**SupremEats2011 Project**

**Grupo 1:**

Gisele Siqueira

Marcelo Dozzi Babugli

Diego Moura

Matheus Higa

Ricardo Geroto

Roberto Eyama

Designed by DALL-e

**Exercício 1 - Modelagem de Dados Relacional**

**Conceitos de Big Data**

**Definição de Big Data:** Discutido através dos três Vs—Volume, Velocidade e Variedade. Detalha como esses fatores caracterizam os dados manipulados nas modernas arquiteturas de Big Data.

**Novos Vs—Veracidade e Valor:** Introduzidos para enfatizar a importância da confiabilidade e do valor econômico dos dados.

**Arquitetura em Camadas**

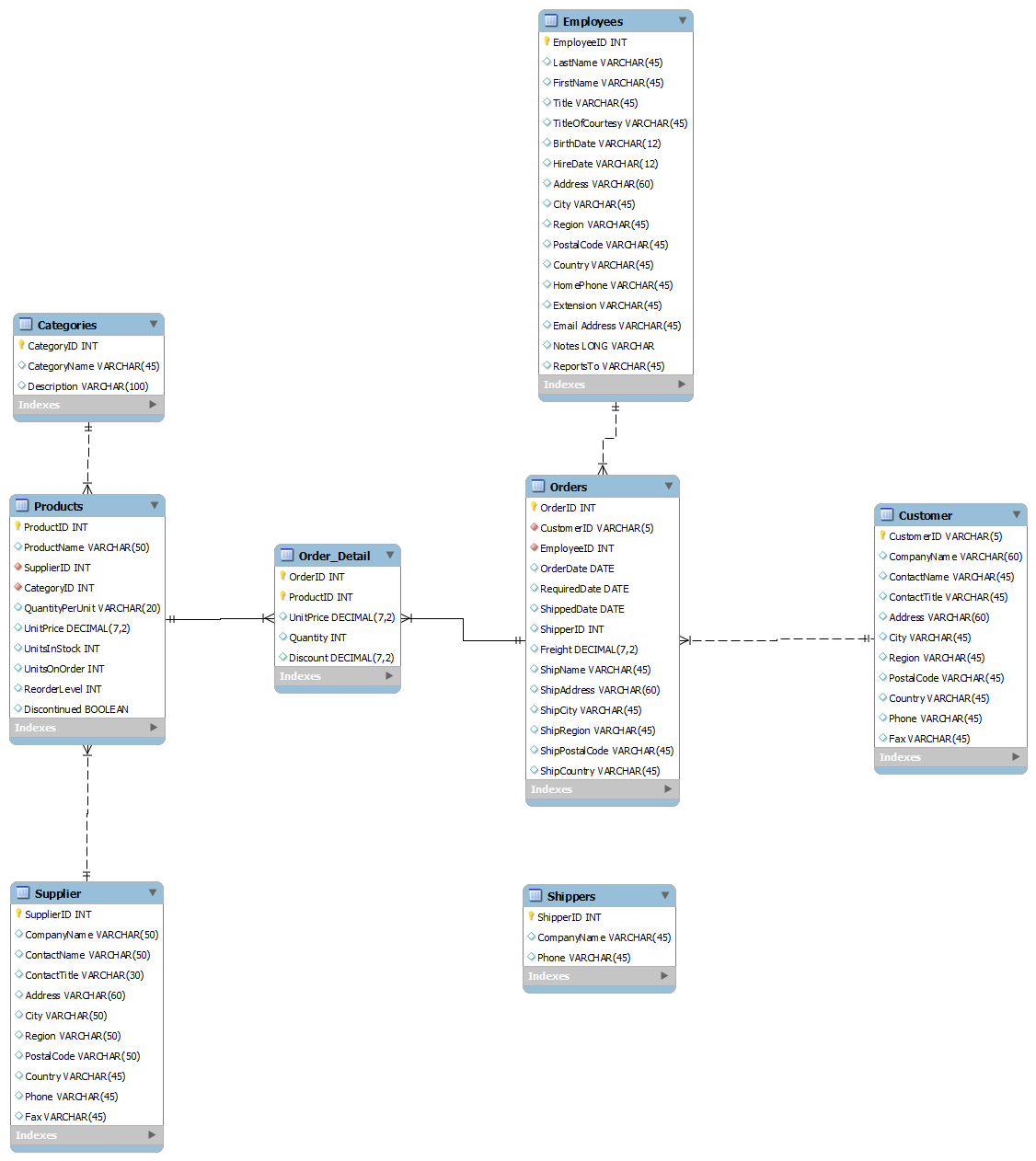
**Fontes de Dados e Área de Staging:** Descreve como os dados são coletados de diferentes fontes e inicialmente armazenados em uma área de staging antes de serem processados (camada Bronze).

**Data Warehouse (DW), Operational Data Stores (ODS), e Data Marts (DM):** Explicados como partes das camadas de informação onde os dados são estruturados para análises (Camada Silver ou Gold).

**Camada Semântica e de Consumo:** Detalha como os dados são apresentados aos usuários finais através de ferramentas de BI e analíticas (camada Gold).

Lab1:

1. Diagrama de entidade-relacionamento das tabelas (próxima página):



1. Entidades e seus atributos:

**1. Categorias**

* **Atributos:**
  + CategoryID (INT)
  + CategoryName (VARCHAR(45))
  + Description (VARCHAR(100))

**2. Clientes (Customer)**

* **Atributos:**
  + CustomerID (VARCHAR(5))
  + CompanyName (VARCHAR(60))
  + ContactName (VARCHAR(45))
  + ContactTitle (VARCHAR(45))
  + Address (VARCHAR(60))
  + City (VARCHAR(45))
  + Region (VARCHAR(45))
  + PostalCode (VARCHAR(45))
  + Country (VARCHAR(45))
  + Phone (VARCHAR(45))
  + Fax (VARCHAR(45))

**3. Empregados (Employees)**

* **Atributos:**
  + EmployeeID (INT)
  + LastName (VARCHAR(45))
  + FirstName (VARCHAR(45))
  + Title (VARCHAR(45))
  + TitleOfCourtesy (VARCHAR(45))
  + BirthDate (DATE)
  + HireDate (DATE)
  + Address (VARCHAR(60))
  + City (VARCHAR(45))
  + Region (VARCHAR(45))
  + PostalCode (VARCHAR(45))
  + Country (VARCHAR(45))
  + HomePhone (VARCHAR(45))
  + Extension (VARCHAR(45))
  + EmailAddress (VARCHAR(45))
  + Notes (LONG VARCHAR)

**4. Detalhes do Pedido (Order\_Detail)**

* **Atributos:**
  + OrderID (INT)
  + ProductID (INT)
  + UnitPrice (DECIMAL(7,2))
  + Quantity (INT)
  + Discount (DECIMAL(7,2))

