

SupremEats2011 Project

Grupo 1:

Gilese Siqueira
Marcelo Dozzi Babugli
Diego Moura
Matheus Higa
Ricardo Geroto
Roberto Eyama

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

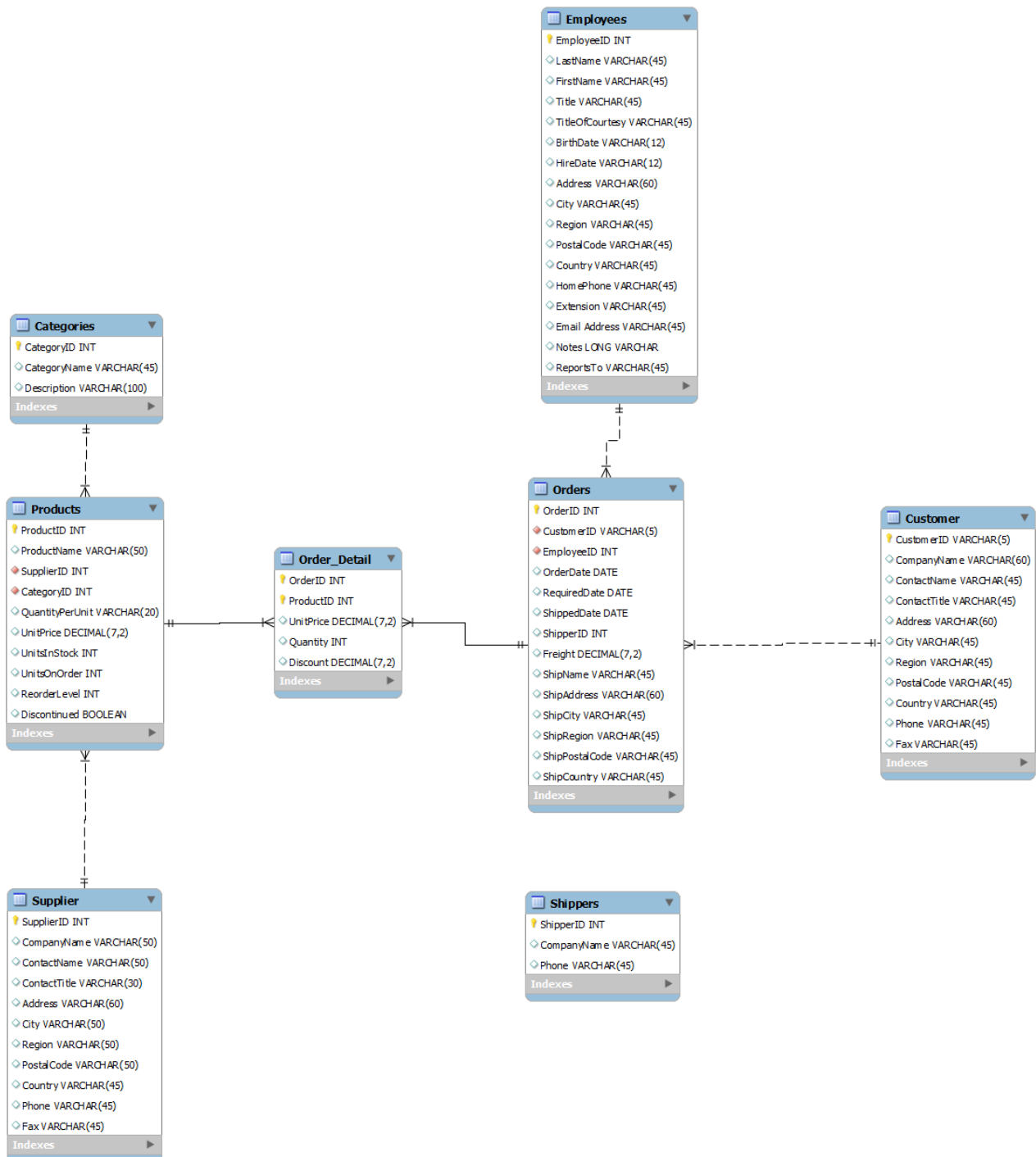
Fontes de Dados e Área de Staging: Descreve como os dados são coletados de diferentes fontes e inicialmente armazenados em uma área de staging antes de serem processados (camada Bronze).

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).

