Universidade Federal da Bahia
Laboratório de Bancos de Dados - MATB09

SISTEMA DE GERENCIAMENTO DE ESTÚDIOS

Regras de negócio
Stored procedures
Triggers
Views

Alunos: Adewale Andrade
Rodrigo Nunes Souto

Regras de Negócio

1. **Valor total do agendamento**
   O valor total do agendamento deve ser um somatório do produto do preço do serviço e sua duração com o produto dos preços internos dos equipamentos alugados e suas respectivas durações.
2. **Valor total do aluguel de equipamentos**
   O valor total do aluguel de equipamentos deve ser um somatório do produto dos preços externos dos equipamentos alugados e suas respectivas durações.
3. **Atualização do valor total do agendamento**
   O valor total do agendamento deve ser atualizado assim que qualquer novo equipamento é agregado ou removido do agendamento.

4. **Status dos agendamentos**
   O status dos agendamentos devem ser analisados e corrigidos diariamente.
5. **Informações sobre banda e agendamentos**
   O sistema deve disponibilizar de forma eficiente dados sobre a banda e seus integrantes, sobre agendamentos relativos a dias específicos e agendamentos de cada banda.

Stored Procedures

1. **Update Agenda Status**

   Postgres
   ```
   create or replace function update_agenda_status()
   returns void as $agenda_status$
   declare
   begin
           update agenda set status = 'done'
           where date - current_date < 3 and status != 'canceled';
   end;
   $agenda_status$
   language plpgsql;
   ```

2. **Members Information**

   Postgres

   ```
   create or replace function members_information(varchar)
   returns setof band_information as $members_information$
           select * from band_information where login=$1;
   $members_information$
   language sql;
   ```

3. **Agenda by Day**

   Postgres

   ```
   create or replace function agenda_by_day(date)
   returns setof agenda_information as $agenda_by_day$
           select * from agenda_information where date=$1;
   ```

```
$agenda_by_day$
language sql;
```

4. **Band Schedule**

   Postgres

```
create or replace function band_schedule(varchar)
returns setof agenda_information as $band_schedule$
        select * from agenda_information where login=$1;
$band_schedule$
language sql;
```

Triggers

1. **Service Total Price**

   Postgres

```
create or replace function service_price()
returns trigger as $service_price$
declare
        service_price float;
        equipments_price float;
begin
        select NEW.duration*service.price into service_price
        from service where service_id = NEW.service_id;

        select sum(ahe.duration*e.internalprice) into equipments_price
        from agenda_has_equipment as ahe inner join equipment as e
        on ahe.equipment_id = e.equipment_id where ahe.agenda_id = NEW.agenda_id;

        if equipments_price is null then
                equipments_price = 0;
        end if;

        NEW.total_price := service_price+equipments_price;
        return NEW;
end;
$service_price$
language plpgsql;

drop trigger if exists calculate_service_price on agenda;
create trigger calculate_service_price before update or insert on agenda
for each row execute procedure service_price();
```

   Mysql

```
DELIMITER $
CREATE TRIGGER total_service_price BEFORE UPDATE ON Agenda
   FOR EACH ROW
   BEGIN
   SET @service_price = 0;
   SET @equipments_price = 0;

   INSERT INTO @service_price
   SELECT  Agenda.duration*Service.Price
   FROM Agenda INNER JOIN Service
   ON Agenda.Service_ID = Service.Service_ID
   WHERE Agenda.Agenda_ID = NEW.Agenda_ID;

   INSERT INTO @equipments_price
      SELECT SUM(ahe.Duration*e.InternalPrice)
   FROM Agenda_has_Equipment AS ahe INNER JOIN Equipment AS e
   ON ahe.Equipment_ID = e.Equipment_ID WHERE ahe.Agenda_ID =
NEW.Agenda_ID;

         IF @equipments_price IS NULL THEN
                  equipments_price = 0;
         END IF;

   SET NEW.Total_Price = @service_price + @equipments_price;

    END;$
DELIMITER ;

DELIMITER $
CREATE TRIGGER total_service_price BEFORE INSERT ON Agenda
    FOR EACH ROW
    BEGIN

   SET @service_price = 0;
   SET @equipments_price = 0;

   INSERT INTO @service_price
   SELECT  Agenda.duration*Service.Price
   FROM Agenda INNER JOIN Service
   ON Agenda.Service_ID = Service.Service_ID
   WHERE Agenda.Agenda_ID = NEW.Agenda_ID;

   INSERT INTO @equipments_price
      SELECT SUM(ahe.Duration*e.InternalPrice)
   FROM Agenda_has_Equipment AS ahe INNER JOIN Equipment AS e
   ON ahe.Equipment_ID = e.Equipment_ID WHERE ahe.Agenda_ID =
NEW.Agenda_ID;

         IF @equipments_price IS NULL THEN
                  equipments_price = 0;
         END IF;

   SET NEW.Total_Price = @service_price + @equipments_price;

    END;$
```

```
DELIMITER ;
```

