**Logística sob Medida**

Gerencie Melhor

Trabalho realizado no âmbito da UC de Base de Dados

Lecionada por: Norberto Luís de Jesus Albino

Aleff Aquino nº 202300054

Louis Idehen nº 202300115

ÍNDICE

Introdução

0. Texto Descritivo Do Tema

1. Modelo Entidade-Relação

2. Criando A Base De Dados

2.1 Criação Da Tabelas

3.2 Alterações Em Tabelas E Relacionamentos

3.2.1 Descrição

3.2.2 Código Sql

3.3 Inserção Dos Dados

3.3.1 Descrição

3.3.2 Código Sql

4.0 Componentes De Banco De Dados

4.1 Stored Procedures

4.1.1 Procedure Usp\_Update\_Data: Atualização De Dados Em Tabelas Específicas

4.1.2 Procedure Usp\_Validate\_Purchase: Validação e Criação De Compras:

4.1.3 Procedure Usp\_Collect\_Value: Coleta e Atribuição Do Tipo De Pacote.

4.1.4 Procedure Usp\_Rescheduleordeliver\_Package: Reagendamento Ou Entrega De Pacote.

4.2 Triggers

4.2.1 Trigger Trg\_Reschedule\_Limit: Limite De Reagendamentos

4.2.2 Trigger Trg\_Update\_Indice\_Sucesso: Atualização Do Índice De Sucesso De Entregas Por Região

4.3 Vistas

4.3.1 View Status\_Of\_Sales: Status Das Vendas

4.3.2 View Courier\_Payment: Pagamento Dos Entregadores

4.3.3 View Finances: Análise Financeira Dos Pacotes Concluídos

4.3.4 View Region\_Sucess: Análise De Sucesso De Entregas Por Região

4.3.5 View Sales\_Of\_Region: Vendas Por Região

Bibliografia

INTRODUÇÃO

Este script SQL estabelece um esquema de banco de dados abrangente para um serviço de entrega, meticulosamente projetado para gerenciar todos os aspectos do processo de entrega. O banco de dados lida com eficiência com informações relacionadas a clientes, fornecedores, pessoal de entrega e aos intrincados processos envolvidos na entrega de pacotes. O script incorpora a criação de tabelas essenciais, define relacionamentos entre essas tabelas e inclui vários procedimentos armazenados e gatilhos para automatizar e agilizar as operações. As principais funcionalidades implementadas são atualização de cadastro, criação de compras, coleta de valores e gerenciamento de cronogramas de entrega, garantindo uma gestão tranquila e eficiente dos serviços.

Características principais

Criação de Esquema: Um esquema dedicado denominado Delivery para organizar logicamente todos os objetos de banco de dados relacionados ao serviço de entrega.

Tabelas: Tabelas estruturadas para armazenar informações detalhadas sobre clientes, fornecedores, entregadores, compras, entregas, pacotes, regiões, status e ocorrências durante a entrega.

Relacionamentos: Restrições de chave estrangeira bem definidas para manter a integridade dos dados e estabelecer relacionamentos entre entidades.

Procedimentos armazenados: um conjunto de procedimentos para automatizar tarefas comuns, como atualização de informações, validação de compras, coleta de valores de pacotes e tratamento de tentativas de entrega e reagendamento.

Gatilhos: Gatilhos automatizados para impor regras de negócios, como limitar o número de tentativas de reagendamento e atualizar índices de sucesso regionais.

Benefícios

Integridade de dados: garantida por restrições abrangentes de chave estrangeira e relacionamentos estruturados entre tabelas.

Automação: Operações simplificadas por meio de procedimentos armazenados e gatilhos, reduzindo a necessidade de intervenção manual.

Eficiência: Gestão eficiente do processo de entrega, desde a validação da compra até a entrega final ou reagendamento, garantindo um alto nível de serviço.

Escalabilidade: Projetado para ser extensível, permitindo futuras melhorias e adaptações a requisitos específicos de negócios.

Ao implementar esse esquema de banco de dados robusto, os serviços de entrega podem alcançar maior eficiência operacional, gerenciamento aprimorado de dados e satisfação superior do cliente.

TEXTO DESCRITIVO DO TEMA

Uma empresa de transportes em Lisboa realiza a logística das compras dos vendedores, gerenciando a coleta e entrega dos pacotes. As compras são realizadas até às 12:00, e após o meio-dia inicia-se o processo de coleta e atribuição do tamanho dos pacotes pelos entregadores. Após a coleta e definição do tipo de pacote, o status é alterado para "em rota" e o pacote é entregue ao respectivo entregador da região.

Para monitorar o desempenho das entregas, cada região possui uma taxa de sucesso, e os gestores são responsáveis por manter essa taxa em níveis elevados. Cada cliente pode realizar várias compras, e para cada cliente são registrados nome, telefone, morada, email e NIF. Cada compra é associada a um único cliente, e os dados da compra incluem o número da fatura e a data/hora da compra. Um vendedor pode estar associado a várias compras, mas cada compra tem apenas um vendedor. As informações do vendedor incluem nome, NIF, telefone, morada e email.

Cada compra gera um pacote, que contém informações sobre o estado atual, uma data prevista de entrega (D+1¹ para compras feitas após o meio-dia e D+0² para compras feitas até o meio-dia). Cada pacote tem um tipo, registrado com seu tamanho e custo.

Os entregadores realizam coletas e entregas dos pacotes, recebendo um valor fixo por cada entrega e coleta. Caso a entrega não seja realizada, o entregador não recebe o valor da entrega nem da coleta. Deve-se registrar o motivo do insucesso, bem como quando a entrega é concluída. Um pacote pode ser recolhido por um entregador e entregue por outro. Uma entrega pode ser reagendada no máximo três vezes; após a terceira tentativa sem sucesso, o pacote é considerado cancelado. Nos reagendamentos, são registrados o motivo, a quantidade de tentativas, a data da tentativa e a nova data de entrega (D+1¹). Cada entregador está atribuído a uma região específica, e cada região pode ter vários entregadores. Cada região possui um ou mais gestores.

O lucro bruto da empresa provém do tipo de pacote entregue; pacotes maiores ou mais pesados custam mais para o vendedor. O lucro líquido é obtido subtraindo-se o custo logístico (custo de coleta e entrega) do valor do pacote.

Notas:

¹ D+1: Refere-se a entregas realizadas no dia seguinte ao pedido. Por exemplo, se o pedido é feito na segunda-feira, a entrega ocorrerá na terça-feira.

² D+0: Indica entregas feitas no mesmo dia do pedido. Por exemplo, se o pedido é feito na segunda-feira, a entrega será concluída ainda na segunda-feira.

1. Modelo Entidade-Relação

Diagrama, Desenho técnico

Descrição gerada automaticamente

MODELO - RELACIONAL

Cliente (Cliente\_id, NIF, Nome, Telefone, Email, Morada)

Compra (Compra\_id, Fatura, Data/Hora, Cliente\_id, Vendedor\_id)

Vendedor (Vendedor\_id, Nome, Telefone, NIF, Email, Morada)

Entregador (Entregador\_id, Nome, Telefone, NIF, Matricula do veículo, Email, Morada, Região\_id)

Entrega (Entrega\_id, Data de entrega, Situação\_id, Pacote\_id, Valor por Entrega, Entregador\_id)

Pacote (Pacote\_id, Status\_id, Compra\_id, Tp\_Pacote\_id, Coleta\_id, Entrega\_id, Data Prevista)

Tipo\_Pacote (Tp\_Pacote\_id, Designação, Valor)

Reagendamento (Reagendamento\_id, Nova Data, Entrega\_id, Numero Tentativa, Data Tentativa, Motivo Insucesso)

Coleta (Coleta\_id, Valor por Coleta, Entregador\_id)

Região (Região\_id, Designação, Taxa de Sucesso, Gestor\_id)

Gestor (Gestor\_id, NIF, Nome, Telefone, Email, Morada)

Status (Status\_id, Tp\_status)

Ocorrencia na entrega (Situação\_id, Tp\_ocorrencia)

Resumo das opções técnicas tomadas e objetos desenvolvidos

Para a criação e implementação da base de dados em questão, foram adotadas várias opções técnicas a fim de atender aos requisitos funcionais e garantir a eficiência do sistema. Os seguintes objetos foram desenvolvidos:

1. Modelo de Dados: Foi utilizado o modelo relacional para representar a estrutura da base de dados. Esse modelo permite a criação de tabelas relacionadas por meio de chaves primárias e estrangeiras, garantindo a integridade dos dados. Também foram definidas as entidades e seus atributos relevantes, como mecenas, requerentes, pedidos, entre outros.