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:



2. 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

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Browser' pane displays the 'mydb' database structure, including tables like 'customer', 'employees', 'orders', and 'order_detail'. The 'Information' pane shows the schema for the 'customer' table, listing columns such as 'CustomerID', 'CompanyName', 'ContactName', 'Address', 'City', 'Region', 'PostalCode', 'Country', 'Phone', and 'Fax'. The main query window displays a SQL query that joins the 'orders' table with the 'customer' and 'employees' tables. The query selects 'o.OrderID', 'o.OrderDate', 'c.CompanyName AS CustomerName', 'e.FirstName AS EmployeeFirstName', and 'e.LastName AS EmployeeLastName' from 'mydb.orders o', joined with 'mydb.customer c' and 'mydb.employees e'. The results are ordered by 'CustomerName'. Below the query, the 'Results' pane shows a table with 7 rows and 5 columns: 'OrderID', 'OrderDate', 'CustomerName', 'EmployeeFirstName', and 'EmployeeLastName'. The first row shows OrderID 10952, OrderDate 2011-04-15, CustomerName 'Alfreds Futterkiste', EmployeeFirstName 'Nancy', and EmployeeLastName 'Davolio'.

```
SELECT
  o.OrderID,
  o.OrderDate,
  c.CompanyName AS CustomerName,
  e.FirstName AS EmployeeFirstName,
  e.LastName AS EmployeeLastName
FROM
  mydb.orders o
JOIN
  mydb.customer c ON o.CustomerID = c.CustomerID
JOIN
  mydb.employees e ON o.EmployeeID = e.EmployeeID
ORDER BY
  CustomerName;
```

OrderID	OrderDate	CustomerName	EmployeeFirstName	EmployeeLastName
10952	2011-04-15	Alfreds Futterkiste	Nancy	Davolio
10643	2010-09-25	Alfreds Futterkiste	Michael	Suyama
10835	2011-02-15	Alfreds Futterkiste	Nancy	Davolio
10692	2010-11-03	Alfreds Futterkiste	Margaret	Peacock
11011	2011-05-09	Alfreds Futterkiste	Janet	Leveling
10702	2010-11-13	Alfreds Futterkiste	Margaret	Peacock
10926	2011-04-03	Ana Trujillo Emparedados y helados	Margaret	Peacock
10308	2009-10-19	Ana Trujillo Emparedados y helados	Robert	King
10625	2010-09-08	Ana Trujillo Emparedados y helados	Janet	Leveling
10759	2010-12-28	Ana Trujillo Emparedados y helados	Janet	Leveling
10856	2011-02-28	Antonio Moreno Taqueria	Janet	Leveling
10573	2010-07-20	Antonio Moreno Taqueria	Robert	King
10535	2010-06-13	Antonio Moreno Taqueria	Margaret	Peacock

```
SELECT
  o.OrderID,
  o.OrderDate,
  c.CompanyName AS CustomerName,
  e.FirstName AS EmployeeFirstName,
  e.LastName AS EmployeeLastName
FROM
  mydb.orders o
JOIN
  mydb.customer 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

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Browser' pane displays the 'mydb' database structure, including tables like 'supplier', 'shippers', 'orders', and 'order_detail'. The 'Information' pane shows the schema for the 'supplier' table, listing columns such as 'SupplierID', 'CompanyName', 'ContactName', 'Address', 'City', 'Region', 'PostalCode', 'Country', 'Phone', and 'Fax'. The main query window displays a SQL query that joins the 'orders' table with the 'shippers' table. The query selects 'o.OrderID', 'o.OrderDate', 's.CompanyName AS ShipperName', and 'o.Freight' from 'mydb.orders o', joined with 'mydb.shippers s'. The results are ordered by 'ShipperName'. Below the query, the 'Results' pane shows a table with 13 rows and 4 columns: 'OrderID', 'OrderDate', 'ShipperName', and 'Freight'. The first row shows OrderID 10248, OrderDate 2009-08-04, ShipperName 'Federal Shipping', and Freight 32.38.

