FESTIVAL DE MÚSICA

Projeto Base de Dados - Parte 2

Turma 6 - Grupo 2
Gaspar Santos Pinheiro - up201704700@fe.up.pt
Sofia de Araújo Lajes - up201704066@fe.up.pt
Tito Alexandre Trindade Griné - up201706732@fe.up.pt

Índice

Introdução 2

Contextualização 3

Diagrama UML 5

Diagrama UML Revisto 6

Esquema Relacional 7

Análise de Dependências Formais e Formas Normais 8

Lista de Restrições e Detalhes de Implementação 13

Interrogações 28

Gatilhos 30

Introdução

O tema do projeto baseia-se na gestão de um **Festival de Música**. Como festival de música, entendemos eventos como o NOS Alive ou o Meo Sudoeste, daí que a nossa base de dados tenha em conta dinâmicas semelhantes a estes festivais, isto é, curta duração (3-5 dias), palcos ao ar livre, com acampamentos. Esta distinção é importante, a título de exemplo, apesar de haver concertos, os bilhetes dão acesso a dias do festival e não a atuações em particular, bem como não têm um lugar associado, visto ser ao ar livre. Noutro contexto, estas considerações poderiam ser bastante relevantes.

Na primeira parte do projeto, o objetivo foi definir o modelo conceptual da base de dados que iriamos implementar, criando um diagrama UML que representasse adequadamente este modelo. Tivemos sempre por base a perspetiva da empresa organizadora do evento, pensando em quais seriam as suas necessidades e incorporá-las na nossa base de dados.

Na segunda parte, após as sugeridas alterações ao modelo conceptual da primeira entrega, foi desenvolvido o esquema relacional, analisadas as dependências funcionais e formas normais (bem como o registo das suas violações). Foi criada a base de dados correspondente em SQLite 3 e, por último, a sua povoação.

Na terceira parte do trabalho, foram realizadas dez interrogações à nossa base de dados bem como três gatilhos que fossem capazes de manter algumas das restrições impostas anteriormente. As interrogações foram escolhidas segundo um critério de pertinência, isto é, de forma a que informação obtida por cada uma tivesse uma relevância do ponto de vista dos gestores de um festival de música, mas também tendo em conta a necessidade de variar a lógica de implementação e aumentar a complexidade das interrogações.

Contextualização

As preocupações de uma Empresa, cujo objetivo é a preparação de um Festival de Música, não devem incidir apenas nos "nomes de cartaz", mas também na estrutura da sua organização, incluindo a logística dos empregados e patrocinadores.

Começando pela superfície: devem ser convidados **Artistas** (o número de artistas convidados é algo ao critério da empresa) dado serem a "cara" do festival, tendo em atenção o seu **Genero** (por exemplo, "Rock", "Blues", "EDM"). O contacto com estes artistas deve ser mantido através do **Agente**, que todos os artistas têm de ter de modo a facilitar o tratamento da sua agenda, garantindo também a privacidade dos artistas, no que toca a contactos pessoais.

Num **Concerto** pode atuar um ou mais artistas, sendo este definido apenas pela sua hora de início, data e palco onde se realiza - uma vez que pode haver dois concertos a decorrer em simultâneo desde que em palcos diferentes (sendo cada um destes definido na classe **Palco**).

A audiência do festival pode ser decomposta em **Convidados** (normalmente são figuras públicas que poderão ajudar a divulgar o evento, representantes das diferentes entidades ou até mesmo jornalistas, daí estarem associados ao seu respetivo tipo) e em compradores regulares (**Normal**) que adquiriram bilhete. Decidiu-se manter um registo de se o participante é menor de idade (abaixo de 16 anos) ou não, dado que estes devem ser acompanhados por um adulto.

No seguimento, foi necessário definir diversos tipos de bilhetes que se assemelhasse à realidade (ainda que no modelo conceptual seja apresentada uma generalização **TipoBilhete**). Podem ser criados diversos tipos de bilhete como, por exemplo, "diário" (para apenas um dia), "pass" (passe para todos os dias do festival), "vip" (pode estar envolvido contacto com os artistas ou acessos a áreas restritas), tendo também em consideração a intenção de usar ou não o acampamento. Cada bilhete (código) é único e deve ser mantido um registo da ativação do bilhete com o objetivo de fazer uma contagem dos participantes e evitar violações na utilização do bilhete (aspetos definidos na classe relação **Tem**).