**5. Pedidos (Orders)**

* **Atributos:**
  + OrderID (INT)
  + CustomerID (VARCHAR(5))
  + EmployeeID (INT)
  + OrderDate (DATE)
  + RequiredDate (DATE)
  + ShippedDate (DATE)
  + ShipperID (INT)
  + Freight (DECIMAL(7,2))
  + ShipName (VARCHAR(45))
  + ShipAddress (VARCHAR(60))
  + ShipCity (VARCHAR(45))
  + ShipRegion (VARCHAR(45))
  + ShipPostalCode (VARCHAR(45))
  + ShipCountry (VARCHAR(45))

**6. Produtos (Products)**

* **Atributos:**
  + ProductID (INT)
  + ProductName (VARCHAR(50))
  + SupplierID (INT)
  + CategoryID (INT)
  + QuantityPerUnit (VARCHAR(20))
  + UnitPrice (DECIMAL(7,2))
  + UnitsInStock (INT)
  + UnitsOnOrder (INT)
  + ReorderLevel (INT)
  + Discontinued (BOOLEAN)

**7. Transportadoras (Shippers)**

* **Atributos:**
  + ShipperID (INT)
  + OrderID (INT)
  + CompanyName (VARCHAR(45))
  + Phone (VARCHAR(45))

**8. Fornecedores (Supplier)**

* **Atributos:**
  + SupplierID (INT)
  + CompanyName (VARCHAR(50))
  + ContactName (VARCHAR(50))
  + ContactTitle (VARCHAR(30))
  + Address (VARCHAR(60))
  + City (VARCHAR(50))
  + Region (VARCHAR(50))
  + PostalCode (VARCHAR(50))
  + Country (VARCHAR(45))
  + Phone (VARCHAR(45))
  + Fax (VARCHAR(45))

**Índices**

Cada entidade possui índices que são utilizados para otimizar as operações de busca e garantir a integridade dos dados.

**Relacionamentos**

1. **Order\_Detail** se relaciona com **Orders** e **Products** através dos atributos OrderID e ProductID.
2. **Orders** se relaciona com **Customers**, **Employees**, e **Shippers** através de CustomerID, EmployeeID, e ShipperID.
3. **Products** se relaciona com **Suppliers** e **Categories** através de SupplierID e CategoryID.

**Exemplos de consultas SQL:**

1. Listar todos os pedidos com detalhes do cliente e nome do empregado:

SELECT

o.OrderID,

o.OrderDate,

c.CompanyName AS CustomerName,

e.FirstName AS EmployeeFirstName,

e.LastName AS EmployeeLastName

FROM

mydb.orders o

JOIN

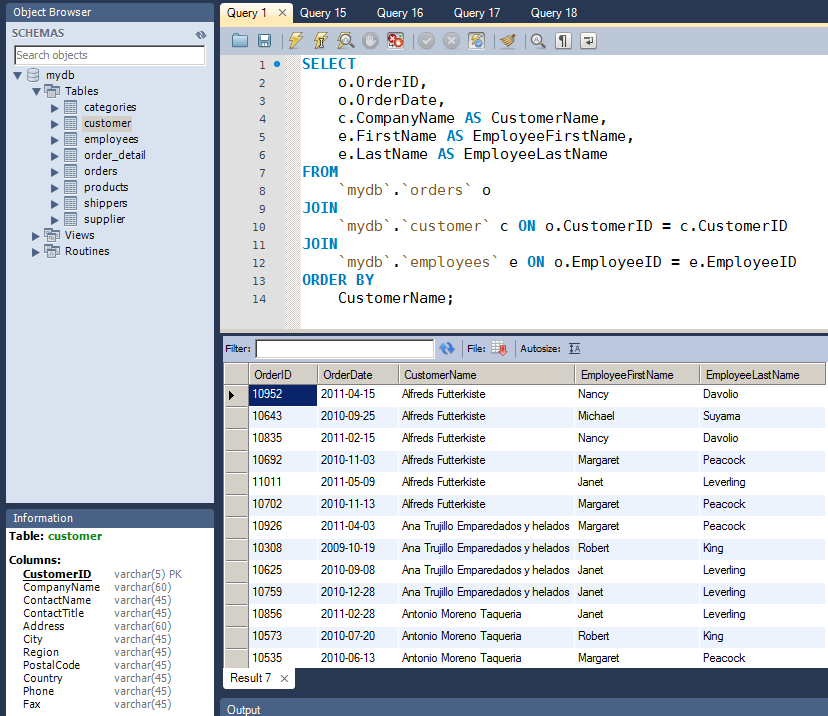
mydb.customers c ON o.CustomerID = c.CustomerID

JOIN

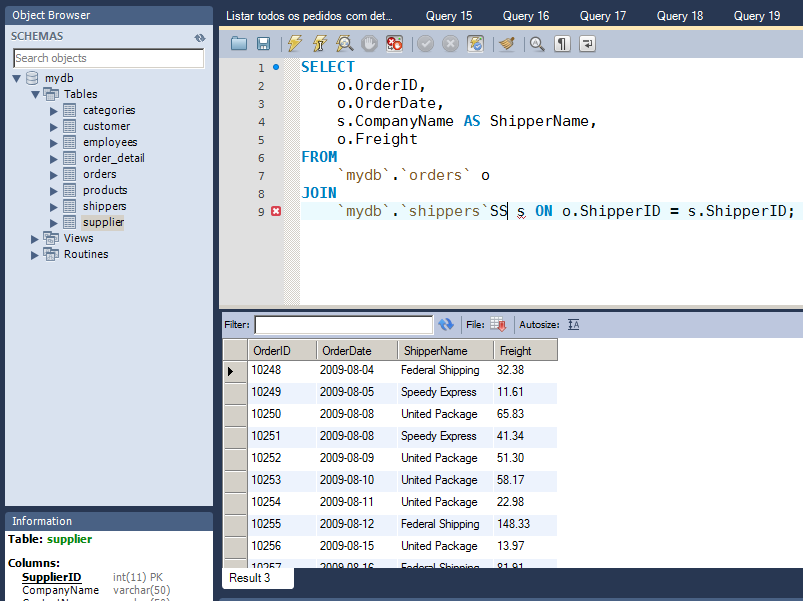
mydb.employees e ON o.EmployeeID = e.EmployeeID

ORDER BY

CustomerName;