2. Procedimentos Armazenados: Com o intuito de melhorar a modularidade e a segurança do sistema, foram desenvolvidos procedimentos armazenados. Esses procedimentos consistem em trechos de código SQL pré-compilados que podem ser chamados por aplicativos ou consultas, permitindo executar operações complexas de maneira mais eficiente e controlada.

3. Triggers: Foram implementados triggers para automatizar certas ações no banco de dados. Os triggers são acionados quando ocorre um evento específico, como uma inserção, atualização ou exclusão de dados em determinadas tabelas. Eles permitem a execução de ações personalizadas, como atualizar outras tabelas ou validar dados antes de serem gravados.

4. Views: As views foram criadas para fornecer diferentes perspectivas dos dados armazenados na base de dados. Elas são consultas predefinidas que retornam um conjunto de resultados virtualmente calculados a partir dos dados armazenados em tabelas. As views facilitam o acesso aos dados de forma mais conveniente e organizada, permitindo que os usuários vejam uma representação específica dos dados de acordo com suas necessidades.

2. CRIANDO A BASE DE DADOS

O código a seguir demonstra o processo de criação e manutenção de uma tabela no banco de dados utilizando a linguagem SQL. Este código inclui a definição da estrutura da tabela, especificando colunas, tipos de dados e quaisquer restrições ou chaves necessárias para garantir a integridade dos dados. Através deste código, é possível criar a tabela, inserir dados, atualizar informações existentes e realizar outras operações de manutenção necessárias para gerenciar o banco de dados de forma eficaz.

2.1 CRIAÇÃO DA TABELAS

USE T1\_AndreAleff

GO

CREATE SCHEMA Delivery

GO

CREATE TABLE Delivery.[Cliente]

(

[Cliente\_id] int NOT NULL,

[NIF] char(9) NOT NULL,

[Nome] varchar(60) NOT NULL,

[Telefone] char(9) NOT NULL,

[Email] varchar (30) NOT NULL,

[Morada] varchar (100) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Cliente\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Compra]

(

[Compra\_id] int NOT NULL,

[Fatura] int NULL,

[Data/Hora] datetime NOT NULL,

[Cliente\_id] int NOT NULL,

[Vendedor\_id] int NOT null

PRIMARY KEY CLUSTERED

(

[Compra\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Vendedor]

(

[Vendedor\_id] int NOT NULL,

[Nome] varchar (30) NOT NULL,

[Telefone] int NOT NULL,

[NIF] char(9) NOT NULL,

[Email] varchar (30) NOT NULL,

[Morada] varchar (150) NOT NULL

PRIMARY KEY CLUSTERED

(

[Vendedor\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Entregador]

(

[Entregador\_id] int NOT NULL,

[Nome] varchar(60) NOT NULL,

[Telefone] int NOT NULL,

[NIF] char(9) NOT NULL,

[ Matricula do veículo] char(9) NULL,

[Email] varchar (30) NOT NULL,

[Morada] varchar(150) NOT NULL,

[Região\_id] int NOT NULL,

PRIMARY KEY CLUSTERED

(

[Entregador\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Entrega]

(

[Entrega\_id] int NOT NULL,

[Data de entrega] datetime NULL,

[Situação\_id] int NOT NULL,

[Pacote\_id] int NOT NULL,

[Valor por Entrega] decimal (5,2) NOT NULL,

[Entregador\_id] int NOT NULL

PRIMARY KEY CLUSTERED

(

[Entrega\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Pacote]

(

[Pacote\_id] int NOT NULL,

[Status\_id] int NOT NULL,

[Compra\_id] INT NOT NULL,

[Tp\_Pacote\_id] int NULL,

[Coleta\_id] int NULL,

[Entrega\_id] int NULL,

[Data Prevista] datetime

PRIMARY KEY CLUSTERED

(

[Pacote\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Tipo\_Pacote]

(

[Tp\_Pacote\_id] int NOT NULL,

[Designação] varchar(60) NULL,

[Valor] int NULL,

PRIMARY KEY CLUSTERED

(

[Tp\_Pacote\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Reagendamento]

(

[Reagendamento\_id] int NOT NULL,

[Nova Data] date NULL,

[Entrega\_id] int NOT NULL,

[ Numero Tentativa] int NULL,

[Data Tentativa] datetime NULL,

[ Motivo Insucesso] VARCHAR (60) NULL,

PRIMARY KEY CLUSTERED

(

[Reagendamento\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Coleta]

(

[Coleta\_id] int NOT NULL,

[Valor por Coleta] decimal (5,2) NULL,

[Entregador\_id] int NOT NULL,

PRIMARY KEY CLUSTERED

(

[Coleta\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Região]

(

[Região\_id] int NOT NULL,

[Designação] varchar(60) NOT NULL,

[Taxa de Sucesso] decimal (3,2) NULL,

[Gestor\_id] int NOT NULL

PRIMARY KEY CLUSTERED

(

[Região\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Gestor]

(

[Gestor\_id] int NOT NULL,

[NIF] char(9) NOT NULL,

[Nome] varchar(60) NOT NULL,

[Telefone] int NULL,

[Email] varchar (30) NOT NULL,

[Morada] varchar (100) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Gestor\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Status]

(

[Status\_id] int NOT NULL,

[Tp\_status] varchar(60) NULL,

PRIMARY KEY CLUSTERED

(

[Status\_id] ASC

)

)

GO

CREATE TABLE Delivery.[Ocorrencia na entrega]

(

[Situação\_id] int NOT NULL,

[Tp\_ocorrencia] varchar(60) NULL,

PRIMARY KEY CLUSTERED

(

[Situação\_id] ASC

)

)

GO

3.2 ALTERAÇÕES EM TABELAS E RELACIONAMENTOS

3.2.1 DESCRIÇÃO

Nesta seção, abordamos as modificações realizadas na estrutura das tabelas e nos relacionamentos entre elas para aprimorar a integridade e a eficiência do banco de dados. As alterações a implementação de restrições de integridade referencial, como chaves estrangeiras. Essas mudanças foram efetuadas para garantir a consistência dos dados, otimizar consultas e suportar novas funcionalidades requeridas pelo sistema. O uso de comandos ALTER TABLE permitiu ajustar o esquema conforme as necessidades evolutivas do projeto, mantendo a flexibilidade e a robustez do modelo de dados.

