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

Alunos: Mateus Azeredo Torres
Matheus Moraes Cruz

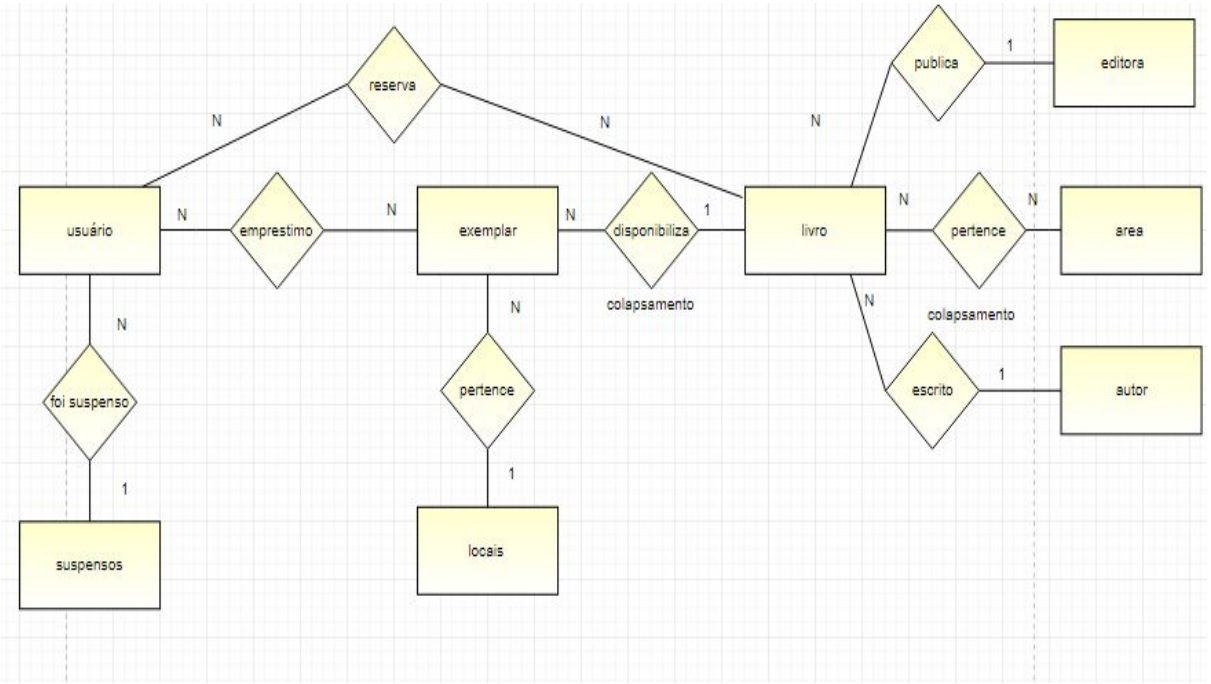
Rede de Bibliotecas Database

SUMÁRIO

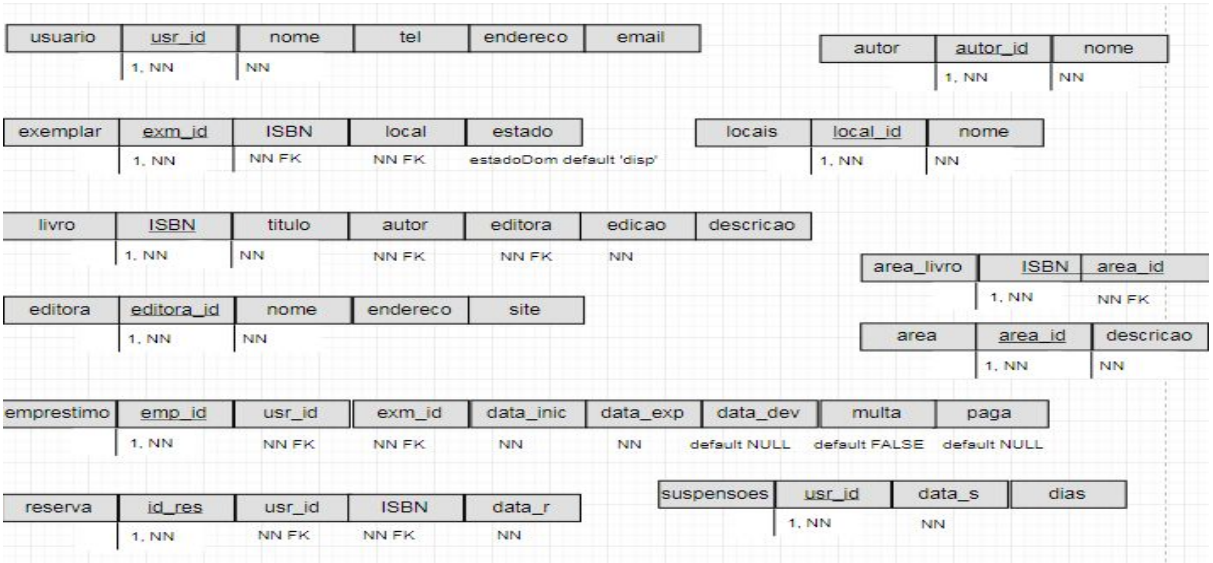
1 - Introdução e Objetivo

O objetivo é criar um sistema de banco de dados de uma biblioteca, que ficará responsável por fazer o controle dos empréstimos de livros, mantendo a integridade do banco de dados, segundo as regras definidas (regras de negócio). O sistema basicamente deve controlar o tempo de empréstimo para cada usuário, informando a penalidade caso ocorra atraso de devolução, assim como garantir que os empréstimos são válidos e também outras regras.

1 - Estrutura e Modelo E-R



Tabelas:



arquivo das tabelas no git: popular

3 - Triggers e Regras de Negócio

Trigger 1 :

- O usuário só poderá fazer empréstimos de um certo livro se cumprir as seguintes regras:
  - Não ter multas pendentes.
  - Não estar suspenso.
  - Ter menos de 5 livros emprestados
  - O livro escolhido está disponível(não reservado ou emprestado).

- A função além de verificar as condições ditas acima que permitem um empréstimo, também checa se a tupla inserida tem valores coerentes:
  - intervalo de 7 dias para empréstimos .
  - Tuplas inseridas não devem iniciar com datas de devolução ou multas.

Toda vez que um empréstimo for realizado(**ocorrer inserção na tabela emprestimo**) a função validaemprestimo() é chamada **antes** da operação de **inserção**.

Arquivo no git : pegaemprestado

Print do Código:

```
BEGIN
    -- regras que definem uma inserção(emprestimo valido)
        IF( ( (NEW.multa = TRUE) OR (NEW.data_paga is not NULL) ) ) THEN
                RAISE EXCEPTION 'emprestimos nao podem iniciar com multa, favor alterar após a inserção';
        END IF;
        -- coerencia de devolução ao inserir
        IF( (NEW.data_dev is not NULL)  ) THEN
                RAISE EXCEPTION 'emprestimos nao podem iniciar com devoluções, favor alterar após a inserção';
