sobre cidade e estado.
tb004_lojas: Armazena os dados das lojas, incluindo detalhes de endereço, bem como informações de identificação, como inscrição estadual e CNPJ.
tb005_006_funcionarios_cargos: Armazena informações dos funcionários, seus cargos, salários, comissões e datas de promoção.

tb005_funcionarios: Armazena informações dos funcionários (como nome, data de nascimento, CPF, RG, status de emprego, além de datas de contratação e demissão).

tb006_cargos: Armazena os nomes dos cargos ocupados pelos funcionários.

tb010_clientes: Armazena informações dos clientes (como nome, CPF e detalhes de contato).

tb010_clientes_antigos: Armazena clientes antigos com base em seu CPF e nome (Indentificado formação ineficiente de dados).

tb011_logins: Armazena informações de logins, incluindo informações de login, senhas, CPF associado e datas de cadastro.

tb012_017_compras: Armazena informações de compras de produtos, fornecendo detalhes sobre produtos e fornecedores.

tb013_categorias: Armazena as categorias dos produtos.

tb012_produtos: Armazena informações detalhadas sobre produtos, incluindo descrição e categoria.

tb010_012_vendas: Armazena informações sobre vendas, incluindo detalhes sobre clientes, produtos, funcionários, datas e valores.

tb014_prd_alimentos: Armazena informações sobre produtos alimentícios, como lote, data de vencimento e valor sugerido.

tb015_prd_eletros: Armazena informações sobre produtos eletroeletrônicos, incluindo tensão, nível de consumo de energia e valor sugerido.

tb016_prd_vestuarios: Armazena informações sobre produtos de vestuário, incluindo tamanho, sexo e valor sugerido.

tb017_fornecedores: Armazena informações sobre fornecedores, incluindo razão social, nome fantasia, telefone e endereço.

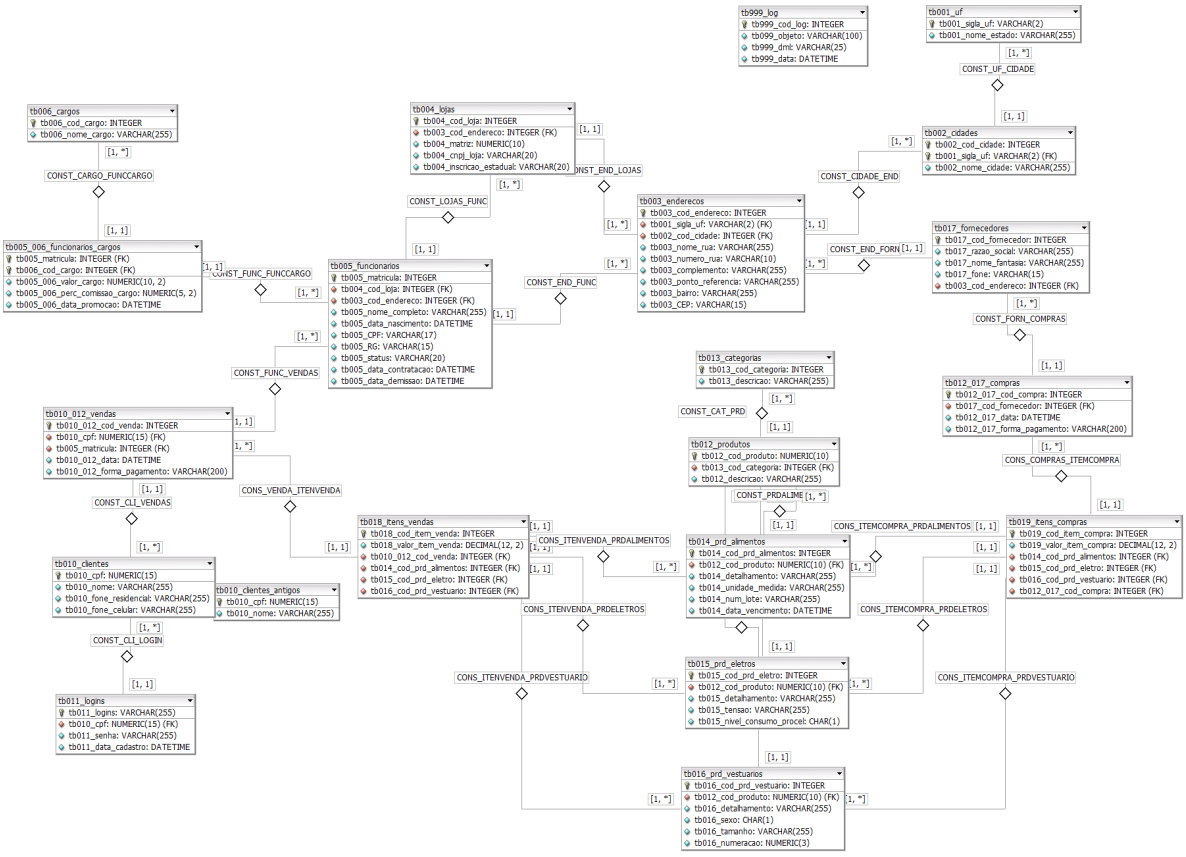
tb999_log: Armazena informações, das atividades de log, dos objetos, dos tipos de operação (DML) , de data e hora para registro de tempo.

2.1. RELACIONAMENTOS E DIVERGÊNCIAS

As chaves primárias e estrangeiras são utilizadas para manter a integridade referencial entre as tabelas, garantindo a consistência dos dados em um modelo de banco de dados relacional. Elas desempenham papel fundamental na relação entre as diferentes tabelas, garantindo que as informações sejam relacionadas e organizadas adequadamente.

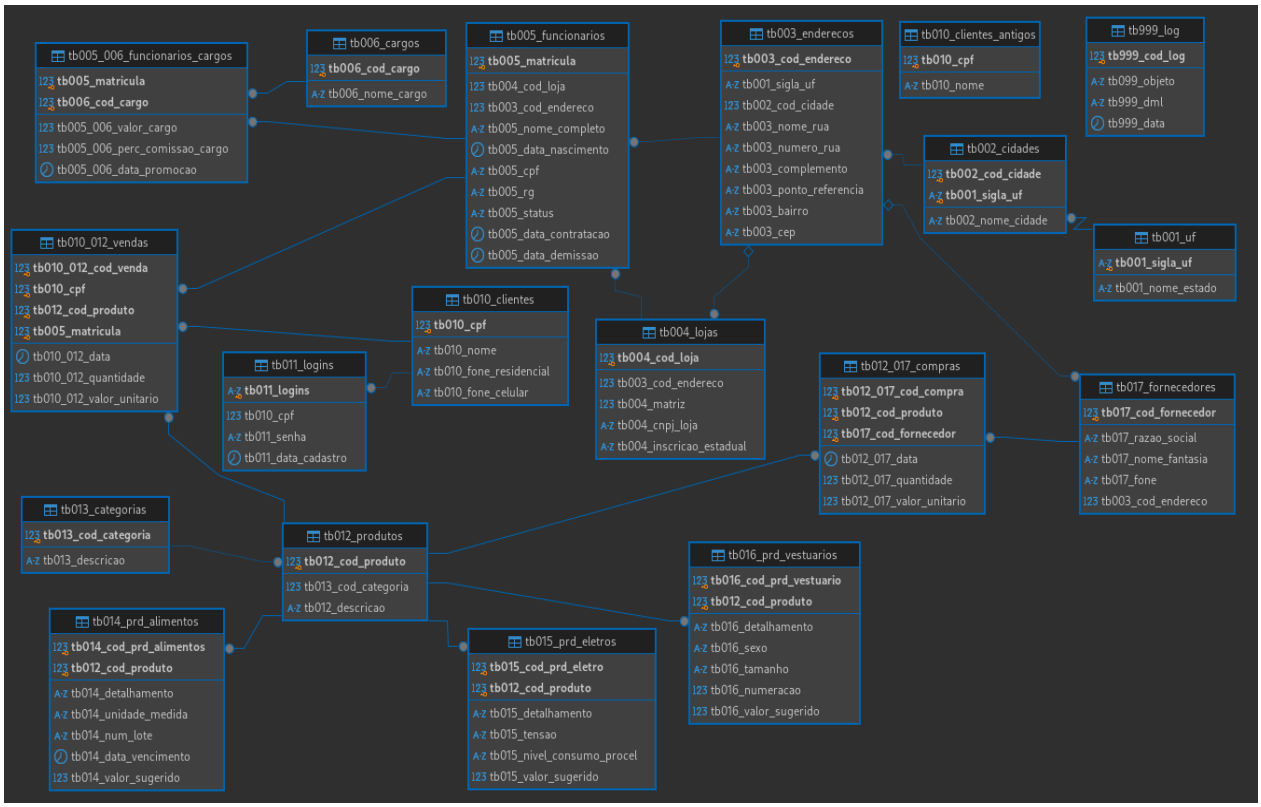
Foram encontradas algumas divergências entre a modelagem e o backup como evidenciado nas figuras abaixos:

Figura 01 – Modelo Lógico Relacional do Banco de Dados Anterior



O modelo acima é o modelo inicial do Banco de Dados, que sofreu algumas alterações importantes frente ao modelo de backup p. Segue abaixo o modelo atual já com as alterações, conforme imagem à seguir:

Figura 02 – Modelo Lógico Relacional do Banco de Dados Atual



Alterações encontradas entre os modelos das figuras 01 e 02:

1. Alteração na tabela tb010_012_vendas:
   1.1 Adicionadas as seguintes colunas: tb012_cod_produto, tb010_012_quantidade, tb010_012_valor_unitario;
   1.2 Adicionada a Foreign Key a coluna tb012_cod_produto, para referenciar a tabela Produtos;
2. Alteração na tabela tb012_017_compras:
   2.1 Adicionadas as seguintes colunas: tb012_017_cod_produto, tb012_017_quantidade, tb012_017_valor_unitario;
   2.2 Adicionada a Foreign Key a coluna tb012_cod_produto, para referenciar a tabela Produtos;
3. Alteração na tabela tb014_prd_alimentos:
   3.1 Adicionada a seguinte coluna: tb014_valor_sugerido;
4. Alteração na tabela tb015_prd_eletros:

4.1 Adicionada a seguinte coluna: tb015_valor_sugerido;

5. Alteração na tabela tb016_prd_vestuario:
   5.1 Adicionada a seguinte coluna: tb015_valor_sugerido;
6. Remoção da tabela tb018_itens_vendas do modelo relacional;
7. Remoção da tabela tb019_itens_compras do modelo relacional;
8. Alterado em todas as tabelas que possuem a coluna CPF, pois foi encontrado divergências: algumas tipavam o tbXXX_cpf como NUMERIC outras como VARCHAR. É boa prática tipar o cpf como VARCHAR, portanto foi alterado em todas as tabelas para VARCHAR.

