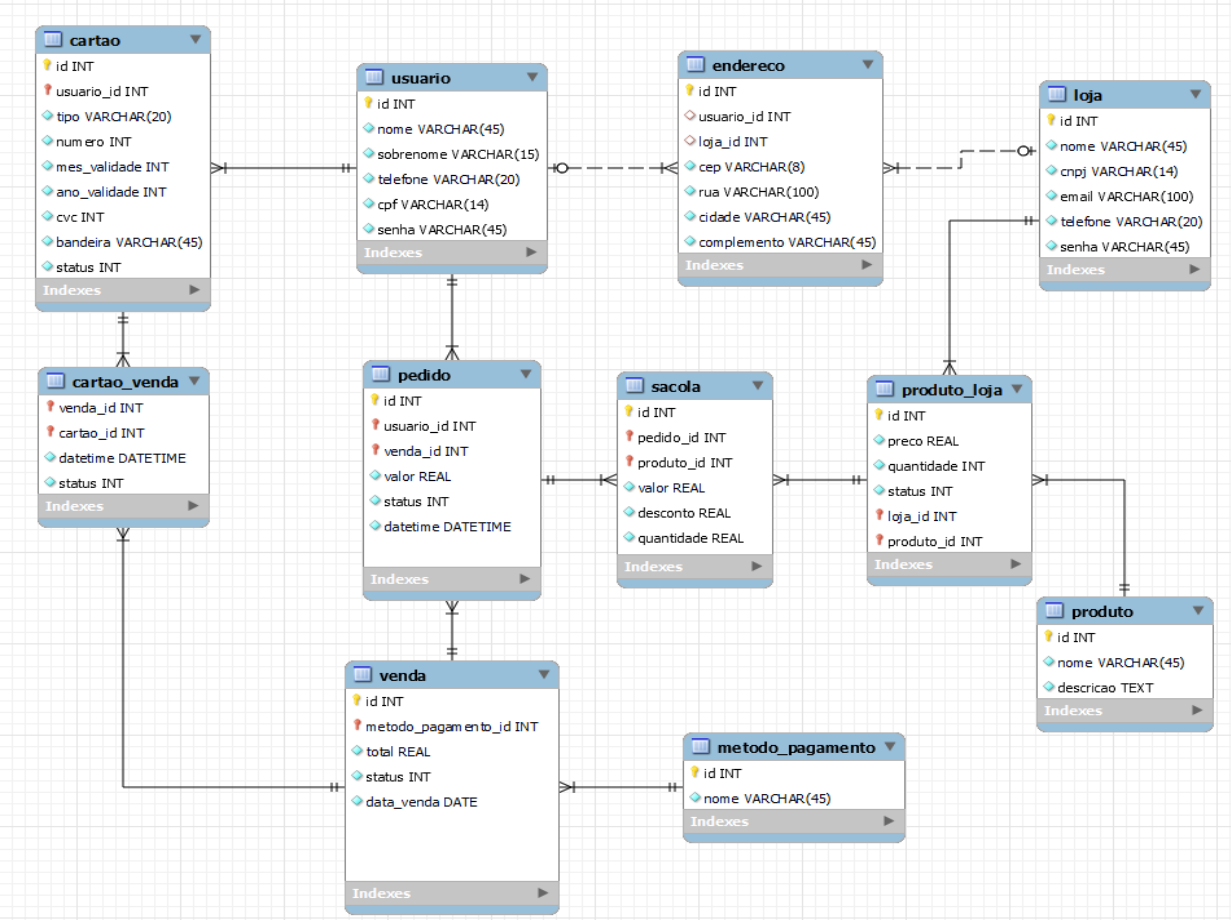
Banco de dados – Steffood (Stefanini)