2. Listar todos os pedidos com detalhes de envio, incluindo nome da transportadora e frete:



SELECT

o.OrderID,

o.OrderDate,

s.CompanyName AS ShipperName,

o.Freight

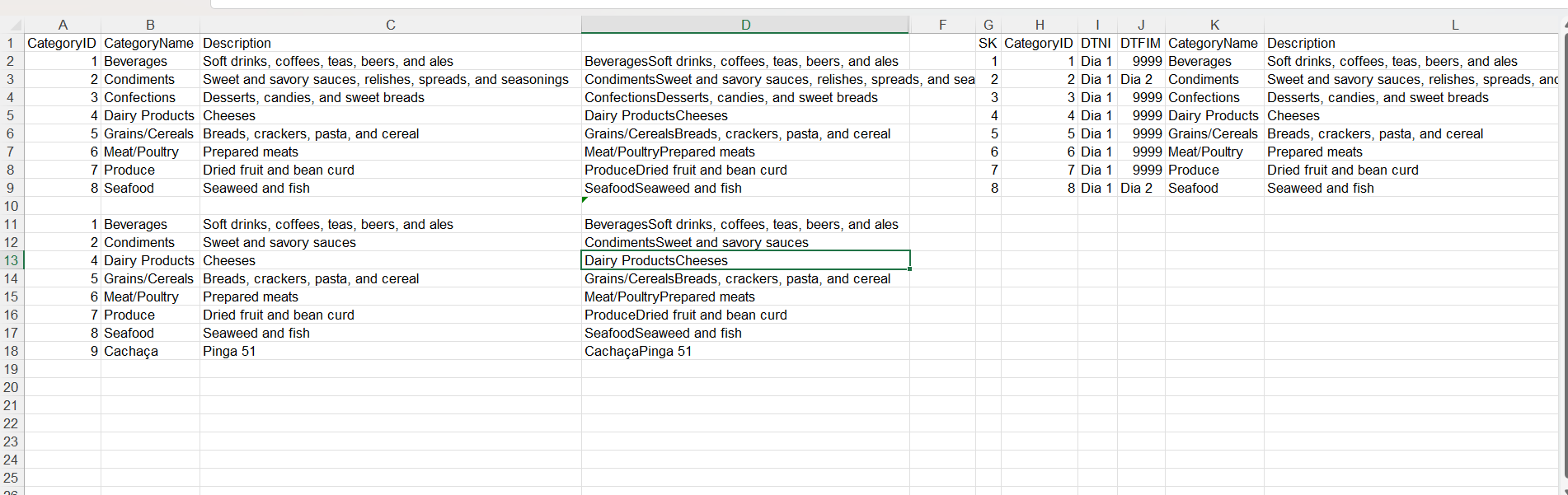
FROM

mydb.orders o

JOIN

mydb.shippers s ON o.ShipperID = s.ShipperID;

**Exercicio2 (parte 1)**

****

\* Atividade feita em classe para compreensão da importância em um DW obter os campos ‘SK’, ‘DTINIT’ e ‘DTFIM’

Em um Data Warehouse (DW), a implementação de campos como **SK** (Surrogate Key), **DTINIT** (Data de Início) e **DTFIM** (Data de Fim) é crucial para suportar efetivamente o armazenamento, a integridade e a análise histórica dos dados. Vamos explorar a importância de cada um desses campos:

**1. SK (Surrogate Key)**

* **Definição:** Uma Surrogate Key é uma chave substituta, não derivada de dados de negócio, geralmente um número sequencial (como um ID) que é usado como chave primária numa tabela de dimensão.
* **Importância:**
  + **Desacoplamento de Chaves de Negócio:** As Surrogate Keys desacoplam as chaves primárias das tabelas de dimensão das chaves de negócio, que podem mudar ao longo do tempo, garantindo estabilidade e integridade nas relações de chave estrangeira.
  + **Performance:** Elas são geralmente mais eficientes para processos de consulta e junção devido ao seu pequeno tamanho e natureza sequencial.
  + **Gerenciamento de Mudanças:** Facilitam o gerenciamento de mudanças históricas em dimensões lentamente mutáveis, permitindo múltiplas versões de um registro sem duplicar dados de negócio.

**2. DTINIT (Data de Início)**

* **Definição:** Este campo registra a data de início da validade de um registro numa tabela de dimensão, indicando quando os dados começaram a ser aplicáveis.
* **Importância:**
  + **Rastreamento Temporal:** Permite o rastreamento temporal de quando um registro se tornou relevante, crucial para análises históricas e tendências ao longo do tempo.
  + **Controle de Versão**: Em conjunto com DTFIM, ajuda a definir a linha do tempo da validade de um registro, facilitando o entendimento de mudanças ao longo do tempo.

**3. DTFIM (Data de Fim)**

* **Definição:** Este campo indica quando a validade de um registro termina ou terminou numa tabela de dimensão.
* **Importância:**
  + **Histórico de Mudanças:** Assim como o DTINIT, o DTFIM é essencial para gerenciar históricos de dados, mostrando quando um registro deixou de ser aplicável.
  + **Facilita Consultas Temporais:** Usado em consultas para filtrar dados dentro de um período específico, permitindo análises que exigem precisão temporal.
  + **Suporte para Dimensões Lentamente Mutáveis:** Em dimensões que mudam lentamente (Slowly Changing Dimensions – SCD), esses campos ajudam a manter diferentes versões de um registro sem ambiguidade.

**Aplicação Prática**

Estes campos são particularmente úteis em ambientes de data warehousing onde a consistência e a precisão dos dados históricos são críticas para a tomada de decisão baseada em dados. Eles garantem que as análises possam ser feitas com confiança na precisão dos dados, refletindo mudanças exatas ao longo do tempo e proporcionando uma base sólida para relatórios históricos e preditivos.

**Exercicio2 (parte 2)**

* + 1. **Exercício Modelagem EDW**

O exercício é referente ao gerenciamento de dados, extração, transformação, carga (ETC), Data Marts (DM), Online Analytical Processing (OLAP), Business Intelligence (BI), Business Analytics (BA), modelagem e metadados, e as estruturas de Data Warehouse (EDW) são fundamentais para a compreensão e implementação de sistemas de análise de dados eficientes. Aqui está um resumo detalhado desses conceitos:

**Script em SQL para criação do DW:**

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci ;

USE `mydb` ;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Category`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Category` (

`SKCategoryID` INT NOT NULL ,

`CategoryID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`CategoryName` CHAR(20) NULL ,

`CategoryDescription` VARCHAR(80) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKCategoryID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Supplier`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Supplier` (

`SKSupplierID` INT NOT NULL ,

`SupplierID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`CompanyName` VARCHAR(50) NULL ,

`ContactName` VARCHAR(50) NULL ,

`ContactTitle` VARCHAR(30) NULL ,

`Address` VARCHAR(50) NULL ,

`City` VARCHAR(50) NULL ,

`Region` VARCHAR(50) NULL ,

`PostalCode` CHAR(10) NULL ,

`Country` VARCHAR(50) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKSupplierID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Product`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Product` (

`SKProductID` INT NOT NULL ,

`ProductID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`ProductName` VARCHAR(50) NULL ,

`SKCategoryID` INT NOT NULL ,

`SKSupplierID` INT NOT NULL ,

`QuantityPerUnit` VARCHAR(20) NULL ,

`UnitPrice` DECIMAL(7,2) NULL ,

`UnitsInStock` INT(11) NULL ,

`UnitsOnOrder` INT(11) NULL ,

`Discontinued` CHAR(1) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKProductID`) ,