A divergencia encontrada, foi quanto a coluna CPF das tabelas, que tinham tipos diferentes. Sendo boa prática no Brasil, modelar o campo CPF como texto e condicionar a inclusão com apenas números, foi corrigido essa divergência em todas as tabelas, se alterando os CREATE TABLE, para onde se constava como NUMERIC(15), foi alterado para CHARACTER(15). Quanto à inclusão dos dados, foi incluídos como estavam e realizado o update substituindo quaisquer caracteres que não fossem números.

Um ponto notável nas tabelas é que todas estavam com suas respectivas chaves corretas e muito bem construídas, não só permitindo que a integridade do banco de dados seja mantida, como mantendo a melhor performance possível.

Outro ponto notável é a presença nos ALTER TABLE, da cláusula “ON DELETE NO ACTION” nas chaves estrangeiras, o que impede que o usuário delete uma linha que está sendo referenciado em outra tabela, garantindo maior segurança dos dados.

A explicação do procedimento realizado para implementação do Banco de Dados LojaDeVarejo e os códigos das tecnologias utilizadas (docker, shellscript e SQL), vão ficar no Apêndice A.

3. DATA WAREHOUSE PROPOSTO

O Data Warehouse proposto oferece uma plataforma unificada e eficiente para armazenamento, transformação e consulta de dados, focando nos fatos de resultado: vendas e lucro. Este projeto incorpora os seguintes elementos essenciais:

1) Fatos que representam os eventos de negócios:

   Com o foco na geração de informação de desempenho e métricas para o negócio, o BDM implementa a pluralidade de dados agregados, mostrando especificamente o que se almeja alcançar.

2) Data Warehouse (DW):

   Envolve um projeto de Banco de Dados Multidimencional (BDM), que é uma representação conceitual das informações fundamentais relacionadas a um negócio ou organização. Ele agrega os atributos e como estão interconectados, para gerar informação. O BDM desempenha um papel crucial ao oferecer uma visão holística das operações de uma empresa, permitindo que todos compreendam os dados essenciais para o negócio.

Nesse caso, há dois tipos de eventos principais: vendas e lucro. Unitariamente esses eventos são importantes para avaliar o desempenho e a eficiência das operações da empresa. Importante é lembrar que não foi necessário criar um Fato para Atendimentos, já que esse foi interpretado como Quantidade de Vendas, deste modo, não há necessidade de criar outra tabela, adotando uma melhor prática, que é a economia, neste caso de memória, sempre que possível, não afetando a necessidade.

No quesito escolha das dimensões, são os atributos pelos quais desejamos analisar os fatos.

No caso das vendas, temos as seguintes dimensões: Categoria, Tipo, Produto, Loja, que é o Endereço da Loja, Funcionário, Cliente, Dia e Mês. Essas medidas permitem uma análise detalhada das vendas, levando em consideração diversos aspectos possíveis.

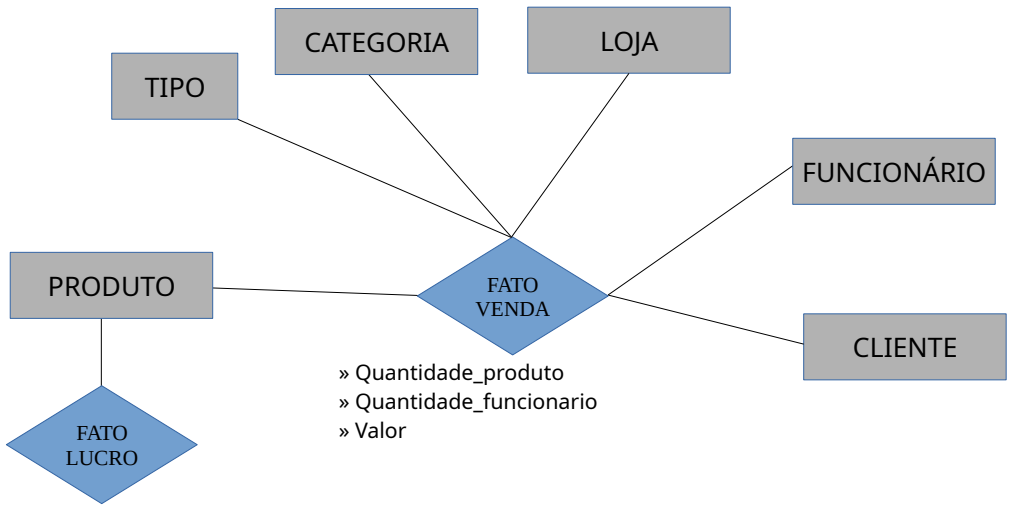
No caso do lucro, temos somente uma única dimensão: produto.

3.1. MODELO CONCEITUAL E LÓGICO DO BDM

3.1.1. MODELO CONCEITUAL

O modelo conceitual do Banco de Dados Multidimensional (BDM) que permitirá uma análise eficiente dos dados complexos relacionados às operações do negório, no caso da Loja de Varejo, está ilustrado na Figura 2:

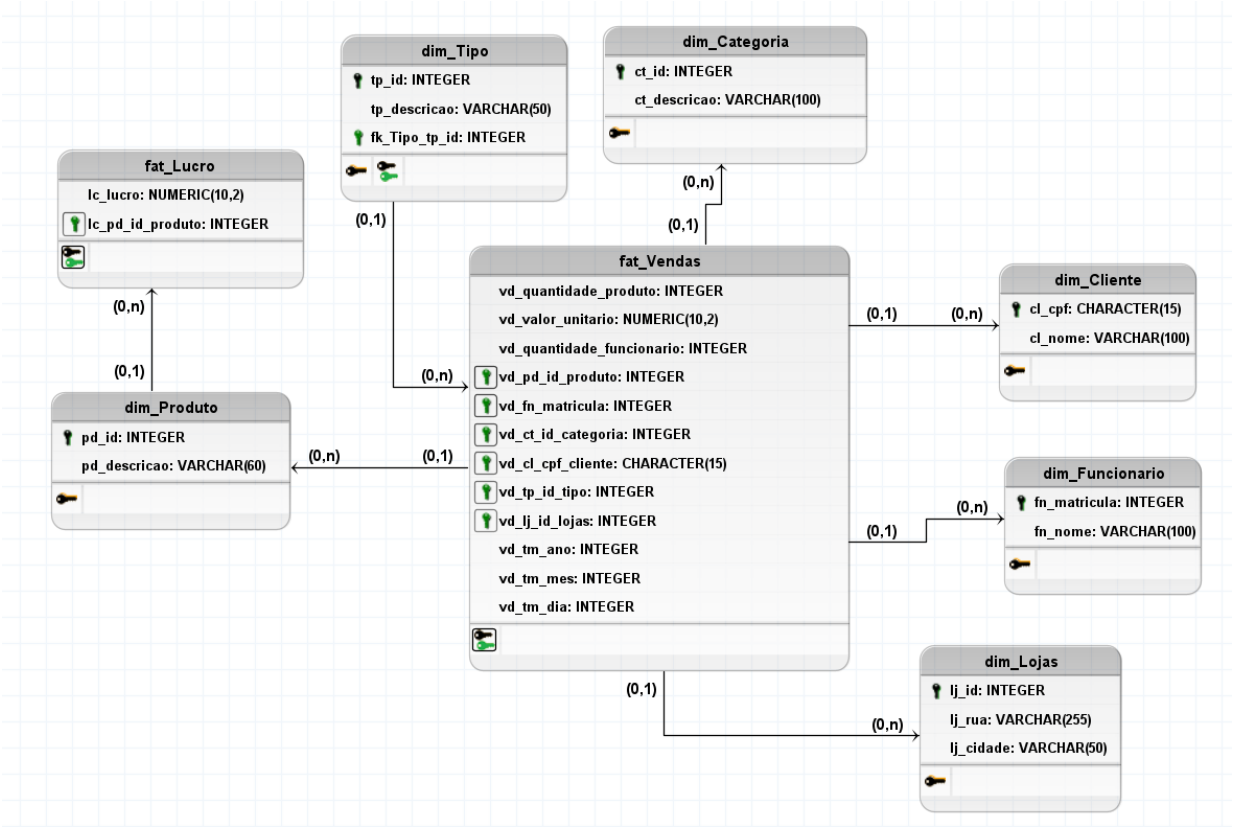
Figura 3: Diagrama Conceitual - Banco de Dados Multidimensional (Estrela de Neve)



3.1.2. MODELO LÓGICO

O esquema relacional/lógico do Banco de Dados Multidimensional (BDM) que está baseado no modelo conceitual apresentado acima, levando em consideração o Banco de Dados da Loja de Varejo, que é a base dados no modelo conceitual. O modelo lógico está exposto na Figura 4, logo abaixo:

Figura 4: Diagrama Lógico - Banco de Dados Multidimensional



3.2. ETL (EXTRACT, TRANSFORM, LOAD)

A Extração, Transformação e Carregamento (ETL) é um processo fundamental na análise de dados e da gestão da informação. Ela desempenha um papel essencial na coleta, preparação e disponibilização de dados para análise e tomada de decisões estratégicas.

