Universidade de Aveiro
Mestrado Integrado em Engenharia de Computadores e Telemática

Unidade Curricular de Bases de Dados

Ano Letivo de 2019/2020

Gestão de uma escola de música

Gabriel Saudade 89304

Ruben Menino 89185

Junho de 2020

Gestão de uma escola de música

Gabriel Saudade 89304

Ruben Menino 89185

Junho de 2020

Índice

1. Introdução

1.1. Reconhecimento do problema

O principal problema consiste no facto de atualmente a escola de música da Santa Cecília em São Bernardo não ter nenhum programa de gestão da sua escola, como por exemplo de alunos, professores e eventos, cingindo-se a uma folha de Excel e a muito papel para efetuar a inscrição de novos alunos, sendo difícil de navegar através de tanta informação. Portanto o trabalho consiste na criação de uma base de dados de modo a facilitar o acesso detalhado a cada uma destas informações, assim como o gestão e criação de eventos, alunos e professores.

1.2. Avaliar a solução e síntese da solução

A base de dados consiste numa simples camada de relações entre entidades. Cada entidade terá vários tipos de atributos para a sua designação. Por exemplo, na entidade Professor e Aluno, terão de ter as características principais de uma pessoa, como nome, número de identificação fiscal, email, telemóvel, entre outros. Posteriormente, sendo uma escola de música haverá uma relação entre alunos, instrumentos e professores. Por exemplo, um aluno poderá estar associado a vários instrumentos e a um professor desse instrumento. Ainda, os alunos poderão estar associados a uma turma de formação musical e a uma ou várias valências com o devido representante. Quaisquer elementos da direção e/ou professor poderá criar um evento com autorização da direção. Posteriormente, será feita uma análise detalhada de todas estas relações. Será uma navegação dinâmica através de toda a informação necessária ao bom funcionamento de uma escola de música e de todas as suas vertentes

1.3. Estrutura do Relatório

Neste relatório descrevemos todas as fases do nosso projeto. Começamos por definir e contextualizar o nosso problema. Também apresentamos os requisitos aos quais achamos serem importantes a nossa BD ser capaz de responder. Depois partimos para as diferentes fases da nossa BD: concetual, lógica e física, validando e justificando cada uma delas.

2. Requisitos

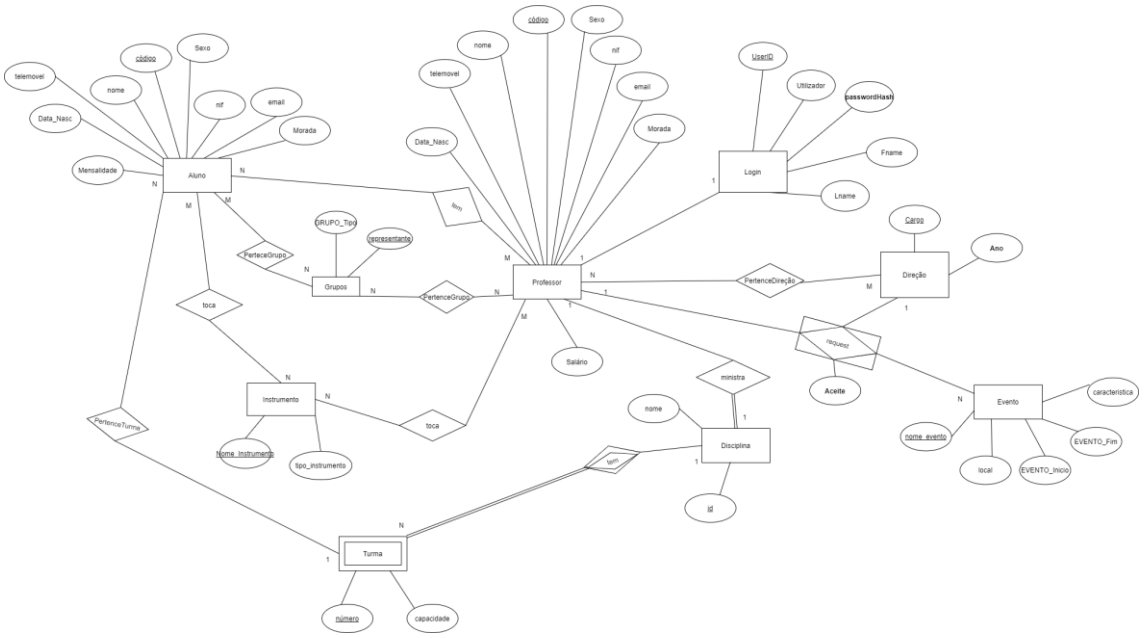
2.1. Requisitos do projeto

De acordo com a pesquisa efetuada constatou-se que a base de dados deve suportar os seguintes requisitos.

