TRABALHO DE IMPLEMENTAÇÃO - BANCO DE DADOS II

SISTEMA DE AGÊNCIA BANCÁRIA

Alunos: Gabriel Duarte e Matheus Tavares

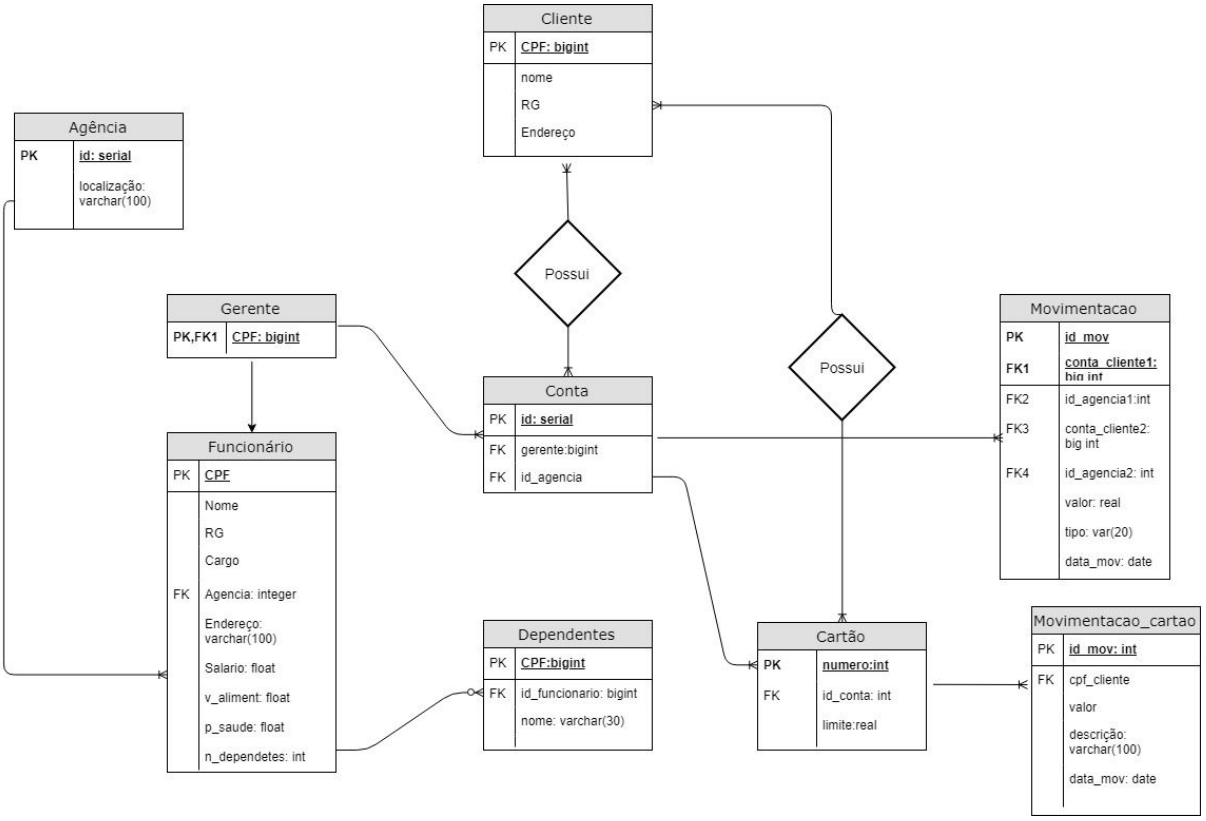
Prof: Luiz André

Niterói, 18/06/2018

Descrição

Modelagem de sistema de banco de dados de uma agência bancária, onde são especificados regras de negócio e relações entre entidades desse sistema