```
SELECT
  o.OrderID,
  o.OrderDate,
  s.CompanyName AS ShipperName,
  o.Freight
FROM
  mydb.orders o
JOIN
  mydb.shippers s ON o.ShipperID = s.ShipperID;
```

OrderID	OrderDate	ShipperName	Freight
10248	2009-08-04	Federal Shipping	32.38
10249	2009-08-05	Speedy Express	11.61
10250	2009-08-08	United Package	65.83
10251	2009-08-08	Speedy Express	41.34
10252	2009-08-09	United Package	51.30
10253	2009-08-10	United Package	58.17
10254	2009-08-11	United Package	22.98
10255	2009-08-12	Federal Shipping	148.33
10256	2009-08-15	United Package	13.97
10257	2009-08-16	Federal Shipping	81.61

```
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)

A		B	C	D	F	G	H	I	J	K	L
CategoryID	CategoryName	Description				SK	CategoryID	DTINI	DTFIM	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales	BeveragesSoft drinks, coffees, teas, beers, and ales			1	1	Dia 1	9999	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	CondimentsSweet and savory sauces, relishes, spreads, and sea			2	2	Dia 1	Dia 2	Condiments	Sweet and savory sauces, relishes, spreads, and
3	Confections	Desserts, candies, and sweet breads	ConfectionsDesserts, candies, and sweet breads			3	3	Dia 1	9999	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses	Dairy ProductsCheeses			4	4	Dia 1	9999	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal	Grains/CerealsBreads, crackers, pasta, and cereal			5	5	Dia 1	9999	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats	Meat/PoultryPrepared meats			6	6	Dia 1	9999	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd	ProduceDried fruit and bean curd			7	7	Dia 1	9999	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish	SeafoodSeaweed and fish			8	8	Dia 1	Dia 2	Seafood	Seaweed and fish
11	1 Beverages	Soft drinks, coffees, teas, beers, and ales	BeveragesSoft drinks, coffees, teas, beers, and ales								
12	2 Condiments	Sweet and savory sauces	CondimentsSweet and savory sauces								
13	4 Dairy Products	Cheeses	Dairy ProductsCheeses								
14	5 Grains/Cereals	Breads, crackers, pasta, and cereal	Grains/CerealsBreads, crackers, pasta, and cereal								
15	6 Meat/Poultry	Prepared meats	Meat/PoultryPrepared meats								
16	7 Produce	Dried fruit and bean curd	ProduceDried fruit and bean curd								
17	8 Seafood	Seaweed and fish	SeafoodSeaweed and fish								
18	9 Cachaça	Pinga 51	CachaçaPinga 51								
19											
20											
21											
22											
23											
24											
25											

