deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

# Festival Planner

## Bases de Dados

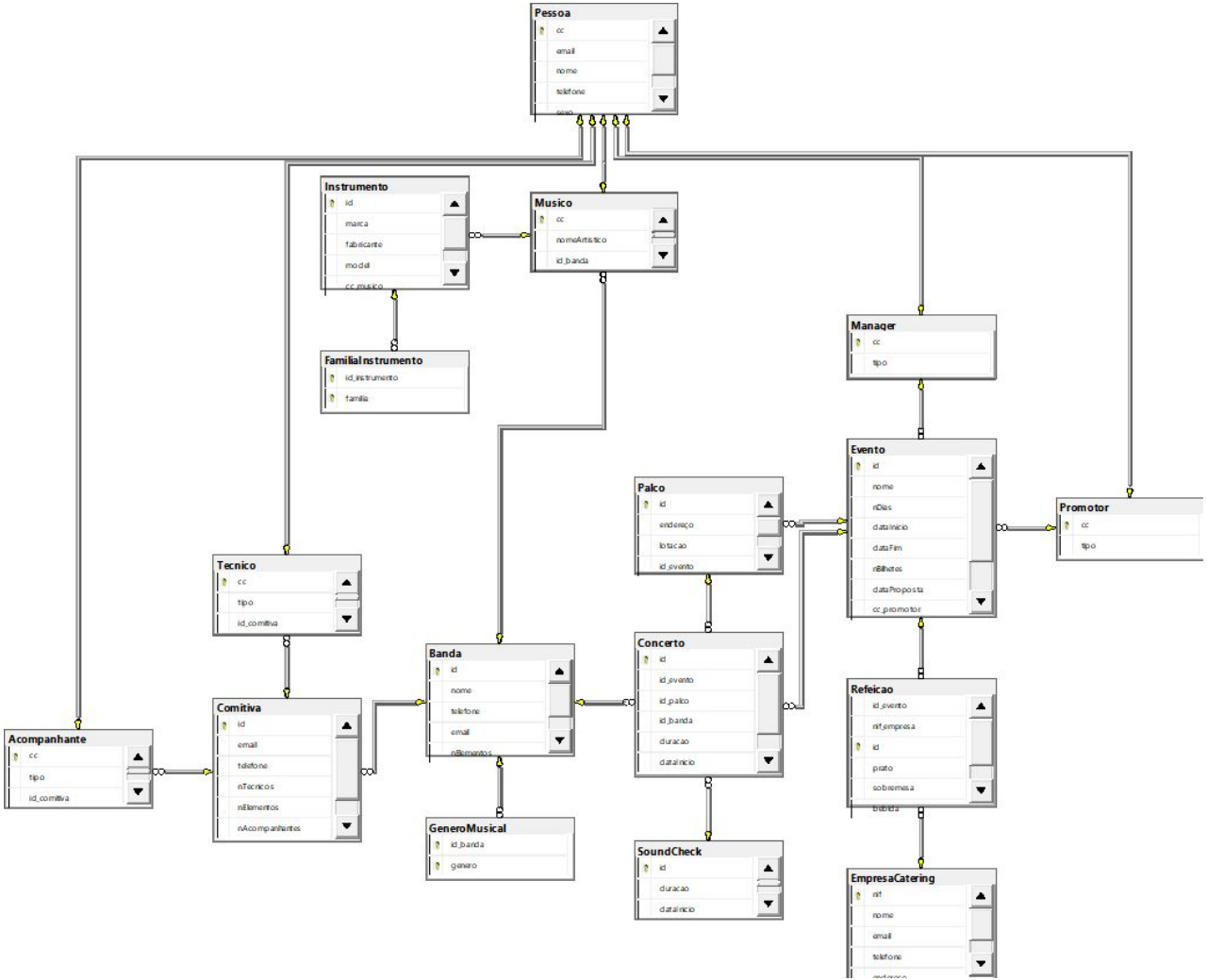
Projeto realizado no âmbito das cadeira de **Base de Dados**, turma **P2**

Miguel Nogueira 93082  David Araújo 93444

23 Junho 2022

# Introdução

- **Festival Planner** pretende gerir Eventos Festivos de Música.
- Estrutura baseada em **12 entidades distintas**, que se estendem a **17** quando **especificadas**.
- Entidade principal é um **Evento**, à qual as restante estão estão direta ou indiretamente dependentes.