O cenário da proposto da Loja de Varejo, foi realizado o processo de migração do banco de dados SQL Server para o banco de dados PostgreSQL. Essa exportação de dados é uma parte do processo de ETL. Sendo assim, a etapa de Extração envolveu a obtenção dos dados do banco de dados SQL Server original (através de um arquivo de backup(.sql) que foi fornecido), criando essencialmente uma staging area no PostgreSQL para armazenar temporariamente esses dados.

A etapa de Transformação incluiu a transcrição dos tipos de dados do SQL Server para o PostgreSQL, ajustes de esquema, e outras transformações necessárias para adaptar os dados ao novo banco de dados.

Importante mencionar que ao importar o schema do banco de dados SQL Server, as tabelas estrangeiras e suas relações foram preservadas, garantindo a integridade referencial dos dados. Foi realizado apenas uma conferência e ajuste conforme já citado anteriormente, na inserção de dados, onde foi necessário a normalização de tipo, de tamanho e limpeza no campo CPF.

Para realizar a transcrição dos dados, foi utilizado a ferramenta de IA (Inteligência Artificial) chatgpt, para converter e transcrever a sintaxe do SGBD SQL Server para o PostgreSQL, resultando em um novo arquivo de backup (.sql) já com a sintaxe do PostgreSQL.

Por fim, a etapa que seguiu foi a verificação de integridade do Banco de Dados, através de conteiner, onde se utilizou o docker para realizar a criação e o carregamento do Banco de Dados Relacional Loja de Varejo, possibilitando assim a Carga que envolveu a inserção dos dados transformados na staging area do PostgreSQL.

3.2.1. STAGING AREA

Foi feita a criação dos scripts de docker-compose e de configuração, sendo por meio destes realizado o backup do SQL Server para o PostgreSQL utilizando o container, que executou o arquivo SQL e fez a criação do Bando de Dados LojaDeVarejo.

Como o propósito do Data Warehouse é análise e integração de dados para fornecer informação, foi criado um segundo container, para que em um novo banco de dados, usando a extenção FDW (Foreign Data Wrapper), fosse possível à implementação do BDM (Banco de Dados Multidimensional) fazendo Cross Database, que era algo que anteriormente não se conseguia fazer diretamente.

Para isso ser possível, utilizou-se extensão postgres_fdw, sendo necessária sua ativação (através do comando CREATE EXTENSION postgres_fdw;), para que fosse possível realizar consultas diretamente do bdm(projetodw) no banco de dados relacional LojaDeVarejo.

Para popular as tabelas do Banco de Dados Multidimensional(BDM), foi necessário executar instruções de inserção (INSERT) com seleções (SELECT) a utilizando os objetos wrapper da ferramenta fdw no staging area do PostgreSQL,

que continha os dados já transformados e adaptados. Os comandos usados para popular as tabelas podem ser visualizados no Anexo B.

Para agregar os dados nas consultas SQL que alimentaram o Data Warehouse, utilizou-se o CUBE, a cláusula CUBE permite que você obtenha resultados agregados para todas as combinações possíveis de valores em várias colunas, criando um conjunto de agregações multidimensional.

3.3. FREQUÊNCIA DE CARGA E TEMPO DE RETENÇÃO

O tempo de retenção dos dados e a frequência de carga são dois aspectos essenciais na gestão e manutenção de dados em um ambiente de armazenamento de informações corporativo, essencialmente em sistemas de gerenciamento de banco de dados e Data Warehouse.

A frequência de carga refere-se à determminação de tempo com que os dados são atualizados ou inseridos em um sistema de banco de dados. No caso da Loja de Varejo, a frequência de carga foi estabelecida como semanalmente, em função do tamanho do BDM resultante após a carga de dados. Foi utilizado a função do PostgreSQL(SELECT pg_size_pretty(pg_database_size('projetodw'));), o que permitiram visualizar o atual tamanho em Bytes, do BDM, sendo tal medida equivale a 12MB para 110 vendas em 9 meses.

Essa frequência de carga semanal é uma escolha levando em conta o crescimento dos dados, e como não são realizadas muitas vendas por dia, não há necessidade de atualizar o banco de hora em hora ou todos os dias.

Além disso, um banco de dados de 12 MB é pequeno em tamanho relativamente. Isso significa que a infraestrutura atual deve ser capaz de lidar facilmente com atualizações semanaius, sem se preocupar com limitações de hardware.

O tempo de retenção é o período durante o qual os dados devem ser mantidos e arquivados antes de serem removidos ou descartados, para o caso da Loja de Varejo, o tempo de retenção foi percebido conforme a Lei de Proteção de Dados, que dita como 5 anos de retenção, o que significa que os dados são preservados por meia década.

Esse tempo de retenção de 5 anos estabelecido é importante pois permite que a Loja acumule um valioso acervo de dados que pode ser essencial para algumas finalidades como: Análises refinadas, sendo por Histórico, sendo por detalhamento por novos escopos e/ou por Previsões de longo prazo.

Além disso, supondo que a loja duplique o número de vendas a cada ano por 5 anos será necessário de 10GB a 15GB de espaço de memória para manter o Data Warehouse, o que na atualidade, não é um problema.

3.4. HARDWARE E SGBD UTILIZADOS E RECOMENDADOS

A expectativa de necessidade de Hardware e Infraestrutura deve andar alinhada com a expectativa de crescimento da Loja de Varejo. Ou seja, existe uma necessidade para o setor de varejo atual, que o tamanho do Data Warehouse atual seja também considerado frente as métricas de vendas. Pois os requisitos de hardware podem ser relativamente modestos na data deste estudo, mas um servidor com um processador moderado (na data deste trabalho, seria um Intel Xeon ou um AMD EPYC) é mais do que suficiente para lidar com essa carga de dados.

Recomenda-se ainda, alocar pelo menos 64 GB de RAM para garantir um bom desempenho, tendo em vista sobrecarga de sistema e overhead. Quanto ao armazenamento, a utilização de unidades de estado sólido (SSD) é essencial, mas é altamente recomendado a utilização da tecnologia NVME para desfrutar de tranquilidade em velocidade e carga, sendo o tamanho de 1TB mais do que suficiente no momento deste estudo, pois ambos oferecem tempos de acesso muito mais rápidos em comparação aos discos rígidos(HDs) tradicionais.

Em termos de SGBD, a opção gratuita como PostgreSQL é mais que adequado, tanto em tecnologia como atualizações para um Data Warehouse desse tamanho. O PostgreSQL oferece soluções em recursos de gerenciamento de dados robustos, como os mostrados neste estudo, mesmo para conjuntos de dados maiores. Se a loja de varejo planeja crescer significativamente no futuro, pode ser útil considerar opções mais escaláveis, como migração de Infraestrutura para Cloud(nuvem), sendo Amazon, Google Cloud ou Azure opções interessantes.

4. ANÁLISE GERENCIAL

4.1. Quantidade de vendas agrupadas por tipo e categoria.

```sql
SELECT
    sum(v.vd_quantidade_produto),
    t.tp_descricao as Tipo,
    c.ct_descricao as Categoria
FROM public.fat_vendas v
INNER JOIN public.dim_tipo t ON t.tp_id = v.vd_tp_id_tipo
INNER JOIN public.dim_categoria c ON c.ct_id = v.vd_ct_id_categoria
WHERE
    v.vd_pd_id_produto IS NULL
    AND v.vd_cl_cpf_cliente IS NULL
    AND v.vd_fn_matricula IS NULL
    AND v.vd_lj_id_lojas IS NULL
    AND v.vd_ct_id_categoria IS NOT NULL
    AND v.vd_tp_id_tipo IS NOT null
group by Tipo, Categoria
ORDER BY Tipo;
```

dim_tipo(+) 1

SELECT sum(v.vd_quantidade_produto), t.tp_descricao as

Enter a SQL expression to filter results (u

Grid

Text

	123 sum	A-Z tipo	A-Z categoria
1	584	Alimentos	Alimentos Perecíveis
2	160	Eletros	CD e DVD
3	240	Eletros	Eletrodomésticos
4	160	Eletros	Eletrônicos
5	320	Vestuários	Roupas Femininas
6	104	Vestuários	Roupas Infantis
7	296	Vestuários	Roupas Masculinas

4.2. Valor das vendas por funcionário, permitindo uma visão hierárquica por tempo.

```sql
SELECT
    sum(v.vd_valor_unitario) AS "Valor das Vendas",
    f.fn_nome as "Nome do Funcionário",
    v.vd_tm_mes as Mes,
    v.vd_tm_ano as Ano
FROM
    public.fat_vendas v
INNER JOIN public.dim_lojas l ON l.lj_id = v.vd_lj_id_lojas
INNER JOIN public.dim_funcionario f ON f.fn_matricula = v.vd_fn_matricula
WHERE
    v.vd_tp_id_tipo IS NULL
    AND v.vd_pd_id_produto IS NULL
    AND v.vd_cl_cpf_cliente IS NULL
    AND v.vd_ct_id_categoria IS NULL
    AND v.vd_fn_matricula IS NOT NULL
    AND v.vd_tm_mes IS NOT null
    AND v.vd_tm_ano IS NOT NULL
    AND v.vd_lj_id_lojas IS NOT null
GROUP BY f.fn_nome, v.vd_tm_mes, v.vd_tm_ano
ORDER BY f.fn_nome;
```

dim_funcionario(+) 1

SELECT sum(v.vd_valor_u | Enter a SQL expression to filter results (use Ctrl+Space)

