UNIVERSIDADE FEDERAL FLUMINENSE

Banco de dados Ingresso.com

Gabriel Fiuza e Gabriela Gomes

Novembro – 2017

Niterói, RJ

Banco de Dados Ingresso.com

O ingresso.com é um site de vendas de ingressos para shows, peças de teatro e filmes. O objetivo desse trabalho foi simular o banco de dados com funcionalidades parecidas com as que são encontradas no site original.

Entidades

1. Usuário
   Entidade que representa os usuários que utilizam os serviços do site.

2. Compra
   Entidade associativa que representa uma compra realizada por um usuário.

3. Cancelamento
   Histórico de compras feitas que foram canceladas.

4. Pagamento
   Histórico de pagamento das compras realizadas no site.

5. Tipopagamento
   Lista com os tipos de pagamento que o site disponibiliza.

6. Ingresso
   Representa um ingresso comprado no site.

7. Tipoingresso
   Lista de tipos de ingresso (inteira, meia, meia especial, etc.).

8. Local
   Locais onde ocorrem os eventos.

9. Espaco
   Espaço onde ocorres os eventos em determinados locais, como salas de cinema.

10. Evento
    Evento que possui ingressos vendidos pelo site.

11. Tipoevento
    Lista de tipos de evento (shows, filmes, peças de teatro, etc.).

12. Exibicao
    Exibição única de um evento em determinado espaço.

13. Distribuicao
    Tabela com as distribuições do lucro da venda dos ingressos.

Relacionamentos

1. c_canc
   Relaciona uma compra com o histórico de cancelamentos.

2. p_canc
   Relaciona uma compra com o histórico de pagamentos.

3. R1
   Relaciona um pagamento ao seu tipo.

4. R2
   Relaciona um ingresso ao seu tipo.

5. Faz_parte
   Relaciona um ingresso a uma única exibição.

6. Realizar
   Relacionamento ternário entre um evento, espaço e exibição. Cada instância representa a exibição de um evento em determinado espaço.

7. Pertence
   Relaciona um espaço a um local (ex.: uma sala de cinema a um shopping).

8. DistribuicaoLucro
   Relaciona a exibição de um evento à determinada divisão de lucro.

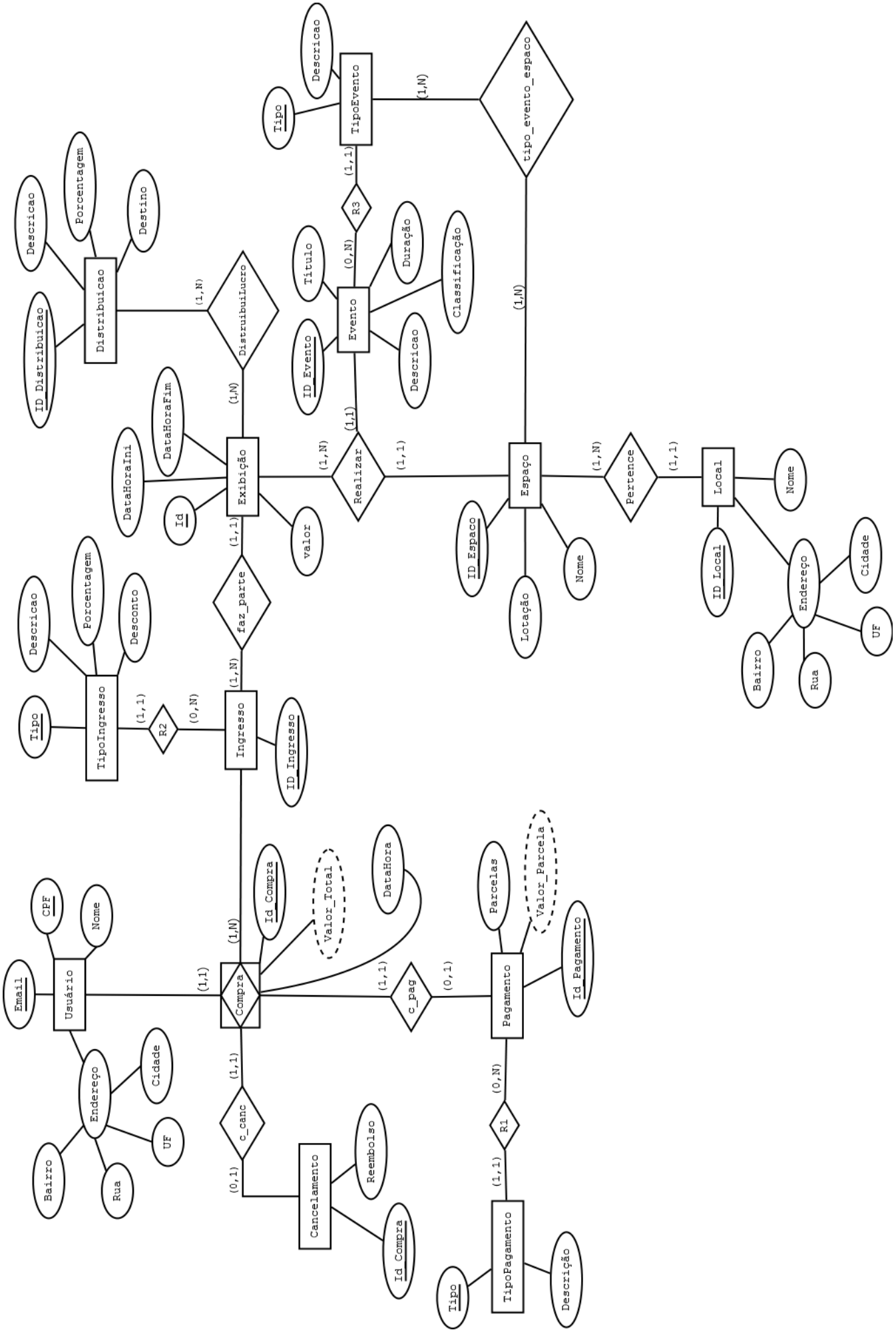
9. R3
   Relaciona um evento ao seu tipo.