Pessoa
cc
email
nome
telefone
Instrumento
id
marca
fabricante
modell
cc_musico
Musico
cc
nomeArtistico
id_banda
FamiliaInstrumento
id_instrumento
familia
Manager
cc
tipo
Evento
id
nome
nDias
dataInicio
dataFim
nBilhetes
dataProposta
cc_promotor
Promotor
cc
tipo
Palco
id
endereço
lotacao
id_evento
Tecnico
cc
tipo
id_comitiva
Banda
id
nome
telefone
email
nElementos
Concerto
id
id_evento
id_palco
id_banda
duracao
dataInicio
Refeicao
id_evento
nif_empresa
id
prato
sobremesa
bebida
Acompanhante
cc
tipo
id_comitiva
Comitiva
id
email
telefone
nTecnicos
nElementos
nAcompanhantes
GeneroMusical
id_banda
genero
SoundCheck
id
duracao
dataInicio
EmpresaCatering
nif
nome
email
telefone

DER

# SQL Scripts

- Setup
  - Criação das tabelas da base de dados, de acordo com o diagrama relacional
- Drops
  - Eliminação das tabelas da base de dados
- Inserts
  - Inserção de dados nas tabelas

# SQL Scripts

- Queries
  - Eventos cujo número de bilhetes seja maior que a média de todos os bilhetes vendidos
  - Promotores dos eventos a acontecer a partir de 2022
  - Empresas de catering e o número de eventos aos quais fornecem serviços
  - entre outras

- Views
  - *V_Bandas* que permite visualizar atributos da entidade banda
  - *V_Geral* que permite ter uma noção geral de um evento, sendo para isso necessário juntar atributos de várias entidades

# SQL Scripts

- Triggers
  - Apagar um concerto apaga também o seu soundcheck
  - Apagar um evento apaga também os seus concertos e soundchecks
  - Soundcheck não pode durar mais de 1 hora (se durar a sua duração é ajustada para 1 hora)
  - não podem existir dois soundchecks ao mesmo tempo

- Stored Procedures
  - Criar um evento;
  - Criar um concerto e um soundcheck para uma determinada banda;
  - Apagar um evento dado o id do mesmo;
  - entre outros

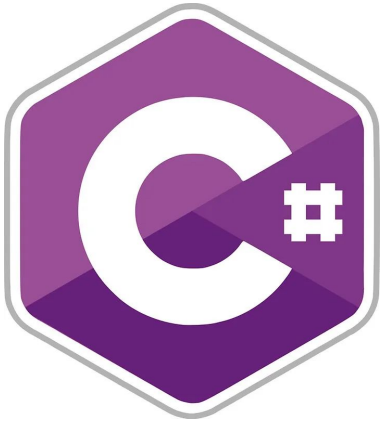
# SQL Scripts

- User Defined Functions
  - Dado um id devolve um evento
  - Dado um nome devolve um evento
  - Dado um ccPromotor devolve um evento
  - entre outras

- Indexes
  - *idx_eventId* sobre a tabela *Evento* e atributo *id*;
  - *idx_concertoId* sobre a tabela *Concerto* e atributo *id*;
  - *idx_BandaId* sobre a tabela *Banda* e atributo *id*;
  - *idx_Musico* sobre a tabela *Musico* e atributo *cc*.