	123 Valor das Vendas	A-Z Nome do Funcionário	123 mes	123 ano
1	543.92	Funcionário 01	2	2,023
2	4,610.48	Funcionário 01	8	2,023
3	732.6	Funcionário 02	3	2,023
4	4,550.32	Funcionário 02	9	2,023
5	75.72	Funcionário 03	3	2,023
6	7,230.4	Funcionário 03	4	2,023
7	77.56	Funcionário 04	3	2,023
8	6,400	Funcionário 04	4	2,023
9	190.4	Funcionário 04	5	2,023
10	163.84	Funcionário 05	3	2,023
11	2,297.6	Funcionário 05	4	2,023
12	201.28	Funcionário 05	6	2,023
13	74.36	Funcionário 06	1	2,023
14	214.16	Funcionário 06	3	2,023
15	2,311.16	Funcionário 06	4	2,023
16	229.04	Funcionário 07	3	2,023

Refresh | Save | Cancel | Expo

200 | 66 | 66 row(s) fetched - 0.008s, on 2024-10-19 at 11:19:29

4.3. Volume das vendas por funcionário, permitindo uma visão por localidade.

```sql
SELECT
    f.fn_nome as "Nome do Funcionário",
    l.lj_rua as "Endereço",
    l.lj_cidade as "Cidade",
    v.vd_tm_mes as Mês,
    v.vd_tm_ano as Ano,
    sum(v.vd_quantidade_funcionario) as "Volume de Vendas"
FROM
    public.fat_vendas v
INNER JOIN public.dim_funcionario f ON f.fn_matricula = v.vd_fn_matricula
INNER JOIN public.dim_lojas l ON l.lj_id = v.vd_lj_id_lojas
WHERE
    f.fn_nome IS NOT NULL
    AND l.lj_rua IS NOT NULL
    AND l.lj_cidade IS NOT NULL
    AND v.vd_tm_ano IS NOT NULL
    AND v.vd_tm_mes IS NOT NULL
GROUP BY
    f.fn_nome, l.lj_rua, l.lj_cidade, v.vd_tm_mes, v.vd_tm_ano
ORDER BY
    f.fn_nome, l.lj_rua, l.lj_cidade, v.vd_tm_mes, v.vd_tm_ano;
```

uncionario(+) 1

CT f.fn_nome as "Nome do | Enter a SQL expression to filter results (use Ctrl+Space)

Nome do Funcionário	Endereço	Cidade	mês	ano	Volume de Vendas
Funcionário 01	Av. Getúlio Vargas	Curitiba	2	2,023	192
Funcionário 01	Av. Getúlio Vargas	Curitiba	8	2,023	192
Funcionário 02	Av. Brasil	Belo Horizonte	3	2,023	608
Funcionário 02	Av. Brasil	Belo Horizonte	9	2,023	128
Funcionário 03	Av. Joaquim Lima	Porto Alegre	3	2,023	160
Funcionário 03	Av. Joaquim Lima	Porto Alegre	4	2,023	192
Funcionário 04	Av. Paulista	São Paulo	3	2,023	256
Funcionário 04	Av. Paulista	São Paulo	4	2,023	64
Funcionário 04	Av. Paulista	São Paulo	5	2,023	128
Funcionário 05	Av. Manoel Ribas	Curitiba	3	2,023	512
Funcionário 05	Av. Manoel Ribas	Curitiba	4	2,023	64
Funcionário 05	Av. Manoel Ribas	Curitiba	6	2,023	64
Funcionário 06	Av. Getúlio Vargas	Curitiba	1	2,023	128
Funcionário 06	Av. Getúlio Vargas	Curitiba	3	2,023	128
Funcionário 06	Av. Getúlio Vargas	Curitiba	4	2,023	64

Refresh Save Cancel Export data

66 66 row(s) fetched - 0.007s, on 2024-10-19 at 11:40:23

4.4. Quantidade de atendimentos realizados por funcionário e localidade.

```sql
SELECT distinct
    v.vd_quantidade_funcionario  "Atendimentos por Funcionário",
    f.fn_nome,
    l.lj_cidade
FROM
    public.fat_vendas v
INNER JOIN dim_lojas l ON l.lj_id = v.vd_lj_id_lojas
INNER JOIN dim_funcionario f ON f.fn_matricula = v.vd_fn_matricula
WHERE
    v.vd_tp_id_tipo IS NULL
    AND v.vd_tm_dia IS NULL
    AND v.vd_pd_id_produto IS NULL
    AND v.vd_cl_cpf_cliente IS NULL
    AND v.vd_tm_mes IS NULL
    AND v.vd_ct_id_categoria IS NULL
    AND v.vd_lj_id_lojas IS NOT NULL
    AND v.vd_fn_matricula IS NOT null
ORDER BY f.fn_nome, v.vd_quantidade_funcionario;
```

fat_vendas(+) 1

SELECT distinct v.vd_quantidade | Enter a SQL expression to filter results (use Ctrl+Space)

	123 Atendimentos por Funcionário	A-Z fn_nome	A-Z lj_cidade
1	12	Funcionário 01	Curitiba
2	23	Funcionário 02	Belo Horizonte
3	11	Funcionário 03	Porto Alegre
4	14	Funcionário 04	São Paulo
5	20	Funcionário 05	Curitiba
6	10	Funcionário 06	Curitiba
7	18	Funcionário 07	Belo Horizonte
8	8	Funcionário 08	Porto Alegre
9	14	Funcionário 09	São Paulo
10	9	Funcionário 10	Curitiba
11	14	Funcionário 11	Curitiba
12	16	Funcionário 12	Belo Horizonte
13	12	Funcionário 13	Porto Alegre
14	13	Funcionário 14	São Paulo
15	11	Funcionário 15	Curitiba
16	7	Funcionário 16	Curitiba
17	11	Funcionário 17	Curitiba
18	13	Funcionário 18	Curitiba

Refresh Save Cancel

33 33 row(s) fetched - 0.005s, on 2024-10-19 at 11:48:40

4.5. Valor das últimas vendas realizadas por cliente.

```sql
SELECT
    vd.vd_cl_cpf_cliente AS "CPF do Cliente",
    vd.vd_tm_dia AS "Dia da Venda",
    vd.vd_tm_mes AS "Mês da Venda",
    vd.vd_tm_ano AS "Ano da Venda",
    SUM(vd.vd_quantidade_produto * vd.vd_valor_unitario) AS "Valor da Última Venda"
FROM
    fat_Vendas vd
INNER JOIN (
    -- Subconsulta para pegar a última venda de cada cliente
    SELECT  vd_cl_cpf_cliente,
            MAX(vd_tm_ano * 10000 + vd_tm_mes * 100 + vd_tm_dia) AS ultima_data
    FROM  fat_Vendas
    GROUP BY vd_cl_cpf_cliente
    ) ult_vd
    ON vd.vd_cl_cpf_cliente = ult_vd.vd_cl_cpf_cliente
    AND (vd.vd_tm_ano * 10000 + vd.vd_tm_mes * 100 + vd.vd_tm_dia) = ult_vd.ultima_data
GROUP BY
    vd.vd_cl_cpf_cliente, vd.vd_tm_ano, vd.vd_tm_mes, vd.vd_tm_dia;
```

ndas 1

CT vd.vd_cl_cpf_cliente AS | Enter a SQL expression to filter results (use Ctrl+Space)

CPF do Cliente	Dia da Venda	Mês da Venda	Ano da Venda	Valor da Última Venda
10000000000	26	3	2,023	1,912.32
10000000001	27	3	2,023	1,912.32
10000000002	28	3	2,023	1,722.88
10000000003	29	3	2,023	2,101.76
10000000004	30	3	2,023	2,778.88
10000000005	31	3	2,023	2,771.2
10000000006	22	3	2,023	5,580.8
10000000007	23	3	2,023	1,912.32
10000000008	24	3	2,023	1,912.32
10000000009	4	4	2,023	6,092.8
10000000010	4	5	2,023	6,092.8
10000000011	4	6	2,023	6,440.96
10000000012	28	3	2,023	4,912.64
10000000013	4	5	2,023	10,442.24