INDEX `fk\_Product\_Category\_idx` (`SKCategoryID` ASC) ,

INDEX `fk\_Product\_Supplier1\_idx` (`SKSupplierID` ASC) ,

CONSTRAINT `fk\_Product\_Category`

FOREIGN KEY (`SKCategoryID` )

REFERENCES `mydb`.`DW\_Category` (`SKCategoryID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Product\_Supplier`

FOREIGN KEY (`SKSupplierID` )

REFERENCES `mydb`.`DW\_Supplier` (`SKSupplierID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Customer`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Customer` (

`SKCustomerID` INT NOT NULL ,

`CustomerID` CHAR(5) NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`CompanyName` VARCHAR(50) NULL ,

`ContactName` VARCHAR(50) NULL ,

`ContactTitle` VARCHAR(50) NULL ,

`Address` VARCHAR(50) NULL ,

`City` VARCHAR(50) NULL ,

`Region` VARCHAR(50) NULL ,

`PostalCode` VARCHAR(50) NULL ,

`Country` VARCHAR(50) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKCustomerID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Shipper`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Shipper` (

`SKShipperID` INT NOT NULL ,

`ShipperID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`CompanyName` VARCHAR(50) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKShipperID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Employee`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Employee` (

`SKEmployeeID` INT NOT NULL ,

`EmployeeID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`LastName` VARCHAR(20) NULL ,

`FirstName` VARCHAR(20) NULL ,

`Title` VARCHAR(50) NULL ,

`BirthDate` DATE NULL ,

`HireDate` DATE NULL ,

`Salary` DECIMAL(7,2) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKEmployeeID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Store`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Store` (

`SKStoreID` INT NOT NULL ,

`StoreID` INT NOT NULL ,

`DT\_INI` DATE NULL ,

`DT\_FIM` DATE NULL ,

`DT\_LOAD` DATE NULL ,

`ManagerID` INT NOT NULL ,

`Address` VARCHAR(45) NULL ,

`City` VARCHAR(45) NULL ,

`Region` VARCHAR(45) NULL ,

`PostalCode` VARCHAR(45) NULL ,

`Country` VARCHAR(45) NULL ,

`CHECKSUM` CHAR(32) NULL ,

PRIMARY KEY (`SKStoreID`) ,

INDEX `fk\_Store\_Employee1\_idx` (`ManagerID` ASC) ,

CONSTRAINT `fk\_Store\_Employee`

FOREIGN KEY (`ManagerID` )

REFERENCES `mydb`.`DW\_Employee` (`SKEmployeeID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Order\_Logical`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Order\_Logical` (

`OrderID` INT NOT NULL ,

PRIMARY KEY (`OrderID`) )

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Order`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Order` (

`DT\_REFE` DATE NOT NULL ,

`OrderID` INT NOT NULL ,

`DT\_LOAD` DATE NULL ,

`SKCustomerID` INT NOT NULL ,

`SKEmployeeID` INT NOT NULL ,

`OrderDate` DATE NULL ,

INDEX `fk\_Customer\_has\_Product\_Customer1\_idx` (`SKCustomerID` ASC) ,

INDEX `fk\_Customer\_has\_Product\_Employee1\_idx` (`SKEmployeeID` ASC) ,

PRIMARY KEY (`DT\_REFE`, `OrderID`) ,

INDEX `fk\_DW\_Order\_DW\_Order1\_idx` (`OrderID` ASC) ,

CONSTRAINT `fk\_Customer\_has\_Product\_Customer1`

FOREIGN KEY (`SKCustomerID` )

REFERENCES `mydb`.`DW\_Customer` (`SKCustomerID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Customer\_has\_Product\_Employee1`

FOREIGN KEY (`SKEmployeeID` )

REFERENCES `mydb`.`DW\_Employee` (`SKEmployeeID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_DW\_Order\_DW\_Order1`

FOREIGN KEY (`OrderID` )

REFERENCES `mydb`.`DW\_Order\_Logical` (`OrderID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Order\_Detail`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Order\_Detail` (

`DT\_REFE` DATE NOT NULL ,

`OrderID` INT NOT NULL ,

`SKProductID` INT NOT NULL ,

`DT\_LOAD` DATE NULL ,

`UnitPrice` DECIMAL(7,2) NULL ,

`Quantity` INT(11) NULL ,

`Discount` DECIMAL(7,2) NULL ,

PRIMARY KEY (`OrderID`, `SKProductID`, `DT\_REFE`) ,

INDEX `fk\_Product\_has\_Order\_Order1\_idx` (`OrderID` ASC, `DT\_REFE` ASC) ,

INDEX `fk\_Product\_has\_Order\_Product1\_idx` (`SKProductID` ASC) ,

CONSTRAINT `fk\_Product\_has\_Order\_Product1`

FOREIGN KEY (`SKProductID` )

REFERENCES `mydb`.`DW\_Product` (`SKProductID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Product\_has\_Order\_Order1`

FOREIGN KEY (`OrderID` , `DT\_REFE` )

REFERENCES `mydb`.`DW\_Order` (`OrderID` , `DT\_REFE` )

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`DW\_Delivery`

-- -----------------------------------------------------

CREATE  TABLE IF NOT EXISTS `mydb`.`DW\_Delivery` (

`DT\_REFE` DATE NOT NULL ,

`OrderID` INT NOT NULL ,

`SKShipperID` INT NOT NULL ,

`DT\_LOAD` DATE NULL ,

`RequiredDate` DATE NULL ,

`ShippedDate` DATE NULL ,

`Freight` DECIMAL(7,2) NULL ,

`Address` VARCHAR(100) NULL ,

`City` VARCHAR(50) NULL ,

`Region` VARCHAR(50) NULL ,

`PostalCode` VARCHAR(10) NULL ,

`Country` VARCHAR(50) NULL ,

PRIMARY KEY (`DT\_REFE`, `OrderID`, `SKShipperID`) ,

INDEX `fk\_incremental\_DW\_Order1\_idx` (`OrderID` ASC) ,

INDEX `fk\_incremental\_DW\_Shipper1\_idx` (`SKShipperID` ASC) ,

CONSTRAINT `fk\_incremental\_DW\_Order1`

FOREIGN KEY (`OrderID` )

REFERENCES `mydb`.`DW\_Order\_Logical` (`OrderID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_incremental\_DW\_Shipper1`

FOREIGN KEY (`SKShipperID` )

REFERENCES `mydb`.`DW\_Shipper` (`SKShipperID` )

ON DELETE NO ACTION

ON UPDATE NO ACTION);

USE `mydb` ;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**Gerenciamento / Serviços**

* **Gerenciamento**: Inclui práticas e processos para coleta, armazenamento, manutenção e distribuição de dados.
* **Serviços**: Referem-se às ferramentas e técnicas usadas para facilitar o acesso e análise dos dados.

