PUC-Rio
Data: 14 de Abril de 2019
Aluno: Wellington Bezerra
Matrícula: 1413383

## CRIAÇÃO DO BANCO DE DADOS

```
CREATE TABLE CLIENTE (
CODIGO INTEGER NOT NULL,
NOME VARCHAR2(30) ,
TELEFONE VARCHAR2 (16),
LOGRADOURO VARCHAR2(50) ,
NUMERO INTEGER NOT NULL,
COMPLEMENTO VARCHAR2(50) ,
CIDADE VARCHAR2(50) ,
ESTADO VARCHAR2(50) ,
NUMEROCONTRIBUINTE VARCHAR2(50) );

CREATE TABLE ITENSNOTA (
NUMERO NUMBER NOT NULL ,
  NUMEROMERCADORIA NUMBER NOT NULL ,
  QUANTIDADE NUMBER NOT NULL ,
  VALORUNITARIO FLOAT NOT NULL ,
  CONSTRAINT NUMERO PRIMARY KEY ( NUMERO , NUMEROMERCADORIA )
ENABLE );

CREATE TABLE MERCADORIAS (
NUMEROMERCADORIA NUMBER NOT NULL,
 DESCRICAO VARCHAR2(20),
 QUANTIDADEESTOQUE NUMBER,
CONSTRAINT MNUMEROMERCADORIA PRIMARY KEY(NUMEROMERCADORIA )
ENABLE );

CREATE TABLE RECIBO (
NUMERO NUMBER NOT NULL,
  DATAEMISSAO TIMESTAMP NOT NULL,
  FORMAPAGAMENTO VARCHAR2(20) NOT NULL ,
  CODIGOCLIENTE NUMBER NOT NULL,
 CPFVENDEDOR VARCHAR2(14) ,
  CONSTRAINT NOTASVENDA_PK PRIMARY KEY ( NUMERO ) ENABLE );
```

```
CREATE TABLE FUNCIONÁRIO
( "CPF" VARCHAR2(14),
"NOME" VARCHAR2(30),
"TELEFONE" VARCHAR2(16),
 "LOGRADOURO" VARCHAR2(50),
"NUMERO" NUMBER(*,0),
"COMPLEMENTO" VARCHAR2(50),
"CIDADE" VARCHAR2(50),
"ESTADO" VARCHAR2(50),
"CODIGODEPARTAMENTO" NUMBER
);

CREATE TABLE DEPARTAMENTO (
CODIGODEPARTAMENTO NUMBER NOT NULL,
NOME VARCHAR2(20) NOT NULL
, CPF_CHEFE VARCHAR2(20) NOT NULL
, CONSTRAINT DEPARTAMENTO_PK PRIMARY KEY
( CODIGODEPARTAMENTO )
ENABLE );



CREATE TABLE CARGO (
"CODIGO" NUMBER,
"DESCRIÇAO" VARCHAR2(50),
"SALARIOBASE" FLOAT(126) )



CREATE TABLE CARGOSFUNC (
CPF VARCHAR2(20) NOT NULL,
  CODIGOCARGO NUMBER NOT NULL ,
  DATAINICIO DATE NOT NULL ,
  DATAFIM DATE ,
  CONSTRAINT CARGOSFUNC_PK PRIMARY KEY
( CPF , CODIGOCARGO , DATAINICIO )
ENABLE );
```

# ALTERAÇÕES

```
ALTER TABLE FUNCIONARIO
ADD CONSTRAINT FUNCIONARIO_FK1 FOREIGN KEY ( CODIGODEPARTAMENTO )
REFERENCES DEPARTAMENTO
( CODIGODEPARTAMENTO )
ENABLE;

ALTER TABLE CLIENTE
ADD CONSTRAINT CLIENTE_PK PRIMARY KEY ( CODIGO )
ENABLE;

ALTER TABLE RECIBO
ADD CONSTRAINT RECIBO_FK1 FOREIGN KEY ( CODIGOCLIENTE )
REFERENCES CLIENTE
( CODIGO )
ENABLE;

ALTER TABLE RECIBO
ADD CONSTRAINT RECIBO_FK2 FOREIGN KEY ( CPFVENDEDOR )
REFERENCES FUNCIONARIO
( CPF )
ENABLE;


ALTER TABLE ITENSNOTA
ADD CONSTRAINT NOTASVENDA_FK1 FOREIGN KEY ( NUMEROMERCADORIA )
REFERENCES MERCADORIAS
( NUMEROMERCADORIA )
ENABLE;


ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT DEPARTAMENTO_FK2 FOREIGN KEY ( CPF_CHEFE )
REFERENCES FUNCIONARIO
( CPF )
ENABLE;


ALTER TABLE CARGOSFUNC
ADD CONSTRAINT CARGOSFUNC_FK2 FOREIGN KEY ( CODIGOCARGO )
REFERENCES CARGO
( CODIGO )
ENABLE;
```

```
ALTER TABLE CARGOSFUNC
ADD CONSTRAINT CARGOSFUNC_FK1 FOREIGN KEY ( CPF )
REFERENCES FUNCIONARIO
( CPF )
ENABLE;

ALTER TABLE CARGO
ADD CONSTRAINT "CODIGO" PRIMARY KEY ("CODIGO")
```

## EXERCÍCIOS

1.