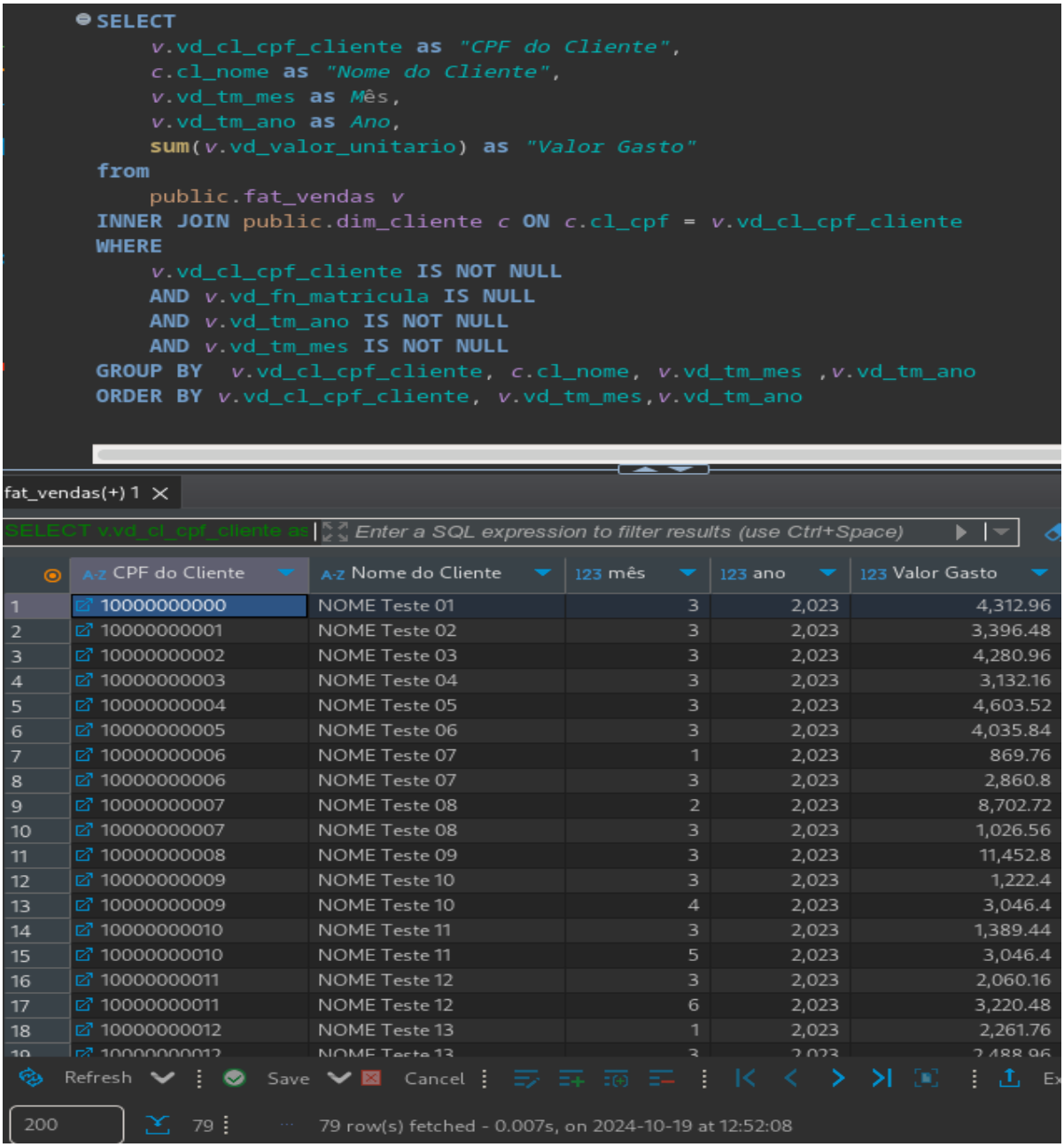
Refresh | Save | Cancel | Export data

51 | 51 row(s) fetched - 0.016s, on 2024-10-19 at 12:09:10

4.6. Clientes que mais compraram na loja virtual com valor acumulado por período.

Analisando as informações fornecidas com o banco de dados transacional Loja de Varejo, não é possível identificar ou inferir de que tipo de vendas foram feitas, seja em loja virtual ou física. Portanto base de dados não consta um campo que às diferenciem. O dado mais próximo que a Loja de Varejo possui são os logins dos clientes, todavia isso não significa que a venda realizada foi em loja virtual.

Em consequência disso, a consulta inferida para essa análise considera os clientes que mais gastaram dentre os dados contidos no BDM, deduzindo que as vendas foram realizadas em loja física e/ou virtual.



```
SELECT
    v.vd_cl_cpf_cliente as "CPF do Cliente",
    c.cl_nome as "Nome do Cliente",
    v.vd_tm_mes as Mês,
    v.vd_tm_ano as Ano,
    sum(v.vd_valor_unitario) as "Valor Gasto"
from
    public.fat_vendas v
INNER JOIN public.dim_cliente c ON c.cl_cpf = v.vd_cl_cpf_cliente
WHERE
    v.vd_cl_cpf_cliente IS NOT NULL
    AND v.vd_fn_matricula IS NULL
    AND v.vd_tm_ano IS NOT NULL
    AND v.vd_tm_mes IS NOT NULL
GROUP BY  v.vd_cl_cpf_cliente, c.cl_nome, v.vd_tm_mes ,v.vd_tm_ano
ORDER BY v.vd_cl_cpf_cliente, v.vd_tm_mes,v.vd_tm_ano
```

fat_vendas(+) 1

SELECT v.vd_cl_cpf_cliente as | Enter a SQL expression to filter results (use Ctrl+Space)

	CPF do Cliente	Nome do Cliente	mês	ano	Valor Gasto
1	10000000000	NOME Teste 01	3	2,023	4,312.96
2	10000000001	NOME Teste 02	3	2,023	3,396.48
3	10000000002	NOME Teste 03	3	2,023	4,280.96
4	10000000003	NOME Teste 04	3	2,023	3,132.16
5	10000000004	NOME Teste 05	3	2,023	4,603.52
6	10000000005	NOME Teste 06	3	2,023	4,035.84
7	10000000006	NOME Teste 07	1	2,023	869.76
8	10000000006	NOME Teste 07	3	2,023	2,860.8
9	10000000007	NOME Teste 08	2	2,023	8,702.72
10	10000000007	NOME Teste 08	3	2,023	1,026.56
11	10000000008	NOME Teste 09	3	2,023	11,452.8
12	10000000009	NOME Teste 10	3	2,023	1,222.4
13	10000000009	NOME Teste 10	4	2,023	3,046.4
14	10000000010	NOME Teste 11	3	2,023	1,389.44
15	10000000010	NOME Teste 11	5	2,023	3,046.4
16	10000000011	NOME Teste 12	3	2,023	2,060.16
17	10000000011	NOME Teste 12	6	2,023	3,220.48
18	10000000012	NOME Teste 13	1	2,023	2,261.76

Refresh | Save | Cancel

200 | 79 | 79 row(s) fetched - 0.007s, on 2024-10-19 at 12:52:08

5. CONCLUSÃO

No presente projeto, foi possível verificar que nem sempre as demandas do cliente se alinham com a realidade da empresa, objetivando a questão de diversidade tecnológica e congruência na escalabilidade das aplicações.

Por meio de todos os dados apresentados, é importante destacar a enorme relevância de um Data Warehouse no ambiente empresarial atualmente, mais ainda para CEOs e as demandas gerenciais. Ele não melhora apenas a eficiência das consultas de dados, mas também ajuda a controlar os custos, ter um espectro visão maior na tomada de decisões e na ampliação da diversidade de possibilidades com os dados. Isso por si só torna-o uma ferramenta de grande importância para aumentar a produtividade da empresa e dos negócios como um todo.

Com um Data Warehouse, os dados se traduzem em informações que são organizadas de maneira fácil para fazer analise ou obter insights importantes para o negócio. Isso significa que os tomadores de decisões têm acesso não apenas rápido a dados confiáveis, mas que ajudem ma tomada de decisões, muito mais informadas, estratégicas e acertivas.

Em outras palavras, o Data Warehouse desempenha um papel fundamental no impulsionamento de grandes negócios para o sucesso, onde pequenos negócios, através das decisões melhoradas e acertivas possibilitadas pelo DW, abrem caminho para que empresas se tornem organizações. O informação é a base atual de toda a Tecnologia da Informação como um todo, e boa parte foi possibilitada pelos Data Warehouses que tornaram as organizações mais ágeis e capazes de enfrentar os desafios do mercado de forma inovadora, num ciclo de evolução contínua, muito mais eficazes e efetivas.

REFERÊNCIAS

CARVALHO, Vinícius. PostgreSQL: Banco de dados para aplicações web modernas. Ed. São Paulo: Casa do Código, 2017.

POSTGRESLQ. PostgreSQL: Documention, c2023. Disponível em: <https://www.postgresql.org/docs/16/index.html>. Acesso em: 18 de Outubro de 2024.

Conjuntos de agrupamentos ou GROUPING SETS do Postgres – ROLLUP & CUBE. PostgreSQL: CUBE, c2022. Disponível em: <https://soloweb.com.br/blog/2023/08/01/conjuntos-de-agrupamentos-ou-grouping-sets-do-postgres-rollup-cube/>. Acesso em: 16 de Outubro de 2024.

POSTGRESQL DOCS. PostgreSQL: Documention, c2024. Disponível em: <https://www.postgresql.org/docs/current/cube.html>. Acesso em: 16 de Outubro de 2024.

ANEXOS

ANEXO A – Configuração dos Containers Docker e ETL (Staging Area).

Arquivo Docker Compose:

File: **docker-compose.yml**

```yaml
services:
  lojadevarejo:
    image: postgres:16-alpine
    container_name: lojadevarejo
    hostname: lojadevarejo
    environment:
      POSTGRES_DB: lojadevarejo
      POSTGRES_USER: equipefoda
      POSTGRES_PASSWORD: "equipefoda"
    ports:
      - 55432:5432
    volumes:
      - ./projetodw/init.sh:/docker-entrypoint-initdb.d/init.sh
      - ./projetodw/backup-full.sql:/tmp/backup-full.sql
    networks:
      - projetodw_bd3
  projetodw:
    image: postgres:17-alpine
    container_name: projetodw
    hostname: projetodw
    environment:
      PGPORT: 5434
      POSTGRES_DB: projetodw
      POSTGRES_USER: equipefoda
      POSTGRES_PASSWORD: "equipefoda"
    ports:
      - 55434:5434
    volumes:
      - ./projetodw/projetodw.sh:/docker-entrypoint-initdb.d/projetodw.sh
      - ./projetodw/projetodw-full.sql:/tmp/projetodw-full.sql
    networks:
      - projetodw_bd3


networks:
  projetodw_bd3:
    name: "projetodw_bd3"
    driver: bridge
```

Shell script que executa a restauração do Banco de Dados Loja de Varejo:

File: **init.sh**

```sh
#!/usr/bin/env bash

# ------------------------------------------------------------------------------ #
# Script      : entrypoint
# Description :
# Version     : 0.1
# Author      : Albano Roberto Drescher Von Maywitz
# Data        : 19 de julho de 2024
# ------------------------------------------------------------------------------ #
# Use :
# ------------------------------------------------------------------------------ #
# Copyright (C) 2024, Albano <allbano@gmail.com>.
# License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
# This is free software: you are free to change and redistribute it.
# There is NO WARRANTY, to the extent permitted by law.
# ------------------------------------------------------------------------------ #


set -e
# Executa o arquivo init.sql usando psql
# psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" -c "CREATE DATABASE $POSTGRES_DB;"
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" -f "/tmp/backup-full.sql"

# Executa o comando original do entrypoint do PostgreSQL
exec "$@"
```