**Extração / Transformação / Carga (ETL)**

* **Extração**: Coleta de dados de diversas fontes.
* **Transformação**: Refinamento e reestruturação de dados conforme necessário.
* **Carga**: Inserção de dados transformados em um sistema de armazenamento ou warehouse.

**OLAP BI / BA**

* **OLAP**: Permite a análise multidimensional de dados, suportando decisões estratégicas e táticas.
* **BI**: Ferramentas e processos para análise de dados que suportam a tomada de decisão baseada em dados.
* **BA**: Utiliza técnicas analíticas avançadas para prever comportamentos futuros e guiar decisões estratégicas.

**Modelagem / Metadados**

* **Modelagem**: Processo de criação de um modelo de dados que visualiza estruturas de dados e suas relações.
* **Metadados**: Informações que descrevem vários aspectos dos dados, como estrutura, origem e uso.

**Data Warehouse (EDW)**

* **EDW**: Um sistema centralizado de armazenamento de dados projetado para apoiar decisões de negócios através da consolidação, organização e armazenamento de dados de várias fontes.

**Data Marts (DM)**

* **DM**: Subconjunto do Data Warehouse, focado em uma área específica de negócio para agilizar o acesso a dados relevantes para decisões departamentais específicas. Pode ser modelado com uma abordagem top-down a partir de um EDW ou bottom-up, começando com DM específicos para cada área de negócio.

**Tabelas Fato e Dimensões**

* **Tabelas Fato**: Armazenam medidas quantificáveis e métricas para análises, conectadas a várias dimensões.
* **Dimensões**: Estruturas que categorizam e descrevem dados dimensionais, suportando o contexto necessário para análises em tabelas fato.

**Modelos Multidimensionais**

* **Star Schema**: Modelo em que uma tabela fato central está diretamente conectada a tabelas de dimensão.
* **Snow Flake Schema**: Uma extensão do modelo star schema onde as tabelas de dimensão são normalizadas.

**Modelagem Multidimensional – Passos Básicos**

* **Definição de Prioridades e Granularidade**: Determinar o que é importante para a análise, a profundidade dos detalhes.
* **Modelagem Convolucional ER para OLAP**: Transformar um modelo relacional em um esquema que suporte processamento analítico

A computer screen shot of a computer

Description automatically generated.

A screenshot of a computer

Description automatically generated

1. imagem 1 – Prints da dela com as tabelas REL\_ e DW\_

1. Imagina 2 - Foi utilizados o EER Diagrama para desenhar as tabelas do DW a partir do nosso primeiro desenho para desenvolver o nosso DW a partir da sugestão do professor.

Explicação:

**1. Contexto Semântico**

* **Tabelas REL\_**: Este prefixo pode sugerir que as tabelas são usadas para operações relacionais regulares, talvez mais orientadas para transações ou para armazenar dados operacionais em um formato normalizado. Por exemplo, REL\_Customers, REL\_Orders, etc., podem armazenar dados de clientes e pedidos como são recebidos.
* **Tabelas DW\_**: O prefixo DW\_ sugere que as tabelas são parte de um Data Warehouse. Estas tabelas podem estar mais orientadas para análises e relatórios, e podem estar estruturadas de maneira diferente (por exemplo, desnormalizadas) para suportar queries de BI e análise de dados de forma eficiente. Exemplos incluem DW\_Customer\_Dimensions, DW\_Sales\_Facts, etc.
  + 1. **Estrutura e Modelagem**
* **Desnormalização vs. Normalização**: Tabelas no contexto de DW\_ podem ser desnormalizadas para melhorar o desempenho das consultas analíticas, o que significa que elas podem conter redundâncias e agrupamentos de dados. Tabelas REL\_, em contraste, podem ser mais normalizadas para evitar redundância e manter a integridade dos dados em um ambiente transacional.
* **Índices e Performance**: Tabelas DW\_ podem ter índices otimizados para leituras rápidas e agregações, enquanto tabelas REL\_ podem ter índices otimizados para transações rápidas, como inserções, atualizações e deletações.

**3. Uso e Aplicações**

* **Consultas de Análise vs. Operacionais**: Tabelas prefixadas com DW\_ são geralmente projetadas para suportar análises OLAP (Online Analytical Processing), enquanto as tabelas REL\_ podem ser utilizadas para processamento OLTP (Online Transaction Processing). Isso implica diferentes tipos de carga de trabalho e consultas.
* **Manutenção e Atualizações**: Tabelas em um data warehouse (DW\_) geralmente são atualizadas através de processos ETL (Extract, Transform, Load) em lotes, enquanto tabelas operacionais (REL\_) podem ser atualizadas em tempo real ou de forma transacional.

**4. Governança e Segurança de Dados**

* **Controle de Acesso**: Pode haver diferentes políticas de controle de acesso para as tabelas DW\_ em comparação com REL\_, refletindo suas diferentes utilizações e sensibilidade dos dados.

**5. Integração e Fluxo de Dados**

* **Fluxo de Dados**: Pode haver processos projetados para mover ou transformar dados das tabelas REL\_ para as DW\_, como parte de rotinas de ETL, onde os dados são limpos, transformados e carregados no warehouse para análises posteriores.

**Queires de validação:**

**Tabela – dw\_category**

A screenshot of a computer

Description automatically generated

**Tabela – rel\_category**

A screenshot of a computer

Description automatically generated

- Como podemos ver, neste exemplo, a tabela no contexto de tabela DW\_, sofre a alteração da Surrogate key = SKCaterotyID, Data de carregamento e fim e por último o CHECKSUM.

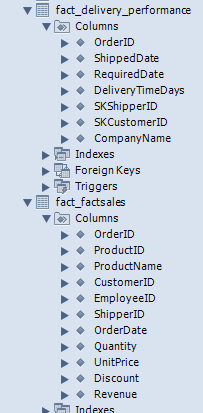
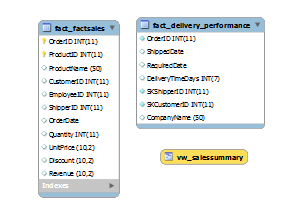
- Já na tabela do contexto Tabela REL\_, ela segue a estrutura do dado ‘raw’, o dado ingerido diratemente via CSV e/ou TXT e/ou inserido diretamente pelo MySQL Workbench.

**Conclusão do Lab2:**

A decisão de usar diferentes prefixos como REL\_ e DW\_ para tabelas no MySQL Workbench deve refletir uma estratégia de design de dados bem pensada que atenda tanto às necessidades operacionais quanto analíticas da organização. Entender e planejar conforme essas distinções pode maximizar o desempenho, escalabilidade e a utilidade do seu sistema de banco de dados, evitando quaisquer gargalos que venham ocorrer futuramente.

**Exercício 3:**

Criando as tabelas Fatos e View:



As tabelas de fatos, comumente encontradas em modelos de dados de data warehouse, são estruturas que armazenam as medidas quantitativas, métricas ou resultados de um processo de negócios. Essas tabelas são o coração de um esquema dimensional e são usadas para armazenar dados transacionais ou eventos que são analisáveis em análises de negócios.

