Trabalho da disciplina banco de dados II - UERJ 2020/2

Integrantes

- Dennis Ribeiro Paiva - 201610050611
- Igor Sousa Silva - 201610053411
- Paulo Victor Coelho - 201610049711
- Vinicius Sathler - 201610051611

Introdução

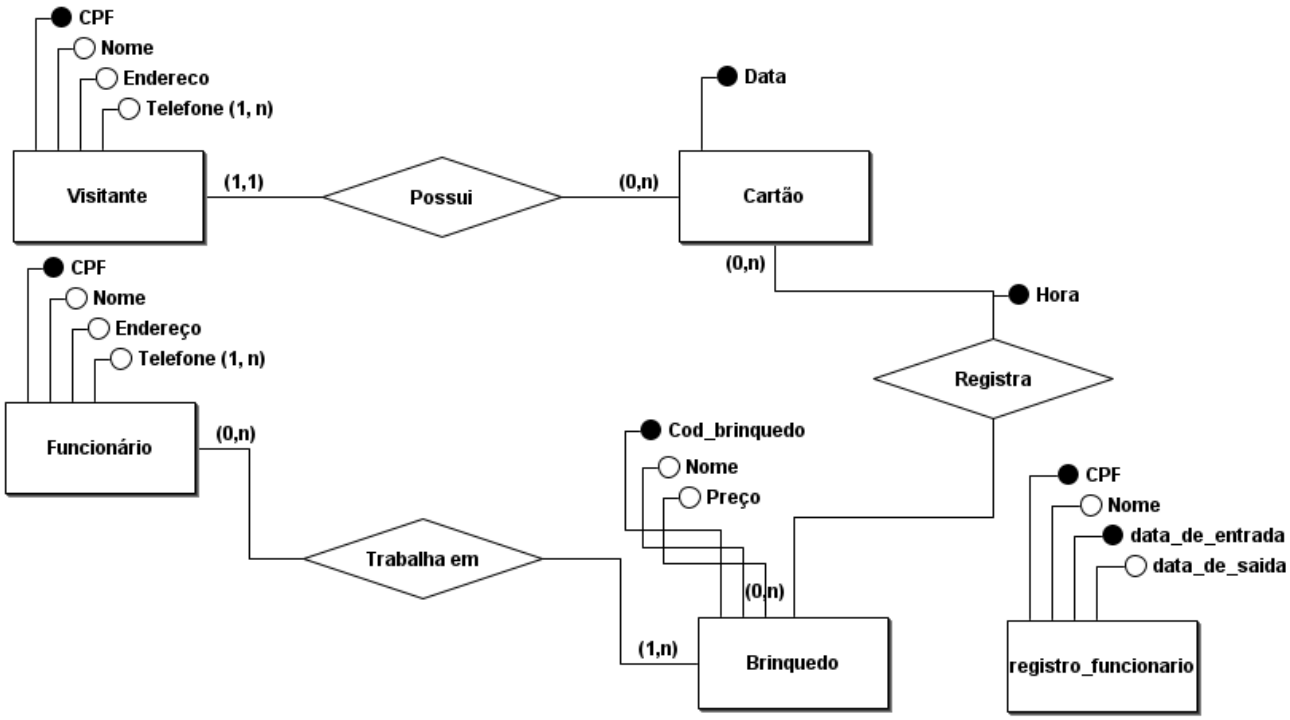
Neste trabalho serão apresentados todos os passos de modelagem de um projeto básico de banco de dados, da descrição do minimundo até sua implementação funcional. O tema abordado será o gerenciamento do cartão de visita de um visitante em um parque de diversões.

Minimundo

O parque de diversões 'tomorrowland' é um parque moderno mas muito ganancioso. Seus visitantes recebem na entrada um cartão digital que deve ser apresentado na entrada de cada brinquedo. Cada brinquedo possui um nome e código de identificação. Sendo a gerência do parque muito gananciosa, cada cartão de visitante deve registrar a cobrança de entrada nos brinquedos cada vez que for utilizado, ou seja, o visitante paga cada vez que for usar um brinquedo. Os funcionários deste parque também são muito ocupados, tendo muitas vezes que trabalhar em mais de um brinquedo, sendo que cada brinquedo pode precisar de um ou mais funcionários. Para estimular uma concorrência saudável entre seus funcionários, a gerência do parque paga um adicional de dois por cento do dinheiro arrecadado em cada brinquedo para cada funcionario responsável por ele. A fim de evitar fraudes, tanto os clientes quanto os funcionários devem ser registrados de acordo com o seu nome completo, CPF, endereço e telefone(s) para contato. Para disfarçar sua ganância o parque permite que cada cartão sejá válido por um dia inteiro. Ao final do dia o visitante deve pagar o valor acumulado de todos os brinquedos que visitou.

Modelo conceitual

Diagrama entidade-relacionamento



Restrições de Domínio

- Visitante:
  - CPF: Número inteiro de onze dígitos.
  - Nome: String de no máximo 45 caracteres.
  - Endereço: String de no máximo 100 caracteres.
  - Telefone: Numero inteiro formado por oito ou nove dígitos.
- Cartão:
  - Data: Data no formato aaaa-mm-dd de acordo com o tipo DATE da linguagem PostgreSQL.
- Brinquedo:
  - Cod_brinquedo: Numero inteiro de cinco dígitos.
  - Nome: String de no máximo 45 caracteres.
  - Preço: Número real positivo com duas casas decimais de precisão.
- Registra (Cartão-Brinquedo):
  - Hora: registro de hora no formato hh-mm-ss de acordo com o tipo TIME da linguagem PostgreSQL.
- Funcionário:
  - CPF: Número inteiro de onze dígitos.
  - Nome: String de no máximo 45 caracteres.
  - Endereço: String de no máximo 100 caracteres.
  - Telefone: Numero inteiro formado por oito ou nove dígitos.

- Registro do funcionario:
    - CPF: número inteiro de onze dígitos
    - Nome: String de no máximo 45 caracteres.
    - data_de_entrada: Data no formato aaaa-mm-dd de acordo com o tipo DATE da linguagem PostgreSQL.
    - data_de_saida: Data no formato aaaa-mm-dd de acordo com o tipo DATE da linguagem PostgreSQL.

Modelo Relacional

Descrição

- **Chave primária**
- *Chave estrangeira*

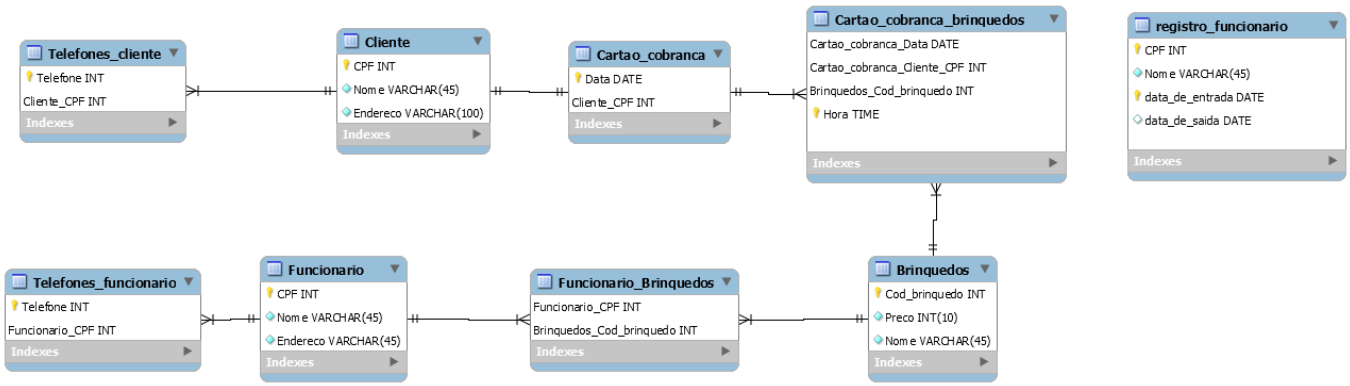
- Cliente (**CPF**, Nome, Endereço)
- Telefones_cliente (**Telefone**, ***Cliente_CPF***)
- Cartao_Cobrança (**Data**, ***Cliente_CPF***)
- Brinquedo (**Cod_brinquedo**, Nome, Preço)
- Cartao_cobranca_brinquedo (**Hora**, ***Cartao_Cobrança_Data***, ***Cartao_Cobranca_Cliente_CPF***, ***Brinquedo_Cod_brinquedo***)
- Funcionario (**CPF**, Nome, Endereço)
- Telefones_funcionario (**Telefone**, ***Funcionario_CPF***)
- Funcionario_brinquedo (***Funcionario_CPF***, ***Brinquedo_Cod_brinquedo***)
- Registro_funcionario: (**CPF**, Nome, **data_entrada**, data_saida)

Sabendo-se que cada Cliente possui nenhum ou vários cartões, mas que cada cartão está vinculado a obrigatoriamente um e apenas um cliente, foi adicionada uma coluna extra na tabela cartão para referenciar o cliente ao qual ele está asssociado

Como cada cartão de cobrança pode registrar nenhum ou vários brinquedos, e cada brinquedo pode estar registrado em nenhum ou vários cartões, criou-se uma nova tabela para relação

Cada funcionário pode trabalhar em nenhum ou mais de um brinquedo, e cada brinquedo possui entre um e varios funcionários. De maneira análoga a relação brinquedo-cartão foi criada uma nova tabela

Diagrama



Restrições Semânticas

1. Cada funcionário não pode receber menos do que o salário minímo, estipulado em $sc 500,00
2. O visitante paga por cada brinquedo visitado, mas não pode pagar menos de $sc 100,00

Visões

Visão 1

Criada com o objetivo de dispor a tabela de preços dos brinquedos do parque, enquanto suprime do usuário o código do brinquedo.

```
CREATE OR REPLACE VIEW preco_brinquedo
AS
    SELECT Nome, Preco
    FROM Brinquedos
    ORDER BY Nome;
```

Visão 2

Lista meios de contato com os funcionários do parque, suprimindo informações pessoais como o CPF.

```
CREATE OR REPLACE VIEW contatos_funcionario
AS
    SELECT nome, endereco, telefone
    FROM funcionario INNER JOIN Telefones_funcionario
    ON cpf = Funcionario_CPF;
```

Visão 3

Lista funcionários e seus respectivos brinquedos, ocultando informações relevantes apenas ao banco de dados (CPF e Código do Brinquedo)

```
CREATE OR REPLACE VIEW funcionarios_brinquedo
AS
    SELECT b.nome AS nome_do_brinquedo, f.nome AS nome_funcionario
    FROM funcionario f, funcionario_brinquedos, brinquedos b
    WHERE cpf = Funcionario_CPF AND cod_brinquedos = brinquedos_cod_brinquedo;
```

Funções

Função 1

Criada para retornar uma string compatível com o tipo DATE dado um mês e um ano com o objetivo de realizar comparações entre datas

```
CREATE OR REPLACE FUNCTION first_day_of(integer, integer)
RETURNS varchar as $$
DECLARE str varchar;
BEGIN
    str := '-01';
    case $1
        when 1, 2, 3, 4, 5, 6, 7, 8, 9 THEN
            str := $2||'-0'||$1||str;
        WHEN 10, 11, 12 THEN
            str := $2||'-'||$1||str;
        WHEN 13 THEN
            str := $2+1||'-01'||str;
    end case;
    return str;
END
$$ LANGUAGE plpgsql
```

Função 2

Criada para listar os funcionários em ordem de clientes atendidos, premiando o melhor com o título "Funcionário do Mês" e parabenizando aqueles que superaram a média de atendimentos.

```
CREATE OR REPLACE FUNCTION bonifica_funcionario(int, int)
RETURNS VARCHAR AS $$
DECLARE
    cursor_freq CURSOR FOR
        SELECT f.nome, sum(freq_table.freq) AS soma
        FROM funcionario F, funcionario_brinquedos func_rel_brinq,
            (SELECT brinquedos_cod_brinquedo, count(cartao_cobranca_cliente_cpf)
AS freq
            FROM cartao_cobranca_brinquedos
            WHERE cartao_cobranca_data >= to_date(first_day_of($1, $2), 'YYYY-MM-
DD') AND
            cartao_cobranca_data < to_date(first_day_of($1+1, $2), 'YYYY-MM-DD')
            GROUP BY brinquedos_cod_brinquedo) AS freq_table
        WHERE cpf = funcionario_cpf AND
        freq_table.brinquedos_cod_brinquedo =
func_rel_brinq.brinquedos_cod_brinquedo
        GROUP BY f.nome
        ORDER BY soma DESC, f.nome;
    freq numeric;
    media_freq numeric;
    media numeric;
    linhas numeric;
    nome varchar;
    str varchar;
BEGIN
    str := '';
    linhas := 0;
    media := 0;
    OPEN cursor_freq;
    LOOP
        FETCH NEXT FROM cursor_freq
        INTO nome, freq;
        EXIT WHEN NOT FOUND;
        linhas := linhas + 1;
        media := media + freq;
    END LOOP;
    IF linhas > 0 THEN
        media := media/linhas;
    END IF;

    FETCH FIRST FROM cursor_freq
    INTO nome, freq;
    IF nome IS NOT NULL THEN
        str := str||nome||' '||'funcionario_do_mes, '||freq;
```

```
    END IF;
    LOOP
        FETCH NEXT FROM cursor_freq
        INTO nome, freq;
        EXIT WHEN NOT FOUND;
        IF freq >= media THEN
            str := str||nome||' '||'parabens, '||freq;
        ELSE
            str := str||nome||' '||'precisa melhorar, '||freq;
        END IF;
    END LOOP;
    CLOSE cursor_freq;
    RETURN str;
END;
$$ LANGUAGE plpgsql
```

Triggers (Gatilho)

A primeira trigger desenvolvida insere na tabela registro_funcionario o registro de entrada de um novo funcionário, enquanto a segunda trigger registra data de saida de um funcionário na deleção do mesmo do quadro de funcionários atual.

Trigger 1

```
CREATE OR REPLACE FUNCTION fc_registra_entrada_funcionario()
RETURNS trigger AS $$
DECLARE
    data_atual date;
BEGIN
    SELECT CURRENT_DATE INTO data_atual;
    INSERT INTO registro_funcionario
    VALUES (new.cpf, data_atual, new.nome, null);
    RETURN null;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER registra_entrada_funcionario AFTER INSERT
ON funcionario
FOR EACH ROW EXECUTE PROCEDURE fc_registra_entrada_funcionario();
```

Trigger 2

```
CREATE OR REPLACE FUNCTION fc_registra_saida_funcionario()
RETURNS trigger AS $$
DECLARE
    data_atual date;
BEGIN
    SELECT CURRENT_DATE INTO data_atual;
    UPDATE registro_funcionario set data_de_saida = data_atual
    WHERE old.cpf = cpf AND
    data_de_saida IS NULL;
    RETURN new;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER atualiza_saida_funcionario AFTER DELETE
ON funcionario
FOR EACH ROW EXECUTE PROCEDURE fc_registra_saida_funcionario();
```

Índice

```
CREATE INDEX indNome ON cliente using hash(nome);
```

O comando acima cria uma tabela hash para o campo nome na tabela cliente, sendo assim quando ocorre uma busca por um nome específico na tabela o sistema restringe a busca ao provável endereço dessa linha, não precisando iterar sobre toda a tabela como normalmente seria feito.

Exemplo

Nem sempre a busca pelo registro do cliente será feita pelo cpf do mesmo, muitas vezes temos disponível apenas o nome, a indexacão da coluna nome melhora o desempenho deste tipo de consulta, como demonstrado nas imagens a seguir.

Consulta não indexada

```
ocadora_igor411.tomorrowland> SELECT *
                              FROM cliente
                              where nome = 'Giorgian Daniel De Arrascaeta Benedetti'
2021-03-27 01:00:00] 1 row retrieved starting from 1 in 63 ms (execution: 7 ms, fetching: 56 ms)
```

Consulta indexada

```
locadora_igor411.tomorrowland> SELECT *
                               FROM cliente
                               where nome = 'Giorgian Daniel De Arrascaeta Benedetti'
[2021-03-27 01:00:48] 1 row retrieved starting from 1 in 28 ms (execution: 7 ms, fetching: 21 ms)
```