Script SQL (backupo-full.sql), que limpa (DROP) de todas as tabelas se existirem:

File: **backup-full.sql**

```sql
-- Procedimento para deletas todas as tabelas caso existam
DO
$$
BEGIN
    DROP TABLE IF EXISTS tb010_clientes_antigos;
    DROP TABLE IF EXISTS tb016_prd_vestuarios;
    DROP TABLE IF EXISTS tb011_logins;
    DROP TABLE IF EXISTS tb015_prd_eletros;
    DROP TABLE IF EXISTS tb014_prd_alimentos;
    DROP TABLE IF EXISTS tb005_006_funcionarios_cargos;
    DROP TABLE IF EXISTS tb006_cargos;
    DROP TABLE IF EXISTS tb010_012_vendas;
    DROP TABLE IF EXISTS tb010_clientes;
    DROP TABLE IF EXISTS tb005_funcionarios;
    DROP TABLE IF EXISTS tb004_lojas;
    DROP TABLE IF EXISTS tb999_log;
    DROP TABLE IF EXISTS tb012_017_compras;
    DROP TABLE IF EXISTS tb017_fornecedores;
    DROP TABLE IF EXISTS tb003_enderecos;
    DROP TABLE IF EXISTS tb002_cidades;
    DROP TABLE IF EXISTS tb001_uf;
    DROP TABLE IF EXISTS tb012_produtos;
    DROP TABLE IF EXISTS tb013_categorias;

END
$$;
```

Ainda no mesmo script citado acima, faz a criação das tabelas(CREATE TABLE):

```sql
-- Create Tables

-- Tabela tb001_uf
CREATE TABLE tb001_uf (
    tb001_sigla_uf       VARCHAR(2)  NOT NULL ,
    tb001_nome_estado    VARCHAR(255)  NOT NULL,
    CONSTRAINT XPKtb001_uf PRIMARY KEY (tb001_sigla_uf)
);

-- Tabela tb002_cidades
CREATE TABLE tb002_cidades (
    tb002_cod_cidade SERIAL,
    tb001_sigla_uf VARCHAR(2) NOT NULL,
    tb002_nome_cidade VARCHAR(255) NOT NULL,
    CONSTRAINT XPKtb002_cidades PRIMARY KEY (tb002_cod_cidade, tb001_sigla_uf)
);

-- Tabela tb003_enderecos
CREATE TABLE tb003_enderecos (
    tb003_cod_endereco SERIAL,
    tb001_sigla_uf VARCHAR(2) NOT NULL,
    tb002_cod_cidade INTEGER     NOT NULL,
    tb003_nome_rua VARCHAR(255) NOT NULL,
    tb003_numero_rua VARCHAR(10) NOT NULL,
    tb003_complemento VARCHAR(255),
    tb003_ponto_referencia VARCHAR(255),
    tb003_bairro VARCHAR(255) NOT NULL,
    tb003_CEP VARCHAR(15) NOT NULL,
    CONSTRAINT XPKtb003_enderecos PRIMARY KEY (tb003_cod_endereco)
);

-- Tabela tb004_lojas
CREATE TABLE tb004_lojas (
    tb004_cod_loja SERIAL,
    tb003_cod_endereco INTEGER,
    tb004_matriz NUMERIC(10),
    tb004_cnpj_loja VARCHAR(20) NOT NULL,
    tb004_inscricao_estadual VARCHAR(20),
    CONSTRAINT XPKtb004_lojas PRIMARY KEY (tb004_cod_loja)
);
```

Continuando a criação das tabelas:

```sql
-- Tabela tb005_006_funcionarios_cargos
CREATE TABLE tb005_006_funcionarios_cargos (
    tb005_matricula INTEGER NOT NULL,
    tb006_cod_cargo INTEGER NOT NULL,
    tb005_006_valor_cargo NUMERIC(10,2) NOT NULL,
    tb005_006_perc_comissao_cargo NUMERIC(5,2) NOT NULL,
    tb005_006_data_promocao DATE NOT NULL,
    CONSTRAINT XPKtb005_006_funcionarios_cargos PRIMARY KEY (tb005_matricula, tb006_cod_cargo)
);

-- Tabela tb005_funcionarios
CREATE TABLE tb005_funcionarios (
    tb005_matricula SERIAL,
    tb004_cod_loja INTEGER NOT NULL,
    tb003_cod_endereco INTEGER NOT NULL,
    tb005_nome_completo VARCHAR(255) NOT NULL,
    tb005_data_nascimento DATE NOT NULL,
    tb005_cpf VARCHAR(15) NOT NULL,
    tb005_RG VARCHAR(15) NOT NULL,
    tb005_status VARCHAR(20) NOT NULL,
    tb005_data_contratacao DATE NOT NULL,
    tb005_data_demissao DATE,
CONSTRAINT XPKtb005_funcionarios PRIMARY KEY (tb005_matricula)
);

-- Tabela tb006_cargos
CREATE TABLE tb006_cargos (
    tb006_cod_cargo SERIAL,
    tb006_nome_cargo VARCHAR(255) NOT NULL,
    CONSTRAINT XPKtb006_cargos PRIMARY KEY (tb006_cod_cargo)
);

-- Tabela tb0_012_vendas
CREATE TABLE tb010_012_vendas (
    tb010_012_cod_venda SERIAL,
    tb010_cpf CHARACTER(15) NOT NULL,
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb005_matricula INTEGER NOT NULL,
    tb010_012_data date NOT NULL,
    tb010_012_quantidade NUMERIC(10) NOT NULL,
    tb010_012_valor_unitario NUMERIC(12,4) NOT NULL,
    CONSTRAINT XPKtb010_012_vendas PRIMARY KEY (tb010_012_cod_venda, tb005_matricula, tb010_cpf, tb012_cod_produto)
);
```

```sql
-- Tabela tb010_clientes
CREATE TABLE tb010_clientes (
    tb010_cpf CHARACTER(15) NOT NULL,
    tb010_nome VARCHAR(255) NOT NULL,
    tb010_fone_residencial VARCHAR(255) NOT NULL,
    tb010_fone_celular VARCHAR(255),
    CONSTRAINT XPKtb010_clientes PRIMARY KEY (tb010_cpf)
);

-- Tabela tb010_clientes_antigos
CREATE TABLE tb010_clientes_antigos (
    tb010_cpf CHARACTER(15) NOT NULL,
    tb010_nome VARCHAR(255),
    CONSTRAINT XPKtb010_clientes_antigos PRIMARY KEY (tb010_cpf)
);
```

Continuando a criação das tabelas:

```
-- Tabela tb011_logins
CREATE TABLE tb011_logins (
    tb011_logins VARCHAR(255) NOT NULL,
    tb010_cpf CHARACTER(15) NOT NULL,
    tb011_senha VARCHAR(255) NOT NULL,
    tb011_data_cadastro date,
    CONSTRAINT XPKtb011_logins PRIMARY KEY (tb011_logins)
);

-- Tabela tb012_produtos
CREATE TABLE tb012_produtos (
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb013_cod_categoria INTEGER NOT NULL,
    tb012_descricao VARCHAR(255) NOT NULL,
    CONSTRAINT XPKtb012_produtos PRIMARY KEY (tb012_cod_produto)
);

-- Tabela tb013_categorias
CREATE TABLE tb013_categorias (
    tb013_cod_categoria SERIAL,
    tb013_descricao VARCHAR(255) NOT NULL,
    CONSTRAINT XPKtb013_categorias PRIMARY KEY (tb013_cod_categoria)
);
Até aqui tá tudo arrumado!
-- Tabela tb012_017_compras
CREATE TABLE tb012_017_compras (
    tb012_017_cod_compra SERIAL,
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb017_cod_fornecedor integer NOT NULL,
    tb012_017_data date,
    tb012_017_quantidade NUMERIC(10),
    tb012_017_valor_unitario NUMERIC(12,2),
    CONSTRAINT XPKtb017_compras PRIMARY KEY (tb012_017_cod_compra, tb012_cod_produto, tb017_cod_fornecedor)
);
```

```
-- Tabela tb014_prd_alimentos
CREATE TABLE tb014_prd_alimentos (
    tb014_cod_prd_alimentos SERIAL,
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb014_detalhamento VARCHAR(255) NOT NULL,
    tb014_unidade_medida VARCHAR(255) NOT NULL,
    tb014_num_lote VARCHAR(255),
    tb014_data_vencimento date,
    tb014_valor_sugerido NUMERIC(10,2),
    CONSTRAINT XPKtb014_prd_alimentos PRIMARY KEY (tb014_cod_prd_alimentos, tb012_cod_produto)
);

-- Tabela tb015_prd_eletros
CREATE TABLE tb015_prd_eletros (
    tb015_cod_prd_eletro SERIAL,
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb015_detalhamento VARCHAR(255) NOT NULL,
    tb015_tensao VARCHAR(255),
    tb015_nivel_consumo_procel CHAR(1),
    tb015_valor_sugerido NUMERIC(10,2),
    CONSTRAINT XPKtb015_prd_tvs PRIMARY KEY (tb015_cod_prd_eletro, tb012_cod_produto)
);
```