Modelagem -



Tabelas -

```
CREATE TABLE usuario(
    id serial primary key,
    nome varchar(15) not null,
    sobrenome varchar(15) not null,
    telefone varchar(20) not null,
    cpf varchar(20),
    senha varchar(45)
);

CREATE TABLE loja (
    id serial primary key,
    nome varchar(45) not null,
    cnpj varchar(14) not null,
    email varchar(100) not null,
    telefone varchar(20),
    senha varchar(45)
);

CREATE TABLE endereco(
    id serial primary key,
    usuario_id INT,
    loja_id INT,
    cep varchar(8) not null,
    rua varchar(100) not null,
    numero varchar(11) not null,
    cidade varchar(45) not null,
    complemento varchar(45),
```

```
  CONSTRAINT fk_usuario_endereco FOREIGN KEY(usuario_id) REFERENCES usuario(id),
CONSTRAINT fk_loja_endereco FOREIGN KEY(loja_id) REFERENCES loja(id)
);

CREATE TABLE cartao(
    id serial primary key,
    usuario_id int not null,
    nome varchar(40) not null,
    numero int not null,
    mes_validade int not null,
    ano_validade int not null,
    cvc int not null,
    bandeira varchar(45) not null,
    status int not null,
    CONSTRAINT fk_usuario FOREIGN KEY(usuario_id) REFERENCES usuario(id),
   unique key(usuario_id, nome)
);

CREATE TABLE metodo_pagamento(
    id serial primary key,
    nome varchar(45)
);

CREATE TABLE venda(
    id serial primary key,
    metodo_pagamento_id int,
    total real not null,
    status int not null,
    data_venda date,
```

```
    CONSTRAINT fk_mp_id FOREIGN KEY(metodo_pagamento_id) REFERENCES
metodo_pagamento(id)
);

CREATE TABLE pedido(
    id serial primary key,
    usuario_id int  not null,
    venda_id int not null,
    valor real not null,
    status int not null,
    datetime timestamp without time zone,
    CONSTRAINT fk_usuario FOREIGN KEY(usuario_id) REFERENCES usuario(id),
    CONSTRAINT fk_venda FOREIGN KEY(venda_id) REFERENCES venda(id)
);


CREATE TABLE cartao_venda(
    venda_id INT not null,
    cartao_id INT not null,
    datetime timestamp without time zone,
    status INT not null,
    CONSTRAINT fk_cartao_venda FOREIGN KEY(venda_id) REFERENCES venda(id),
    CONSTRAINT fk_meu_cartao FOREIGN KEY(cartao_id) REFERENCES cartao(id)
);

CREATE TABLE produto (
    id serial primary key,
    nome varchar(45) not null,
    descricao varchar(255) not null
);
```

```
CREATE TABLE produto_loja(
    id serial primary key,
    produto_id int not null,
    loja_id int not null,
    preco real not null,
    quantidade int not null,
    status int,
    CONSTRAINT fk_produtos FOREIGN KEY(produto_id) REFERENCES produto(id),
    CONSTRAINT fk_loja FOREIGN KEY(loja_id) REFERENCES loja(id)
);

CREATE TABLE sacola(
    pedido_id int not null,
    produto_id int not null,
    valor real not null,
    quantidade int not null,
    datetime timestamp without time zone,
    CONSTRAINT fk_sacola_pedido FOREIGN KEY(pedido_id) REFERENCES pedido(id),
    CONSTRAINT fk_produto_loja FOREIGN KEY(produto_id) REFERENCES
produto_loja(id)
);
```

SELECT'S

```
SELECT * FROM usuario;
SELECT * FROM produto;
SELECT * FROM estoque;
```

```
SELECT * FROM pedido;
SELECT * FROM sacola;
SELECT * FROM produto;
SELECT * FROM loja;
SELECT * FROM produto_loja;
```

INSERT'S

```
INSERT INTO usuario (nome, sobrenome, telefone, cpf)
VALUES('Jefferson', 'Junior', '(11)93848-3434', '300.233.234-89')

INSERT INTO loja (nome, cnpj, email, telefone, senha)
VALUES
('Mc Donalds', '32132312313234', 'mcdonalds@email.com', '(11)93848-3434'),
('ipsum',78799925154996,'ut.nulla@protonmail.net',52442753307416,'d1234'),
 ('dolor',67580076301826,'ac.urna@outlook.ca',85060815245192,'d1234'),
 ('non,',54379165526639,'ut.ipsum.ac@icloud.com',45105854326043,'d1234'),
 ('neque.',61158669064049,'sed@protonmail.com',61644381722178,'d1234')

INSERT INTO produto (nome, descricao)
VALUES
('Café', '3 Corações'),
('Leite', 'Leite 1L'),
('Pão', 'Pão ao Leite'),
('Guaraná', 'Guaraná Antarctica'),
('X-Hamgurger', '1 Hamburguer, 2 Tomate, 3 Pickles, 1 Salada, 2 Pão')

INSERT INTO produto_loja
```

```
(preco, quantidade, status, loja_id, produto_id)
VALUES (20.00, 100, 1, 1, 5), (20.00, 100, 1, 1, 3),

INSERT INTO metodo_pagamento (nome) VALUES
('Cartão de Crédito'),
('Cartão de Débito'),
('Dinheiro')

INDEX'S
SELECT * FROM usuario order by nome
CREATE INDEX idx_usuario ON usuario (nome)
DROP INDEX idx_usuario

SELECT * FROM usuario order by cpf
CREATE INDEX idx_usuario_cpf ON usuario (cpf)
DROP INDEX idx_usuario_cpf

SELECT * FROM loja order by nome
CREATE INDEX idx_loja ON loja (nome)
DROP INDEX idx_loja

SELECT * FROM loja order by cnpj
CREATE INDEX idx_loja_cnpj ON loja (cnpj)
DROP INDEX idx_loja_cnpj
```

```
========================================
CRIAR PEDIDO
========================================

INSERT INTO produto_loja (produto_id, loja_id, preco, quantidade, status)
VALUES (5, 1, 5, 100, 1)

SELECT * FROM venda;
SELECT * FROM metodo_pagamento;

CALL novaCompra(1);

CREATE OR REPLACE PROCEDURE novaCompra (
  idUser INT
)
LANGUAGE plpgsql AS $$
DECLARE
numberCod INT;
validUser INT;

BEGIN
   numberCod = numberRandom();
   validUser = (SELECT id FROM usuario WHERE id = idUser);

   IF validUser IS NOT NULL THEN
      INSERT INTO venda (id, metodo_pagamento_id, total, status, data_venda)
```

```
        VALUES (numberCod, null, 0.0, 0, NOW());

        INSERT INTO pedido (usuario_id, venda_id, valor, status, datetime)
        VALUES (idUser, numberCod, 0, 0, CURRENT_TIMESTAMP);
        RAISE NOTICE 'Pedido realizado código: %', numberCod;
        COMMIT;
    ELSE
        RAISE NOTICE 'Pedido não realizado, usuário inexistente!';
    END IF;
END;
$$
```

==========

FUNÇÃO DE NÚMERO RANDOM

==========

```
CREATE OR REPLACE FUNCTION numberRandom()
            RETURNS INT LANGUAGE plpgsql AS $$
            DECLARE
               NumeroRandom INT;
            BEGIN
              RETURN floor(random() * 99999999 + 1000)::int;
            END;
$$
```

==========

INSERIR SACOLA

==========

```
SELECT * FROM pedido;
```

```
-- Inserir Sacola
-- ID PEDIDO
-- ID PRODUTO (FK)
-- ID LOJA
-- QUANTIDADE
CALL inserirSacola(6, 7, 1, 8);

SELECT * FROM produto_loja;
SELECT * FROM sacola;

-- PROCEDURE
CREATE OR REPLACE PROCEDURE inserirSacola(
    idPedido INT,
    idProduto INT,
    idLoja INT,
    qtd INT
) LANGUAGE plpgsql AS $$

DECLARE
validStatusPedido INT;
validStatusProduto INT;
validIdPedido INT;
validIdProduto INT;
qtdProdutoReturn INT;
qtdProdutoSacola INT;
novaQtdEstoque INT;

precoProduto REAL;
novoPreco REAL;
```

```
precoTotal REAL;
valorAtualSacola REAL;
valorAtualPedido REAL;
novoValorPedido REAL;
novoValorSacola REAL;
dateSacola TIMESTAMP WITHOUT TIME ZONE;

BEGIN
   precoProduto = (SELECT preco FROM produto_loja
   WHERE id = idProduto);

   qtdProdutoReturn = (SELECT quantidade FROM produto_loja
   WHERE id = idProduto);

   validStatusProduto = (SELECT status FROM produto_loja
   WHERE id = idProduto);

   validStatusPedido = (SELECT status FROM pedido
   WHERE id = idPedido);

   validIdPedido = (SELECT id FROM pedido
   WHERE id = idPedido);

   validIdProduto = (SELECT produto_id FROM sacola
   WHERE produto_id = idProduto);

   precoTotal = (SELECT valor FROM pedido
   WHERE id = idPedido);
```

```
valorAtualSacola = (SELECT valor FROM sacola
WHERE pedido_id = idPedido AND produto_id = idProduto);

 qtdProdutoSacola = (SELECT quantidade FROM sacola
WHERE produto_id = idProduto AND pedido_id = idPedido);

IF qtd <= qtdProdutoReturn THEN
   novoPreco = precoProduto * qtd;

   IF validIdPedido != idPedido AND validIdPedido IS NULL OR validIdProduto IS NULL THEN
   RAISE NOTICE 'Inseriu';
      INSERT INTO sacola (pedido_id, produto_id, valor, quantidade)
      VALUES (idPedido, idProduto, novoPreco, qtd);
   ELSE
    RAISE NOTICE 'Update Sacola';
      novoValorSacola = valorAtualSacola + (precoProduto * qtd);
      UPDATE sacola SET valor = novoValorSacola, quantidade = qtdProdutoSacola + qtd
      WHERE pedido_id = idPedido AND produto_id = idProduto;
   END IF;

    IF validStatusPedido != 1 THEN
      RAISE NOTICE 'Update pedido';
      novoValorPedido = precoTotal + (precoProduto * qtd);
      UPDATE pedido SET valor = novoValorPedido WHERE id = idPedido;

      IF validStatusProduto != 1 THEN
         RAISE NOTICE 'ROLLBACK';
         ROLLBACK;
      END IF;
```

```
        COMMIT;
      ELSE
        RAISE NOTICE 'ROLLBACK';
        ROLLBACK;
      END IF;
   END IF;
END;
$$
```

==========

FUNÇÃO DE VERIFICAR ESTOQUE

==========

```
CREATE OR REPLACE FUNCTION verificarEstoque(idPedido INT)
RETURNS BOOLEAN LANGUAGE plpgsql AS $$
DECLARE
sts INT = 0;
pl record;
qtdProduto INT;
BEGIN
    FOR pl in SELECT produto_id, quantidade
      FROM sacola WHERE pedido_id = idPedido
    LOOP
    qtdProduto = (SELECT quantidade FROM produto_loja
                WHERE id = pl.produto_id);
       IF qtdProduto < pl.quantidade THEN
         sts =+ 1;
       END IF;
```

```
    END LOOP;

    IF sts != 0 THEN
      RETURN FALSE;
    ELSE
      RETURN TRUE;
    END IF;
END;
$$
```

==========

PROCEDURE DE GERAR NOVO CARTÃO

==========

```
CREATE OR REPLACE PROCEDURE gerarCartao(
  idUser INT,
  nome VARCHAR
) LANGUAGE plpgsql AS $$
  DECLARE
  numero VARCHAR;
  mes INT;
  ano INT;
  cvc INT;
  bandeira VARCHAR;
  validUser INT;
BEGIN

  numero = floor(random() * 9999999999999 + 999999999999)::VARCHAR;
  mes = 1 + round(CAST (random()*(12-1) AS NUMERIC),0);
  ano = 2023 + round(CAST (random()*(2030-2023) AS NUMERIC),0);
```

```
  cvc = 100 + round(CAST (random()*(999-100) AS NUMERIC),0);

  validUser = (SELECT id FROM usuario WHERE id = idUser);

  IF validUser IS NOT NULL THEN
      INSERT INTO cartao (usuario_id, nome, numero, mes_validade, ano_validade, cvc, bandeira, status)
      VALUES (idUser, nome, numero, mes, ano, cvc, 'MasterCard', 1);
   ELSE
      RAISE NOTICE 'Usuário com ID % inexistente.', idUser;
   END IF;
END;
$$

========================================
FINALIZAR COMPRA
========================================

CREATE OR REPLACE PROCEDURE finalizarVenda(
   codPedido INT,
   metodPagamento INT,
   nameCard VARCHAR
)
LANGUAGE plpgsql AS $$
DECLARE
codigo INT;
validEstoque BOOLEAN;
validCard BOOLEAN;
pd record;
novaQtd INT;
```

```
idCartao INT;

BEGIN
   codigo = (SELECT id FROM venda WHERE id = codPedido);
   IF codigo IS NOT NULL THEN

      validEstoque = (SELECT verificarEstoque(codigo));
      IF validEstoque THEN
         FOR pd IN SELECT S.produto_id, S.quantidade, PL.quantidade AS estoque FROM venda AS V
                  JOIN pedido AS P ON V.id = P.venda_id
                  JOIN sacola AS S ON S.pedido_id = P.id
                  JOIN produto_loja AS PL ON S.produto_id = PL.id
                  WHERE V.id = codigo
         LOOP
            novaQtd = pd.estoque - pd.quantidade;
            UPDATE produto_loja SET quantidade = novaQtd WHERE id = pd.produto_id;
         END LOOP;

         validCard = (SELECT validarCartao(codigo, nameCard));

        IF metodPagamento = 1 OR metodPagamento = 2 THEN
            IF validCard THEN
               idCartao = (SELECT C.id FROM usuario AS U
               JOIN cartao AS C ON U.id = C.usuario_id
               JOIN pedido AS P ON P.usuario_id = U.id
               WHERE C.nome = nameCard AND P.venda_id = codigo LIMIT 1);

               INSERT INTO cartao_venda VALUES (codigo, idCartao, CURRENT_TIMESTAMP, 1);
               UPDATE venda SET status = 1 WHERE id = codigo;
```

```
            UPDATE pedido SET status = 1 WHERE venda_id = codigo;
            RAISE NOTICE 'Compra efetuada com sucesso!';
            RAISE NOTICE 'Pagamento em cartão!';
            COMMIT;

        ELSE
            RAISE NOTICE 'Compra negada!';
            ROLLBACK;
        END IF;
    ELSIF metodPagamento = 3 THEN
        UPDATE venda SET status = 1 WHERE id = codigo;
        UPDATE pedido SET status = 1 WHERE venda_id = codigo;
        RAISE NOTICE 'Pagamento em dinheiro!';
    END IF;
  END IF;
ELSE
  RAISE NOTICE 'Compra inexistente';
END IF;
END;
$$
```

==========

VIEWS

==========

SELECT * FROM produtos_loja

CREATE VIEW produtos_loja AS SELECT L.id AS id_loja, L.nome AS nome_loja,PL.id AS id_produto, P.nome AS nome_produto,

```
PL.quantidade AS quantidade, PL.preco AS preco_produto
FROM loja AS L
JOIN produto_loja AS PL ON PL.loja_id = L.id
JOIN produto AS P ON P.id = PL.produto_id;
```

SELECT * FROM pedidosRealizados

```
CREATE VIEW pedidosRealizados AS
SELECT venda_id AS codigo_Pedido, PP.nome AS produto,
S.quantidade, S.valor AS sub_total, U.nome AS cliente
FROM pedido AS P
JOIN sacola AS S ON S.pedido_id = P.id
JOIN produto_loja AS PL ON PL.id = S.produto_id
JOIN produto AS PP ON PP.id = PL.produto_id
JOIN usuario AS U ON U.id = P.usuario_id
```

SELECT * FROM vendasRealizadas

```
CREATE VIEW vendasRealizadas AS
SELECT V.id AS "Código Venda", MP.nome AS "método",
C.numero AS "Número Card",
V.total AS "Total da venda",
V.data_venda
FROM venda AS V
LEFT JOIN metodo_pagamento AS MP ON MP.id = V.metodo_pagamento_id
LEFT JOIN cartao_venda AS CV ON CV.venda_id = V.id
LEFT JOIN cartao AS C ON C.id = CV.cartao_id
LEFT JOIN usuario AS U ON U.id = C.usuario_id
```

```
WHERE V.status = 1
```