***Aqui estão algumas características principais das tabelas de fatos:***

**Granularidade:** A granularidade de uma tabela de fatos refere-se ao nível de detalhe que os dados representam.

**Medidas:** As medidas são os dados quantitativos armazenados na tabela de fatos.

**Chaves Estrangeiras:** As tabelas de fatos contêm chaves estrangeiras que referenciam tabelas de dimensões. Estas chaves estrangeiras permitem que a tabela de fatos seja relacionada com tabelas de dimensões, que contêm dados descritivos, como informações sobre produtos, tempo, lojas, clientes etc.

As tabelas de fatos são cruciais para o suporte à tomada de decisões em ambientes de business intelligence, permitindo análises detalhadas e agregadas através da combinação com várias dimensões. Veja abaixo algumas sugestões escolhidas pelo nosso projeto.

**Exemplos das atividades:**

Tabela: **fact\_factsales**

A computer screen shot of a computer

Description automatically generated

**Descrição da Query:**

Esta consulta SQL é usada para selecionar todos os registros da tabela **fact\_factsales** no esquema **mydb**. A tabela **fact\_factsales** parece ser uma tabela fato no data **warehouse** que contém dados agregados de vendas.

**Uso da Query:**

Esta query é típica para visualização de dados ou para operações iniciais de verificação de dados em um ambiente de desenvolvimento ou teste. Ela fornece uma visão abrangente de todas as colunas da tabela **fact\_factsales,** o que é útil para entender a estrutura dos dados e para realizar análises ad hoc.

**Utilidade da Tabela fact\_factsales:**

A tabela **fact\_factsales** é crucial para análises de negócios relacionadas a vendas, permitindo que a empresa análise tendências de vendas, desempenho de produtos, eficácia de empregados, e logística de entrega. É uma tabela central em um ambiente de data warehousing para responder a perguntas estratégicas e operacionais sobre a performance de vendas.

Query:

CREATE TABLE mydb.fact\_FactSales (

OrderID INT,

ProductName VARCHAR(50),

CustomerID INT,

EmployeeID INT,

ShipperID INT,

OrderDate DATE,

Quantity INT,

UnitPrice DECIMAL(10,2),

Discount DECIMAL(10,2),

Revenue DECIMAL(10,2),

PRIMARY KEY (OrderID, ProductName)

);

INSERT INTO mydb.fact\_FactSales (OrderID, ProductName, CustomerID, EmployeeID, ShipperID, OrderDate, Quantity, UnitPrice, Discount, Revenue)

SELECT

o.OrderID,

of.ProductName,

o.SKCustomerID,

o.SKEmployeeID,

d.SKShipperID,

o.OrderDate,

od.Quantity,

od.UnitPrice,

od.Discount,

(od.Quantity \* od.UnitPrice \* (1 - od.Discount / 100)) AS Revenue

FROM

mydb.dw\_order o

JOIN

mydb.dw\_order\_detail od ON o.OrderID = od.OrderID

JOIN

mydb.dw\_product of ON of.SKProductID = od.SKProductID

JOIN

mydb.dw\_delivery d ON o.OrderID = d.OrderID;

## Número de Pedidos em Períodos Específicos:

SELECT DISTINCT COUNT(\*) AS NumberOfOrders, OrderDate

FROM mydb.fact\_FactSales

GROUP BY OrderDate;

Tabela: **fact\_delivery\_performance**

A screenshot of a computer

Description automatically generated

**Descrição da Query:**

Esta consulta SQL é usada para criar uma tabela fact**\_delivery\_performance** no esquema **mydb**. A tabela é preenchida com dados extraídos e transformados a partir de três tabelas de dimensão**: dw\_delivery, dw\_order**, e **dw\_customer.**

**Componentes da Query:**

* **CREATE TABLE mydb.fact\_delivery\_performance AS:** Cria uma nova tabela chamada fact\_delivery\_performance dentro do esquema mydb.
* **SELECT:** Inicia a seleção de colunas específicas das tabelas mencionadas para inclusão na nova tabela fato.
* **Colunas Selecionadas:**
  + d.OrderID: ID do pedido, identifica unicamente cada pedido.
  + d.ShippedDate: Data em que o pedido foi enviado.
  + d.RequiredDate: Data em que o pedido é esperado para chegar ou ser completado.
  + DATEDIFF(d.RequiredDate, d.ShippedDate) AS DeliveryTimeDays: Calcula o número de dias entre a data esperada e a data de envio, fornecendo uma métrica de desempenho de entrega.
  + d.SKShipperID: ID do transportador, chave estrangeira que referencia o transportador envolvido no envio.
  + o.SKCustomerID: ID do cliente, chave estrangeira que referência o cliente que fez o pedido.
  + c.CustomerName: Nome do cliente, extraído da tabela de clientes.
* **FROM mydb.dw\_delivery d:** Indica que os dados começarão a ser coletados a partir da tabela dw\_delivery.
* **Joins:**
  + JOIN mydb.dw\_order o ON d.OrderID = o.OrderID: Junta a tabela dw\_delivery com dw\_order utilizando OrderID para correlacionar registros relacionados ao mesmo pedido.
  + JOIN mydb.dw\_customer c ON o.SKCustomerID = c.SKCustomerID: Junta a tabela resultante com dw\_customer para adicionar informações do cliente ao conjunto de dados usando SKCustomerID.

**Uso da Tabela:**

A tabela fact\_delivery\_performance é destinada a fornecer uma visão integrada do desempenho de entrega dos pedidos, incluindo quanto tempo os pedidos levam para ser enviados versus a data em que eram esperados. Isso é útil para análise de desempenho operacional e satisfação do cliente.

Query:

CREATE TABLE mydb.fact\_delivery\_performance AS

SELECT

    d.OrderID,

    d.ShippedDate,

    DATEDIFF(d.RequiredDate, d.ShippedDate) AS DeliveryTimeDays,

    d.SKUShipperID,

    c.SKUCustomerID,

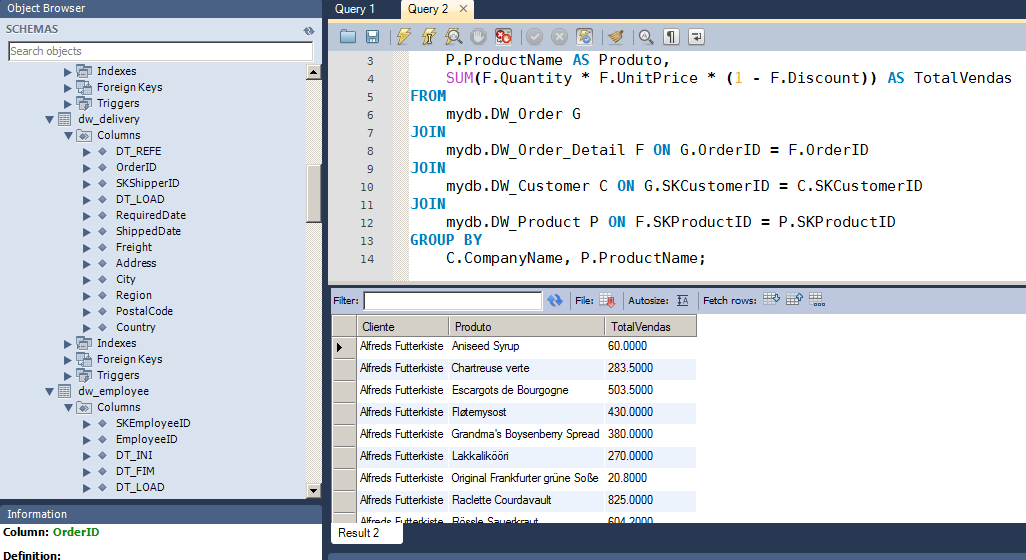
    c.CompanyName

FROM mydb.dw\_delivery d

JOIN mydb.dw\_order o ON o.OrderID = d.OrderID

JOIN mydb.dw\_customer c ON c.SKUCustomerID = d.SKUCustomerID;

**Uso da tabela DW\_Product:**



**Detalhes da Consulta:**

* **Seleção de Campos:**
  + ProductName da tabela DW\_Product é renomeado para Produto.
  + O campo TotalVendas é calculado multiplicando a quantidade pelo preço unitário e aplicando o desconto de cada item.
* **Joins:**
  + DW\_Order é unida com DW\_Order\_Detail através do OrderID.
  + DW\_Order é também unida com DW\_Customer usando SKCustomerID.
  + DW\_Order\_Detail é unida com DW\_Product por SKProductID.
* **Agrupamento:**
  + Os dados são agrupados por ProductName para fornecer a soma de vendas para cada produto.

**Resultados da Consulta**

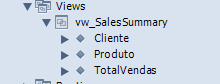
A consulta gera uma lista de produtos com o total de vendas para cada um. Por exemplo, o produto "Aniseed Syrup" teve um total de vendas de 60.00, enquanto "Raclette Courdavault" teve 825.00.

**Conclusão**

Esta consulta é fundamental para analisar o desempenho de vendas por produto, fornecendo insights valiosos sobre quais produtos são mais lucrativos ou populares. A estrutura de dados utilizada e a consulta SQL projetada são eficientes para relatórios de vendas detalhados e análise de dados dentro do ambiente de data warehouse.

Este tipo de relatório é crucial para tomada de decisões estratégicas em vendas e marketing, bem como para o ajuste de estratégias de estoque e suprimento baseadas em desempenho de produtos específicos.

**Crianção de View para consultas rápidas:** Criando uma view – **vw\_SalesSummary**



**O propósito de criar views para esta solução:**

A view **vw\_SalesSummary** foi criada para fornecer um resumo das vendas por cliente e produto. Ela simplifica a análise das vendas ao agregar dados de múltiplas tabelas e calcular o total das vendas, considerando quantidade, preço unitário e descontos aplicados. A view é especialmente útil para análises financeiras e operacionais, permitindo uma visão rápida e clara das vendas sem necessidade de execução de joins e cálculos repetitivos em consultas subsequentes.

**Estrutura**

A view vw\_SalesSummary inclui as seguintes colunas:

* **Cliente**: Nome da empresa cliente, derivado da tabela DW\_Customer.
* **Produto**: Nome do produto vendido, derivado da tabela DW\_Product.
* **TotalVendas**: Valor total das vendas para cada combinação de cliente e produto, calculado como a soma do produto da quantidade vendida, preço unitário e o complemento do desconto. Este valor é formatado como um decimal com até sete dígitos no total e duas casas decimais.

**Fontes de Dados**

* **mydb.DW\_Order**: Contém informações sobre as ordens de venda.
* **mydb.DW\_Order\_Detail**: Detalha cada item dentro de uma ordem, incluindo quantidade e preço.
* **mydb.DW\_Customer**: Armazena informações sobre os clientes.
* **mydb.DW\_Product**: Fornece detalhes dos produtos vendidos.

**Relacionamentos**

* A tabela DW\_Order é relacionada com DW\_Order\_Detail por OrderID.
* DW\_Order é também relacionada com DW\_Customer por SKCustomerID.
* DW\_Order\_Detail é relacionada com DW\_Product por SKProductID.

Query:

DROP VIEW IF EXISTS mydb.vw\_SalesSummary;

CREATE VIEW mydb.vw\_SalesSummary AS

SELECT

    C.CompanyName AS Cliente,

    P.ProductName AS Produto,

    CAST(SUM(F.Quantity \* F.UnitPrice \* (1 - F.Discount)) AS DECIMAL(7,2)) AS TotalVendas

FROM

    mydb.DW\_Order G

JOIN

    mydb.DW\_Order\_Detail F ON G.OrderID = F.OrderID

JOIN

    mydb.DW\_Customer C ON G.SKCustomerID = C.SKCustomerID

JOIN

    mydb.DW\_Product P ON F.SKProductID = P.SKProductID

GROUP BY

    C.CompanyName,

    P.ProductName;

-- Fazer uma consulta na view

SELECT \* FROM mydb.vw\_SalesSummary;

**Conslusão:**

Esta consulta retorna todos os registros da view, mostrando o total das vendas por cliente e produto, o que é ideal para relatórios e análises de desempenho de vendas.

**Dashboard de Vendas:**

**Descrição Geral**

O dashboard exibe gráficos de vendas por produto, vendas por região e vendas ao longo do tempo. A aplicação lê dados de três arquivos CSV e permite ao usuário filtrar os dados com base em intervalo de datas, produtos e regiões.

**Inicialização da Aplicação Dash (Python)**

A aplicação Dash é inicializada com um layout que inclui:

* Seletor de datas para filtrar as vendas por intervalo de datas.
* Dropdowns para seleção de produtos e regiões.
* Três gráficos interativos que exibem as vendas por produto, vendas por região e vendas ao longo do tempo.

**Callbacks para Atualização dos Gráficos**

Os gráficos são atualizados com base nas entradas do usuário nos seletores de datas e nos *dropdowns*.

**Como Usar**

Para visualizar o dashboard, execute o script em um ambiente Python que tenha as bibliotecas necessárias instaladas. Após a inicialização, acesse o servidor local especificado pelo Dash (geralmente http://127.0.0.1:8050/) em um navegador web.