10. Tipo_evento_local
    Relaciona um tipo de evento a um local, ou seja, que tipos de eventos um local pode exibir.

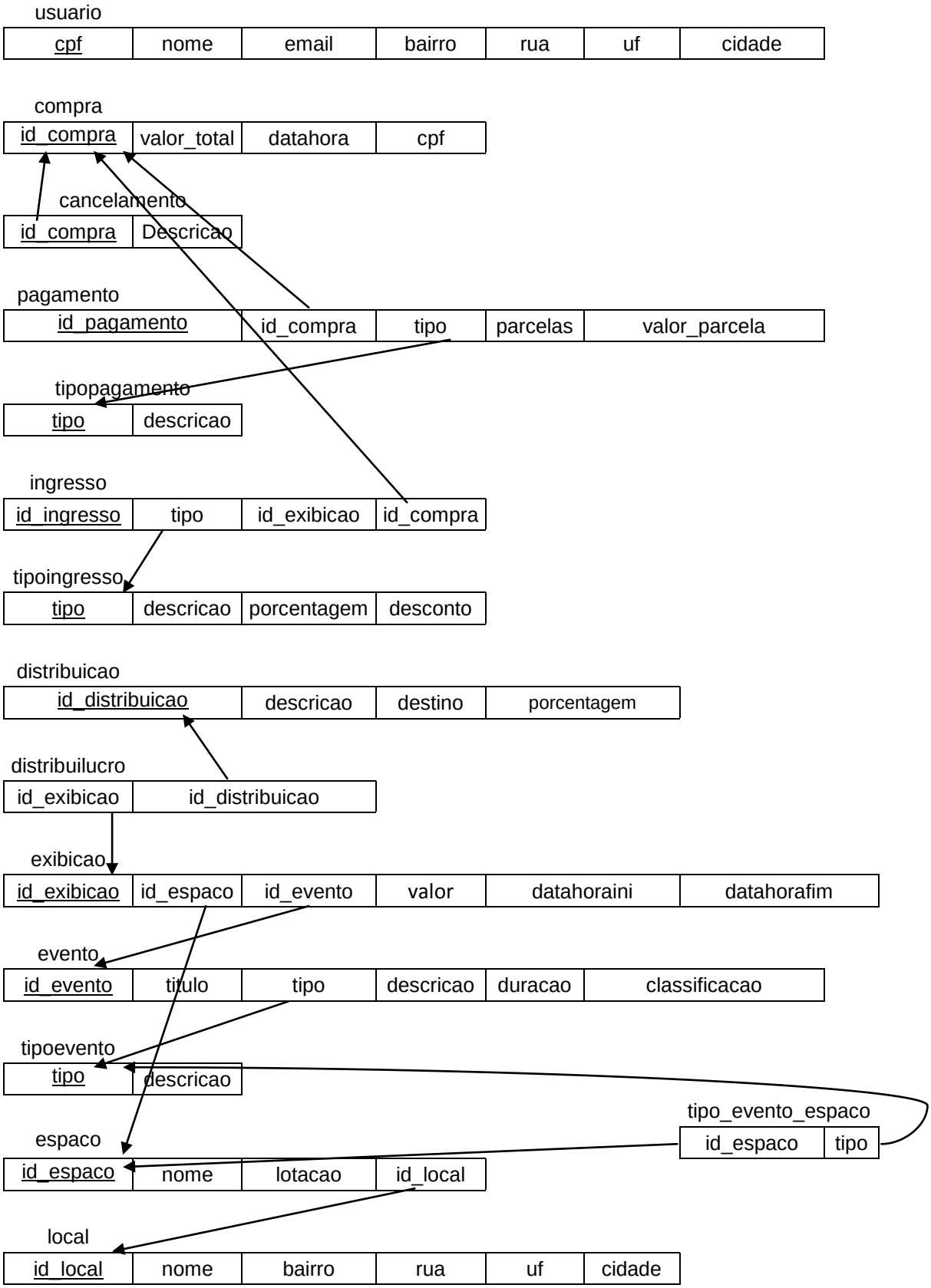
Restrições de Negócio:

- Distribuição de valores não pode ser acima de 100% para uma determinada sessão
- Impedir que se venda ingressos para uma sessão que esteja lotada
- Impedir que se insira uma exibição se sobreponha a outra
- Impedir que ingressos que possuem limite de venda ultrapassem esse limite nas vendas
- Impedir que exibições de um certo tipo sejam alocadas a espaços de outro tipo
- Não permitir que usuários peçam reembolso em compras que ocorreram há 7 dias, ou caso o evento ocorra antes de 7 dias, impedir que haja reembolso antes da véspera do evento.

Modelo Entidade-Relacionamento



Mapeamento Lógico-Relacional



Tabelas

```
CREATE TABLE usuario(
        cpf integer,
        nome text NOT NULL,
        email text UNIQUE NOT NULL,
        bairro text NOT NULL,
        rua text NOT NULL,
        uf varchar(2) NOT NULL,
        cidade text NOT NULL,
        CONSTRAINT pk_usuario PRIMARY KEY(cpf));

CREATE TABLE compra(
        id_compra integer,
        valor_total money NOT NULL,
        datahora timestamp NOT NULL,
        cpf integer NOT NULL,
        CONSTRAINT pk_compra PRIMARY KEY(id_compra),
        CONSTRAINT fk_compra FOREIGN KEY(cpf) REFERENCES usuario(cpf));

CREATE TABLE cancelamento(
        id_compra integer NOT NULL,
        reembolso money,
        CONSTRAINT pk_cancelamento PRIMARY KEY(id_compra),
        CONSTRAINT fk_cancelamento FOREIGN KEY (id_compra) REFERENCES
compra(id_compra));

CREATE TABLE tipopagamento(
        tipo integer,
        descricao text NOT NULL,
        CONSTRAINT pk_tipopagamento PRIMARY KEY(tipo));

CREATE TABLE pagamento(
        id_pagamento integer,
        id_compra integer NOT NULL,
        tipo integer NOT NULL,
        parcelas integer NOT NULL,
        valor_parcela money NOT NULL,
        CONSTRAINT pk_pagamento PRIMARY KEY(id_pagamento),
        CONSTRAINT fk_pagamento_compra FOREIGN KEY (id_compra) REFERENCES
compra(id_compra),
        CONSTRAINT fk_pagamento_tipo FOREIGN KEY (tipo) REFERENCES
tipopagamento(tipo));

CREATE TABLE tipoevento (
        tipo integer,
        descricao text NOT NULL,
        CONSTRAINT pk_tipoevento PRIMARY KEY(tipo));
```

```
CREATE TABLE evento (
	id_evento integer,
	titulo text,
	tipo integer,
	descricao text,
	duracao time,
	classificacao varchar(3),
	CONSTRAINT pk_evento PRIMARY KEY(id_evento),
	CONSTRAINT fk_evento FOREIGN KEY (tipo) REFERENCES tipoevento(tipo));

CREATE TABLE local (
	id_local integer,
	nome text NOT NULL,
	bairro text NOT NULL,
	rua text NOT NULL,
	uf varchar(2) NOT NULL,
	cidade text NOT NULL,
	CONSTRAINT pk_local PRIMARY KEY(id_local));

CREATE TABLE espaco (
	id_espaco integer,
	nome text NOT NULL,
	lotacao integer NOT NULL,
	id_local integer NOT NULL,
	CONSTRAINT pk_espaco PRIMARY KEY(id_espaco),
	CONSTRAINT fk_espaco FOREIGN KEY (id_local) REFERENCES local(id_local));

CREATE TABLE exibicao (
	id_exibicao integer,
	id_espaco integer NOT NULL,
	id_evento integer NOT NULL,
	valor money NOT NULL,
	datahoraini timestamp NOT NULL,
	datahorafim timestamp NOT NULL,
	CONSTRAINT pk_exibicao PRIMARY KEY(id_exibicao),
	CONSTRAINT fk_exibicao_espaco FOREIGN KEY (id_espaco) REFERENCES
espaco(id_espaco),
	CONSTRAINT fk_exibicao_evento FOREIGN KEY (id_evento) REFERENCES
evento(id_evento));

CREATE TABLE tipoingresso (
	tipo integer,
	descricao text NOT NULL,
	porcentagem real NOT NULL,
	desconto real,
	CONSTRAINT pk_tipoingresso PRIMARY KEY(tipo));
```

```
CREATE TABLE ingresso(
	id_ingresso integer,
	tipo integer NOT NULL,
	id_exibicao integer NOT NULL,
	id_compra integer NOT NULL,
	CONSTRAINT pk_ingresso PRIMARY KEY(id_ingresso),
	CONSTRAINT fk_ingresso_tipo FOREIGN KEY (tipo) REFERENCES tipoingresso(tipo),
	CONSTRAINT fk_ingresso_exibicao FOREIGN KEY (id_exibicao) REFERENCES
exibicao(id_exibicao),
	CONSTRAINT fk_ingresso_compra FOREIGN KEY (id_compra) REFERENCES
compra(id_compra));

CREATE TABLE distribuicao (
	id_distribuicao integer,
	descricao text NOT NULL,
	destino text NOT NULL,
	porcentagem real NOT NULL,
	CONSTRAINT pk_distribuicao PRIMARY KEY(id_distribuicao));

CREATE TABLE distribuilucro (
	id_exibicao integer,
	id_distribuicao integer,
	CONSTRAINT pk_distribuilucro PRIMARY KEY(id_exibicao, id_distribuicao),
	CONSTRAINT fk_distribuilucro_exibicao FOREIGN KEY (id_exibicao) REFERENCES
exibicao(id_exibicao),
	CONSTRAINT fk_distribuilucro_distribuicao FOREIGN key (id_distribuicao) REFERENCES
distribuicao(id_distribuicao));

CREATE TABLE tipo_evento_espaco (
	id_espaco integer,
	tipo integer,
	CONSTRAINT pk_tipo_evento_espaco PRIMARY KEY (id_espaco, tipo),
	CONSTRAINT fk_tipo_evento_espaco_espaco FOREIGN KEY (id_espaco) REFERENCES
espaco(id_espaco),
	CONSTRAINT fk_tipo_evento_espaco_tipo FOREIGN KEY (tipo) REFERENCES
tipoevento(tipo));
```