# Interface

- Desenvolvido como **WFA .NET** em **C#**.
- Suportado numa estrutura de **classes correlacionadas** com as diferentes **entidades**.
- Suporta múltiplas **views** para **edição** de cada **entidade.**



# Interaction & SQL

**Festivals view**, displays the following view

```
CREATE VIEW FP.V_GERAL AS
        SELECT      FP.Evento.id, FP.Evento.nome AS EVENTO_NOME, FP.Evento.dataInicio, FP.Evento.nDias,
                    FP.Pessoa.nome AS PROMOTOR_NOME, FP.Pessoa.email AS PROMOTOR_EMAIL,
                    FP.Pessoa.telefone AS PROMOTOR_TELEFONE, FP.Banda.nome
        FROM        FP.Pessoa INNER JOIN
                    FP.Promotor ON FP.Pessoa.cc = FP.Promotor.cc INNER JOIN
                    FP.Evento ON FP.Promotor.cc = FP.Evento.cc_promotor LEFT JOIN
                    FP.Concerto  ON FP.Concerto.id_evento = FP.Evento.id LEFT JOIN
                    FP.Banda ON FP.Banda.id = FP.Concerto.id_banda;
```

# Interaction & SQL

## Criação de novos Eventos

**Stored Procedure**

```
CREATE PROCEDURE create_evento( @id VARCHAR(20), @nome VARCHAR(150), @numdias INT, @dataini DATE,
@datafim DATE, @numbilhetes INT, @dataproposta DATE, @cc_promotor VARCHAR(12), @cc_manager VARCHAR(12))
AS
BEGIN
	BEGIN TRY
		INSERT INTO FP.Evento VALUES(@id, @nome, @numdias, @dataini, @datafim, @numbilhetes,
@dataproposta, @cc_promotor, @cc_manager);
		PRINT 'Event created successfully!'
	END TRY
	BEGIN CATCH
		PRINT ERROR_MESSAGE()
	END CATCH
END
```

# Interaction & SQL

## Apagar Evento

### Stored Procedure

```
CREATE PROCEDURE delete_evento_byId( @id
VARCHAR(20))
AS
BEGIN
	BEGIN TRY
		BEGIN TRANSACTION
			DELETE FROM FP.EVENTO WHERE
id=@id;
			PRINT 'Event deleted successfully!'
		COMMIT
	END TRY
	BEGIN CATCH
		PRINT ERROR_MESSAGE()
		ROLLBACK
	END CATCH
END
```

### Trigger

```
CREATE TRIGGER delete_event ON FP.Evento
INSTEAD OF DELETE
AS
BEGIN
	DECLARE @ev_id AS VARCHAR(20);

	SELECT @ev_id = id FROM deleted;

	DELETE FROM FP.Concerto WHERE id_evento
= @ev_id;
	DELETE FROM FP.Evento WHERE id = @ev_id;
END
```

# Interaction & SQL

## Apagar Evento

```
CREATE PROCEDURE alter_evento( @id VARCHAR(20), @nome VARCHAR(150), @numdias INT, @dataini DATETIME, @datafim DATETIME, @numBilhetes INT,
@dataproposta DATETIME, @cc_promotor VARCHAR(12), @cc_manager VARCHAR(12) )
AS
BEGIN
	BEGIN TRY
		BEGIN TRANSACTION
			DECLARE @id_old AS VARCHAR(20);
			DECLARE @nome_old AS VARCHAR(150);
			DECLARE @numdias_old AS INT;
			DECLARE @dataini_old AS DATE;
			DECLARE @datafim_old AS DATE;
			DECLARE @numBilhetes_old INT;
			DECLARE @dataproposta_old DATE;
			DECLARE @cc_promotor_old VARCHAR(12);
			DECLARE @cc_manager_old VARCHAR(12);

			SELECT @id_old = id, @nome_old = nome, @numdias_old = nDias, @dataini_old = dataInicio, @datafim_old = dataFim, @numBilhetes_old =
nBilhetes, @dataproposta_old = dataProposta, @cc_promotor_old = cc_promotor, @cc_manager_old = cc_manager
			FROM FP.EVENTO
			WHERE FP.EVENTO.id = @id;

			IF @id_old != @id
			BEGIN
				UPDATE FP.EVENTO SET id = @id WHERE id=@id_old;
				PRINT 'Event id updated with success'
			END
…
```