Renomear a tabela NotasVenda para Recibo
ALTER TABLE NotasVenda
 RENAME TO Recibo;

Adição de uma nova coluna na tabela Recibo
ALTER TABLE Recibo
ADD venda INT default 1 NOT NULL;

Adição de uma chave estrangeira
ALTER TABLE ItensNota
ADD CONSTRAINT fk_itensNota FOREIGN KEY (numero)
REFERENCES Recibo (numero);

Criação de uma View
CREATE VIEW RecibosCompra AS
SELECT numero, quantidade, ValorUnitario,  ValorUnitario * Quantidade, dataEmissao, cpfVendedor
FROM Recibo, ItensNota
WHERE venda = 0 AND Recibo.numero = ItensNota.numero

2.

O nome da tabela Cliente será substituído por Pessoa
ALTER TABLE Cliente
RENAME TO Pessoa;

A tabela Cliente receberá mais uma coluna:
ALTER TABLE CLIENTE
ADD fornecedor INT default 0 NOT NULL;

CREATE VIEW Cliente AS
SELECT codigoCliente, nome, telefone, logradouro, numero, complemento, cidade, estado, numeroContribuinte
FROM Cliente
WHERE Cliente.fornecedor = 0

3.

**Altera nome da tabela**

```
ALTER TABLE Mercadorias
RENAME TO Mercadoria;
```

**A tabela receberá mais uma coluna**

```
ALTER TABLE Mercadoria
ADD (quantidadeMinima NUMBER );
```

**A tabela receberá mais uma coluna**

```
ALTER TABLE Cliente
ADD (email TEXT );
```

**Criação de uma view**

```
CREATE VIEW MERCADORIAS AS
SELECT      email, quantidadeEstoque, quantidadeMinima
FROM Mercadoria, ItensNota, Recibo, Cliente
WHERE Mercadoria.numeroMercadoria = Itensnota.numeroMercadoria AND
            ItensNota.numero = Recibo.numero AND
            Recibo.codigoCliente = Client.codigoCliente ;
```

**Criação de uma procedure**

```
CREATE FUNCTION avisoLimiteEstoque() RETURNS trigger AS $avisoLimiteEstoque$
   BEGIN

      IF NEW.quantidadeEstoque < NEW.quantidateMinima
            pgmail('admin@admin.com.br' , 'NEW.email', 'Atenção: Estoque baixo!' )
      END IF
   END;
$avisoLimiteEstoque$ LANGUAGE plpgsql;

CREATE TRIGGER avisoLimiteEstoque BEFORE INSERT OR UPDATE ON Mercadorias
   FOR EACH ROW EXECUTE PROCEDURE avisoLimiteEstoque();
```

OBS: http://brandolabs.com/pgmail

4.

**Criação de uma view**

```
CREATE VIEW UltimaCompra AS
SELECT      valorUnitario, quantidadeEstoque
FROM        Mercadoria, ItensNota, Recibo
WHERE       Mercadoria.numeroMercadoria = "12345" AND
            Mercadoria.numeroMercadoria = Itensnota.numeroMercadoria AND
            ItensNota.numero = Recibo.numero AND
            Recibo.venda = 0 AND
            MAX(dataEmissão) ;
```

**Essa procedure será executada sempre que um determinado produto for vendido:**

**Criação de uma procedure**

```
CREATE FUNCTION impedePrecoVendaMaiorPrecoCompra() RETURNS trigger AS
$impedePrecoVendaMaiorPrecoCompra$
   BEGIN

      IF NEW.valorUnitario < VALORVENDA
            “Venda não autorizada"
      END IF
   END;
$impedePrecoVendaMaiorPrecoCompra$ LANGUAGE plpgsql;

CREATE TRIGGER impedePrecoVendaMaiorPrecoCompra BEFORE INSERT OR
UPDATE ON UltimaCompra
   FOR EACH ROW EXECUTE PROCEDURE impedePrecoVendaMaiorPrecoCompra();
```

5.