- A base de dados deve permitir guardar informações (nome, email, nif…) sobre os utilizadores;
- O utilizador poderá criar uma conta de utilizador;
- Após efetuado o registo, o utilizador pode fazer quantas adições desejar;
- Um aluno ou professor poderá ser adicionado à base de dados;
- Poderá ser atualizado cada um dos parâmetros do utilizador;
- Cada utilizador tem um respetivo id;
- Pesquisa por número id;
- Adicionar instrumento ao professor ou ao aluno;
- O número de instrumentos é dependente do cargo na base de dados;
- É possível saber o número de alunos por professor;
- Criar uma turma e adicionar o aluno a uma respetiva turma;
- A partir do id, adicionar um cargo a um professor na direção;
- Adicionar um evento;
- Possibilidade de um membro da direção aceitar o evento ou não;

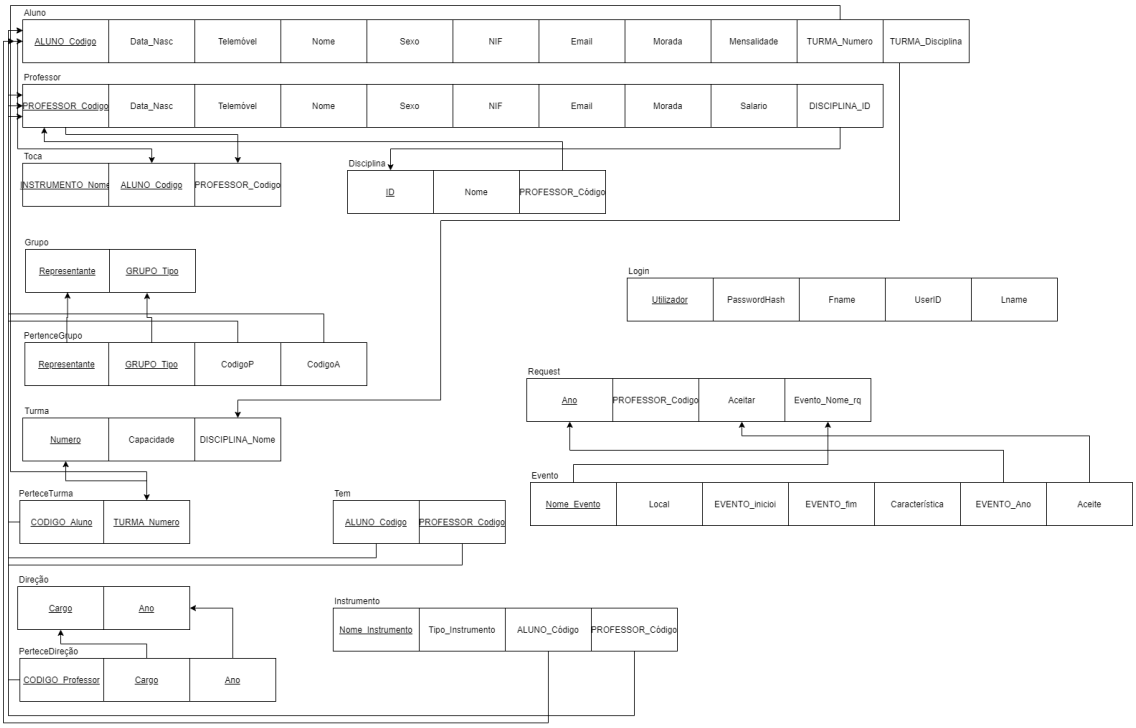
3. Metodologia para construção do modelo conceptual

3.1. Diagrama Entidade-Relação (DER)



3.2. Modelo Relacional

Nesta secção apresentamos o modelo relacional utilizado para a base de dados da plataforma, derivado do diagrama entidade-relação anterior.



4. Definição da estrutura

4.1 SQL-DDL

Nesta secção apresentamos a estrutura da base de dados, apresentada em forma de comandos SQL DDL. As tabelas inseridas encontram-se no schema “projeto”.

Um professor e aluno surge como uma das entidades centrais do sistema, devido ao facto de estar “conectada” com muitas das outras entidades. Muitas delas vão utilizar o id determinado professor ou aluno, e conseguir realizar muitas das funcionalidades.