* 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

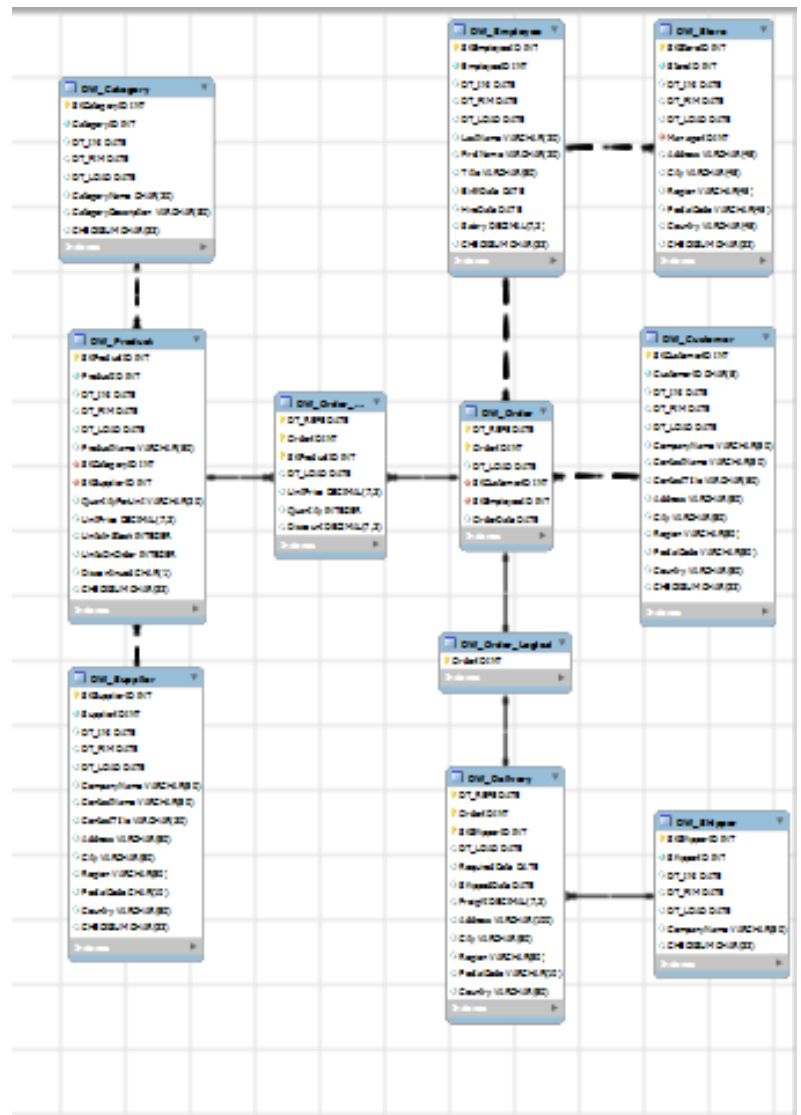
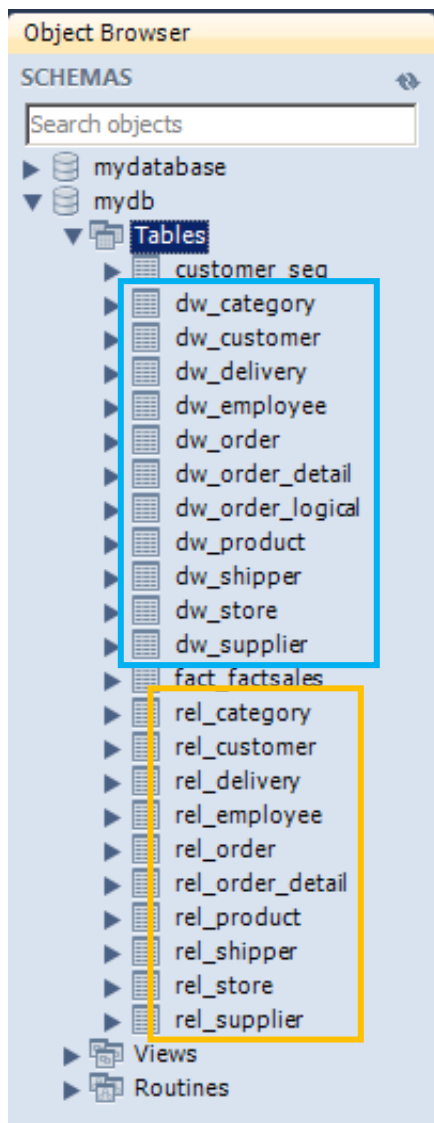
- **Tabelas Fato:** Armazenam medidas quantificáveis e métricas para análises, conectadas a várias dimensões.
- **Dimensões:** Estruturas que categorizam e descrevem dados dimensionais, suportando o contexto necessário para análises em tabelas fato.

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 Convolutacional ER para OLAP:** Transformar um modelo relacional em um esquema que suporte processamento analítico



1. imagem 1 – Prints da tela com as tabelas REL_ e DW_

1. Imagem 2 - Foi utilizado 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.

2. Estrutura e Modelagem

3.