2. **Service Total Price Equipment Update**

   Postgres

```
create or replace function update_agenda()
returns trigger as $update_agenda$
declare
d integer;
begin
        if (TG_OP = 'INSERT' or TG_OP = 'UPDATE') then
                select duration into d from agenda where agenda_id=NEW.agenda_id;
                update agenda set duration=d where agenda_id=NEW.agenda_id;
                return NEW;
        elsif (TG_OP = 'DELETE') then
                select duration into d from agenda where agenda_id=OLD.agenda_id;
                update agenda set duration=d where agenda_id=OLD.agenda_id;
                return OLD;
        end if;
        return null;
end;
$update_agenda$
language plpgsql;

drop trigger if exists call_agenda_trigger on agenda_has_equipment;
create trigger call_agenda_trigger after update or insert or delete on
agenda_has_equipment
for each row execute procedure update_agenda();
```

3. **Rent Total Price**

   Postgres

```
create or replace function rent_price()
returns trigger as $rent_price$
declare
        equipments_price float;
begin
        equipments_price = 0;
        select sum(cre.duration*e.externalprice) into equipments_price
        from client_rents_equipment as cre inner join equipment as e
        on cre.equipment_id = e.equipment_id where cre.client_login =
NEW.client_login;

        if equipments_price is null then
                equipments_price = 0;
        end if;

        NEW.total_price := equipments_price;
        return NEW;
end;
```

```
$rent_price$
language plpgsql;

drop trigger if exists calculate_rent_price on client_rents_equipment;
create trigger calculate_rent_price before update or insert on client_rents_equipment
for each row execute procedure rent_price();
```

Mysql

```
DELIMITER $
CREATE TRIGGER total_rent_price BEFORE UPDATE ON Client_rents_Equipment
    FOR EACH ROW
    BEGIN

   SET @equipments_price = 0;

   INSERT INTO @equipments_price
      SELECT SUM(cre.Duration*e.ExternalPrice)
   FROM Client_rents_Equipment AS cre INNER JOIN Equipment AS e
   ON cre.Equipment_ID = e.Equipment_ID WHERE cre.Client_Login =
NEW.Client_Login;

         IF @equipments_price IS NULL THEN
                  equipments_price = 0;
         END IF;

   SET NEW.Total_Price = @equipments_price;

    END;$
DELIMITER ;

DELIMITER $
CREATE TRIGGER total_rent_price BEFORE INSERT ON Client_rents_Equipment
    FOR EACH ROW
    BEGIN

   SET @equipments_price = 0;

   INSERT INTO @equipments_price
      SELECT SUM(cre.Duration*e.ExternalPrice)
   FROM Client_rents_Equipment AS cre INNER JOIN Equipment AS e
   ON cre.Equipment_ID = e.Equipment_ID WHERE cre.Client_Login =
NEW.Client_Login;

         IF @equipments_price IS NULL THEN
                  equipments_price = 0;
         END IF;

   SET NEW.Total_Price = @equipments_price;

    END;$
DELIMITER ;
```

Views

1. Band Information

```
CREATE OR REPLACE VIEW Band_Information AS
SELECT Band.Name, Band.Login, Band.Style, Band.HomePage, Person.Name as
Member_Name, Person.Phone1, Person.Email FROM Band
INNER JOIN Band_Has_Member
ON Band_Has_Member.Band_Login = Band.Login
INNER JOIN Member
ON Member.Member_ID = Band_Has_Member.Member_ID
INNER JOIN Person
ON Person.Cpf = Member.Person_Cpf
Order by Band.Name asc;
```

2. Agenda Information

```
CREATE OR REPLACE VIEW Agenda_Information AS
SELECT b.Name, b.Login, a.Date, a.Time, a.Duration, a.Room, a.Status
FROM Band b
INNER JOIN Agenda a
ON a.Band_Login = b.Login
Order by a.Date asc;
```