3.2.2 CÓDIGO SQL

ALTER TABLE Delivery.[Compra] WITH CHECK ADD CONSTRAINT [FK\_Compra\_Cliente] FOREIGN KEY([Cliente\_id])

REFERENCES Delivery.[Cliente] ([Cliente\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Compra] WITH CHECK ADD CONSTRAINT [FK\_Compra\_vendedor] FOREIGN KEY([Vendedor\_id])

REFERENCES Delivery.[Vendedor] ([Vendedor\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Pacote] WITH CHECK ADD CONSTRAINT [FK\_Pacote\_Compra] FOREIGN KEY([Compra\_id])

REFERENCES Delivery.[Compra] ([Compra\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Pacote] WITH CHECK ADD CONSTRAINT [FK\_Pacote\_tp\_pacote] FOREIGN KEY([Tp\_Pacote\_id])

REFERENCES Delivery.[Tipo\_pacote] ([Tp\_Pacote\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Pacote] WITH CHECK ADD CONSTRAINT [FK\_Pacote\_Coleta] FOREIGN KEY([Coleta\_id])

REFERENCES Delivery.[Coleta] ([Coleta\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Pacote] WITH CHECK ADD CONSTRAINT [FK\_Pacote\_Status] FOREIGN KEY([Status\_id])

REFERENCEs Delivery.[Status] ([Status\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Pacote] WITH CHECK ADD CONSTRAINT [FK\_Pacote\_Entrega] FOREIGN KEY([Entrega\_id])

REFERENCES Delivery.[Entrega] ([Entrega\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Coleta] WITH CHECK ADD CONSTRAINT [FK\_Coleta\_Entregregador] FOREIGN KEY([Entregador\_id])

REFERENCES Delivery.[Entregador] ([Entregador\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Entregador] WITH CHECK ADD CONSTRAINT [FK\_Entregador\_Região] FOREIGN KEY([Região\_id])

REFERENCES Delivery.[Região] ([Região\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Região] WITH CHECK ADD CONSTRAINT [FK\_Região\_Gestor] FOREIGN KEY([Gestor\_id])

REFERENCES Delivery.[Gestor] ([Gestor\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Entrega] WITH CHECK ADD CONSTRAINT [FK\_Entrega\_Situação] FOREIGN KEY([Situação\_id])

REFERENCES Delivery.[Ocorrencia na entrega] ([Situação\_id])

ON UPDATE CASCADE

GO

ALTER TABLE Delivery.[Entrega] WITH CHECK ADD CONSTRAINT [FK\_Entrega\_Entregador] FOREIGN KEY([Entregador\_id])

REFERENCES Delivery.[Entregador] ([Entregador\_id])

GO

ALTER TABLE Delivery.[Reagendamento] WITH CHECK ADD CONSTRAINT [FK\_Reagendamento\_Entrega] FOREIGN KEY([Entrega\_id])

REFERENCES Delivery.[Entrega] ([Entrega\_id])

ON UPDATE CASCADE

GO

3.3 INSERÇÃO DOS DADOS

3.3.1 DESCRIÇÃO

Nesta seção, detalhamos o processo de inserção de dados nas tabelas do banco de dados. Realizamos a inclusão de registos nas tabelas de clientes, vendedores, entregadores e outras entidades relevantes. Cada operação de inserção foi cuidadosamente planejada para garantir a integridade referencial e a consistência dos dados. Utilizando comandos INSERT INTO, os dados foram adicionados conforme as necessidades do sistema, assegurando que todos os relacionamentos e dependências entre as tabelas fossem respeitados. Este processo é crucial para a população inicial do banco de dados e para a validação das estruturas e relacionamentos definidos previamente.

3.3.2 CÓDIGO SQL

INSERT INTO Delivery.[Região] ([Região\_id], [Designação], [Taxa de Sucesso], [Gestor\_id])

VALUES

(1, 'Lisboa Centro', 0.95, 1),

(2, 'Lisboa Norte', 0.90, 2),

(3, 'Lisboa Sul', 0.85, 3),

(4, 'Lisboa Oeste', 0.92, 4),

(5, 'Lisboa Leste', 0.88, 5);

GO

INSERT INTO Delivery.[Tipo\_Pacote] ([Tp\_Pacote\_id], [Designação], [Valor])

VALUES

(1, 'Pequeno', 10),

(2, 'Médio', 20),

(3, 'Grande', 30),

(4, 'Extra Grande', 50),

(5, 'Cancelado', 0);

GO

INSERT INTO Delivery.[Ocorrencia na entrega] ([Situação\_id], [Tp\_ocorrencia])

VALUES

(1, 'Cliente Ausente'),

(2, 'Endereço Incorreto'),

(3, 'Entrega Reagendada'),

(4, 'Pacote Danificado'),

(5, 'Entrega Bem Sucedida'),

(6, 'Em Rota');

GO

INSERT INTO Delivery.[Status] ([Status\_id], [Tp\_status])

VALUES

(1, 'Aguardando Coleta'),

(2, 'Coletado'),

(3, 'Aguardando Despacho'),

(4, 'Em Trânsito'),

(5, 'Entregue'),

(6, 'Reagendado'),

(7, 'Cancelado');

GO

INSERT INTO Delivery.[Gestor] ([Gestor\_id], [NIF], [Nome], [Telefone], [Email], [Morada])

VALUES

(1, 123456789, 'Ana Silva', '912345678', 'ana.silva@example.com', 'Rua A, Lisboa'),

(2, 987654321, 'Bruno Costa', '932145678', 'bruno.costa@example.com', 'Rua B, Lisboa'),

(3, 192837465, 'Carla Santos', '952367890', 'carla.santos@example.com', 'Rua C, Lisboa'),

(4, 564738291, 'Diogo Ferreira', '912365478', 'diogo.ferreira@example.com', 'Rua D, Lisboa'),

(5, 374829105, 'Eva Martins', '923456789', 'eva.martins@example.com', 'Rua E, Lisboa');

GO

INSERT INTO Delivery.[Cliente] ([Cliente\_id], [NIF], [Nome], [Telefone], [Email], [Morada])

VALUES

(1, '123456789', 'José Almeida', '912345678', 'jose.almeida@example.com', 'Avenida A, 10, Lisboa'),

(2, '098765432', 'Maria Ferreira', '913245678', 'maria.ferreira@example.com', 'Rua B, 20, Lisboa'),

(3, '112233445', 'Carlos Silva', '914345678', 'carlos.silva@example.com', 'Travessa C, 30, Lisboa'),

(4, '667788990', 'Ana Sousa', '915445678', 'ana.sousa@example.com', 'Alameda D, 40, Lisboa'),

(5, '554433221', 'Luís Costa', '916545678', 'luis.costa@example.com', 'Praça E, 50, Lisboa');

GO

INSERT INTO Delivery.[Vendedor] ([Vendedor\_id], [Nome], [Telefone], [NIF], [Email], [Morada])

VALUES

(1, 'Pedro Mendes', 917654321, '123456789', 'pedro.mendes@example.com', 'Rua F, 60, Lisboa'),

(2, 'Rita Santos', 918754321, '987654321', 'rita.santos@example.com', 'Rua G, 70, Lisboa'),

(3, 'João Pereira', 919854321, '192837465', 'joao.pereira@example.com', 'Rua H, 80, Lisboa'),

(4, 'Paula Rocha', 910954321, '564738291', 'paula.rocha@example.com', 'Rua I, 90, Lisboa'),

(5, 'Nuno Gomes', 911054321, '374829105', 'nuno.gomes@example.com', 'Rua J, 100, Lisboa');

GO

INSERT INTO Delivery.[Entregador] ([Entregador\_id], [Nome], [Telefone], [NIF], [Matricula do veículo], [Email], [Morada], [Região\_id])

VALUES

(1, 'Fábio Ribeiro', 920123456, '456789123', '00-AA-01', 'fabio.ribeiro@example.com', 'Rua K, 110, Lisboa', 1),

(2, 'Sofia Martins', 920223456, '654321987', '00-AA-02', 'sofia.martins@example.com', 'Rua L, 120, Lisboa', 2),

(3, 'Tiago Neves', 920323456, '789123654', '00-AA-03', 'tiago.neves@example.com', 'Rua M, 130, Lisboa', 3),

(4, 'Helena Costa', 920423456, '321987654', '00-AA-04', 'helena.costa@example.com', 'Rua N, 140, Lisboa', 4),

(5, 'Ricardo Lima', 920523456, '987321654', '00-AA-05', 'ricardo.lima@example.com', 'Rua O, 150, Lisboa', 5);

GO

4.0 COMPONENTES DE BANCO DE DADOS

4.1 STORED PROCEDURES

Conjuntos de instruções SQL que podem ser armazenados no servidor de banco de dados e executados sob demanda. Eles podem aceitar parâmetros e retornar resultados.