- **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

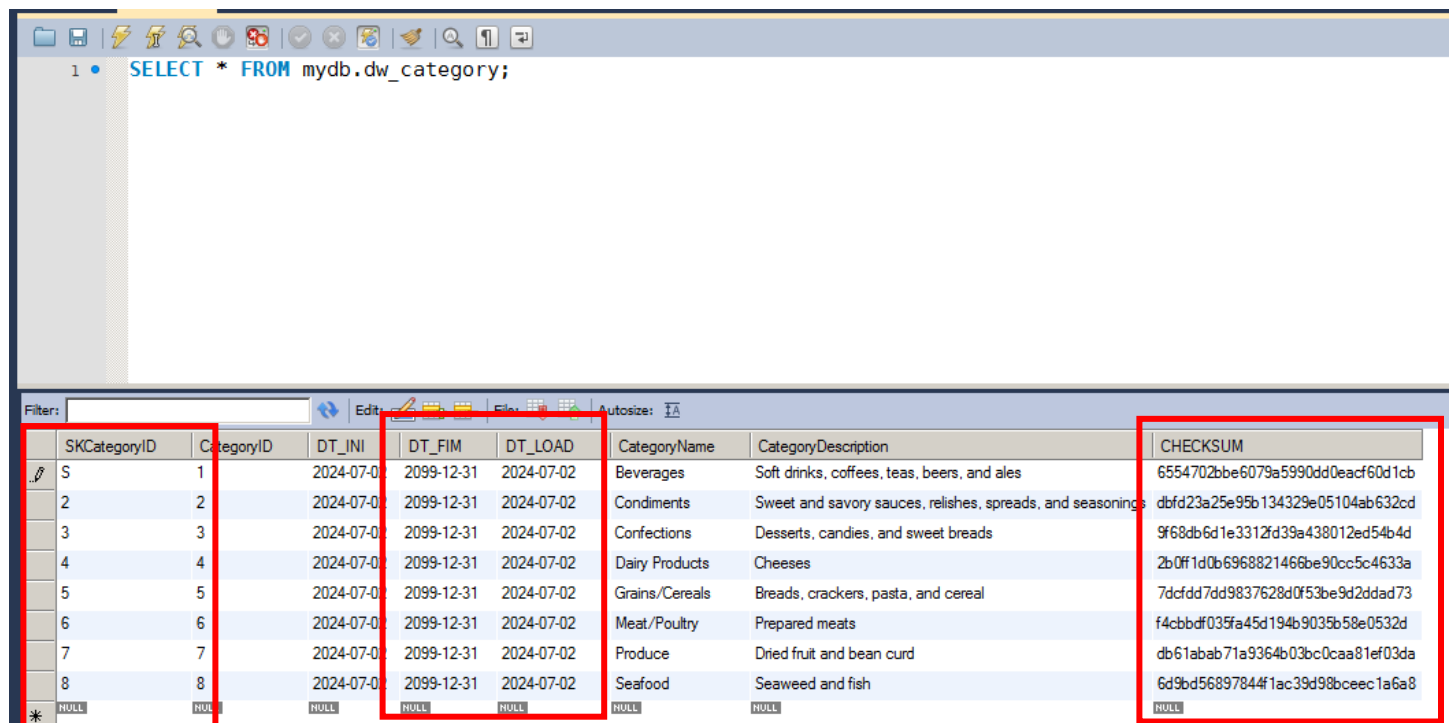
- **Controle de Acesso:** Pode haver diferentes políticas de controle de acesso para as tabelas DW_ em comparação com REL_, refletindo suas diferentes utilizações e sensibilidade dos dados.

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.

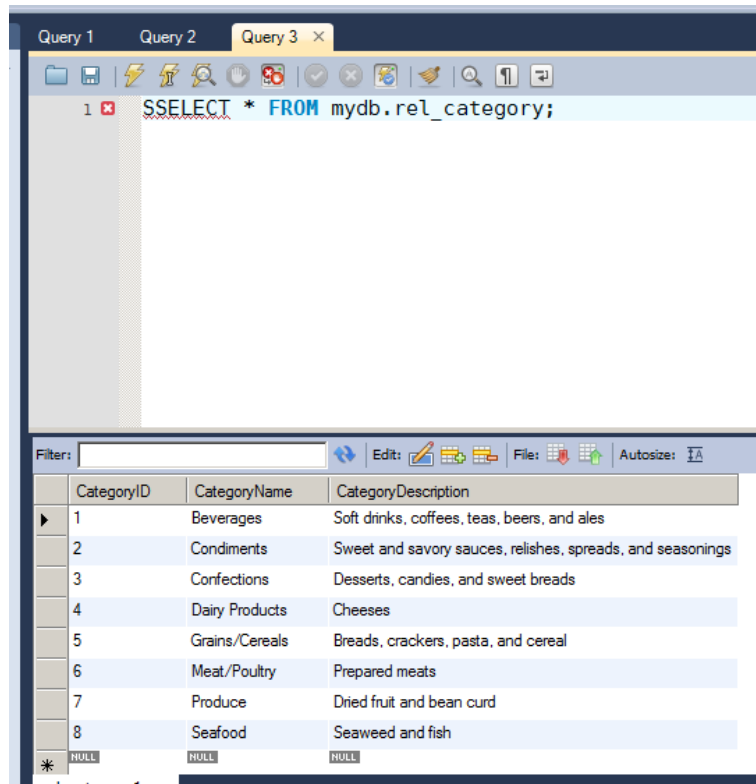
Queires de validação:

Tabela – dw_category



SKCategoryID	CategoryID	DT_INI	DT_FIM	DT_LOAD	CategoryName	CategoryDescription	CHECKSUM
1	1	2024-07-01	2099-12-31	2024-07-02	Beverages	Soft drinks, coffees, teas, beers, and ales	6554702bbe6079a5990dd0eac60d1cb
2	2	2024-07-01	2099-12-31	2024-07-02	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	dbfd23a25e95b134329e05104ab632cd
3	3	2024-07-01	2099-12-31	2024-07-02	Confections	Desserts, candies, and sweet breads	9f68db6d1e3312fd39a438012ed54b4d
4	4	2024-07-01	2099-12-31	2024-07-02	Dairy Products	Cheeses	2b0ff1d0b6968821466be90cc5c4633a
5	5	2024-07-01	2099-12-31	2024-07-02	Grains/Cereals	Breads, crackers, pasta, and cereal	7dcfdd7dd9837628d0f53be9d2ddad73
6	6	2024-07-01	2099-12-31	2024-07-02	Meat/Poultry	Prepared meats	f4cbbdf035fa45d194b9035b58e0532d
7	7	2024-07-01	2099-12-31	2024-07-02	Produce	Dried fruit and bean curd	db61abab71a9364b03bc0caa81ef03da
8	8	2024-07-01	2099-12-31	2024-07-02	Seafood	Seaweed and fish	6d9bd5689784f1ac39d98bceec1a6a8
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela – rel_category



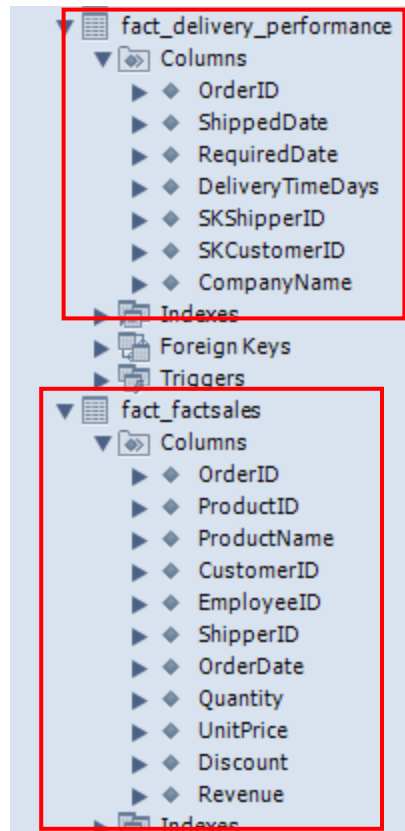
- Como podemos ver, neste exemplo, a tabela no contexto de tabela DW_, sofre a alteração da Surrogate key = SKCaterotyID, Data de carregamento e fim e por último o CHECKSUM.
- Já na tabela do contexto Tabela REL_, ela segue a estrutura do dado 'raw', o dado ingerido diretamente via CSV e/ou TXT e/ou 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:



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.

Tabela: **fact_factsales**

The screenshot shows the MySQL Workbench interface. On the left, the 'Object Browser' displays the 'mydb' database schema with various tables and views. The 'fact_factsales' table is highlighted. In the center, the 'SQL Editor' shows a query: `SELECT * FROM mydb.fact_factsales;`. On the right, the 'Query Results' pane displays the data from the `fact_factsales` table.

OrderID	ProductID	ProductName	CustomerID	EmployeeID	ShipperID	OrderDate	Quantity	UnitPrice	Discount	Revenue
10248	11	Queso Cabrales	85	5	3	2009-08-04	12	14.00	0.00	168.00
10248	42	Singaporean Hokkien Fried Mee	85	5	3	2009-08-04	10	9.80	0.00	98.00
10248	72	Mozzarella di Giovanni	85	5	3	2009-08-04	5	34.80	0.00	174.00
10249	14	Tofu	79	6	1	2009-08-05	9	18.60	0.00	167.40
10249	51	Manjimup Dried Apples	79	6	1	2009-08-05	40	42.40	0.00	1696.00
10250	41	Jack's New England Clam Chowder	34	4	2	2009-08-08	10	7.70	0.00	77.00
10250	51	Manjimup Dried Apples	34	4	2	2009-08-08	35	42.40	0.15	1481.77
10250	65	Louisiana Fiery Hot Pepper Sauce	34	4	2	2009-08-08	15	16.80	0.15	251.62
10251	22	Gustaf's Knäckebröd	84	3	1	2009-08-08	6	16.80	0.05	100.75