Funções e Triggers

- Distribuição de valores não pode ser acima de 100% para uma determinada sessão:

```
create or replace function distribuicao_100()
returns trigger
language 'plpgsql'
as $$
declare
	pctg real;
	valor real;
	c1 cursor for
```

```
        select id_distribuicao
        from distribuilucro dl
        where dl.id_exibicao = new.id_exibicao;
        c2 cursor(iid integer) for
        select porcentagem
        from distribuicao
        where iid = id_distribuicao;
begin
        pctg := 0;
        for r1 in c1 loop
                OPEN c2(r1.id_distribuicao);
                FETCH c2 into valor;
                pctg := pctg + valor;
                close c2;
        end loop;
        if (pctg > 1) then
                raise exception 'porcentagem total acima de 100';
        end if;
        return null;
end;
$$;
create trigger distribuicao_acima_de_100 after insert on distribuilucro for each row execute
procedure distribuicao_100();

create or replace function ingressos_validos() returns table(
        id_ingresso integer)
language 'plpgsql' as
$$
declare
begin
        return query
        with    t1 as(  select aa.id_ingresso
                        from ingresso aa)
        select t1.id_ingresso
        from t1
        where t1.id_ingresso NOT IN (   select i1.id_ingresso
                        from cancelamento c1 natural join ingresso i1);
end;
$$;

create or replace function lucro_bruto_ingressos_por_filme(nome_filme text)
returns money
language 'plpgsql' as
$$
declare
        resultado money;
begin
select  sum(lucro) into resultado
        from (  select id_ingresso, valor*desconto as lucro
```

```
                from (  select iv1.id_ingresso, t3.valor, 1 - iv1.desconto as desconto
                        from (select * from ingressos_validos() natural join ingresso natural
join tipoingresso) iv1
                        natural join (  select id_exibicao, valor
                                        from evento tex natural join exibicao
                                        where tex.titulo = nome_filme) as t3) as t2 ) as t1;
return resultado;
end;
$$;

create or replace function qtd_ingresso_vendido_local(nome_local text)
returns integer
language 'plpgsql' as
$$
declare
        resultado integer;
begin
select count(id_ingresso) into resultado
        from (  select id_exibicao
                from local l1 inner join espaco e1 on l1.id_local = e1.id_local
                inner join exibicao e2 on e2.id_espaco = e1.id_espaco
                where l1.nome = nome_local) t1
                inner join
        (       select id_ingresso, id_exibicao
                from (ingressos_validos() natural join ingresso)) t2
                on t1.id_exibicao = t2.id_exibicao;
return resultado;
end;
$$;
```

- Impedir que se venda ingressos para uma sessão que esteja lotada:

```
create or replace function checa_lotacao()
returns trigger
language 'plpgsql' as
$$
declare
        lotacao_sala integer;
        qtd_ingressos integer;
begin
        select e2.lotacao into lotacao_sala
        from exibicao e1 natural join espaco e2
        where new.id_exibicao = e1.id_exibicao;
        select count(id_ingresso) into qtd_ingressos
        from ingressos_validos() natural join ingresso it
        where new.id_exibicao = it.id_exibicao;
        if lotacao_sala < qtd_ingressos then
                RAISE EXCEPTION 'sala lotada';
        end if;
        return null;
```

```
end;
$$;
CREATE TRIGGER lotacao AFTER INSERT ON ingresso for each row EXECUTE PROCEDURE
checa_lotacao();
```

- Impedir que se insira uma exibição se sobreponha a outra:

```
create or replace function intervalo_entre_exibicoes()
returns trigger
language 'plpgsql'
as $$
declare
        c1 cursor for
        select id_exibicao, datahoraini, datahorafim
        from exibicao e1
        where e1.id_espaco = new.id_espaco;
begin
        for r1 in c1 loop
                if r1.id_exibicao = new.id_exibicao then
                                continue;
                end if;
                if ((r1.datahoraini, r1.datahorafim) OVERLAPS (new.datahoraini,
new.datahorafim)) then
                        raise exception 'Horario indisponivel';
                end if;
        end loop;
        return null;
end;
$$;

CREATE TRIGGER intervalo_exibicao AFTER INSERT ON exibicao for each row EXECUTE
PROCEDURE intervalo_entre_exibicoes();
```

- Impedir que ingressos que possuem limite de venda ultrapassem esse limite nas vendas:

```
create or replace function e_aquele_1porcento()
returns trigger
language 'plpgsql' as
$$
declare
        porc_max float;
        total_ingressos float;
        total_ingressos_tipo float;
begin
        select ting.porcentagem into porc_max
        from ingresso ing natural join tipoingresso ting
        where ing.id_ingresso = new.id_ingresso;
        select t2.lotacao into total_ingressos
```

```
        from ((ingresso natural join exibicao) t1 natural join espaco) t2
        where id_exibicao = new.id_exibicao;
        select count(t1.id_ingresso) into total_ingressos_tipo
        from (ingresso natural join ingressos_validos()) t1
        where id_exibicao = new.id_exibicao and tipo = new.tipo;
        if porc_max < (total_ingressos_tipo / total_ingressos) then
                raise exception 'limite do tipo de ingresso alcançado';
        end if;
        return new;
end;
$$;
CREATE TRIGGER limite_tipo_ingresso AFTER INSERT ON ingresso for each row EXECUTE
PROCEDURE e_aquele_1porcento();
```