4.1.1 Procedure USP\_Update\_Data: Atualização de Dados em Tabelas Específicas

1. DESCRIÇÃO: A procedure USP\_Update\_Data foi criada para facilitar a atualização de registros nas tabelas Cliente, Entregador e Vendedor dentro do esquema Delivery. Esta procedure aceita vários parâmetros que representam os campos a serem atualizados e o identificador do registro. Com base no parâmetro @Tabela, que indica a tabela alvo, a procedure realiza a atualização dos dados de forma apropriada.
2. PARAMETROS:

* @Tabela (varchar(20)): Este parâmetro identifica a tabela que será atualizada. Os valores permitidos são 'Cliente', 'Entregador' e 'Vendedor',
* @Id (int): O identificador único do registro que será atualizado. Dependendo da tabela, pode ser Cliente\_id, Entregador\_id ou Vendedor\_id,
* @Nome (varchar(60)): O novo nome a ser atualizado no registro,
* @Telefone (char(9)): O novo telefone a ser atualizado no registro,
* @Email (varchar(30)): O novo email a ser atualizado no registro,
* @Morada (varchar(100)): A nova morada (endereço) a ser atualizada no registro.

1. CÓDIGO SQL :

Create or alter Procedure USP\_Update\_Data (

@Tabela varchar(20), -- Esse parâmetro identifica a tabela que será atualizada

@Id int,

@Nome varchar(60),

@Telefone char(9),

@Email varchar(30),

@Morada varchar(100))

as

begin

set nocount on

if @Tabela = 'Cliente'

UPDATE Delivery.Cliente

SET Nome = @Nome,

Telefone = @Telefone,

Email = @Email,

Morada = @Morada

WHERE Cliente\_id = @id;

else if @tabela = 'Entregador'

UPDATE Delivery.Entregador

SET Nome = @Nome,

Telefone = @Telefone,

Email = @Email,

Morada = @Morada

WHERE Entregador\_id = @id;

else if @tabela = 'Vendedor'

UPDATE Delivery.Vendedor

SET Nome = @Nome,

Telefone = @Telefone,

Email = @Email,

Morada = @Morada

WHERE Vendedor\_id = @id;

else

print('A tabela fornecida não existe, os parâmetros de tabela que podem ser utilizados são: Cliente, Entregador ou Vendedor')

end

go

1. LÓGICA DE ATUALIZACAO E EXEMPLO DE USO:

* A procedure verifica o valor do parâmetro @Tabela.
  + Se @Tabela for 'Cliente', os campos Nome, Telefone, Email e Morada serão atualizados na tabela Delivery.Cliente para o registro com o Cliente\_id correspondente a @Id.
  + Se @Tabela for 'Entregador', os mesmos campos serão atualizados na tabela Delivery.Entregador para o registro com o Entregador\_id correspondente a @Id.
  + Se @Tabela for 'Vendedor', os campos serão atualizados na tabela Delivery.Vendedor para o registro com o Vendedor\_id correspondente a @Id.
  + Caso @Tabela não corresponda a nenhuma das opções válidas ('Cliente', 'Entregador', 'Vendedor'), uma mensagem será impressa indicando que a tabela fornecida não existe, sugerindo os parâmetros corretos.
* Exemplo de uso: Para atualizar o registro de um cliente com Cliente\_id igual a 1:
  + EXEC USP\_Update\_Data 'Cliente', 1, 'Novo Nome', '123456789', 'novoemail@gmail.com', 'Nova Morada 123';

4.1.2 Procedure USP\_Validate\_Purchase: Validação e Criação de Compras:

1. DESCRICAO

A procedure USP\_Validate\_Purchase tem como objetivo validar a existência de um cliente antes de registrar uma nova compra e criar um novo pacote associado a essa compra. Esta procedure é usada para assegurar que o cliente existe no sistema antes de proceder com a operação de compra e para definir a data prevista de entrega do pacote com base no horário da compra.

1. PARAMETROS:

* @Cliente\_id (INT): Identificador único do cliente que está realizando a compra.
* @Vendedor\_id (INT): Identificador único do vendedor responsável pela venda.

1. CÓDIGO SQL:

CREATE or alter PROCEDURE USP\_Validate\_Purchase

@Cliente\_id INT,

@Vendedor\_id INT

AS

BEGIN

SET NOCOUNT ON

declare @pacoteid int

SET @pacoteid = (SELECT MAX([Pacote\_id]) FROM Delivery.Pacote)

declare @compraid int

SET @compraid = (SELECT MAX([Compra\_id]) FROM Delivery.Compra)

IF NOT EXISTS (SELECT 1 FROM Delivery.Cliente WHERE Cliente\_id = @Cliente\_id) --- se nao existe entao adicionar uma cliente

BEGIN

RAISERROR('Cliente não existe. Por favor, crie o cliente primeiro.', 16, 1) ---- dizendo que o clienete nao existe

RETURN

END

INSERT INTO Delivery.Compra (Compra\_id, Cliente\_id, Vendedor\_id, [Data/Hora])

VALUES (@compraid+1,@Cliente\_id, @Vendedor\_id, getdate());

IF DATEPART(HOUR, getdate()) < 12

BEGIN

INSERT INTO Delivery.Pacote (Pacote\_id,Status\_id, Compra\_id, [Data Prevista])

VALUES (@pacoteid+1,1, @Compraid+1, DATEADD(HOUR, 24, GETDATE()));

END

ELSE

BEGIN

INSERT INTO Delivery.Pacote (Pacote\_id,Status\_id, Compra\_id, [Data Prevista])

VALUES (@pacoteid+1,3, @Compraid+1, DATEADD(HOUR, 48, GETDATE()));

END

END

go

1. LÓGICA DE ATUALIZACAO E EXEMPLO DE USO:

Verificação da Existência do Cliente:

A procedure verifica se o cliente com o Cliente\_id fornecido existe na tabela Delivery.Cliente.

* + Se o cliente não existir, a procedure lança um erro com a mensagem 'Cliente não existe. Por favor, crie o cliente primeiro.' e interrompe a execução.

Inserção de Nova Compra:

A procedure calcula o próximo Compra\_id e insere um novo registro na tabela Delivery.Compra com o Cliente\_id, Vendedor\_id e a data/hora atual (getdate()).

Inserção de Novo Pacote:

A procedure calcula o próximo Pacote\_id.

Com base na hora atual (getdate()):

* + Se a hora for antes de meio-dia (12:00), a data prevista de entrega do pacote é definida para 24 horas a partir do momento da compra.
  + Se a hora for após meio-dia, a data prevista de entrega do pacote é definida para 48 horas a partir do momento da compra.

O status inicial do pacote é definido (1 ou 3) com base na hora da compra, e um novo registro é inserido na tabela Delivery.Pacote.

Exemplo para validar e registrar uma compra para um cliente com Cliente\_id igual a 1 e um vendedor com Vendedor\_id igual a 2:

* + EXEC USP\_Validate\_Purchase 1, 2;

4.1.3 Procedure USP\_Collect\_Value: Coleta e Atribuição do Tipo de Pacote

1. DESCRIÇÃO: A procedure USP\_Collect\_Value foi criada ou alterada para realizar a coleta de um pacote, atualizando o status e o tipo do pacote, além de registrar uma nova entrada de coleta. Abaixo está uma explicação detalhada de cada parte desta procedure.
2. PARÂMETROS:
   * @entregador\_id int: Identificador do entregador que está realizando a coleta.
   * @tipo\_pacote\_id int: Identificador do tipo de pacote.
   * @pacote\_id int: Identificador do pacote que está sendo coletado.
3. CÓDIGO SQL:

Create or alter procedure USP\_Collect\_Value(

@entregador\_id int,

@tipo\_pacote\_id int,

@pacote\_id int)

as

begin

set nocount on

declare @coleta\_id int

SET @coleta\_id = (SELECT MAX([Coleta\_id]) FROM Delivery.Coleta)

insert into Delivery.Coleta(Coleta\_id, [Valor por Coleta], Entregador\_id)

values(@coleta\_id+1,2.5, @entregador\_id)

UPDATE Delivery.Pacote

SET Status\_id = 2,

Tp\_Pacote\_id = @tipo\_pacote\_id,

Coleta\_id = @coleta\_id+1

WHERE Pacote\_id = @pacote\_id;

end

go

1. LÓGICA DE ATUALIZACAO E EXEMPLO DE USO:

Determinação do Próximo Identificador de Coleta: A procedure consulta o máximo Coleta\_id na tabela Delivery.Coleta e atribui o próximo valor disponível a @coleta\_id.

Inserção na Tabela Delivery.Coleta:

* É inserido um novo registro na tabela Delivery.Coleta com o Coleta\_id calculado, o valor fixo de coleta de 2.5 unidades monetárias e o @entregador\_id.

Atualização do Pacote:

* O pacote correspondente identificado por @pacote\_id é atualizado na tabela Delivery.Pacote: O Status\_id do pacote é atualizado para 2, indicando que o pacote está em rota para entrega.
* O Tp\_Pacote\_id é definido como @tipo\_pacote\_id, especificando o tipo de pacote coletado.
* Coleta\_id é atualizado com o identificador da coleta recém-registrada.

Exemplo para registrar a coleta de um pacote pelo entregador com @entregador\_id igual a 1, associando-o ao tipo de pacote com @tipo\_pacote\_id igual a 3 e identificado pelo @pacote\_id igual a 10:

* + EXEC USP\_Collect\_Value 1, 3, 10;

4.1.4 Procedure USP\_RescheduleOrDeliver\_Package: Reagendamento ou Entrega de Pacote.

1. DESCRIÇÃO: A procedure USP\_RescheduleOrDeliver\_Package é responsável por gerenciar as ações relacionadas ao reagendamento ou entrega de um pacote atribuído a um entregador. Dependendo da situação e da ocorrência solicitada, ela pode criar uma nova entrega caso o pacote ainda não tenha sido atribuído, finalizar uma entrega existente, ou reagendar a entrega do pacote até um máximo de três tentativas.
2. PARÂMETROS:
   * @entregador\_id (INT): Identificador único do entregador responsável pela entrega do pacote.
   * @pacote\_id (INT): Identificador único do pacote que está sendo manipulado.
   * @ocorrencia\_id (INT): Identificador da ocorrência solicitada. Pode indicar a finalização da entrega (ocorrência 5) ou o reagendamento da entrega (outras ocorrências).
3. CÓDIGO SQL:

Create or alter procedure USP\_RescheduleOrDeliver\_Package(

@entregador\_id int,

@pacote\_id int,

@ocorrencia\_id int)

as

begin

set nocount on

declare @entrega\_id int,

@reagendamento\_id int,

@num\_tentativa int,

@ocorrencia varchar(60)

select @entrega\_id = Entrega\_id

from Delivery.Pacote

where Pacote\_id = @pacote\_id

if @entrega\_id is NULL and @ocorrencia\_id = 6 -- se a entrega ainda nao esta atribuida a um pacote criamos a entrega e atribuimos o pacote a "em rota"

begin

SET @entrega\_id = (SELECT MAX(Entrega\_id) FROM Delivery.Entrega)

insert into Delivery.Entrega(Entrega\_id, Entregador\_id, Pacote\_id,Situação\_id, [Valor por Entrega])

values(@entrega\_id+1,@entregador\_id,@pacote\_id,6, 5)

UPDATE Delivery.Pacote

SET Status\_id = 4,

Entrega\_id = @entrega\_id+1

WHERE Pacote\_id = @pacote\_id;

return

end

else if @entrega\_id is NULL and @ocorrencia\_id <> 6

begin

RAISERROR('Objeto não está em rota para ocorrencia solicitada', 16, 1)

RETURN

end

else if @entrega\_id is not null and @ocorrencia\_id = 5 -- se o pacote ja estiver atribuido a uma entrega e formos finalizar a entrega isso correrá

begin

UPDATE Delivery.Entrega

SET Situação\_id = @ocorrencia\_id,

[Data de entrega] = getdate()

WHERE Entrega\_id = @entrega\_id;

UPDATE Delivery.Pacote

SET Status\_id = 5

WHERE Pacote\_id = @pacote\_id;

end

else if @entrega\_id is not null and @ocorrencia\_id <> 5 --se o pacote ja estiver atribuido a uma entrega e formos realizar uma ação que reagende o pacote.

begin

UPDATE Delivery.Entrega

SET Situação\_id = @ocorrencia\_id

WHERE Entrega\_id = @entrega\_id;

UPDATE Delivery.Pacote

SET Status\_id = 6

WHERE Pacote\_id = @pacote\_id;

set @ocorrencia = (select Tp\_ocorrencia from Delivery.[Ocorrencia na entrega] where Situação\_id = @ocorrencia\_id)

SET @reagendamento\_id = ISNULL((SELECT MAX(Reagendamento\_id) FROM Delivery.Reagendamento), 0)

SET @num\_tentativa = ISNULL((SELECT MAX([Numero Tentativa]) FROM Delivery.Reagendamento where Entrega\_id = @entrega\_id), 0)

insert into Delivery.Reagendamento(Reagendamento\_id, Entrega\_id, [Nova Data],[Numero Tentativa],[Data Tentativa], [Motivo Insucesso])

values(@reagendamento\_id+1, @entrega\_id, DATEADD(HOUR, 24, GETDATE()), @num\_tentativa+1, getdate(), @ocorrencia)

end

end;

go

1. LÓGICA DE ATUALIZACAO E EXEMPLO DE USO:

Verificação da Entrega Associada ao Pacote:

* + A procedure verifica se o pacote já está associado a uma entrega (@entrega\_id não é NULL). Caso não esteja e a ocorrência solicitada seja 6, uma nova entrega é criada para o pacote.

Criação de Nova Entrega (Se Aplicável):

* + Se o pacote não estiver associado a uma entrega e a ocorrência solicitada for 6:
    - Insere um novo registro na tabela Delivery.Entrega com o Entrega\_id calculado, associando o entregador, pacote e definindo a situação como "em rota".
    - Atualiza o Status\_id do pacote para indicar que está em rota (4).

Finalização de Entrega (Ocorrência 5):

* + Se o pacote já estiver associado a uma entrega e a ocorrência solicitada for 5:
    - Atualiza a tabela Delivery.Entrega marcando a entrega como concluída (Situação\_id = 5) e registrando a data atual como data de entrega ([Data de entrega] = GETDATE()).
    - Atualiza o Status\_id do pacote para indicar que foi entregue (5).

Reagendamento da Entrega (Outras Ocorrências):

* + Se o pacote já estiver associado a uma entrega e a ocorrência solicitada não for 5:
    - Atualiza a tabela Delivery.Entrega com a nova ocorrência solicitada.
    - Atualiza o Status\_id do pacote para indicar que a entrega foi reagendada (6).
    - Insere um registro na tabela Delivery.Reagendamento com detalhes do reagendamento, incluindo a nova data de entrega calculada para D+1, o motivo do reagendamento e a data da tentativa.

Exemplo para reagendar a entrega do pacote com @pacote\_id igual a 15, atribuído ao entregador com @entregador\_id igual a 3, devido à ocorrência de não entrega:

* EXEC USP\_RescheduleOrDeliver\_Package 3, 15, 6;

4.2 TRIGGERS

São blocos de código SQL que são automaticamente executados ou acionados em resposta a certos eventos em uma tabela ou visualização, como inserções, atualizações ou deleções de dados.