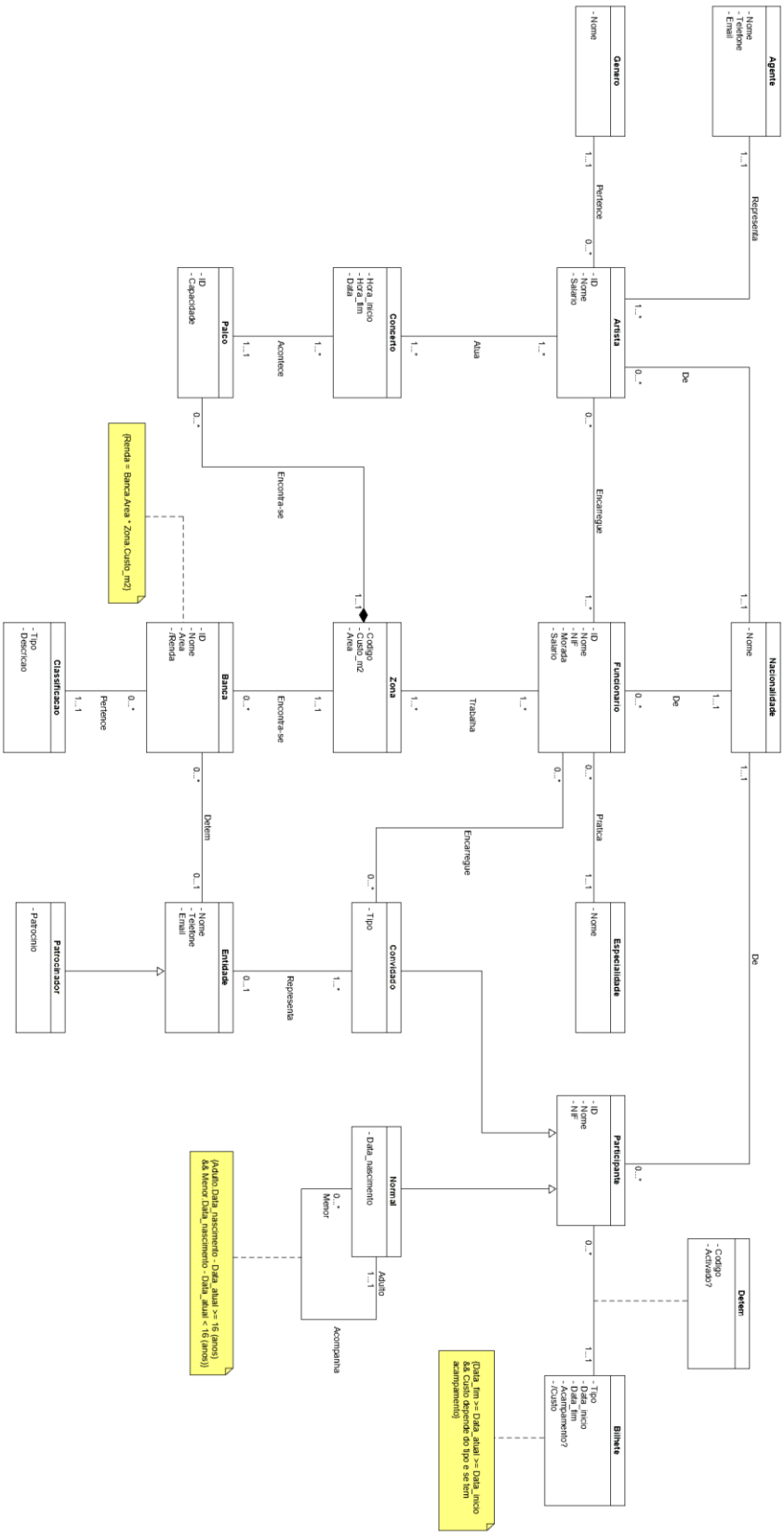
Naturalmente, um evento com grande dimensão dependerá muito de patrocínios, sendo então necessário guardar informação sobre cada **Patrocinador** com o qual se chegou a acordo favorável.

O festival é dividido em zonas, tal que uma **Zona** pode conter palcos (destinados a concertos), bancas relacionadas dos patrocinadores para fins comerciais ou até os dois se o espaço assim o possibilitar. Uma **Banca** está associada a uma **Entidade**, regularmente, patrocinadora do evento (podendo haver exceções nas quais uma entidade não patrocinadora detém uma banca). Deverão haver diversos tipos de bancas (por exemplo, "Restauração", "Informação", "Lazer") tendo, portanto, de estar associadas à sua **Classificacao**.

Note-se que os convidados podem ser representantes de uma determinada entidade, e ainda que a cada banca está associada uma renda que é o resultado do custo por m2 da zona onde se encontra pela área ocupada pela banca.