Modelo Entidade Relacionamento



Regras de Negócio

1) A consistência das movimentações e sua aplicação às respectivas contas participantes devem ser garantidas. Exemplo: Em uma transferência de cliente A para cliente B, a operação só é realizada se o cliente A possui saldo para tal, e além disso, os valores de saldo das respectivas contas são alterados

```
CREATE OR REPLACE FUNCTION verifica_movimentacoes()
returns trigger as $$

DECLARE
	r1 record;
	r2 record;

BEGIN

	select * into r1 from conta where new.conta_cliente1 = conta.id;

	if r1 = null then
		raise exception 'todos os tpos de movimentacao requerem no minimo
uma conta';
		return null;

	elsif new.tipo = 'saque' then
		if new.valor>r1.saldo then
			raise exception 'valor maior que seu saldo';
			return null;
		end if;

	elsif new.tipo = 'emprestimo' then
		if r1.saldo>0 then
			raise exception 'O emprestimo só pode ser feito por conta com
saldo<=0';
			return null;
		elsif new.valor>r1.limite then
			raise exception 'valor maior que seu limite';
			return null;
		end if;

	elsif new.tipo = 'deposito' then
		update conta set saldo = saldo+new.valor where
id=new.conta_cliente1;
		return new;

	elsif new.tipo = 'transferencia' then
		select * into r2 from conta where new.conta_cliente2 = conta.id;

		if r2=null then
			raise exception 'o tipo trasnferencia requer duas contas';
```

```
                return null;

            elsif new.conta_cliente1 = new.conta_cliente2 then
                raise exception 'o tipo trasnferencia requer duas contas
diferentes';
                return null;

            else
                if r1.saldo<new.valor then
                    raise exception 'transferencia não foi possivel, saldo
menor que valor de transferencia';
                    return null;
                else
                    update conta set saldo = saldo-new.valor where
id=new.conta_cliente1;
                    update conta set saldo = saldo+new.valor where
id=new.conta_cliente2;
                    return new;
                end if;
            end if;

        end if;

        update conta set saldo = saldo-new.valor where id=new.conta_cliente1;
        return new;
end;
$$
language plpgsql;

drop trigger verifica_movimentacoes on movimentacao_emp;
CREATE trigger verifica_movimentacoes before insert on movimentacao_emp
        FOR EACH ROW EXECUTE PROCEDURE verifica_movimentacoes();
```

2) O funcionário comum tem direito a 2 dependentes associados enquanto o gerente tem direito a 4. A inserção e as atualizações de dependentes devem ser feitas de maneira que o atributo n_dependentes da tabela funcionário seja consistente com o banco

CREATE OR REPLACE FUNCTION n_dependentes() returns TRIGGER AS $$

```
-- trigger que garante o valor correto do atributo derivado "n_dependentes" da entidade funcionario

DECLARE

	c_funcionario cursor for select*from funcionario where funcionario.cpf=new.cpf
	limite_gerente int = 4;
	limite_comum int =2;
	n_dep int;

BEGIN

	if(TG_OP = 'INSERT') then

		select n_dependentes into n_dep from funcionario as f where f.cpf = new.id_funcionario;

		if (c_funcionario.cargo='gerente'and n_dep<limite_gerente or c_funcionario.cargo='comum' and n_dep<limite_comum) then
			UPDATE funcionario SET n_dependentes = n_dependentes+1 WHERE cpf=new.id_funcionario;
			return new;
		else
			raise exception 'O funcionário já atingiu o limite de dependentes';
			return null;
		end if;
	elsif (TG_OP = 'DELETE') then
		select n_dependentes into n_dep from funcionario as f where f.cpf = old.id_funcionario;

		--raise notice '% n dep', old.cpf;

		if(n_dep>0)then
```

```
                    UPDATE funcionario SET n_dependentes = n_dependentes-1
WHERE cpf=old.id_funcionario;
                end if;
                return old;

        elsif (TG_OP = 'UPDATE')then
                if(new.id_funcionario <> old.id_funcionario)then
                    select n_dependentes into n_dep from funcionario as f where
f.cpf = new.id_funcionario;

                    if(n_dep>=limite_dep)then
                        raise exception 'novo funcionario ja esta no limite de
dependentes';
                        return null;
                    end if;

                    UPDATE funcionario SET n_dependentes = n_dependentes+1
WHERE cpf=new.id_funcionario;
                    UPDATE funcionario SET n_dependentes = n_dependentes-1
WHERE cpf=old.id_funcionario;
                    return new;
                end if;
        end if;

        return null;

END;
$$
LANGUAGE plpgsql;

drop trigger n_dependentes on dependente;
CREATE TRIGGER n_dependentes before INSERT or update or delete on
dependente
FOR EACH ROW EXECUTE PROCEDURE n_dependentes();
```

3) Os benefícios (salário, vale alimentação e plano de saúde) de um funcionário que é gerente são necessariamente superiores comparada ao funcionário comum do banco.

```
-- função para garantir que os beneficios de um gerente sejam maiores que o do
resto dos funcionarios
```