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import plotly.express as px

import pandas as pd

# Carregar dados dos arquivos CSV com a codificação correta

sales\_data = pd.read\_csv('doc/dashboard/ft\_FactSales.csv', encoding='latin1')

order\_details = pd.read\_csv('doc/dashboard/oder\_details.csv', encoding='latin1')

sales\_summary = pd.read\_csv('doc/dashboard/view\_SalesSummary.csv', encoding='latin1')

categories = pd.read\_csv('doc/dashboard/categories.csv', encoding='latin1')

employees = pd.read\_csv('doc/dashboard/employee.csv', encoding='latin1')

products = pd.read\_csv('doc/dashboard/products.csv', encoding='latin1')

# Adicionar coluna fictícia de Region

sales\_summary['Region'] = sales\_summary['Cliente'].apply(lambda x: 'Region A' if 'A' in x else 'Region B')

# Mesclar dados e resolver conflitos de nomes de colunas

merged\_data = sales\_data.merge(products, on='ProductID', suffixes=('\_sales', '\_product'))

merged\_data = merged\_data.merge(categories, on='CategoryID')

merged\_data = merged\_data.merge(employees, on='EmployeeID')

# Renomear colunas para manter consistência

merged\_data['ProductName'] = merged\_data['ProductName\_sales']

# Função para criar gráfico de vendas por produto

def create\_sales\_by\_product\_chart(df):

    product\_sales = df.groupby('ProductName').agg({'Revenue': 'sum'}).reset\_index()

    fig = px.bar(product\_sales, x='ProductName', y='Revenue', title="Sales by Product")

    return fig

# Função para criar gráfico de vendas por região

def create\_sales\_by\_region\_chart(df):

    region\_sales = sales\_summary.groupby('Region').agg({'TotalVendas': 'sum'}).reset\_index()

    fig = px.bar(region\_sales, x='Region', y='TotalVendas', title="Sales by Region")

    return fig

# Função para criar gráfico de vendas ao longo do tempo

def create\_sales\_over\_time\_chart(df):

    df['OrderDate'] = pd.to\_datetime(df['OrderDate'])

    sales\_over\_time = df.groupby('OrderDate').agg({'Revenue': 'sum'}).reset\_index()

    fig = px.line(sales\_over\_time, x='OrderDate', y='Revenue', title="Sales Over Time")

    return fig

# Função para criar gráfico dos top 10 clientes

def create\_top\_customers\_chart(df):

    top\_customers = df.groupby('CustomerID').agg({'Revenue': 'sum'}).nlargest(10, 'Revenue').reset\_index()

    fig = px.bar(top\_customers, x='CustomerID', y='Revenue', title="Top 10 Customers")

    return fig

# Função para criar gráfico das melhores categorias

def create\_top\_categories\_chart(df):

    top\_categories = df.groupby('CategoryName').agg({'Revenue': 'sum'}).reset\_index()

    fig = px.bar(top\_categories, x='CategoryName', y='Revenue', title="Top Categories")

    return fig

# Layout do Dash

app = dash.Dash(\_\_name\_\_)

app.layout = html.Div(

    style={'backgroundColor': '#ADD8E6', 'padding': '20px'},

    children=[

        html.Img(src='assets/supremeats-logo.png', style={'width': '200px', 'display': 'block', 'margin-left': 'auto', 'margin-right': 'auto'}),

        html.H1(

            "Sales Dashboard",

            style={'textAlign': 'center'}

        ),

        dcc.DatePickerRange(

            id='date-picker-range',

            start\_date=sales\_data['OrderDate'].min(),

            end\_date=sales\_data['OrderDate'].max(),

            display\_format='YYYY-MM-DD'

        ),

        dcc.Dropdown(

            id='product-dropdown',

            options=[{'label': prod, 'value': prod} for prod in merged\_data['ProductName'].unique()],

            value=merged\_data['ProductName'].unique().tolist(),

            multi=True,

            placeholder="Select Products"

        ),

        dcc.Dropdown(

            id='region-dropdown',

            options=[{'label': region, 'value': region} for region in sales\_summary['Region'].unique()],

            value=sales\_summary['Region'].unique().tolist(),

            multi=True,

            placeholder="Select Regions"

        ),

        dcc.Graph(id='sales-by-product-chart'),

        dcc.Graph(id='sales-by-region-chart'),

        dcc.Graph(id='sales-over-time-chart'),

        dcc.Graph(id='top-customers-chart'),

        dcc.Graph(id='top-categories-chart')

    ]

)

# Callbacks para atualizar os gráficos com base nos filtros

@app.callback(

    [

        Output('sales-by-product-chart', 'figure'),

        Output('sales-by-region-chart', 'figure'),

        Output('sales-over-time-chart', 'figure'),

        Output('top-customers-chart', 'figure'),

        Output('top-categories-chart', 'figure')

    ],

    [

        Input('date-picker-range', 'start\_date'),

        Input('date-picker-range', 'end\_date'),

        Input('product-dropdown', 'value'),

        Input('region-dropdown', 'value')

    ]

)

def update\_charts(start\_date, end\_date, selected\_products, selected\_regions):

    filtered\_sales\_data = merged\_data[

        (merged\_data['OrderDate'] >= start\_date) &

        (merged\_data['OrderDate'] <= end\_date) &

        (merged\_data['ProductName'].isin(selected\_products))

    ]

    filtered\_sales\_summary = sales\_summary[sales\_summary['Region'].isin(selected\_regions)]

    return (

        create\_sales\_by\_product\_chart(filtered\_sales\_data),

        create\_sales\_by\_region\_chart(filtered\_sales\_summary),

        create\_sales\_over\_time\_chart(filtered\_sales\_data),

        create\_top\_customers\_chart(filtered\_sales\_data),

        create\_top\_categories\_chart(filtered\_sales\_data)

    )

# Rodar o servidor

if \_\_name\_\_ == '\_\_main\_\_':

    app.run\_server(debug=True)

**Evidencias do Dashboard:**

**Vendas por Produto:**

**A screenshot of a computer

Description automatically generated**

**Vendas por Região:**

**A blue and white rectangles

Description automatically generated**

**Vendas por período:**

**A graph showing a graph

Description automatically generated with medium confidence**

**Top 10 Compradores:**

**A blue and white bar

Description automatically generated with medium confidence**

**Melhores Categorias de Produtos:**

**A graph with a blue rectangle

Description automatically generated**

**Conclusão final do projeto:**

A conclusão deste projeto de arquitetura de Data Warehouse destaca a proficiência do grupo na implementação e compreensão das práticas essenciais de modelagem e gerenciamento de dados. Com a criação de um sistema robusto que emprega estruturas de Data Warehouse, Data Marts e processos ETL, o grupo não apenas atendeu aos requisitos técnicos, mas também demonstrou habilidade em adaptar essas tecnologias para resolver problemas reais e apoiar decisões de negócios. A integração de novas abordagens com ferramentas de Business Intelligence e Analytics assegura que a solução desenvolvida está preparada para enfrentar os desafios da big data, destacando o compromisso do grupo com a excelência e a inovação na área de dados. Este projeto não apenas reforça a teoria estudada, mas também serve como um sólido exemplo de como a tecnologia de Data Warehouse pode ser utilizada para extrair valor significativo dos dados.

Obrigado – Grupo 1!

Link github: