UFF - UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROF.º LUIZ ANDRÉ PORTES PAES LEME

PEDRO PAULO BASTOS TEIXEIRA
RENATO ARAÚJO BASTOS

IFOOD - DATABASE

Niterói
Dezembro de 2017

1 Introdução do Problema

O objetivo é desenvolver um sistema gerenciador de entregas e pedidos de vários restaurantes com delivery, similar ao iFood, a manutenção da integridade bem como a geração de relatórios de dados é feita através de funções que podem ser acessadas por aplicações cliente, seja para a gerência do estabelecimento, seja para o cliente final.

2 Estrutura do Banco de Dados e Modelo Entidade-Relacionamento

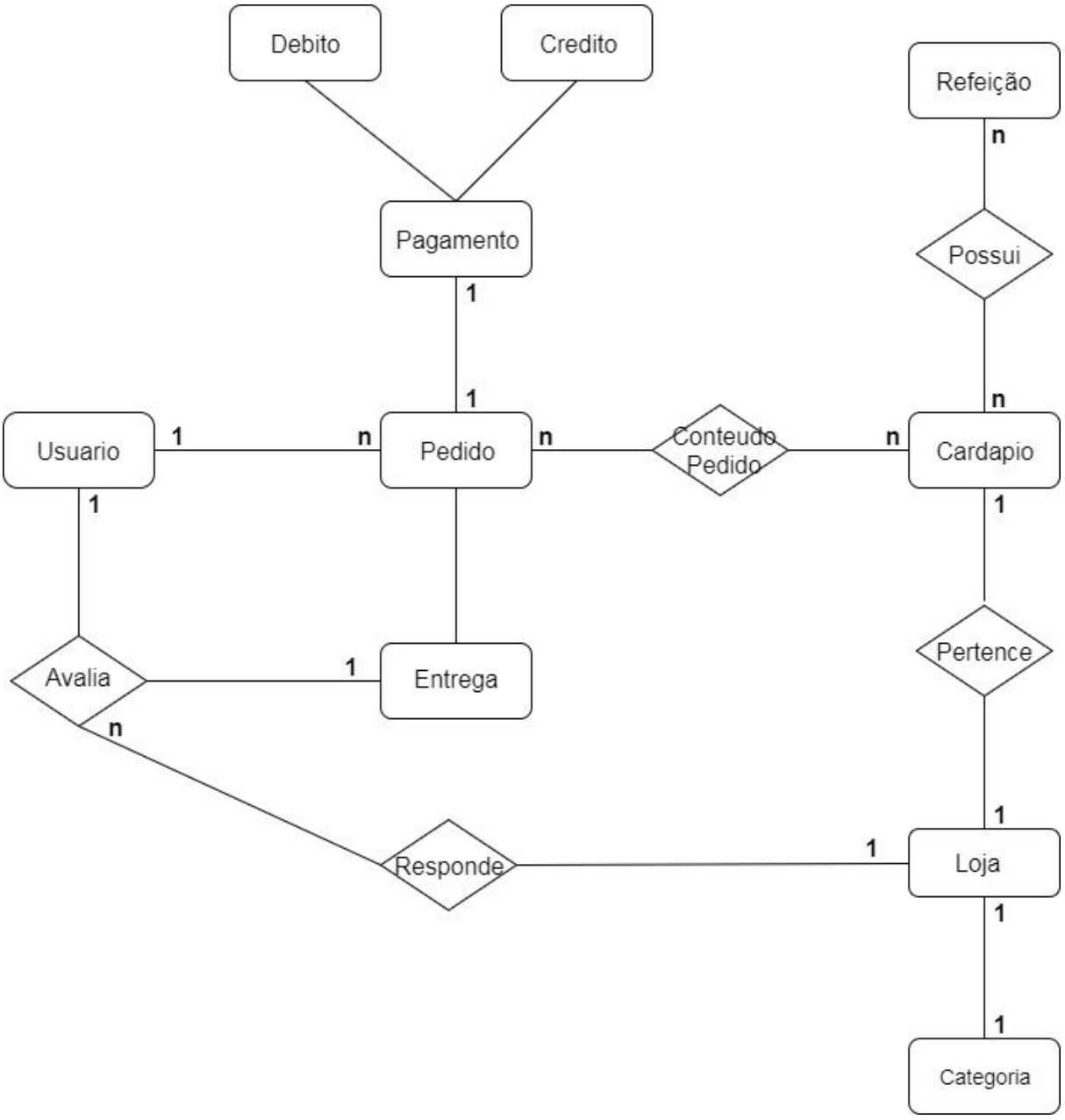

Figura 1 : Modelo ER

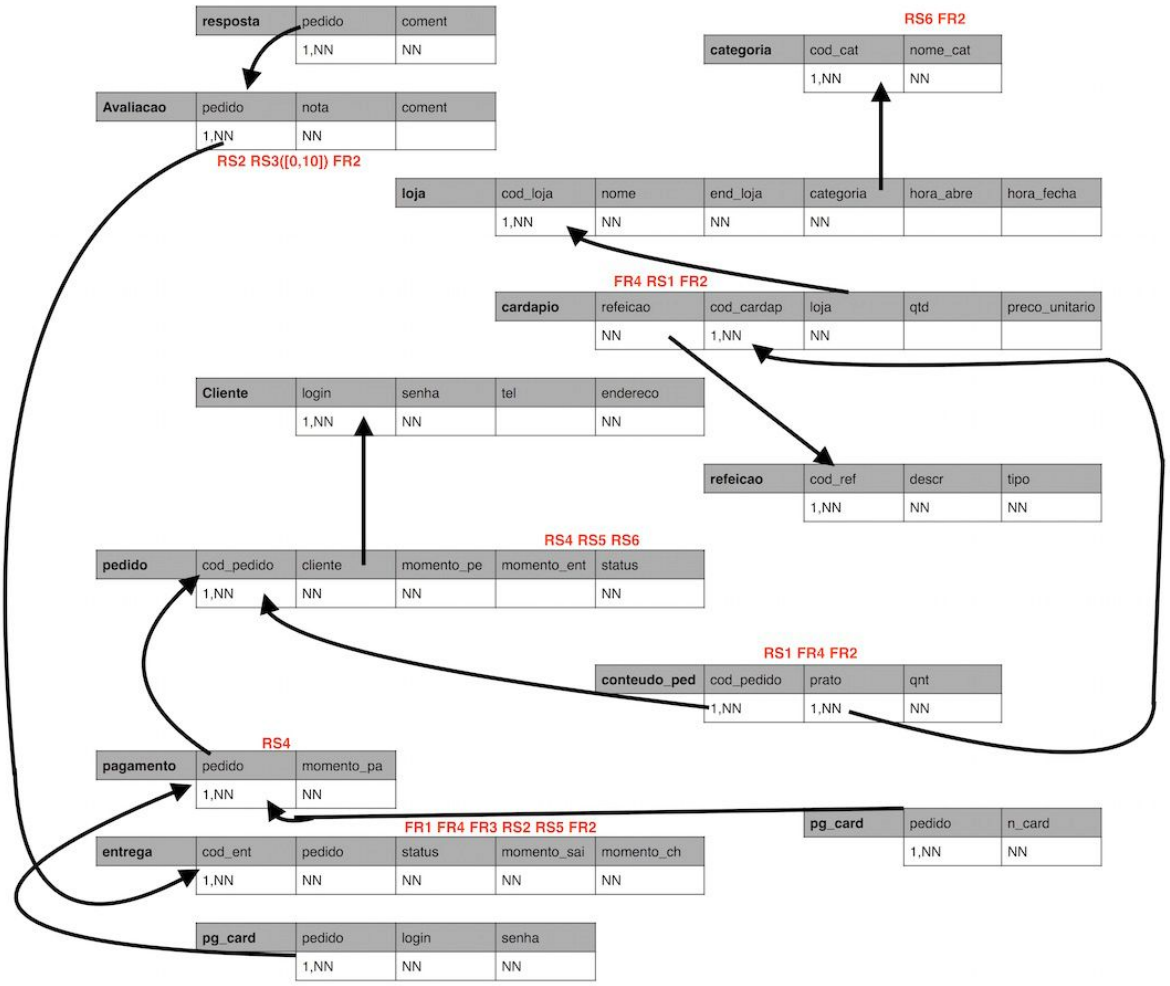

Figura 2 :Modelagem de Tabelas.

3 Triggers e Regras de Negócio

Trigger 1 :

O estoque deve ser atualizado após um novo pedido ser inserido, evitando que haja pedidos de refeições que não possuem estoque suficiente.

Código:

```
CREATE OR REPLACE FUNCTION novo_estoque() RETURNS trigger AS $$
 BEGIN
    UPDATE cardapio
    SET cardapio.qnt=cardapio.qnt - NEW.qnt
    WHERE NEW.prato=cardapio.cod_cardap;
  END;
  $$
  LANGUAGE plpgsql;

CREATE TRIGGER att_estoque AFTER INSERT ON conteudo_pedido
FOR EACH ROW EXECUTE PROCEDURE novo_estoque();
```

Figura 3: Código SQL do trigger de atualização do estoque

Explicação do código:

É um trigger de linha que após uma inserção na tabela conteudo_pedido, é executada a função novo_estoque.A função novo_estoque atualiza a tabela cardápio diminuindo a quantidade disponível da refeição que foi pedida pela quantidade pedida por um cliente, fazendo com que o estoque esteja sempre atualizado.

Trigger 2 :

Não deve ser permitida a compra de refeições que não tenham estoque disponível.
Código:

```
CREATE OR REPLACE FUNCTION estoque_disponivel() RETURNS trigger AS $$
  BEGIN
    IF((SELECT qnt FROM cardapio WHERE NEW.prato=cardapio.cod_cardap) - NEW.qnt <0) THEN
      RAISE EXCEPTION 'Estoque insuficiente';
    END IF;
    RETURN NEW;
  END;
  $$
  LANGUAGE plpgsql;

CREATE TRIGGER pode_comprar BEFORE INSERT ON conteudo_pedido
FOR EACH ROW EXECUTE PROCEDURE estoque_disponivel();
```

Figura 4 : Código SQL do trigger de estoque disponível

Explicação do código:

É um trigger de linha que antes da inserção na tabela conteudo_pedido executa a função estoque_disponivel.A função estoque_disponivel verifica se a quantidade disponível da refeição na tabela cardápio subtraído pela quantidade pedida da mesma refeição é menor que 0, se for, ele levanta uma exceção e o novo pedido não é realizado.

Trigger 3:

Um pedido só pode ser realizado se a loja que vende a refeição estiver no horário de funcionamento.

Código:

```
CREATE OR REPLACE FUNCTION loja_aberta() RETURNS trigger AS $$
  BEGIN
    IF ( (SELECT hora_pedido FROM pedido WHERE NEW.cod_pedido=pedido.cod_pedido) >
                            (SELECT hora_abre FROM Loja WHERE Loja.cod_loja =
                            (SELECT loja FROM cardapio WHERE NEW.prato=cardapio.cod_cardap))
      AND (SELECT hora_pedido FROM pedido WHERE NEW.cod_pedido=pedido.cod_pedido) <
                            (SELECT hora_fecha FROM Loja WHERE Loja.cod_loja =
                            (SELECT loja FROM cardapio WHERE NEW.prato=cardapio.cod_cardap)))
      THEN RETURN NEW;
    END IF;
    RETURN OLD;
    END;
    $$
    LANGUAGE plpgsql;

CREATE TRIGGER pode_comprar_loja_aberta BEFORE INSERT ON conteudo_pedido
FOR EACH ROW EXECUTE PROCEDURE loja_aberta();
```

Figura 5 : Código SQL do trigger de verificação do horário de funcionamento da loja

Explicação do código:

É um trigger de linha que antes da inserção na tabela conteudo_pedido executa a função loja_aberta.A função loja_aberta confere (através de SELECT) se a hora do pedido está entre o horário de funcionamento da loja,se estiver , retorna o NEW.

Trigger 4:

Um pedido só pode possuir refeições de uma mesma loja.
Codigo:

```
CREATE OR REPLACE FUNCTION mesma_loja() RETURNS trigger AS $$
  BEGIN
    IF EXISTS(SELECT *
        FROM conteudo_pedido INNER JOIN cardapio ON conteudo_pedido.prato=cardapio.cod_cardap
        GROUP BY conteudo_pedido.prato
        HAVING COUNT(DISTINCT loja) > 1)  THEN
          RAISE EXCEPTION 'Pedido com mais de uma loja ao mesmo tempo';
    END IF;
  RETURN NULL;
  END;
  $$
  LANGUAGE plpgsql;

CREATE TRIGGER confere_pedido AFTER INSERT ON conteudo_pedido
FOR EACH statement EXECUTE PROCEDURE mesma_loja();
```

Figura 6 : Código SQL do trigger de verificação de pedidos

Explicação do código:

É um trigger de função que depois da inserção na tabela conteudo_pedido executa a função mesma_loja.A função mesma_loja confere se existe alguma tupla da união(INNER JOIN) entre as tabelas cardápio e conteudo_pedido, onde o mesmo código do pedido está associado a mais de um código de loja.

Trigger 5

O conteúdo de um pedido não pode ser modificado (tuplas podem apenas ser incluídas).

Código:

```
CREATE OR REPLACE FUNCTION fnprevent_update()
  RETURNS trigger AS $$
  BEGIN
    RAISE EXCEPTION 'Proibido modificação de pedidos, cancele o pedido
                    e crie um novo, ou apenas crie um novo.';
  END
  $$
  LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_update
  BEFORE UPDATE OF cod_pedido, prato, qnt ON conteudo_pedido
  FOR EACH ROW EXECUTE PROCEDURE fnprevent_update();
```

Figura 7 : Código SQL do trigger de impedimento de modificação do conteúdo do pedido

Explicação do código:

É um trigger que impede que o conteúdo do pedido seja modificado, a fim de proteger a integridade do banco de dados, no que se refere ao estoque, uma vez que, por exemplo, alterar a quantidade de um pedido, não implicará em modificação no estoque, o que implicaria em perda de integridade, se for desejado a modificação para menos da quantidade ou troca do produto, se deve cancelar o pedido, e criar um novo com os dados desejados. Se for desejado aumentar a quantidade de um determinado produto, basta criar um novo pedido.

4 Relatórios de Saída

Relatório 1:
Relatório para saber quanto foi o valor total de um pedido realizado por um cliente.

Código:

```
CREATE OR REPLACE FUNCTION comanda (nPedido INT)
RETURNS DEC(10,2) AS $$
  DECLARE
    curs CURSOR FOR SELECT * FROM conteudo_pedido WHERE nPedido=cod_pedido;
    c_linha conteudo_pedido%rowtype;
    soma DEC(10,2);
  BEGIN
    soma:=0;
    IF nPedido IS NOT NULL THEN
      FOR c_linha IN curs LOOP
        soma=soma + (SELECT preco_unitario FROM cardapio WHERE cod_cardap=c_linha.prato);
      END LOOP;
    END IF;
    RETURN soma;
  END;
$$
LANGUAGE plpgsql;
```

Figura 8 : Código SQL do relatório que retorna o valor total do pedido.

Explicação do código:
A função recebe um código de pedido e calcula a soma do preço de todos os pratos pedidos. A função usa um cursor para andar em todos as refeições compradas naquele pedido, e para cada pedido há um SELECT para recuperar o preço das refeições na tabela cardápio.

Relatório 2:
Relatório para saber qual o ranking das lojas pela avaliação.

Código:

```
CREATE OR REPLACE FUNCTION ranking()
  RETURNS TABLE( loja CHARACTER VARYING,
                 nota DEC(10,2)
                 ) AS $$
  BEGIN
    RETURN
    QUERY
      SELECT loja.nome,avg(nota)
      FROM Avaliacao INNER JOIN Entrega ON Avaliacao.pedido=Entrega.cod_ent INNER JOIN conteudo_pedido ON conteudo_pedido.cod_pedido=Entrega.pedido
           INNER JOIN cardapio ON conteudo_pedido.prato=cardapio.cod_cardap INNER JOIN Loja on cardapio.loja=Loja.cod_loja
      GROUP BY Loja.cod_loja;
  END;
  $$
  LANGUAGE plpgsql;
```

Figura 9: Código SQL do relatório que retorna o ranking das lojas pela avaliação.

Explicação do código:
A função retorna uma tabela com o nome das lojas e suas respectivas notas através de uma query, que com a junção de tabelas calcula a média das avaliações de cada loja separadas por um GROUP BY.