-- *detalhe: entre funcionarios da mesma agencia*

CREATE OR REPLACE FUNCTION gerente_privilegios()returns TRIGGER AS $$

```
DECLARE
	r1 funcionario%rowtype;
BEGIN

	if(new.cargo = 'gerente')then
		for r1 in select * from funcionario WHERE funcionario.cargo<>'gerente'
and new.agencia=funcionario.agencia loop
			if new.salario<r1.salario or new.v_aliment<r1.v_aliment
				or new.p_saude<r1.p_saude THEN

				RAISE EXCEPTION 'Um gerente não pode possuir
algum beneficio inferior ao de algum funcionario';
				return null;
			end if;
		end loop;

	--caso o seja um funcionario 'comum' vem para cá
	else
		for r1 in select * from funcionario WHERE
funcionario.cargo='gerente'and new.agencia=funcionario.agencia loop
			if new.salario>r1.salario or new.v_aliment>r1.v_aliment
				or new.p_saude>r1.p_saude THEN
				RAISE EXCEPTION 'O funcionario não pode possuir
algum beneficio superior ao de algum gerente';
				return null;
			end if;
		end loop;
	end if;

	return new;
END;
$$
LANGUAGE plpgsql;
```

CREATE TRIGGER gerente_privilegios BEFORE INSERT OR UPDATE ON FUNCIONARIO
FOR EACH ROW EXECUTE PROCEDURE gerente_privilegios();

4) Uma conta necessariamente tem um responsável (gerente) de sua agência registrada.

CREATE OR REPLACE FUNCTION responsavel() returns TRIGGER AS $$

```
DECLARE

c_contas cursor for select * from conta where conta.gerente_id = new.id;

BEGIN

        if c_contas<>null then

                if new.agencia<>old.agencia then
                        RAISE EXCEPTION " É necessário atualizar o responsável das contas sob a responsabilidade desse gerente";

                end if;
        end if;

        return new;
END;
$$
LANGUAGE plpgsql;
```

CREATE trigger responsavel before update gerente
FOR EACH ROW execute procedure responsavel()

5) A tabela gerente é preenchida sempre que um funcionário gerente é registrado ou um funcionário comum é promovido.

CREATE OR REPLACE FUNCTION up_gerente() returns trigger AS &&

```
DECLARE

BEGIN

        IF (TG_OP= 'INSERT' and new.cago='gerente') THEN
                INSERT into GERENTE(cpf) values (new.cpf);


        ELSIF TG_OP= 'UPDATE'

                if funcionario.cargo='comum' and new.cargo='gerente' THEN
                        INSERT into GERENTE(cpf) values (new.cpf);
                END IF;


                IF funcionario.cargo='gerente' and new.cargo='comum' THEN

                        DELETE FROM gerente WHERE gerente.cpf=new.cpf;

                END IF;

        END IF;

        return new;
END
$$
LANGUAGE plpgsql;
```

CREATE TRIGGER up_gerente AFTER INSERT or BEFORE UPDATE on FUNCIONARIO
FOR EACH ROW EXECUTE PROCEDURE up_gerente();