4.2.1 Trigger TRG\_Reschedule\_Limit: Limite de Reagendamentos

1. DESCRIÇÃO: A trigger TRG\_Reschedule\_Limit é acionada após a inserção ou atualização de registros na tabela Delivery.Reagendamento. Ela verifica se o número de tentativas de reagendamento para uma entrega atingiu ou excedeu o limite de 3 tentativas. Caso esse limite seja alcançado, a trigger executa ações para cancelar o pacote associado à entrega afetada.
2. CÓDIGO SQL:

CREATE TRIGGER TRG\_Reschedule\_Limit

ON Delivery.Reagendamento

AFTER INSERT, UPDATE

AS

BEGIN

set nocount on

DECLARE @Entrega\_id INT;

SELECT @Entrega\_id = Entrega\_id

FROM inserted;

-- Verifique se o número de tentativas de reagendamento é 3 ou mais

IF (SELECT MAX([Numero Tentativa]) AS N\_TENTATIVA FROM Delivery.Reagendamento WHERE Entrega\_id = @Entrega\_id) >= 3

BEGIN

-- Atualizar tipo de pacote para cancelado, isto vai zerar o valor do pacote

UPDATE Delivery.Pacote

SET Tp\_Pacote\_id = 5

WHERE Entrega\_id = @Entrega\_id;

-- Zerar o valor do pacote

UPDATE Delivery.Pacote

SET [Status\_id] = 7

WHERE Entrega\_id = @Entrega\_id;

-- Zerar o valor da coleta associada

UPDATE Delivery.Coleta

SET [Valor por Coleta] = 0

WHERE Coleta\_id = (SELECT Coleta\_id FROM Delivery.Pacote WHERE Entrega\_id = @Entrega\_id);

-- Zerar o valor da entrega associada

UPDATE Delivery.Entrega

SET [Valor por Entrega] = 0

WHERE Entrega\_id = (SELECT Entrega\_id FROM Delivery.Pacote WHERE Entrega\_id = @Entrega\_id);

    END

END;

GO

1. LÓGICA E FUNCIONAMENTO:

* Acionamento da Trigger: A trigger é ativada sempre que um novo registro é inserido ou atualizado na tabela Delivery.Reagendamento.
* Verificação de Tentativas: Ela verifica o número máximo de tentativas de reagendamento para a entrega associada ao novo registro ou atualização.
* Cancelamento do Pacote: Se o número de tentativas for igual ou superior a 3:
  + Atualiza o tipo de pacote para "cancelado" (Tp\_Pacote\_id = 5), indicando que a entrega foi cancelada devido a múltiplos reagendamentos.
  + Define o status do pacote como "cancelado" (Status\_id = 7).
* Zerar Valores Associados:
  + Zera o valor da coleta associada ao pacote, garantindo que não haja custos de coleta.
  + Zera o valor da entrega associada, assegurando que o entregador não receba pelo serviço não realizado.

4.2.2 Trigger trg\_Update\_Indice\_Sucesso: Atualização do Índice de Sucesso de Entregas por Região

1. DESCRIÇÃO

A trigger trg\_Update\_Indice\_Sucesso é acionada após a inserção ou atualização de registros na tabela Delivery.Entrega. Seu objetivo é calcular e atualizar o índice de sucesso de entregas para uma determinada região, com base nas entregas concluídas e nos reagendamentos registrados.

1. CÓDIGO SQL:

CREATE TRIGGER trg\_Update\_Indice\_Sucesso

ON Delivery.Entrega

AFTER INSERT, UPDATE

AS

BEGIN

set nocount on

DECLARE @EntregasFeitas int,

@Reagendadas int,

@Entrega\_id int,

@Região\_id int

set @Entrega\_id = (select Entrega\_id from Inserted)

set @Região\_id = (select top 1(Rg.Região\_id)

from Delivery.Reagendamento R

Join Delivery.Entrega E on R.Entrega\_id = E.Entrega\_id

join Delivery.Entregador En on E.Entregador\_id = En.Entregador\_id

join Delivery.Região Rg on En.Região\_id = Rg.Região\_id

where E.Entrega\_id = @Entrega\_id)

---- conta quantos reagendamentos tiveram na região

set @Reagendadas = (select count(isnull(R.[Data Tentativa],0))

from Delivery.Reagendamento R

Join Delivery.Entrega E on R.Entrega\_id = E.Entrega\_id

join Delivery.Entregador En on E.Entregador\_id = En.Entregador\_id

join Delivery.Região Rg on En.Região\_id = Rg.Região\_id

where Rg.Região\_id = @Região\_id)

--- conta quantas entregas foram completadas na região

set @EntregasFeitas = (select count(isnull([Data de entrega],0))

from Delivery.Entrega E

join Delivery.Entregador En on E.Entregador\_id = En.Entregador\_id

join Delivery.Região Rg on En.Região\_id = Rg.Região\_id

where Rg.Região\_id = @Região\_id)

update Delivery.Região

set [Taxa de Sucesso] = cast(@EntregasFeitas as float)/(isnull(@EntregasFeitas,0)+isnull(@Reagendadas,0))

where Região\_id = @Região\_id

END;

GO

1. LÓGICA E FUNCIONAMENTO:
   * Acionamento da Trigger: A trigger é ativada sempre que um novo registro é inserido ou atualizado na tabela Delivery.Entrega.
   * Obtenção dos Dados: Ela obtém o Entrega\_id da tabela Inserted, que contém os registros inseridos ou atualizados na tabela Delivery.Entrega.
   * Determinação da Região: Utiliza o Entrega\_id para determinar a região correspondente através de consultas nas tabelas relacionadas (Delivery.Reagendamento, Delivery.Entregador, Delivery.Região).
   * Contagem de Reagendamentos: Calcula o número de reagendamentos (@Reagendadas) para entregas na região específica.
   * Contagem de Entregas Feitas: Calcula o número de entregas feitas (@EntregasFeitas) na região, considerando apenas as entregas completas.
   * Atualização do Índice de Sucesso: Calcula a taxa de sucesso de entregas para a região com a fórmula:

4.3 Vistas

São consultas SQL armazenadas no banco de dados sob um nome e que podem ser tratadas como tabelas virtuais. Elas não armazenam dados por si mesmas, mas sim a consulta que define como os dados são apresentados.

4.3.1 View Status\_of\_sales: Status das Vendas

1. DESCRIÇÃO:

A view Status\_of\_sales fornece informações detalhadas sobre o status das vendas de pacotes realizadas pelos vendedores em diversos marketplaces. Ela apresenta dados como o ID da compra, o nome do vendedor, o nome do cliente, a data da compra e o status atual do pacote.

1. CÓDIGO SQL

create view Status\_of\_sales

as

select C.Compra\_id,

V.Nome as NomeVendedor,

CLT.nome as NomeCliente,

C.[Data/Hora] as [Data da compra],

ST.Tp\_status as [Status]

from Delivery.Vendedor V

join Delivery.Compra C on V.Vendedor\_id = C.Vendedor\_id

join Delivery.Pacote P on P.Compra\_id = C.Compra\_id

join Delivery.Tipo\_Pacote TP on TP.Tp\_Pacote\_id = P.Tp\_Pacote\_id

join Delivery.[Status] ST on P.Status\_id = ST.Status\_id

join Delivery.Coleta CL on P.Coleta\_id = CL.Coleta\_id

join Delivery.Entrega E on E.Entrega\_id = P.Entrega\_id

JOIN delivery.Cliente CLT on CLT.Cliente\_id = C.Cliente\_id

Go

1. ESTRUTURAS E RELACIONAMENTOS

* Delivery.Vendedor: Contém informações sobre os vendedores responsáveis pelas vendas.
* Delivery.Compra: Registra as informações de cada compra, incluindo o vendedor associado.
* Delivery.Pacote: Relaciona as compras aos pacotes enviados, contendo o status atual do pacote.
* Delivery.Tipo\_Pacote: Define os tipos de pacotes conforme sua categoria.
* Delivery.Status: Define os diferentes estados de status que um pacote pode ter.
* Delivery.Coleta: Registra as informações relacionadas à coleta do pacote.
* Delivery.Entrega: Contém os detalhes das entregas realizadas para cada pacote.
* Delivery.Cliente: Mantém informações dos clientes que realizaram as compras.

1. COLUNAS RETORNADAS

* Compra\_id: Identificador único da compra.
* NomeVendedor: Nome completo do vendedor responsável pela venda.
* NomeCliente: Nome completo do cliente que realizou a compra.
* Data da compra: Data e hora exatas em que a compra foi realizada.
* Status: Estado atual do pacote, indicando se está em coleta, em rota, entregue, ou outro estado específico.

1. FUNCIONAMENTO E EXEMPLO

* **Relacionamentos**: A view utiliza joins entre várias tabelas para garantir que todas as informações relevantes sejam incluídas na consulta final.
* **Status do Pacote**: Utiliza a tabela Delivery.Status para obter o status atual do pacote, proporcionando uma visão clara do estado de cada compra.
* **Exemplo**: A view pode ser consultada para obter um panorama completo das vendas realizadas por vendedores específicos, incluindo o status atual de cada pacote associado às compras.
  + SELECT \* FROM Status\_of\_sales WHERE NomeVendedor = 'Nome do Vendedor';

4.3.2 View Courier\_Payment: Pagamento dos Entregadores

1. DESCRIÇÃO

A view Courier\_Payment apresenta um resumo dos pagamentos atribuídos a cada entregador com base nos serviços de coleta e entrega realizados. Ela calcula o valor total das coletas e entregas para pacotes que foram entregues com sucesso (status 5) e os agrupa por entregador.

1. CÓDIGO SQL

create view Courier\_Payment

as

select EN.Entregador\_id as [Entregador ID],

EN.Nome as Entregador,

SUM(C.[Valor por Coleta]) as [Valor das coletas],

SUM(E.[Valor por Entrega]) as [Valor das entregas],

SUM(C.[Valor por Coleta])+SUM(E.[Valor por Entrega]) as TOTAL

from Delivery.[Status] ST

join Delivery.Pacote P on ST.Status\_id = P.Status\_id

join Delivery.Coleta C on P.Coleta\_id = C.Coleta\_id

join Delivery.Entrega E on P.Entrega\_id = E.Entrega\_id

join Delivery.Entregador EN on C.Entregador\_id = EN.Entregador\_id

where P.Status\_id = 5

group by EN.Nome, EN.Entregador\_id

go

1. ESTRUTURAS E RELACIONAMENTOS

* Delivery.[Status]: Contém os status dos pacotes, incluindo o status que indica entrega concluída.
* Delivery.Pacote: Relaciona os pacotes aos entregadores e às coletas associadas.
* Delivery.Coleta: Armazena informações sobre as coletas realizadas pelos entregadores.
* Delivery.Entrega: Registra os detalhes das entregas, incluindo os valores associados.
* Delivery.Entregador: Contém os dados dos entregadores responsáveis pelas coletas e entregas.

1. COLUNAS RETORNADAS

* Entregador ID: Identificação única do entregador.
* Entregador: Nome do entregador.
* Valor das coletas: Valor total recebido pelo entregador pelas coletas realizadas.
* Valor das entregas: Valor total recebido pelo entregador pelas entregas realizadas.
* TOTAL: Valor total recebido pelo entregador, somando coletas e entregas.

1. FUNCIONAMENTO E EXEMPLO

* A view Courier\_Payment utiliza a função de agregação SUM para calcular o valor total das coletas (Valor das coletas) e das entregas (Valor das entregas) para cada entregador, com base nos pacotes que têm status de entrega concluída (status 5). Ela agrupa os resultados pelo nome e ID do entregador.
* Exemplo: Este exemplo retorna todos os registros da view Courier\_Payment, mostrando o pagamento detalhado de cada entregador com base nos pacotes entregues com sucesso.
  + SELECT \* FROM Courier\_Payment;

4.3.3 View Finances: Análise Financeira dos Pacotes Concluídos

1. DESCRIÇÃO

A view Finances fornece uma análise financeira dos pacotes que foram concluídos com sucesso, calculando a faturação bruta, o custo logístico e a faturação líquida com base nos valores associados aos pacotes, coletas e entregas realizadas.

1. CÓDIGO SQL

create view Finances

as

select TP.Designação As [Tamanho do pacote],

TP.Valor as [Valor por pacote],

Count(P.[Status\_id]) as [Total Pacotes Concluidos],

SUM(C.[Valor por Coleta])+SUM(E.[Valor por Entrega]) as [Custo Logístico],

Count(P.[Status\_id])\*TP.Valor as [Faturação Bruta],

(Count(P.[Status\_id])\*TP.Valor)-(SUM(C.[Valor por Coleta])+SUM(E.[Valor por Entrega])) as [Faturação líquida]

from Delivery.[Status] ST

join Delivery.Pacote P on ST.Status\_id = P.Status\_id

join Delivery.Coleta C on P.Coleta\_id = C.Coleta\_id

join Delivery.Entrega E on P.Entrega\_id = E.Entrega\_id

join Delivery.Entregador EN on C.Entregador\_id = EN.Entregador\_id

join Delivery.Tipo\_Pacote TP on P.Tp\_Pacote\_id = TP.Tp\_Pacote\_id

where P.Status\_id = 5

group by TP.Designação, TP.Valor

go

1. ESTRUTURA E RELACIONAMENTOS

* Delivery.[Status]: Contém os status dos pacotes, incluindo o status que indica entrega concluída.
* Delivery.Pacote: Relaciona os pacotes aos tipos de pacote, coletas e entregas associadas.
* Delivery.Coleta: Armazena informações sobre as coletas realizadas pelos entregadores.
* Delivery.Entrega: Registra os detalhes das entregas, incluindo os valores associados.
* Delivery.Entregador: Contém os dados dos entregadores responsáveis pelas coletas e entregas.
* Delivery.Tipo\_Pacote: Define os tipos de pacotes e seus respectivos valores.

1. COLUNAS RETORNADAS

* Tamanho do pacote: Designação do tamanho do pacote.
* Valor por pacote: Valor associado ao tipo de pacote.
* Total Pacotes Concluídos: Número total de pacotes que foram entregues com sucesso.
* Custo Logístico: Custo total associado às coletas e entregas dos pacotes concluídos.
* Faturação Bruta: Total faturado com os pacotes concluídos (quantidade de pacotes multiplicada pelo valor do pacote).
* Faturação líquida: Lucro líquido obtido com os pacotes concluídos (faturação bruta menos o custo logístico).

1. FUNCIONAMENTO E EXEMPLO

* A view Finances utiliza a função de agregação COUNT para contar o número de pacotes concluídos e SUM para calcular o custo logístico, a faturação bruta e líquida com base nos valores definidos para cada tipo de pacote e nas coletas/entregas realizadas.
* Este exemplo retorna todos os registros da view Finances, mostrando a análise financeira detalhada dos pacotes que foram entregues com sucesso, incluindo o valor por pacote, custos logísticos e lucro líquido obtido.
  + SELECT \* FROM Finances;
    1. View Region\_sucess: Análise de Sucesso de Entregas por Região

1. DESCRIÇÃO

A view Region\_sucess fornece uma análise detalhada do desempenho de entregas em cada região, incluindo a quantidade de pacotes entregues, cancelados e reagendados, juntamente com a taxa de sucesso de entrega definida para cada região.

1. CÓDIGO SQL

create view Region\_sucess

as

select R.Designação AS Região,

SUM(case when s.Tp\_status = 'Entregue' then 1 else 0 end) as [Quantidade Entregue],

SUM(case when s.Tp\_status = 'Cancelado' then 1 else 0 end) as [Quantidade Cancelado],

SUM(case when s.Tp\_status = 'Reagendado' then 1 else 0 end) as [Quantidade Reagendado],

R.[Taxa de Sucesso] as [Taxa de sucesso da região]

From Delivery.Entrega E

join Delivery.Entregador EN ON E.Entregador\_id = EN.Entregador\_id

join Delivery.Região r ON en.Região\_id = r.Região\_id

join Delivery.[Status] s ON e.Situação\_id = s.Status\_id

group by R.Designação, R.[Taxa de Sucesso]

go

1. ESTRUTURA E RELACIONAMENTOS

* Delivery.Entrega: Contém informações sobre as entregas realizadas, incluindo o status da entrega.
* Delivery.Entregador: Relaciona os entregadores às suas respectivas regiões.
* Delivery.Região: Define as regiões onde as entregas são realizadas.
* Delivery.[Status]: Contém os status das entregas, como entregue, cancelado ou reagendado.

1. COLUNAS RETORNADAS

* Região: Designação da região onde as entregas foram realizadas.
* Quantidade Entregue: Número total de pacotes que foram entregues com sucesso na região.
* Quantidade Cancelado: Número de pacotes que foram cancelados na região.
* Quantidade Reagendado: Número de pacotes que foram reagendados na região.
* Taxa de sucesso da região: Taxa de sucesso das entregas na região, previamente calculada e armazenada.

1. FUNCIONAMENTO E EXEMPLO

* A view Region\_sucess utiliza cláusulas CASE dentro da função de agregação SUM para contar a quantidade de pacotes entregues, cancelados e reagendados, com base nos status definidos na tabela Delivery.[Status]. O campo Taxa de sucesso da região é obtido diretamente da tabela Delivery.Região.
* Este exemplo retorna todos os registros da view Region\_sucess, mostrando a análise detalhada do desempenho de entregas por região, incluindo a quantidade de pacotes entregues, cancelados e reagendados, junto com a taxa de sucesso definida para cada região.
  + SELECT \* FROM Region\_sucess;
    1. View Sales\_of\_Region: Vendas por Região

1. DESCRIÇÃO

A view Sales\_of\_Region fornece um resumo das vendas concluídas por região, incluindo a quantidade total de vendas concluídas e o valor total das vendas realizadas.

1. CÓDIGO SQL

create view Sales\_of\_Region

as

select

R.Designação AS Região,

Count(C.Compra\_id) AS [Quantidade Vendas Concluídas],

SUM(TP.Valor) AS [Valor Total Vendas]

from Delivery.Compra c

join Delivery.Cliente cli ON c.Cliente\_id = cli.Cliente\_id

join Delivery.Pacote p ON c.Compra\_id = p.Compra\_id

join Delivery.Tipo\_Pacote tp ON p.Tp\_Pacote\_id = tp.Tp\_Pacote\_id

join Delivery.Entrega e ON p.Entrega\_id = e.Entrega\_id

join Delivery.Entregador en ON e.Entregador\_id = en.Entregador\_id

join Delivery.Região r ON en.Região\_id = r.Região\_id

where P.status\_id = 5

group by R.Designação

Go

1. ESTRUTURAS E RELACIONAMENTOS

* Delivery.Compra: Contém informações sobre as compras realizadas pelos clientes.
* Delivery.Cliente: Relaciona as compras aos clientes que as efetuaram.
* Delivery.Pacote: Associa as compras aos pacotes gerados para entrega.
* Delivery.Tipo\_Pacote: Define o tipo de cada pacote e seu respectivo valor.
* Delivery.Entrega: Relaciona os pacotes às entregas realizadas.
* Delivery.Entregador: Associa as entregas aos entregadores responsáveis.
* Delivery.Região: Define as regiões onde as entregas foram realizadas.

1. COLUNAS RETORNADAS

* Região: Designação da região onde as vendas foram realizadas.
* Quantidade Vendas Concluídas: Número total de vendas concluídas na região.
* Valor Total Vendas: Valor total das vendas realizadas na região, calculado somando os valores dos pacotes associados às vendas.

1. FUNCIONAMENTO E EXEMPLO

* A view Sales\_of\_Region utiliza a cláusula GROUP BY para agrupar os resultados por R.Designação, que representa a designação da região na tabela Delivery.Região. Isso permite contar o número de vendas (Count(C.Compra\_id)) e calcular o valor total das vendas (SUM(TP.Valor)), considerando apenas os pacotes com status 5 (que indicam vendas concluídas).
* Este exemplo retorna todos os registros da view Sales\_of\_Region, mostrando um resumo das vendas concluídas por região, incluindo a quantidade total de vendas e o valor total das vendas realizadas em cada uma das regiões da empresa de transportes.
  + SELECT \* FROM Sales\_of\_Region;

CONCLUSÃO

Este esquema de banco de dados, acompanhado de procedimentos e gatilhos meticulosamente projetados, fornece uma solução robusta e abrangente para gerenciar eficientemente um serviço de entrega. Ao garantir a integridade dos dados através de restrições de chave estrangeira bem definidas, garante a precisão e consistência dos dados em todo o sistema.

Os procedimentos armazenados automatizam operações rotineiras e essenciais, como atualização de registros, criação de compras, coleta de valores e gerenciamento de cronogramas de entrega, reduzindo assim a intervenção manual e o risco de erros. Esses procedimentos são elaborados para lidar com vários cenários, garantindo fluxos de trabalho tranquilos e eficientes.

Os gatilhos impõem regras comerciais críticas e garantem que o banco de dados reaja adequadamente a eventos específicos, mantendo a integridade e a confiabilidade dos dados. Por exemplo, os gatilhos podem atualizar automaticamente registros relacionados, validar alterações de dados e aplicar medidas de segurança, garantindo que todas as operações sigam a lógica de negócios predefinida.

Essa configuração não apenas agiliza todo o processo de entrega, mas também fornece uma base escalonável e adaptável. Ele foi projetado para crescer junto com o negócio, permitindo fácil personalização e extensão para atender aos requisitos de negócios em evolução. A combinação de gerenciamento estruturado de dados, automação e aplicação de regras de negócios torna esta solução ideal para um ambiente de serviços de entrega dinâmico e exigente.

Em essência, esse esquema de banco de dados oferece uma estrutura sólida, flexível e eficiente para gerenciar todos os aspectos de um serviço de entrega, desde as interações com clientes e fornecedores até o gerenciamento do pessoal de entrega e o rastreamento de pacotes. É uma ferramenta poderosa que aumenta a eficiência operacional, garante a precisão dos dados e dá suporte ao crescimento contínuo e às necessidades de personalização do negócio.

Bibliografia

<https://www.w3schools.com/sql/>

<https://www.w3schools.com/SQL/sql_view.asp>

<https://www.w3schools.com/SQl/func_mysql_if.asp>

<https://www.w3schools.com/sql/sql_primarykey.asp>

<https://www.w3schools.com/sql/sql_top.asp>

<https://www.w3schools.com/SQL/sql_orderby.asp>

<https://www.w3schools.com/SQL/sql_stored_procedures.asp>

<https://www.w3schools.com/SQL/sql_min_max.asp>

<https://stackoverflow.com/>

<https://stackoverflow.com/questions/20853725/what-is-the-use-of-with-ties-keyword-in-select-statement-in-sql-queries>

<https://stackoverflow.com/questions/63447/how-do-i-perform-an-if-then-in-an-sql-select>

<https://stackoverflow.com/questions/15683975/if-elif-else-in-t-sql>

<https://pt.stackoverflow.com/questions/371331/copiar-dados-de-umbanco-de-dados-para-outro>