        END IF;
        -- coerencia de datas
        IF( cast(DATE_PART('day',NEW.data_exp - NEW.data_inic) AS integer) = 7 ) THEN
            RAISE EXCEPTION 'periodo de emprestimo INVALIDO!';

        END IF;
        -- regras que definem que um usuario nao pode pegar o livro emprestado
        SELECT estado INTO teste2 FROM exemplar WHERE exm_id = NEW.exm_id;
        IF(estado != 'disp') THEN
                RAISE EXCEPTION 'ERRO! O Exemplar referido nao esta disponivel para emprestimo';
        END IF;
        -- SE EXISTIR PELO MENOS UMA MULTA em nome desse usuario que nao tenha sido paga nao pode pegar livros ainda
        IF (EXISTS(SELECT 1 FROM emprestimo WHERE multa = TRUE AND data_paga is NULL AND usr_id = NEW.usr_id)) THEN
            RAISE EXCEPTION 'o usuario tem multas pendentes';
        END IF;
        -- verifica se nao esta suspenso
        SELECT * INTO teste4 FROM suspensoes WHERE usr_id = NEW.usr_id;
        IF(cast(DATE_PART('day',now()::timestamp - teste4.data_s::timestamp) AS INTEGER) < teste4.dias ) THEN
            RAISE EXCEPTION 'O usuario esta suspenso';
        END IF;
        -- limite de 5 livros por usuario
        SELECT COUNT(*) AS total INTO teste1 FROM emprestimo WHERE usr_id = NEW.usr_id AND data_dev is NULL;
        IF(cast(teste1.total AS INTEGER) = 5 ) THEN
            RAISE EXCEPTION 'o usuario nao pode pegar mais livros';
        END IF;
        -- verifica se nao tem RESERVA DESSE LIVRO em nome de outro usuario!(reservas só valem até 3 dias!)
        IF(EXISTS(SELECT ISBN from reserva
                    WHERE ( (ISBN IN (SELECT exemplar.ISBN  from exemplar WHERE exm_id = NEW.exm_id) ) AND
                    (DATE_PART('day',now()::timestamp - data_r::timestamp) =< 3   ) AND (reserva.usr_id != NEW.usr_id) ) ))
                    THEN raise EXCEPTION 'o livro tem uma reserva feita antes do seu pedido';
        END IF;
        RETURN NEW; -- se nao foi filtrado por nenhuma dessas ele pode inserir
```