**Criação de uma view**

```
CREATE VIEW MercadoriaSelecionada AS
SELECT      numeroMercadoria, quantidadeEstoque
FROM        Mercadoria
WHERE       Mercadoria.numeroMercadoria = "12345";
```

**Essa procedure será executada sempre que um produto for vendido. Ou seja, o update na tabela Mercadoria somente será realizado se a quantidade do estoque for maior ou igual a quantidade**

**Criação de uma procedure**

```
CREATE FUNCTION impedaVendaProdutoSemEstoque() RETURNS trigger AS
$impedaVendaProdutoSemEstoque$
   BEGIN

      IF NEW. quantidadeEstoque < QTDDESEJADA
            “Venda não autorizada"
      END IF
   END;
$impedaVendaProdutoSemEstoque$ LANGUAGE plpgsql;

CREATE TRIGGER impedaVendaProdutoSemEstoque BEFORE INSERT OR UPDATE
ON MercadoriaSelecionada
   FOR EACH ROW EXECUTE PROCEDURE impedaVendaProdutoSemEstoque();
```

6. Nao pode  ter aquisição de um produto que ultrapasse seu estoque máximo

**A tabela receberá mais uma coluna**

```
ALTER TABLE Mercadoria
ADD (quantidadeMaxima NUMBER );
```

Um Recibo de compra (Recibo.venda = 0) somente terá um ItensNota com uma Mercadoria, enquanto não atingir o valor do estoque máximo

**Criação de uma procedure**

```
CREATE FUNCTION verificaQtdMaximoEstoque() RETURNS trigger AS
$verificaQtdMaximoEstoque$
   BEGIN

      IF NEW. quantidadeEstoque < quantidadeMaxima
               “Permite a compra desse produto"
      END IF
   END;
$verificaQtdMaximoEstoque$ LANGUAGE plpgsql;

CREATE TRIGGER verificaQtdMaximoEstoque BEFORE INSERT OR UPDATE ON
MercadoriaSelecionada
   FOR EACH ROW EXECUTE PROCEDURE verificaQtdMaximoEstoque();
```

7. MECANISMO PARA GARANTIR A INTEGRIDADE DOS DADOS

Para garantir a integridade dos dados de vendedores com a base de funcionários, basta que o **cpfVendedor** da tabela **Recibo** seja uma **chave estrangeira** para **cpf** da tabela **Funcionarios**

Adição de uma chave estrangeira
ALTER TABLE Recibo
ADD CONSTRAINT fk_cpfVendedor FOREIGN KEY (cpfVendedor)
REFERENCES Funcionario (cpf);

**8.**

**Criação de uma view**
CREATE VIEW PagamentoFuncionarios AS

SELECT cpf, SUM(quantidade * valorUnitario * 0,05), salario_Base

FROM Recibo JOIN Funcionario ON (Recibo.cpfVendedor = Funcionario.cpf) AND
Recibo JOIN ItensNota ON (Recibo.numero = ItensNota.numero) AND
Funcionario JOIN CargosFunc ON (Funcionario.cpf = CargosFunc.cpf) AND
CargosFunc JOIN Cargo ON (CargosFunc.codigoCargo = Cargo.codigo);

WHERE Recibo.venda = 1;

**9.**

- **Cadastro de produtos com todas as suas informações;**

CREATE PROCEDUREA CadastraProduto @numeromercadoria int,
@descricao nvarchar(30),
@quantidadeEstoque int,
@quantidadeMinima int,
@quantidadeMaxima int
AS

INSERT INTO Mercadoria
VALUES (@numeromercadoria, @descricao, @quantidadeEstoque,
@quantidadeMinima, @quantidadeMaxima);

GO

**Para executar essa procedure:**

```
EXEC CadastraProduto numeromercadoria = 123,
                                descricao = “Descrição da mercadoria",
                                quantidadeEstoque = 100,
                                quantidadeMinima = 10,
                                quantidadeMaxima = 1000;
```

- **Consulta dos N maiores clientes;**

```
CREATE PROCEDUREA MaioresClientes @nClientes int,

AS

SELECT      codigoCliente, SUM(quantidade * valorUnitario)

FROM        Recibo JOIN ItensNota on (Recibo.numero = ItensNota.numero)

ORDER BY   SUM(quantidade * valorUnitario)  DESC

FETCH NEXT @nClientes ROWS ONLY;

GO
```

- **Consulta a dados de um fornecedor/cliente X.**

```
CREATE PROCEDURE ConsultaFornecedorOuCliente @codigoCliente nvarchar(30),

AS

SELECT *
FROM Cliente
WHERE codigoCliente = @ codigoCliente

GO
```

- **Consulta o estoque e preço de um produto.**

```
CREATE PROCEDURE ConsultaEstoqueEPrecoMercadoria @codigoMercadoria
nvarchar(30),

AS

SELECT      quantidadeEstoque, valorUnitario
```

```
FROM        ItensNota JOIN Mercadoria ON (ItensNota.numeroMercadoria =
Mercadoria.numeroMercadoria)

WHERE       Mercadoria.numeroMercadoria = @codigoMercadoria

GO
```

- **Consulta a dados de um pedido X.**

```
CREATE PROCEDURE ConsultaDadosPedido @numeroRecibo nvarchar(30),

AS

SELECT      *

FROM ItensNota JOIN Recibo ON (ItensNota.numero = Recibo.numero)

WHERE       Recibo.numero = @numeroRecibo

GO
```