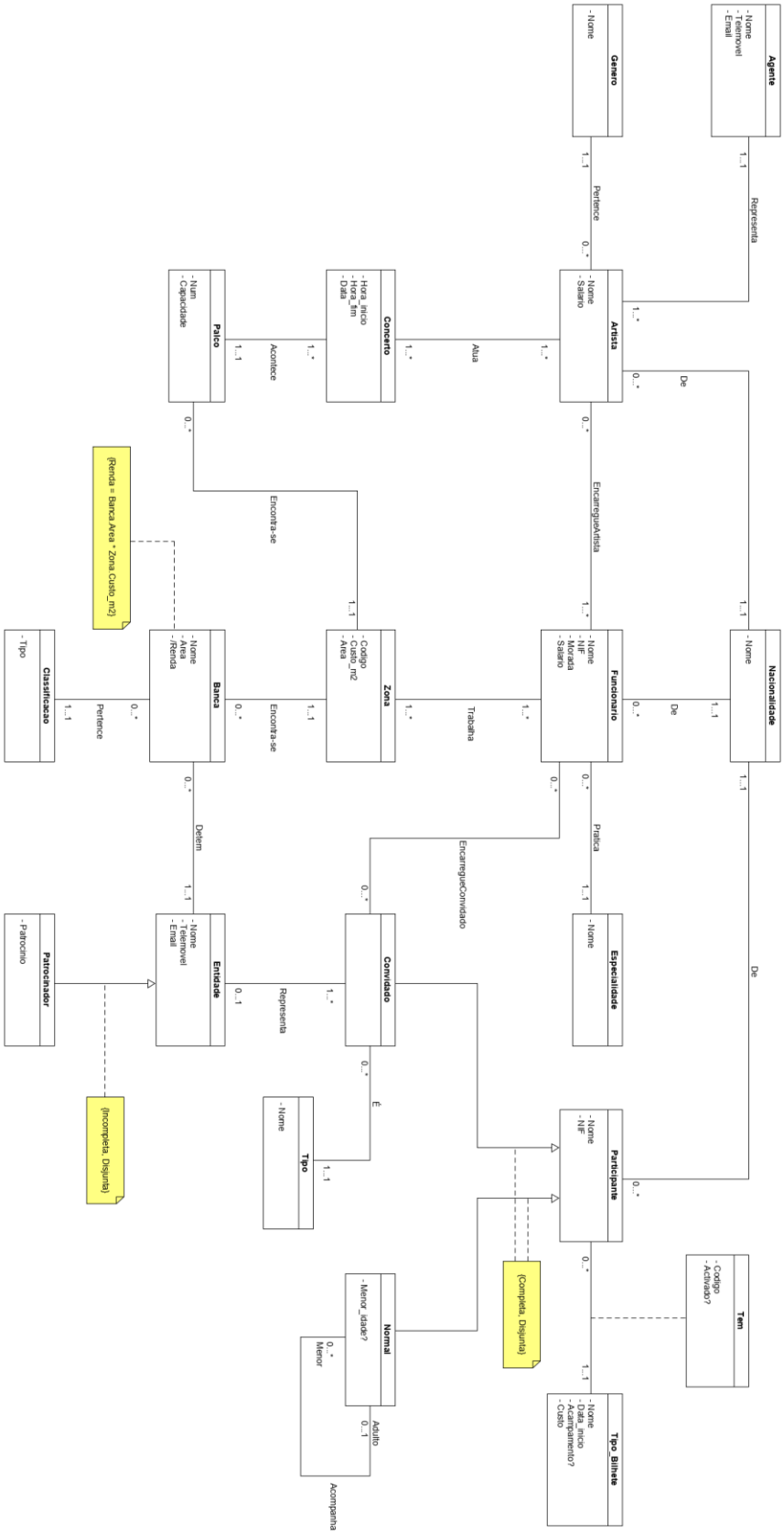
Por último, é necessário distribuir a mão de obra. Um **Funcionario** tem uma **Especialidade**, que poderá ser, por exemplo, "Organizador", "Eletricista", "Segurança", entre outros. Os funcionários são designados para uma Zona, e, caso a sua especialidade seja adequada, também a um Artista ou Convidado, podendo exercer diversas funções em prol da orientação destes durante o evento.

Diagrama UML*



* O diagrama UML encontra-se também anexado para facilitar a análise

Diagrama UML Revisto*



* O diagrama UML revisto encontra-se também anexado para facilitar a análise

Esquema Relacional

- **Acompanha**(menor → Normal, adulto → Normal);
- **Agente**(id, nome, telemovel, email);
- **Artista**(id, nome, salario, agente → Agente, genero → Genero, nacionalidade → Nacionalidade);
- **Atua**(artista → Artista, hora_inicio → Concerto.hora_inicio, data → Concerto.data, palco_num → Concerto.palco_num, palco_zona → Concerto.palco_zona);
- **Banca**(id, nome, area, zona → Zona, renda, classificacao → Classificacao, entidade → Entidade);
- **Classificacao**(id, tipo);
- **Concerto**(hora_inicio, hora_fim, data, palco_num → Palco.num, palco_zona → Palco.zona);
- **Convidado**(participante → Participante, tipo → Tipo, entidade → Entidade);
- **EncarregueArtista**(artista → Artista, funcionario → Funcionario);
- **EncarregueConvidado**(convidado → Convidado, funcionario → Funcionario);
- **Entidade**(id, nome, telemovel, email);
- **Especialidade**(id, nome);
- **Funcionario**(id, nome, nif, morada, salario, especialidade → Especialidade, nacionalidade → Nacionalidade);
- **Genero**(id, nome);
- **Nacionalidade**(id, nome);
- **Normal**(participante → Participante, menor_idade);
- **Palco**(num, capacidade, zona → Zona);
- **Participante**(id, nome, nif, nacionalidade → Nacionalidade);
- **Patrocinador**(entidade → Entidade, patrocinio);
- **Tem**(codigo, ativado, participante → Participante, bilhete → TipoBilhete);
- **Tipo**(id, nome);
- **TipoBilhete**(id, nome, data_inicio, acampamento, custo);
- **Trabalha**(funcionario → Funcionario, zona → Zona);
- **Zona**(codigo, custo_m2, area);

Análise de Dependências Formais e Formas Normais

Acompanha:

- FD's: menor ⟶ adulto
- Keys: {menor}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Agente:

- FD's: id ⟶ telemovel, email, nome
  telemovel ⟶ email, nome, id
  email ⟶ telemovel, nome, id
- Keys: {id}, {telemovel}, {email}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Artista:

- FD's: id ⟶ nome, salario, agente, genero, nacionalidade
  nome ⟶ id, genero, nacionalidade, agente, salario
- Keys: {id}, {nome}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Atua:

- FD's: Não tem FD's não triviais, todos os atributos compõem a chave primária.
- Keys: {artista, hora_inicio, data, palco_num, palco_zona}
- Violações: Não tem.