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 analise 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

Object Browser

Search objects

mydatabase

mydb

Tables

customer_seq

dw_category

dw_customer

dw_delivery

dw_employee

dw_order

dw_order_detail

dw_order_logical

dw_product

dw_shipper

dw_store

dw_supplier

fact_delivery_performance

Columns

OrderID

ShippedDate

RequiredDate

DeliveryTimeDays

SKShipperID

SKCustomerID

CompanyName

Indexes

Foreign Keys

Triggers

fact_factsales

rel_category

rel_customer

rel_delivery

rel_employee

rel_order

rel_order_detail

rel_product

rel_shipper

rel_store

rel_supplier

Views

Routines

Information

Column: ShippedDate

Object Info

Session

Query 1

Query 2

Query 3

Query 4

Query 5

Query 6

Query 7

1

2

3

4

5

6

7

8

9

10

11

12

CREATE TABLE mydb.fact_delivery_performance AS

SELECT

d.OrderID,

d.ShippedDate,

d.RequiredDate,

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

d.SKShipperID,

o.SKCustomerID,

c.CompanyName

FROM mydb.dw_delivery d

JOIN mydb.dw_order o ON d.OrderID = o.OrderID

JOIN mydb.dw_customer c ON o.SKCustomerID = c.SKCustomerID;

Output

History Output

Date	Time	SQL
2024-06-19	20:42:51	CREATE TABLE mydb.fact_delivery_performance AS SELE
2024-06-20	21:58:12	CREATE TABLE mydb.fact_delivery_performance AS SELE
2024-06-25	21:58:45	CREATE TABLE mydb.fact_delivery_performance AS SELE
2024-07-02	22:00:12	CREATE TABLE mydb.fact_delivery_performance AS SELE
2024-07-16	22:00:22	CREATE TABLE mydb.fact_delivery_performance AS SELE
2024-07-26	22:00:43	SELECT * FROM mydb.fact_delivery_performance

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 referencia 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.SKShipperID,
    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;
```


Tabela:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Browser' pane shows the 'dw_delivery' schema with columns: DT_REFE, OrderID, SKShipperID, DT_LOAD, RequiredDate, ShippedDate, Freight, Address, City, Region, PostalCode, and Country. Below it, the 'Information' pane shows the 'Column: OrderID' definition. The main window shows 'Query 1' with the following SQL code:

```
3 P.ProductName AS Produto,
4 SUM(F.Quantity * F.UnitPrice * (1 - F.Discount)) AS TotalVendas
5 FROM
6 mydb.DW_Order G
7 JOIN
8 mydb.DW_Order_Detail F ON G.OrderID = F.OrderID
9 JOIN
10 mydb.DW_Customer C ON G.SKCustomerID = C.SKCustomerID
11 JOIN
12 mydb.DW_Product P ON F.SKProductID = P.SKProductID
13 GROUP BY
14 C.CompanyName, P.ProductName;
```

Below the query, the 'Filter' pane is empty. The 'Results' pane shows a table with 3 columns: 'Cliente', 'Produto', and 'TotalVendas'. The data is as follows:

Cliente	Produto	TotalVendas
Alfreds Futterkiste	Aniseed Syrup	60.0000
Alfreds Futterkiste	Chartreuse verte	283.5000
Alfreds Futterkiste	Escargots de Bourgogne	503.5000
Alfreds Futterkiste	Fløtemysost	430.0000
Alfreds Futterkiste	Grandma's Boysenberry Spread	380.0000
Alfreds Futterkiste	Lakkalikööri	270.0000
Alfreds Futterkiste	Original Frankfurter grüne Soße	20.8000
Alfreds Futterkiste	Raclette Courdavault	825.0000
Alfreds Futterkiste	Rösti Räucherfleisch	604.2000

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.

3. Resultados da Consulta

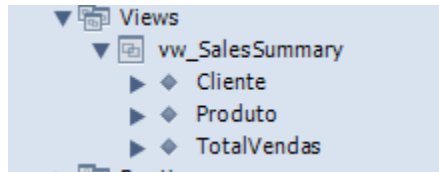
A consulta gera uma lista de produtos com o total de vendas para cada um. Por exemplo, o produto "Aniseed Syrup" teve um total de vendas de 60.00, enquanto "Raclette Courdavault" teve 825.00.

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.

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;
```

Conclusão:

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.