Trigger 2 :

- Um exemplar só pode ser movimentado de uma biblioteca para outra ou para manutenção se cumprir as seguintes condições:
  - Não é o último exemplar de um livro naquela biblioteca .
  - Só pode ir para a manutenção se for um dos livros mais requisitados no período de 6 meses.

- A função usa uma **<u>função auxiliar</u>(olhar funções: Pg.8)** maisemprestados(intervalo), que devolve uma tabela com os livros mais emprestados no período de “x” meses.

Toda vez que uma movimentação for realizada(**<u>ocorrer atualização na tabela exemplares modificando o local daquele exemplar</u>**) a função validalocaliza() é chamada **antes** da operação de **<u>atualização</u>**.

Arquivo no git : movimenta
Print do Código:

```
CREATE OR REPLACE FUNCTION validalocaliza()
  RETURNS trigger AS $$

DECLARE
teste1 RECORD;

BEGIN
    -- ele já naturalmente só pode estar em um local que exista(propriedade da FK)
    -- agora resta ver se a movimentação é valida!
    IF(NEW.local_id <> OLD.local_id) THEN
        -- a biblioteca nao pode ficar sem nenhum daquele livro
        -- conta quantos livros desse tem naquela biblioteca antes da movimentação
        SELECT COUNT(*) AS total INTO teste1 FROM exemplar
        WHERE ISBN = NEW.ISBN AND estado != 'manu' AND local_id = OLD.local_id;
        IF(cast(teste1.total AS INTEGER) = 1 ) THEN
            RAISE EXCEPTION 'Não é possível movimentar o ultimo exemplar para outra biblioteca ou manutenção!';
        END IF;
        -- só pode levar para manutenção se for um dos livros mais emprestados nos ultimos 6 meses
        IF(NEW.estado = 'manu' ) THEN
            IF(EXISTS(SELECT 1 from exemplar WHERE (exm_id IN (SELECT exem_id FROM maisemprestados(6) ) and nomev= NEW.exm_id ))) THEN
                RETURN NEW;
            ELSE
                RAISE EXCEPTION 'o livro não deve ir para a manutenção ainda!';
            END IF;
        END IF;

   END IF;

RETURN NEW;

END;
$$ LANGUAGE plpgsql;
```

Trigger 3 :

- O usuário será multado toda vez que atrasar a devolução de um empréstimo. Essa multa será calculada com base nos dias de atraso **por outra função(olhar funções: Pg.8)**.

- O usuário será suspenso por um período de tempo caso ele seja multado pelo menos 2 vezes no período de um mês. Seu tempo de suspensão será calculado pelo número de livros não entregues (5 dias) somado a 2 dias para cada dia que ele atrasou cada um desses livros.

- O usuário que for suspenso pela 1ª vez terá uma entrada na tabela de suspensões. Ou acumulará/sobrescreverá mais dias/novo intervalo de suspensão numa entrada já existente.

- A função além de realizar as condições ditas acima que definem penalidades para o usuário, também checa se a tupla que será atualizada em empréstimo ainda tem valores coerentes:
  - Não Permitindo que um mesmo empréstimo tenha o seu usuário ou exemplar trocado .
  - O intervalo de empréstimo ainda deve ser de 7 dias .
  - Usuário só pode ser multado se a data de devolução realmente exceder o tempo limite de empréstimo.
  - O usuário não pode ter pago uma multa que não exista, e só pode pagar uma multa no dia ou após a devolução do exemplar.

Toda vez que uma devolução for realizada(**ocorrer atualização na tabela emprestimo modificando o atributo de multa ou atribuindo uma data de devolução**) a função atualizadev() é chamada **antes** da operação de **atualização.**

Arquivo no git : devolucao
Print do Código:

```
-- entrega de livro(update de dev) ou nao de de livro(update de multa) resulta no calculo ou nao suspensao no
IF((NEW.data_dev <> OLD.data_dev AND NEW.data_dev > NEW.data_exp) OR NEW.multa = TRUE) THEN
    SELECT COUNT(*) AS total INTO teste3 FROM emprestimo
    WHERE ( ( DATE_PART('month',data_exp) = DATE_PART('month',now()) )
          AND ( DATE_PART('year',data_exp) = DATE_PART('year',now()) )
          AND (multa = TRUE) AND (usr_id = NEW.usr_id)  );
    IF(cast(teste3.total AS INTEGER) >= 2 ) THEN
        -- suspenso!
        diastodos = cast(teste3.total AS INTEGER)*5; -- para cada livro atrasado anteriormente
        IF(NEW.data_dev is NULL OR NEW.data_dev = OLD.data_dev) THEN -- livro nao foi devolvido ainda
            diaslivro = cast(DATE_PART('day',now()::timestamp - NEW.data_exp::timestamp) AS integer)*2;
        ELSE
            diaslivro = cast(DATE_PART('day',NEW.data_dev::timestamp - NEW.data_exp::timestamp) AS integer)*2;
        END IF;
        diassusp = diastodos + diaslivro; -- total
        IF(EXISTS(SELECT 1 from suspensoes WHERE usr_id = NEW.usr_id ) ) THEN
            SELECT * INTO teste5 from suspensoes WHERE usr_id = NEW.usr_id;
            IF(cast(DATE_PART('day',now() - teste5.data_s)AS INTEGER) > teste5.dias) THEN -- suspensao que ja
                -- atualiza com uma nova suspensao
                UPDATE suspensoes
                SET suspensoes.dias = diasusp, -- nao acumula com as anteriores(anteriores nao mais validas)
                suspensoes.data_s = NEW.data_exp
                WHERE usr_id = NEW.usr_id;
                RETURN NEW;
            ELSE
                UPDATE suspensoes
                SET suspensoes.dias = suspensoes.dias + diaslivro +(diastodos-5), -- evita contar 2 vezes!
                suspensoes.data_s = NEW.data_exp
                WHERE usr_id = NEW.usr_id;
                RETURN NEW;
            END IF;
        ELSE
            INSERT INTO suspensoes(usr_id,data_s,dias) VALUES (NEW.usr_id,NEW.data_,diassusp);
            RETURN NEW;
        END IF;
```

Trigger 4 :

- Um usuário cadastrado no sistema deverá ter seu RG validado. O processo será feito **por outra função(olhar funções: Pg.8)**.

Toda vez que um usuário for registrado(**ocorrer inserção na tabela usuario**) a função validaUsuario() é chamada **antes** da operação de **inserção**.

```
CREATE OR REPLACE FUNCTION validaUsuario()
  RETURNS trigger AS $$

BEGIN
        if(verificaRg(new.rg)) then
                return new;
        else
                raise exception 'RG invalido, tente novamente';
        end if;
END

CREATE TRIGGER addUsuario BEFORE INSERT ON usuario
FOR EACH ROW EXECUTE PROCEDURE validaUsuario();
```

Trigger 5 e 6 :

- Após pegar um livro emprestado ou devolver um livro o seu estado deve ser atualizado na tabela de exemplares.

Toda vez que um exemplar for emprestado(**ocorrer inserção na tabela emprestimo**) a função estado_up_func() é chamada **depois** da operação de **inserção**.

Toda vez que um exemplar for devolvido(**ocorrer atualização na tabela emprestimo**) a função estado_up_funcdev() é chamada **depois** da operação de **atualização**.