Banca:

- FD's: id ⟶ area, nome, zona, renda, classificacao, entidade
  area, zona ⟶ renda
- Keys: {id}
- Violações: BCNF: area, zona ⟶ renda é uma violação.
  3NF: area, zona ⟶ renda é uma violação.

Classificacao:

- FD's: id → tipo
  tipo → id
- Keys: {id}, {tipo}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Concerto:

- FD's: hora_inicio, palco_num, palco_zona, data → hora_fim
- Keys: {hora_inicio, palco_num, palco_zona, data}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Convidado:

- FD's: participante → tipo, entidade
- Keys: {participante}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

EncarregueArtista:

- FD's: Não tem FD's não triviais, todos os atributos compõem a chave primária.
- Keys: {artista, funcionario}
- Violações: Não tem.

EncarregueConvidado:

- FD's: Não tem FD's não triviais, todos os atributos compõem a chave primária.
- Keys: {convidado, funcionario}
- Violações: Não tem.

Entidade:

- FD's: id → nome, telemovel, email
  nome → id, telemovel, email
  telemovel → id, nome, email
  email → id, nome, telemovel
- Keys: {id}, {nome}, {telemovel}, {email}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Especialidade:

- FD's: id → nome
  nome → id
- Keys: {id}, {nome}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Funcionario:

- FD's: id → nif, nome, morada, salario, nacionalidade, especialidade
  nif → id, nome, morada, salario, nacionalidade, especialidade
- Keys: {id}, {nif}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Genero:

- FD's: id → nome
  nome → id
- Keys: {id}, {nome}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Nacionalidade:

- FD's: id → nome
  nome → id
- Keys: {id}, {nome}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Normal:

- FD's: participante → menor_idade
- Keys; {participante}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Palco:

- FD's: num, zona → capacidade
- Keys: {num, zona}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Participante:

- FD's: id → nome, nif, nacionalidade
  nif → id, nome, nacionalidade
- Keys: {id}, {nif}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Patrocinador:

- FD's: entidade → patrocinio
- Keys: {entidade}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Tem:

- FD's: participante → bilhete, codigo, ativado
  codigo → participante, bilhete, ativado
- Keys; {participante}, {codigo}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Tipo:

- FD's: id → nome
  nome → id
- Keys: {id}, {nome}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

TipoBilhete:

- FD's: id → nome, data_inicio, acampamento, custo
  nome, data_inicio, acampamento → id, custo
- Keys: {id}, {nome, data_inicio, acampamento}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Trabalha:

- FD's: Não tem FD's não triviais, todos os atributos compõem a chave primária.
- Keys: {funcionario, zona}
- Violações: Não tem.

Zona:

- FD's: codigo → custo_m2, area
- Keys: {codigo}
- Violações: Não tem, todas as FD's têm uma chave candidata no seu lado esquerdo.

Lista de Restrições e Detalhes de Implementação

Acompanha:

- menor:
    1. Restrição: Cada menor tem de ser acompanhado por um e um só adulto.
       Implementação: Restrição chave (PRIMARY KEY)
    2. Restrição: Cada tuplo da relação tem de ser sempre constituído por um menor.
       Implementação: Restrição NOT NULL
- adulto:
    1. Restrição: Cada tuplo da relação tem de ser sempre constituído por um adulto.
       Implementação: Restrição NOT NULL

Agente:

- id:
    1. Restrição: Não existem dois ou mais agentes com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Cada Agente tem de ter sempre um nome
       Implementação: Restrição NOT NULL
- telemovel:
    1. Restrição: Não existe dois ou mais agentes com o mesmo número de telemóvel.
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada agente tem de ter sempre um número de telemóvel.
       Implementação: Restrição NOT NULL
    3. Restrição: O número de telemóvel tem de ter 12 dígitos
       Implementação: Restrição CHECK
- email:
    1. Restrição: Não existem dois ou mais agentes com o mesmo email
       Implementação: Restrição UNIQUE
    2. Restrição: Cada agente tem de ter sempre um email
       Implementação: Restrição NOT NULL

Artista:

- id:
    1. Restrição: Não existem dois ou mais artistas com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Cada artista tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL
- salario:
    1. Restrição: Um artista não pode ter salário negativo
       Implementação: Restrição CHECK
- agente:
    1. Restrição: Cada artista tem de ter sempre um agente associado
       Implementação: Restrição de integridade referencial
    2. Restrição: O género deve ser válido *
       Implementação: Restrição NOT NULL
- genero:
    1. Restrição: Um artista deve ter sempre o seu género de música associado
       Implementação: Restrição de integridade referencial
    2. Restrição: O género deve ser válido *
       Implementação: Restrição NOT NULL
- nacionalidade:
    1. Restrição: Um artista deve ter sempre uma nacionalidade associada
       Implementação: Restrição de integridade referencial
    2. Restrição: A nacionalidade de um artista deve ser válida *
       Implementação: Restrição NOT NULL

Atua:

- artista:
    1. Restrição: Cada tuplo da relação tem de ter o artista que irá atuar associado
       Implementação: Restrição de integridade referencial
    2. Restrição: O artista tem de ser válido *
       Implementação: Restrição NOT NULL

- hora_inicio:
    1. Restrição: Cada tuplo da relação tem de ter sempre uma hora de início associada
       Implementação: Restrição de integridade referencial
    2. Restrição: A hora de inicio ao qual está associado tem de ser válida
       Implementação: Restrição NOT NULL
- data:
    1. Restrição: Cada tuplo da relação atua tem de ter sempre uma data associada
       Implementação: Restrição de integridade referencial
    2. Restrição: A data ao qual está associado tem de ser válida
       Implementação: Restrição NOT NULL
- palco_num e palco_zona:
    1. Restrição: Uma atuação acontece num palco, tendo, portanto, de estar sempre associada a um
       Implementação: Restrição de integridade referencial
    2. Restrição: Os atributos palco_num e palco_zona têm de pertencer um palco válido *
       Implementação: Restrição NOT NULL

Restrição: Não podem haver dois ou mais tuplos da relação atua com os mesmos atributos
Implementação: Restrição chave (PRIMARY KEY)
Restrição: O palco associado a palco_num e palco_zona tem de ser o mesmo
Implementação: Gatilho

Banca:

- id:
    1. Restrição: Não existem duas ou mais bancas com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver duas ou mais bancas com o mesmo nome
       Implementação: Restrição UNIQUE
    2. Restrição: Cada banca tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL

- area:
    1. Restrição: Cada banca tem de ter sempre uma área associada
       Implementação: Restrição NOT NULL
    2. Restrição: Cada banca tem de ter uma área positiva
       Implementação: Restrição CHECK
- zona:
    1. Restrição: Cada banca tem de ter sempre uma zona associada
       Implementação: Restrição de integridade referencial
    2. Restrição: A zona associada tem de ser válida *
       Implementação: Restrição NOT NULL
- renda:
    1. Restrição: Cada banca tem de ter sempre uma renda associada
       Implementação: Restrição NOT NULL
    2. Restrição: A renda deve ser calculada fazendo o produto entre a área ocupada pela banca e o custo por metro quadrado da zona em que se encontra
       Implementação: Gatilho

- classificacao:
    1. Restrição: Cada banca deve ter associada a sua classificação
       Implementação: Restrição de integridade referencial
    2. Restrição: Esta classificação tem de ser válida *
       Implementação: Restrição NOT NULL
- entidade:
    1. Restrição: Cada banca deve pertencer a uma entidade
       Implementação: Restrição de integridade referencial
    2. Restrição: A entidade não pode ser inválida
       Implementação: Restrição NOT NULL

Classificacao:

- id:
    1. Restrição: Não podem haver duas ou mais classificações com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- tipo:
    1. Restrição: Não podem haver duas ou mais classificações com o mesmo tipo
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada classificação tem de ter sempre um tipo associado
       Implementação Restrição NOT NULL

Concerto:

- hora_inicio:
    1. Restrição: Cada concerto tem sempre uma hora de início associada
       Implementação: Restrição NOT NULL
- hora_fim:
    1. Restrição: Cada concerto tem sempre uma hora de fim associada
       Implementação: Restrição NOT NULL
- data:
    1. Restrição: Cada concerto tem sempre uma data associada
       Implementação: Restrição NOT NULL
- palco_num e palco_zona:
    3. Restrição: Um concerto acontece num palco, tendo, portanto, de estar sempre associado a um
       Implementação: Restrição de integridade referencial
    4. Restrição: Os atributos palco_num e palco_zona têm de pertencer um palco válido *
       Implementação: Restrição NOT NULL

Restrição: Não existem dois concertos com a mesma hora de início, data e palco.
Implementação: Restrição chave (PRIMARY KEY)
Restrição: O palco associado a palco_num e palco_zona tem de ser o mesmo
Implementação: Gatilho

Convidado:

- participante:
    1. Restrição: Não podem haver dois ou mais tuplos da relação convidado que representem o mesmo participante
       Implementação: Restrição chave (PRIMARY KEY)
- tipo:
    1. Restrição: Cada convidado tem de ter sempre um tipo associado
       Implementação: Restrição de integridade referencial
    2. Restrição: Esse tipo tem de ser válido *
       Implementação: Restrição NOT NULL
- entidade:
    1. Restrição: Cada convidado pode estar associado a uma entidade (podendo esta ser NULL)
       Implementação: Restrição de integridade referencial

EncarregueArtista:

- artista:
    1. Restrição: Cada tuplo da relação tem de ter sempre um artista associado
       Implementação: Restrição de integridade referencia
    2. Restrição: Esse artista não pode ser inválido *
       Implementação: Restrição NOT NULL
- funcionario:
    1. Restrição: Cada tuplo da relação tem de ter sempre um funcionário associado
       Implementação: Restrição de integridade referencia
    2. Restrição: O funcionário tem de existir (ser válido) *
       Implementação: Restrição NOT NULL
    3. Restrição: Cada artista deve ter pelo menos um supervisor como seu encarregado
       Implementação: Gatilhos

Restrição: Não podem haver dois tuplos da relação com os mesmos atributos

Implementação: Restrição chave (PRIMARY KEY)

EncarregueConvidado:

- convidado:
    1. Restrição: Cada tuplo da relação tem de ter sempre um convidado associado
       Implementação: Restrição de integridade referencia
    2. Restrição: Tendo um convidado associado, este não pode ser inválido *
       Implementação: Restrição NOT NULL
- funcionario:
    1. Restrição: Cada tuplo da relação tem de ter sempre um funcionário associado
       Implementação: Restrição de integridade referencia
    2. Restrição: O funcionário encarregue tem de ser válido *
       Implementação: Restrição NOT NULL

Restrição: Não podem haver dois tuplos da relação com os mesmos atributos
Implementação: Restrição chave (PRIMARY KEY)

Entidade:

- id:
    1. Restrição: Não podem haver duas ou mais entidades com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver duas ou mais entidades com o mesmo nome
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada entidade tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL
- telemovel:
    1. Restrição: Não existe duas ou mais entidades com o mesmo número de telemóvel.
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada entidade tem de ter sempre um número de telemóvel
       Implementação: Restrição NOT NULL
    3. Restrição: O número de telemóvel tem de ter 12 dígitos
       Implementação: Restrição CHECK

- email:
    1. Restrição: Não existem duas ou mais entidades com o mesmo email
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada entidade tem de ter sempre um email associado
       Implementação: Restrição NOT NULL

Especialidade:

- id:
    1. Restrição: Não podem haver duas ou mais especialidades com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver duas ou mais especialidades com o mesmo nome
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada especialidade tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL

Funcionario:

- id:
    1. Restrição: Não podem haver dois ou mais funcionários com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Cada funcionário tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL
- nif:
    1. Restrição: Não podem haver dois ou mais funcionários com o mesmo nif
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada funcionário tem de ter sempre um nif associado
       Implementação: Restrição NOT NULL
    3. Restrição: O nif tem de ter sempre nove dígitos
       Implementação: Restrição CHECK
- morada:
    1. Restrição: Cada funcionário tem de ter sempre uma morada associada
       Implementação: Restrição NOT NULL

- salario:
    1. Restrição: Cada funcionário tem de ter sempre um salário associado
       Implementação: Restrição NOT NULL
    2. Restrição: O salário tem de ser sempre positivo
       Implementação: Restrição CHECK
- especialidade:
    1. Restrição: Cada funcionário tem de ter sempre uma especialidade associada
       Implementação: Restrição de integridade referencia
    2. Restrição: É obrigatório ser uma especialidade bem definida (válida)
       Implementação: Restrição NOT NULL
    3. Restrição: Todos os funcionários da mesma especialidade devem ter o mesmo salário
       Implementação: Gatilho
- nacionalidade:
    1. Restrição: Cada funcionário tem de ter sempre uma nacionalidade associada
       Implementação: Restrição de integridade referencia
    2. Restrição: A nacionalidade tem de ser válida
       Implementação: Restrição NOT NULL

Genero:

- id:
    1. Restrição: Não podem haver dois ou mais géneros com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver dois ou mais géneros com o mesmo nome
       Implementação: Restrição chave (PRIMARY KEY)
    2. Restrição: Cada género tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL

Nacionalidade:

- id:
    1. Restrição: Não podem haver duas ou mais nacionalidades com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver duas ou mais nacionalidades com o mesmo nome
       Implementação: Restrição chave (PRIMARY KEY)
    2. Restrição: Cada nacionalidade tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL

Normal:

- participante:
    1. Restrição: Cada tuplo da relação tem de ter sempre um participante associado.
       Implementação: Restrição de integridade referencia
    2. Restrição: Não podem haver dois ou mais tuplos da relação normal que representem o mesmo participante.
       Implementação: Restrição chave (PRIMARY KEY)
    3. Restrição: O participante tem de ser válido *
       Implementação: Restrição NOT NULL
- menor_idade:
    1. Restrição: Cada tuplo da relação tem de ter sempre uma indicação se o participante é ou não menor de idade.
       Implementação: Restrição NOT NULL
    2. Restrição: A menor_idade é zero, indicando que não é menor de idade, ou 1, indicando que é menor de idade.
       Implementação: Restrição CHECK
    3. Restrição: Por omissão, assume-se que o participante é maior de idade (menor_idade igual a 0).
       Implementação: Restrição DEFAULT

Palco:

- num:
    1. Restrição: Cada palco tem de ter sempre um número associado.
       Implementação: Restrição NOT NULL

- capacidade:
    1. Restrição: Cada palco tem de ter sempre uma capacidade associada
       Implementação: Restrição NOT NULL
    2. Restrição: A capacidade tem de ser sempre positiva
       Implementação: Restrição CHECK
- zona:
    1. Restrição: Cada palco tem de ter sempre uma zona associada
       Implementação: Restrição de integridade referencia
    2. Restrição: A zona tem de ser válida *
       Implementação: Restrição NOT NULL

Restrição: Não podem haver dois palcos com o mesmo número e zona
Implementação: Restrição chave (PRIMARY KEY)

Participante:

- id:
    1. Restrição: Não podem haver dois ou mais participantes com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Cada participante tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL
- nif:
    1. Restrição: Não podem haver dois ou mais participantes com o mesmo nif
       Implementação: Restrição chave (UNIQUE)
    2. Restrição: Cada participante tem de ter sempre um nif associado
       Implementação: Restrição NOT NULL
    3. Restrição: O nif tem de ter sempre 9 dígitos
       Implementação: Restrição CHECK
- nacionalidade:
    1. Restrição: Cada participante tem de ter sempre uma nacionalidade associada
       Implementação: Restrição de integridade referencia

2. Restrição: A nacionalidade tem de ser válida *
   Implementação: Restrição NOT NULL

Patrocinador:

- entidade:
  1. Restrição: Não pode haver dois patrocinadores com a mesma entidade
     Implementação: Restrição chave (PRIMARY KEY)
  2. Restrição: Cada patrocinador tem de ter sempre uma entidade associada
     Implementação: Restrição de integridade referencia
- patrocinio:
  1. Restrição: Cada patrocinador tem de ter sempre um patrocínio associado
     Implementação: Restrição NOT NULL
  2. Restrição: O patrocínio tem de ser sempre maior que 100
     Implementação: Restrição CHECK

Tem:

- codigo:
  1. Restrição: Cada tuplo da relação tem de ter sempre um código associado.
     Implementação: Restrição NOT NULL
  2. Restrição: Não podem haver dois ou mais tuplos da relação com o mesmo código.
     Implementação: Restrição chave (UNIQUE)
- ativado:
  1. Restrição: Cada tuplo da relação tem de ter sempre uma indicação se o bilhete já foi ativado.
     Implementação: Restrição NOT NULL
  2. Restrição: A indicação da ativação do bilhete (ativado) tem de ser 0, indicando que o bilhete ainda não foi ativado, ou 1, indicando que já foi ativado.
     Implementação: Restrição CHECK
  3. Restrição: Assume-se que, aquando da inserção, o bilhete ainda não foi ativado, ou seja, ativado igual a 0.
     Implementação: Restrição DEFAULT

- participante:
    1. Restrição: Cada tuplo da relação tem de ter ser um participante associado.
       Implementação: Restrição de integridade referencia
    2. Restrição: O participante deve ser válido *
       Implementação: Restrição NOT NULL
    3. Restrição: Um participante não pode ter mais do que um bilhete
       Implementação: Restrição chave (PRIMARY KEY)
- bilhete:
    1. Restrição: Cada tuplo da relação deve ter sempre uma indicação do tipo de bilhete
       Implementação: Restrição de integridade referencia
    2. Restrição: O bilhete deverá ser válido *
       Implementação: Restrição NOT NULL

Restrição: No caso de o participante ser um Convidado, dependendo ainda do tipo (Celebridade, Representante, Media, Outro), devem ser associados tipos de bilhete específicos (VIP, VIP/PASS, PASS, VIP/PASS, respetivamente)
Implementação: Gatilho

Tipo:

- id:
    1. Restrição: Não podem haver dois ou mais tipos de convidado com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- nome:
    1. Restrição: Não podem haver dois ou mais tipos de convidado com o mesmo nome
       Implementação: Restrição chave (PRIMARY KEY)
    2. Restrição: Cada tipo de convidado tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL

TipoBilhete:

- id:
    1. Restrição: Não podem haver dois ou mais tipos de bilhete com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)

- nome:
    1. Restrição: Cada tipo de bilhete tem de ter sempre um nome associado
       Implementação: Restrição NOT NULL
- data_inicio:
    1. Restrição: Cada tipo de bilhete tem de ter sempre uma data de início associada
       Implementação: Restrição NOT NULL
- acampamento:
    1. Restrição: Cada tipo de bilhete tem de ter sempre uma indicação se inclui acampamento ou não
       Implementação: Restrição NOT NULL
    2. Restrição: A indicação do acesso ao acampamento tem de ser 0, indicando que não tem acesso, ou 1, indicando que tem acesso.
       Implementação: Restrição CHECK
- custo:
    1. Restrição: Cada tipo de bilhete tem de ter sempre um custo associado
       Implementação: Restrição NOT NULL
    2. Restrição: O custo de um bilhete não pode ser negativo
       Implementação: Restrição CHECK

Trabalha:

- funcionario:
    1. Restrição: Cada tuplo da relação tem de ter sempre um funcionário associado
       Implementação: Restrição de integridade referencia
    2. Restrição: O funcionário deve ser válido *
       Implementação: Restrição NOT NULL
- zona:
    1. Restrição: Cada tuplo da relação tem de ter sempre uma zona associada
       Implementação: Restrição de integridade referencia
    2. Restrição: A zona deve ser válida *
       Implementação: Restrição NOT NULL

Restrição: Não podem haver dois ou mais tuplos com os mesmos atributos

Implementação: Restrição chave (PRIMARY KEY)

Zona:

- codigo:
    1. Restrição: Não podem haver duas ou mais zonas com o mesmo id
       Implementação: Restrição chave (PRIMARY KEY)
- custo_m2:
    1. Restrição: Cada zona tem de ter sempre um custo por metro quadrado associado
       Implementação: Restrição NOT NULL
- area:
    1. Restrição: Cada zona tem de ter sempre uma área associada
       Implementação: Restrição NOT NULL
    2. Restrição: A área tem de ser sempre positiva
       Implementação: Restrição CHECK
    3. Restrição: A área de uma zona não pode ser menor que a soma das áreas de todas as suas bancas
       Implementação: Gatilho

Nota: Nas restrições apontadas por um '*', é considerado valor válido como não podendo ser nulo (NOT NULL).

Nota: As restrições com designação "Gatilho" visam a implementação desta ferramenta de modo a obter uma reação após uma remoção, inserção ou mudança que implique a alteração de um ou mais tuplos das tabelas, com os objetivos descritos pelas restrições onde estão indicados.

Interrogações

1. Lista todos os **dias** nos quais existem **concertos** em todos os **palcos** do festival.
2. Obtém, para cada **participante menor** e seu respetivo **acompanhante**, os seus nomes e código dos bilhetes.
3. Obtém o nome, NIF, código do bilhete, nome da entidade e patrocínio, de todos os **convidados**, com **bilhete ativado**, que representam uma **entidade**, em que esta, por sua vez, tem pelo menos uma **banca**, é **patrocinadora** e cujo patrocínio é superior a 200000.
4. Obtém o nome de todos **artistas**, cujo respetivo **agente** representa mais que um artista, e que são acompanhados por pelo menos um **funcionário** da mesma **nacionalidade**, sendo o seu nome também mostrado.
5. Obtém, para cada **tipo** de banca, o nome, a área e **zona** da banca que ocupa maior área indicando também se pertence a uma zona que contém **palcos** (Coluna Palcos: "existem" – a zona contém palcos; "não existem" – a zona não contém palcos)
6. Obtém o **dia** com maior adesão, isto é, o dia com mais **bilhetes adquiridos**, mostrando tanto o número de bilhetes comprados como quantos desses foram já **ativados**. Tem em conta não só os bilhetes diários, mas também os passes gerais para efeitos de contagem.
7. Obtém, para cada **dia**, os **artistas** que atuam apenas uma única vez, que têm um salário acima da média daquele dia, e cujo **palco** em que atuam também tenha uma capacidade acima da média relativa a esse dia.
8. Obtém, para cada hora do dia (00:00 – 00:59, 01:00 – 01:59, …, 23:00 – 23:59), o **gênero** de música mais popular, isto é, do qual houve mais **concertos** em todo o festival, e indica quantos foram. Caso não haja nenhum concerto, no festival todo, a uma determinada hora, a mesma não é mostrada. Se houver mais que um gênero com o máximo de concertos numa dada hora, é mostrado apenas o primeiro.
9. Calcula uma estimativa do número de **bilhetes** que é necessário vender para não obter prejuízo. Ou seja, dado os patrocínios, rendas, salários com **artistas** e **funcionários,** quantos bilhetes é necessário vender para obter lucro 0. Assume-se que o custo de um bilhete é a média do custo de todos os tipos de bilhetes que o festival oferece.
10. Obtém todos os pares de **funcionários** que estão responsáveis por, pelo menos um **artista** em comum. Indica os seus nomes bem como o artista em comum que acompanham.

Nota: Inicialmente, o nosso objetivo era ter interrogações o mais úteis e pertinentes para um possível utilizador da base de dados. Contudo, obedecer a isto nas 10 interrogações provou-se difícil, tendo em conta que também era do nosso interesse diversificar a forma de implementação de cada

uma (i.e não ter duas interrogações com implementações semelhantes em termos lógicos) e aumentar a sua complexidade, sendo que, a maioria das interrogações pertinentes para a nossa base de dados eram de fácil implementação (pouca complexidade). Assim, decidimos abdicar um pouco da pertinência das nossas interrogações de modo a compensar os restantes fatores.

Gatilhos

1. VerificaArea

Antes de qualquer inserção de tuplos na tabela Banca, deve ser verificado se a adição deste provoca uma violação, caso a soma das áreas de todas as bancas inseridas nessa **Zona** juntamente com a nova área da nova **Banca**, ultrapasse a área da própria zona. Em caso afirmativo, a inserção é abortada e uma mensagem de erro é escrita no ecrã.

Note-se que, para garantir a aproximação realista do problema, seria necessário criar mais dois **triggers**:

- No caso de haver **UPDATE** de uma ou mais bancas que afetasse a sua área (por exemplo, se uma banca quisesse ter mais espaço), o pedido teria de ser verificado antes do UPDATE acontecer.
- No caso de haver **UPDATE** da área de uma zona, deveria ser verificada a possibilidade de tal acontecer com base nas bancas já inseridas, se ocorresse uma diminuição da mesma.

2. DefineRenda

Aquando da inserção de um novo tuplo na tabela Banca, a renda desta deve ser definida como sendo o produto da área da **Banca** pelo custo por metro quadrado da **Zona** onde esta se encontra. A renda é sempre definida, independentemente do valor colocado pelo utilizador na inserção. De modo a garantir que está condição se verifica sempre, era necessário criar triggers para controlar também as situações em que:

- Ocorre **UPDATE** do custo por metro quadrado de uma Zona, no qual deveria atualizar a renda de todas as bancas pertencentes a essa zona.
- Ocorre **UPDATE** da área de uma banca, devendo a sua renda também ser atualizada.
- Ocorre **UPDATE** da renda de uma banca, que não deveria ser permitido, mesmo que o valor fosse correto.

3. DefineConvidado

Antes de definirmos que um **Participante** é um **Convidado** devemos garantir que este não foi já definido como um **Normal**. Caso este já tenha sido definido como um Normal, a inserção deve ser abortada e uma mensagem de erro deve ser impressa no ecrã.

Para a restrição fazer sentido e ser completa, seria necessário criar mais um trigger para garantir a outra metade da disjunção, ou seja, na inserção de um Normal, teríamos de verificar se o participante associado a este já se encontra na tabela como um Convidado.