- Impedir que exibições de um certo tipo sejam alocadas a espaços de outro tipo:

```
create or replace function espaco_tipo_exibicao()
returns trigger
language 'plpgsql' as
$$
declare
        tipo_evento integer;
        id_do_espaco integer;
        c1 CURSOR (idespaco integer) FOR select tipo from tipo_evento_espaco where
id_espaco = idespaco;
begin
        select e1.tipo into tipo_evento
        from exibicao e0 natural join evento e1
        where e0.id_exibicao = new.id_exibicao;
        id_do_espaco := new.id_espaco;
        FOR linha_cursor IN c1(id_do_espaco) LOOP
                if linha_cursor.tipo = tipo_evento then
                        return null;
                end if;
        end loop;
        raise exception 'Espaco nao pode abrigar este tipo de evento';
end;
$$;
CREATE TRIGGER espaco_tipo_exibicao AFTER INSERT ON exibicao FOR EACH ROW EXECUTE
PROCEDURE espaco_tipo_exibicao();
```

- Não permitir que usuários peçam reembolso em compras que ocorreram há 7 dias, ou caso o evento ocorra antes de 7 dias, impedir que haja reembolso antes da véspera do evento:

```
create or replace function compra_reembolso()
returns trigger
language 'plpgsql'
as $$
declare
        inicio_evento timestamp;
```

```
	data_compra timestamp;
	intervalao_da_compra interval;
	dias_desde_compra integer;
	intervalao_do_evento interval;
	dias_ate_evento integer;
	intervalo_compra_evento interval;
	dias_entre_compra_evento integer;
begin
	select datahoraini  into inicio_evento
	from ingresso i1 natural join exibicao e1
	where i1.id_compra = new.id_compra;
	select datahora into data_compra
	from compra c1
	where c1.id_compra = new.id_compra;

	intervalao_da_compra := localtimestamp - data_compra;
	dias_desde_compra := extract(day from intervalao_da_compra);

	intervalao_do_evento := inicio_evento - localtimestamp;
	dias_ate_evento := extract(day from intervalao_do_evento);

	intervalo_compra_evento := inicio_evento - data_compra;
	dias_entre_compra_evento := extract(day from intervalo_compra_evento);

	if ( dias_ate_evento > 0 and (dias_desde_compra <= 7 or dias_entre_compra_evento
<= 7)) then
		return null;
	end if;
	raise exception 'Compra nao pode ser cancelada.';
end;
$$;
CREATE TRIGGER validade_cancelamento AFTER INSERT ON cancelamento FOR EACH ROW
EXECUTE PROCEDURE compra_reembolso();

create or replace function valor_total_da_compra()
returns trigger
language 'plpgsql'
as $$
declare
	idcompra integer;
	valorinteira money;
	des real;
	valortotal money;
begin
	select valor into valorinteira
	from ingresso natural join exibicao
	where ingresso.id_ingresso = new.id_ingresso;

	select desconto into des
```

```
        from tipoingresso t
        where t.tipo = new.tipo;

        select valor_total into valortotal
        from ingresso natural join compra
        where ingresso.id_compra = new.id_compra;

        valortotal := valortotal + valorinteira*(1-des);

        update compra set valor_total = valortotal where id_compra = new.id_compra;

        return null;
end;
$$;
CREATE TRIGGER valor_total_compra AFTER INSERT ON ingresso FOR EACH ROW EXECUTE
PROCEDURE valor_total_da_compra();

create or replace function valor_parcela_compra()
returns trigger
language 'plpgsql'
as $$
declare
        parcela money;
        valortotal money;
begin
        select valor_total into valortotal
        from compra
        where compra.id_compra = new.id_compra;
        parcela := valortotal/new.parcelas;
        update pagamento set valor_parcela = parcela where id_compra = new.id_compra;
        return null;
end;
$$;
CREATE TRIGGER valor_parcela AFTER INSERT ON pagamento FOR EACH ROW EXECUTE
PROCEDURE valor_parcela_compra();
```