Arquivo no git : atualizaestado
Print do Código:

```
CREATE OR REPLACE FUNCTION estado_up_func()
RETURNS  TRIGGER AS $$

BEGIN
        UPDATE exemplar
        SET exemplar.estado = 'empres'
        WHERE exemplar.exm_id = NEW.exm_id;
END;
$$
LANGUAGE plpgsql;


CREATE TRIGGER atualizaestado
AFTER INSERT ON emprestimo
FOR EACH ROW
EXECUTE PROCEDURE estado_up_func();
```

Arquivo no git: atualizaestadodevo
Print do Código:

```
CREATE OR REPLACE FUNCTION estado_up_funcdev()
RETURNS  TRIGGER AS $$

BEGIN
        IF(NEW.data_dev is not NULL) THEN
                UPDATE exemplar
                SET exemplar.estado = 'disp'
                WHERE exemplar.exm_id = NEW.exm_id;
        END IF;

END;
$$
LANGUAGE plpgsql;


CREATE TRIGGER atualizaestadodevo
AFTER UPDATE ON emprestimo
FOR EACH ROW
EXECUTE PROCEDURE estado_up_funcdev();
```

4 - Funções

Arquivo do Git: funcoes

- CalculaDivida(ID_USER):
  - Calcula quanto um usuário deve decorrente de multas sendo: Sendo 5 reais para cada exemplar + e 0.50 centavos para cada dia a mais de atraso atualmente.

```
CREATE OR REPLACE FUNCTION divida(usr_id INTEGER)
RETURNS DEC(10,2) AS $$

DECLARE
    cur CURSOR(usr_id INTEGER) FOR SELECT data_inic,data_exp,data_dev
    FROM emprestimo WHERE usr_id = usr_id AND multa = TRUE AND data_paga is NULL ;
    c_row RECORD;
    total DEC(10,2);

BEGIN
    OPEN cur(usr_id);
    total :=0;
    IF usr_id IS NOT NULL THEN
        FOR c_row IN cur(usr_id)
        LOOP
            IF (c_row.data_dev is NULL) THEN -- ainda nao devolveu o livro
                total:= total + (5+(cast(DATE_PART('day',now()::timestamp - c_row.data_exp::timestamp) AS DEC(10,2))*0.5));
            ELSE -- devolveu eventualmente
                total:= total + (5+(cast(DATE_PART('day',c_row.data_dev::timestamp - c_row.data_exp::timestamp) AS DEC(10,2))*0.5));
            END IF;
        END LOOP;
    END IF;
    RETURN total;
    RETURN total;
END;
$$
LANGUAGE plpgsql;
```

- MaisEmprestados(intervalo):
  - Devolve uma tabela com os livros mais emprestados em um intervalo de x meses.

```
CREATE OR REPLACE FUNCTION maisemprestados(mesinter integer)
RETURNS TABLE(exm_id INTEGER, ISBN INTEGER, titulo character varying, edicao character varying, total BIGINT) AS $$

BEGIN
    RETURN QUERY EXECUTE'
        SELECT emprestimo.exm_id, exemplar.ISBN, livro.titulo, livro.edicao, COUNT(emp_id)
        FROM
        emprestimo INNER JOIN exemplar ON emprestimo.exm_id = exemplar.exm_id
        INNER JOIN livro ON exemplar.ISBN = livro.ISBN
        WHERE cast(DATE_PART(''month'',now() - emprestimo.data_inic) AS INTEGER) = $1
        GROUP BY
        emprestimo.exm_id,
        exemplar.ISBN,
        livro.titulo,
        livro.edicao
        HAVING COUNT(emp_id) >30'
        USING mesinter;


END;
$$
LANGUAGE plpgsql;
```

- validaRg(rg):
  - Verifica os números inseridos e confirma o dígito verificador. Usado para evitar erros.

```
CREATE OR REPLACE FUNCTION verificaRg(rg text)
RETURNS boolean AS $$

DECLARE
    aux integer;
    soma integer := 0;
    i integer;
    dvEsperado integer;
    teste char[];
BEGIN
    teste = string_to_array(rg,null);
    if(array_length(teste, 1) != 9) then
        raise exception 'tamanho nao compativel';
    end if;
    for i in 1..8 loop
        aux = CAST(teste[i] AS INTEGER);
        if(i%2 = 1) then
            aux = 2*aux;
            if(aux>9) then
                aux = aux - 9;
            end if;
        end if;
        soma = soma + aux;
        i = i -1;
    end loop;

    aux = CAST(teste[1] AS INTEGER);
    dvEsperado = 10 - mod(soma, 10);

    if(aux = dvEsperado) then
        return true;
    else
        return false;
    end if;
END;
```