Continuando a criação das tabelas:

```
-- Tabela tb016_prd_vestuarios
CREATE TABLE tb016_prd_vestuarios (
    tb016_cod_prd_vestuario SERIAL,
    tb012_cod_produto NUMERIC(10) NOT NULL,
    tb016_detalhamento VARCHAR(255) NOT NULL,
    tb016_sexo CHAR(1) NOT NULL,
    tb016_tamanho VARCHAR(255),
    tb016_numeracao NUMERIC(3),
    tb016_valor_sugerido NUMERIC(10,2),
    CONSTRAINT XPKtb016_refrigeradores PRIMARY KEY (tb016_cod_prd_vestuario, tb012_cod_produto)
);

-- Tabela tb017_fornecedores
CREATE TABLE tb017_fornecedores (
    tb017_cod_fornecedor SERIAL,
    tb017_razao_social VARCHAR(255),
    tb017_nome_fantasia VARCHAR(255),
    tb017_fone VARCHAR(15),
    tb003_cod_endereco integer,
    CONSTRAINT XPKtb017_fornecedor PRIMARY KEY (tb017_cod_fornecedor)
);

-- Tabela tb999_log
CREATE TABLE tb999_log (
    tb999_cod_log SERIAL,
    tb099_objeto VARCHAR(100) NOT NULL,
    tb999_dml VARCHAR(25) NOT NULL,
    tb999_data TIMESTAMP NOT NULL,
    CONSTRAINT XPKtb999_log PRIMARY KEY (tb999_cod_log)
);
```

Agora realizando as modificações nas tabelas (ALTER TABLE):

```
-- Procedimento para alterar as tabelas, criando as foreign keys
DO
$$
BEGIN
    -- Tabela tb002_cidades
    ALTER TABLE tb002_cidades
        ADD CONSTRAINT CONST_UF_CIDADE FOREIGN KEY (tb001_sigla_uf)
        REFERENCES tb001_uf(tb001_sigla_uf)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;

    -- Tabela tb003_enderecos
    ALTER TABLE tb003_enderecos
        ADD CONSTRAINT CONST_CIDADE_END FOREIGN KEY (tb002_cod_cidade, tb001_sigla_uf)
        REFERENCES tb002_cidades(tb002_cod_cidade, tb001_sigla_uf)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;
    -- Tabela tb004_lojas
    ALTER TABLE tb004_lojas
        ADD CONSTRAINT CONST_END_LOJAS FOREIGN KEY (tb003_cod_endereco)
        REFERENCES tb003_enderecos(tb003_cod_endereco)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;
```

Continuando as modificações nas tabelas (ALTER TABLE):

```
-- Tabela tb005_006_funcionarios_cargos - Vincula Funcionários
ALTER TABLE tb005_006_funcionarios_cargos
    ADD CONSTRAINT CONST_FUNC_FUNCCARGO FOREIGN KEY (tb005_matricula)
    REFERENCES tb005_funcionarios(tb005_matricula)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb005_006_funcionarios_cargos - Vincula Cargos
ALTER TABLE tb005_006_funcionarios_cargos
    ADD CONSTRAINT CONST_CARGO_FUNCCARGO2 FOREIGN KEY (tb006_cod_cargo)
    REFERENCES tb006_cargos(tb006_cod_cargo)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb005_funcionarios - Vincula Endereços
ALTER TABLE tb005_funcionarios
    ADD CONSTRAINT CONST_END_FUNC FOREIGN KEY (tb003_cod_endereco)
    REFERENCES tb003_enderecos(tb003_cod_endereco)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb005_funcionarios - Vincula Lojas
ALTER TABLE tb005_funcionarios
    ADD CONSTRAINT CONST_LOJAS_FUNC FOREIGN KEY (tb004_cod_loja)
    REFERENCES tb004_lojas(tb004_cod_loja)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb010_012_vendas - Vincula Funcionários
ALTER TABLE tb010_012_vendas
    ADD CONSTRAINT CONST_FUNC_VENDAS FOREIGN KEY (tb005_matricula)
    REFERENCES tb005_funcionarios(tb005_matricula)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb010_012_vendas - Vincula Clientes
ALTER TABLE tb010_012_vendas
    ADD CONSTRAINT CONST_CLI_VENDAS FOREIGN KEY (tb010_cpf)
    REFERENCES tb010_clientes(tb010_cpf)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;

-- Tabela tb010_012_vendas - Vincula Produtos
ALTER TABLE tb010_012_vendas
    ADD CONSTRAINT CONST_PRD_VENDAS FOREIGN KEY (tb012_cod_produto)
    REFERENCES tb012_produtos(tb012_cod_produto)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;
```

Finalizando as modificações nas tabelas (ALTER TABLE):

```
    -- Tabela tb012_017_compras - Vincula Fornecedores
    ALTER TABLE tb012_017_compras
        ADD CONSTRAINT CONST_FORN_COMPRAS FOREIGN KEY (tb017_cod_fornecedor)
        REFERENCES tb017_fornecedores(tb017_cod_fornecedor)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;



    -- Tabela tb014_prd_alimentos - Vincula Produtos
    ALTER TABLE tb014_prd_alimentos
        ADD CONSTRAINT CONST_PRD_ALIM FOREIGN KEY (tb012_cod_produto)
        REFERENCES tb012_produtos(tb012_cod_produto)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;

    -- Tabela tb015_prd_eletros - Vincula Produtos
    ALTER TABLE tb015_prd_eletros
        ADD CONSTRAINT CONST_PRD_ELET FOREIGN KEY (tb012_cod_produto)
        REFERENCES tb012_produtos(tb012_cod_produto)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;

    -- Tabela tb016_prd_vestuarios - Vincula Produtos
    ALTER TABLE tb016_prd_vestuarios
        ADD CONSTRAINT CONST_PRD_VEST FOREIGN KEY (tb012_cod_produto)
        REFERENCES tb012_produtos(tb012_cod_produto)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;

    -- Tabela tb017_fornecedores - Vincula Endereços
    ALTER TABLE tb017_fornecedores
        ADD CONSTRAINT CONST_END_FORN FOREIGN KEY (tb003_cod_endereco)
        REFERENCES tb003_enderecos(tb003_cod_endereco)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION;

END
$$;
```

Realizando as alterações nos dados de CPF na tabela (UPDATE TABLE):

```
1536  DO
1537  $$
1538  BEGIN
1539    UPDATE tb005_funcionarios
1540    SET tb005_cpf = REPLACE(REPLACE(tb005_cpf, '.', ''), '-', '');
1541  END;
1542  $$;
```

ANEXO B – Configuração do Container Docker do DW.

Parte do Arquivo Docker Compose que faz a composição do serviço projetodw:

```yaml
  projetodw:
    image: postgres:17-alpine
    container_name: projetodw
    hostname: projetodw
    environment:
      PGPORT: 5434
      POSTGRES_DB: projetodw
      POSTGRES_USER: equipefoda
      POSTGRES_PASSWORD: "equipefoda"
    ports:
      - 55434:5434
    volumes:
      - ./projetodw/projetodw.sh:/docker-entrypoint-initdb.d/projetodw.sh
      - ./projetodw/projetodw-full.sql:/tmp/projetodw-full.sql
    networks:
      - projetodw_bd3
```

Shell script que executa um comando psql, para que na criação do container, ele já execute o Script SQL (projetodw-full.sql), que faz a configuração da extenção FDW do BDM:

File: **projetodw.sh**

```bash
#!/usr/bin/env bash

# ------------------------------------------------------------------------------ #
# Script      : projetodw.sh
# Description :
# Version     : 0.1
# Author      : Albano Roberto Drescher Von Maywitz
# Data        : 17 de outubro de 2024
# ------------------------------------------------------------------------------ #
# Use :
# ------------------------------------------------------------------------------ #
# Copyright (C) 2024, Albano <allbano@gmail.com>.
# License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
# This is free software: you are free to change and redistribute it.
# There is NO WARRANTY, to the extent permitted by law.
# ------------------------------------------------------------------------------ #


set -e
# Executa o arquivo init.sql usando psql
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" -f "/tmp/projetodw-full.sql"

# Executa o comando original do entrypoint do PostgreSQL
exec "$@"
```

Script SQL (projetodw-full.sql), que no primeiro procedimento standalone (DO), começa configurando o FDW, conforme figura abaixo:

File: **projetodw-full.sql**

```sql
-- Habilitar a extensão FDW
DO
$$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_extension WHERE extname = 'postgres_fdw') THEN
    CREATE EXTENSION postgres_fdw;
  END IF;
END
$$;

-- Criar o servidor para o contêiner lojadevarejo
DO
$$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_foreign_server WHERE srvname = 'lojadevarejo') THEN
    CREATE SERVER lojadevarejo
    FOREIGN DATA WRAPPER postgres_fdw
    --OPTIONS (host 'lojadevarejo', dbname 'lojadevarejo', port '5432');
    OPTIONS (host 'lojadevarejo', dbname 'lojadevarejo', port '5432');
  END IF;
END
$$;

-- Criar o mapeamento de usuário
DO
$$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_user_mappings
      WHERE srvname = 'lojadevarejo'
      AND umuser = (SELECT usesysid FROM pg_user WHERE usename = 'equipefoda')) THEN
    CREATE USER MAPPING FOR equipefoda
    SERVER lojadevarejo
    OPTIONS (user 'equipefoda', password 'equipefoda');
  END IF;
END
$$;
```

```sql

-- Importar o esquema public
DO
$$
BEGIN
  IF EXISTS (SELECT 1 FROM information_schema.schemata WHERE schema_name = 'public') THEN
    IMPORT FOREIGN SCHEMA public
    FROM SERVER lojadevarejo
    INTO public;
  END IF;
END
$$;
```