Criação das tabelas

```
create table if not exists cliente(
	CPF bigint not null unique,
	nome varchar(30) not null,
	RG bigint not null unique,
	endereco varchar(100) not null,

	constraint pk_cliente primary key(CPF)
);

create table agencia(
	id serial unique,
	localizacao varchar(100) not null unique,
	constraint pk_agencia primary key(id)
);

create table if not exists funcionario(
	CPF bigint not null unique,
	nome varchar(30) not null,
	RG bigint not null unique,
	cargo varchar(12) not null DEFAULT 'comum',
	agencia integer not null,
	endereco varchar(100),
	salario float not null,
	v_aliment float not null DEFAULT '450',
	p_saude float not null DEFAULT '500',
	n_dependentes int DEFAULT '0',

	constraint pk_funcionario primary key(CPF),

	constraint fk_agencia foreign key(agencia) references agencia(id)
		on delete cascade on update cascade

);
```

```
create table if not exists gerente(
	CPF bigint not null unique,

	constraint pk_gerente primary key(CPF),
	constraint fk_gerente_funcionario foreign key(CPF) references
funcionario(CPF) --updates e deletes feito via trigger

);

create table if not exists conta(
	id serial,
	gerente bigint not null,
	id_agencia int not null,
	saldo bigint not null DEFAULT '0',

	constraint pk_conta primary key (id),

	constraint fk_conta_cliente foreign key(gerente) references gerente(CPF)
		on delete restrict on update cascade,

	constraint fk_agencia foreign key(id_agencia) references agencia(id)
);


CREATE TABLE if not exists movimentacao_emp(
	id_mov serial unique not null,
	conta_cliente1 bigint not null,
	id_agencia1 int not null,
	conta_cliente2 bigint,
	id_agencia2 int,
	valor integer not null,
	tipo varchar(20) not null,
	data_mov date not null,
	situacao varchar(20) not null DEFAULT ‘quitado’;

	CONSTRAINT pk_movimentacao_emp
		primary key (id_mov),

	CONSTRAINT fk1_movemp_cliente
		foreign key (conta_cliente1) references conta(id)
		on delete cascade on update cascade,
```

```
	CONSTRAINT fk2_movemp_agencia1
		foreign key (id_agencia1) references agencia(id)
		on delete cascade on update cascade,

	CONSTRAINT fk3_movemp_cliente
		foreign key (conta_cliente2) references conta(id)
		on delete cascade on update cascade,

	CONSTRAINT fk4_movemp_agencia2
		foreign key (id_agencia2) references agencia(id)
		on delete cascade on update cascade,

	CHECK (valor>0)
);

create table if not exists possui_conta(
	id_cliente bigint,
	id_conta int,

	constraint pk_possui primary key(id_cliente, id_conta),

	constraint fk_possui_cliente foreign key(id_cliente) references cliente(CPF)
		on delete cascade on update cascade,
	constraint fk_possui_conta foreign key(id_conta) references conta(id)
		on delete cascade on update cascade
);

create table if not exists dependente(
	CPF bigint,
	id_funcionario bigint,
	nome varchar(30),

	constraint pk_dependente primary key(CPF),

	constraint fk_dep_funcionario foreign key(id_funcionario) references
funcionario(CPF)
		on delete cascade on update cascade
);
```

```
create table if not exists cartao (
	numero int not null,
	id_conta int not null,
	limite real not null,
	constraint pk_cartao primary key(numero),

	constraint fk_cartao_conta foreign key(id_conta) references conta(id)
);

create table if not exists possui_cartao(
	id_cliente bigint not null,
	numero_cartao int not null,
	saldo_cartao real not null,

	constraint pk_possui_cartao primary key (id_cliente, numero_cartao),

	constraint fk_possui_cliente foreign key(id_cliente) references cliente(CPF)
		on delete cascade on update cascade,

	constraint fk_possui_cartao foreign key(numero_cartao) references
cartao(numero)
		on delete cascade on update cascade
);

create table if not exists movimentacao_cart(
	id_mov serial unique not null,
	cpf_cliente bigint not null,
	valor integer not null,
	descricao varchar(100),
	data_mov date not null,

	CONSTRAINT pk_movimentacao_cart
		primary key (id_mov),

	CONSTRAINT fk1_movcart_cliente
		foreign key (cpf_cliente) references cliente(CPF)
		on delete cascade on update cascade
);
```

Funções criadas

1) *Função que calcula multa/juros ao ultrapassar o limite estabelecido para uma conta. O saldo é atualizado de acordo com a respectiva multa.*

```
CREATE OR REPLACE FUNCTION calcula_juros()
returns void as $$

DECLARE

	emprestimos cursor for select * from movimentacao_emp
                                                where tipo = 'emprestimo';

	r1 record;
BEGIN

	for r1 in emprestimos loop
		if r1.situacao <> 'quitado' then
			update conta set saldo = saldo+(r1.juros*saldo) where id =
r1.conta_cliente1;
		end if;
	end loop;

END;
$$
language plpgsql;
```

2) *Função que retorna todos os clientes que possuem dívida com a agência*

```
CREATE OR REPLACE FUNCTION clientes_agencia(id_agencia int)
returns table(cpf_cliente bigint, nome_cliente varchar) as $$

DECLARE

	contas_agencia cursor for select c.id as id_conta from conta as c
                                                inner join agencia on
c.id_agencia = agencia.id
                                                where c.saldo<0;
```

```
        r1 record;
        r2 record;
BEGIN

        for r1 in contas_agencia LOOP

                for r2 in (select cl.cpf as cpf, cl.nome as nome from possui_conta pc
                                        inner join cliente cl on pc.id_cliente = cl.cpf
                                        where r1.id_conta = pc.id_conta) LOOP

                        return query select r2.cpf, r2.nome;

                END LOOP;

        END LOOP;

        return;
END;
$$
language plpgsql;
```