Ainda no mesmo script citado acima, faz a criação das tabelas(CREATE TABLE) do projetodw:

```
-- Criação das Tabelas do Data Warehouse(DW)
DO
$$
BEGIN
-- CREATE TABLE
CREATE TABLE dim_Tipo (
    tp_id INTEGER,
    tp_descricao VARCHAR(50),
    fk_Tipo_tp_id INTEGER,
    CONSTRAINT pk_dim_tipo PRIMARY KEY (tp_id)
);

CREATE TABLE dim_Categoria (
    ct_id INTEGER,
    ct_descricao VARCHAR(100),
    CONSTRAINT pd_dim_categoria PRIMARY KEY (ct_id)
);

CREATE TABLE dim_Cliente (
    cl_cpf CHARACTER(15),
    cl_nome VARCHAR(100),
    CONSTRAINT pk_dim_clientes PRIMARY KEY (cl_cpf)
);

CREATE TABLE dim_Funcionario (
    fn_matricula INTEGER,
    fn_nome VARCHAR(100),
    CONSTRAINT pk_dim_funcionario PRIMARY KEY (fn_matricula)
);

CREATE TABLE dim_Lojas (
    lj_id INTEGER,
    lj_rua VARCHAR(255),
    lj_cidade VARCHAR(50),
    CONSTRAINT pk_dim_loja PRIMARY KEY (lj_id)
);

CREATE TABLE dim_Produto (
    pd_id INTEGER,
    pd_descricao VARCHAR(60),
    CONSTRAINT pk_dim_produto PRIMARY KEY (pd_id)
);

CREATE TABLE fat_Lucro (
    lc_lucro NUMERIC(10,2),
    lc_pd_id_produto INTEGER
);
```

Continuando a criação das tabelas:

```
97
98  CREATE TABLE fat_Vendas (
99      vd_quantidade_produto INTEGER,
100     vd_valor_unitario NUMERIC(10,2),
101     vd_quantidade_funcionario INTEGER,
102     vd_pd_id_produto INTEGER,
103     vd_fn_matricula INTEGER,
104     vd_ct_id_categoria INTEGER,
105     vd_cl_cpf_cliente CHARACTER(15),
106     vd_tp_id_tipo INTEGER,
107     vd_lj_id_lojas INTEGER,
108     vd_tm_ano INTEGER,
109     vd_tm_mes INTEGER,
110     vd_tm_dia INTEGER
111 );
112 END
113 $$;
```

Agora realizando as modificações nas tabelas (ALTER TABLE) do projetodw:

```
115 DO
116 $$
117 BEGIN
118 -- ALTER TABLE
119 ALTER TABLE public.fat_lucro
120     ADD CONSTRAINT fk_pd_id_fat_lucro FOREIGN KEY (lc_pd_id_produto)
121     REFERENCES dim_produto(pd_id)
122     ON DELETE NO ACTION
123     ON UPDATE NO ACTION;
124
125 ALTER TABLE public.fat_vendas
126     ADD CONSTRAINT fk_pd_id_fat_venda FOREIGN KEY (vd_pd_id_produto)
127     REFERENCES dim_produto(pd_id)
128     ON DELETE NO ACTION
129     ON UPDATE NO ACTION;
130
131 ALTER TABLE public.fat_vendas
132     ADD CONSTRAINT fk_fn_matricula_fat_venda FOREIGN KEY (vd_fn_matricula)
133     REFERENCES dim_funcionario(fn_matricula)
134     ON DELETE NO ACTION
135     ON UPDATE NO ACTION;
136
137 ALTER TABLE public.fat_vendas
138     ADD CONSTRAINT fk_ct_id_fat_venda FOREIGN KEY (vd_ct_id_categoria)
139     REFERENCES dim_categoria(ct_id)
140     ON DELETE NO ACTION
141     ON UPDATE NO ACTION;
```

Continuando as modificações nas tabelas (ALTER TABLE) do projetodw:

```
142
143  ALTER TABLE public.fat_vendas
144      ADD CONSTRAINT fk_cl_cpf_fat_venda FOREIGN KEY (vd_cl_cpf_cliente)
145      REFERENCES dim_cliente(cl_cpf)
146      ON DELETE NO ACTION
147      ON UPDATE NO ACTION;
148
149  ALTER TABLE public.fat_vendas
150      ADD CONSTRAINT fk_tp_id_fat_venda FOREIGN KEY (vd_tp_id_tipo)
151      REFERENCES dim_tipo(tp_id)
152      ON DELETE NO ACTION
153      ON UPDATE NO ACTION;
154
155  ALTER TABLE public.fat_vendas
156      ADD CONSTRAINT fk_lj_id_fat_venda FOREIGN KEY (vd_lj_id_lojas)
157      REFERENCES dim_lojas(lj_id)
158      ON DELETE NO ACTION
159      ON UPDATE NO ACTION;
160  END
161  $$;
162
```

Agora realizando as inserções das tabelas mais simples (INSERT) do projetodw:

```
163  DO
164  $$
165  BEGIN
166  -- SIMPLE INSERTs
167  INSERT INTO public.dim_cliente(cl_cpf,cl_nome)
168      SELECT tb010_cpf, tb010_nome
169      FROM tb010_clientes;
170
171  INSERT INTO public.dim_funcionario(fn_matricula,fn_nome)
172      SELECT tb005_matricula, tb005_nome_completo
173      FROM tb005_funcionarios;
174
175  INSERT INTO public.dim_categoria(ct_id,ct_descricao)
176      SELECT tb013_cod_categoria, tb013_descricao
177      FROM tb013_categorias;
178
179  INSERT INTO public.dim_produto(pd_id,pd_descricao)
180      SELECT tb012_cod_produto, tb012_descricao
181      FROM tb012_produtos;
182
183  INSERT INTO public.dim_lojas(lj_id,lj_rua,lj_cidade)
184      SELECT l.tb004_cod_loja,t.tb003_nome_rua, c.tb002_nome_cidade
185      FROM tb002_cidades c
186      INNER JOIN tb003_enderecos t ON t.tb002_cod_cidade = c.tb002_cod_cidade
187      INNER JOIN tb004_lojas l ON l.tb003_cod_endereco = t.tb003_cod_endereco
188      ORDER BY l.tb004_cod_loja;
189
190  INSERT INTO public.dim_tipo
191      VALUES (1,'Alimentos'), (2,'Eletros'), (3,'Vestuários');
192  END
193  $$;
```

Agora realizando as inserções das tabelas mais complexas (INSERT) do projetodw.

O primeiro insert complexo é o da tabela fat_venda:

```sql
-- COMPLEX INSERTs
INSERT INTO public.fat_vendas(
    vd_quantidade_produto,
    vd_valor_unitario,
    vd_quantidade_funcionario,
    vd_pd_id_produto,
    vd_fn_matricula,
    vd_ct_id_categoria,
    vd_cl_cpf_cliente,
    vd_tp_id_tipo,
    vd_lj_id_lojas,
    vd_tm_dia,
    vd_tm_mes,
    vd_tm_ano)
SELECT
  COUNT(v.tb010_012_quantidade) as vd_quantidade_produto,
  SUM(v.tb010_012_valor_unitario) as valor_unitario,
  SUM(v.tb010_012_quantidade) as quantidade_funcionario,
  v.tb012_cod_produto,      You, 51 minutes ago • Adicionado os arquivos do estágio ET
  v.tb005_matricula,
  p.tb013_cod_categoria,
  cl.tb010_cpf,
  CASE
    WHEN v.tb012_cod_produto = a.tb012_cod_produto  THEN 1
    WHEN v.tb012_cod_produto = e.tb012_cod_produto  THEN 2
    WHEN v.tb012_cod_produto = ve.tb012_cod_produto THEN 3
  END AS vd_tp_id_tipo,
  l.tb004_cod_loja,
  EXTRACT(DAY FROM v.tb010_012_data) as dia,
  EXTRACT(MONTH FROM v.tb010_012_data) as mes,
  EXTRACT(YEAR FROM v.tb010_012_data) as ano
FROM
  tb010_012_vendas v
  INNER JOIN tb012_produtos p ON v.tb012_cod_produto = p.tb012_cod_produto
  INNER JOIN tb005_funcionarios f ON v.tb005_matricula = f.tb005_matricula
  INNER JOIN tb013_categorias c ON c.tb013_cod_categoria = p.tb013_cod_categoria
  INNER JOIN tb004_lojas l ON l.tb004_cod_loja = f.tb004_cod_loja
  LEFT JOIN tb014_prd_alimentos a ON v.tb012_cod_produto = a.tb012_cod_produto
  LEFT JOIN tb015_prd_eletros e ON v.tb012_cod_produto = e.tb012_cod_produto
  LEFT JOIN tb016_prd_vestuarios ve ON v.tb012_cod_produto = ve.tb012_cod_produto
  INNER JOIN tb010_clientes cl ON cl.tb010_cpf = v.tb010_cpf
GROUP BY CUBE (
  v.tb012_cod_produto,
  v.tb005_matricula,
  p.tb013_cod_categoria,
  l.tb004_cod_loja,
  EXTRACT(DAY FROM v.tb010_012_data),
  EXTRACT(MONTH FROM v.tb010_012_data),
  EXTRACT(YEAR FROM v.tb010_012_data),
  cl.tb010_cpf,
  vd_tp_id_tipo);
END
$$;
```

O segundo insert complexo é o da tabela fat_lucro:

```
251
252   DO
253   $$
254   BEGIN
255   -- COMPLEX INSERTs
256   INSERT INTO public.fat_lucro(lc_lucro,lc_pd_id_produto)
257   SELECT
258     SUM(tb010_012_valor_unitario - tb012_017_valor_unitario) AS lucro,
259     p.tb012_cod_produto AS produto
260   FROM
261     tb012_produtos p
262   INNER JOIN tb010_012_vendas v ON p.tb012_cod_produto = v.tb012_cod_produto
263   INNER JOIN tb012_017_compras c ON p.tb012_cod_produto = c.tb012_cod_produto
264   GROUP by GROUPING SETS (
265     (p.tb012_cod_produto,(tb010_012_valor_unitario - tb012_017_valor_unitario)),
266     (p.tb012_cod_produto),
267     ());
268   END
269   $$;
270
```