A assinatura das entidades presentes na base de dados está listada abaixo. As tables completas podem ser visualizadas do ficheiro “Tables.sql”.

```
GO
CREATE SCHEMA projeto
go

CREATE TABLE projeto.Toca();

CREATE TABLE projeto.Disciplina();

CREATE TABLE projeto.Participa();

CREATE TABLE projeto.Grupo();

CREATE TABLE projeto.Turma();

CREATE TABLE projeto.PertenceTurma();

CREATE TABLE projeto.PertenceDirecao();

CREATE TABLE projeto.PertenceGrupo();

CREATE TABLE projeto.Direcao();

CREATE TABLE projeto.Login();

CREATE TABLE projeto.Request();

CREATE TABLE projeto.Evento();

CREATE TABLE projeto.Aluno();

CREATE TABLE projeto.Tem();

CREATE TABLE projeto.Professor();
```

4.2 Views

Views são simples instruções SQL que nos permite listar e visualizar as tabelas e entidades da base de dados, permitindo selecionar e juntar informação entre entidades, facilitando o uso depois no código C#.

- Listar os alunos e o seu respectivo instrumento
- Listar os professores e o seu respectivo instrumentos

```
drop view projeto.profInst
CREATE VIEW projeto.profInst AS
SELECT projeto.Professor.PROFESSOR_Codigo, projeto.Toca.INTRUMENTO_Nome
FROM projeto.Professor
inner JOIN projeto.Toca
ON Toca.PROFESSOR_Codigo = Professor.PROFESSOR_Codigo
```

- Lista de eventos e lista de eventos aceites
- Lista da turma

4.3 Triggers

Ao criar instâncias de uma entidade optou-se por usar *Stored Procedures*, no entanto achamos a necessidade de usar triggers ou outras opções.

No primeiro trigger, “removePrevious” achamos a necessidade de o utilizar para um caso específico. Havendo alunos que tocam o mesmo instrumento, houve a necessidade de ao remover um dos instrumentos, não remover a totalidade dos alunos, daí a ativação deste trigger, removendo só o aluno com o respetivo instrumento selecionado.

```
GO
CREATE Trigger removePrevious ON projeto.Toca
    AFTER INSERT, UPDATE
        AS
        BEGIN
            DECLARE @INSTRUMENTO_Nome AS VARCHAR(30);
            DECLARE @ALUNO_Codigo AS INT;

            SELECT @INSTRUMENTO_Nome, @ALUNO_Codigo FROM inserted;


            if @ALUNO_Codigo IN (SELECT ALUNO_Codigo FROM projeto.Toca WHERE ALUNO_Codigo = @ALUNO_Codigo)
            BEGIN
                DELETE FROM projeto.Toca  WHERE ALUNO_Codigo = @ALUNO_Codigo
                DELETE FROM projeto.Aluno WHERE ALUNO_Codigo = @ALUNO_Codigo
            END


            INSERT INTO projeto.Toca(INTRUMENTO_Nome, ALUNO_Codigo, PROFESSOR_Codigo) VALUES(@INSTRUMENTO_Nome, @ALUNO_Codigo, null)
        END

GO
```

Utilizamos também outros trigger, para garantir que um professor adicionado não tinha um id já existente de um aluno, e vice-versa.

```
GO
CREATE TRIGGER projeto.VerProfessor ON projeto.Professor
AFTER INSERT, UPDATE
AS
    IF (EXISTS(SELECT Aluno_Codigo FROM projeto.Aluno WHERE Aluno_Codigo in (SELECT PROFESSOR_Codigo FROM inserted)))
        BEGIN
            RAISERROR ('Professor não pode ser inserido/atualizado - já existe um aluno com esse numero', 16,1);
            ROLLBACK TRAN;
        END

GO
CREATE TRIGGER projeto.VerAluno ON projeto.Aluno
AFTER INSERT, UPDATE
AS
    IF (EXISTS(SELECT PROFESSOR_Codigo FROM projeto.Professor WHERE PROFESSOR_Codigo in (SELECT ALUNO_Codigo FROM inserted)))
        BEGIN
            RAISERROR ('Aluno não pode ser inserido/atualizado - já existe um professor com esse numero.', 16,1);
            ROLLBACK TRAN;
        END
```

4.4 Stored Procedures

Os Stored Procedures foram usados na criação de objetos de uma dada entidade derivada de outra, executar algumas condições necessárias e fazer na atualização de informação.

- Efetuar o login na plataforma

As stored procedures “ projeto.adicionarRegisto” e “projeto.logar” permitem conseguir registar uma conta e utilizá-la para entrar na aplicação. Decidimos encriptar os dados de password criados da maneira mais segura. Basicamente recorremos a uma criptografia salt, onde é gerado um hash a partir da combinação de uma senha e texto gerado aleatoriamente.

```
ALTER PROCEDURE projeto.adicionarRegisto
    @pLogin NVARCHAR(50),
    @pPassword NVARCHAR(50),
    @pFirstName NVARCHAR(40) = NULL,
    @pLastName NVARCHAR(40) = NULL,
    @responseMessage NVARCHAR(250) OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @salt UNIQUEIDENTIFIER=NEWID()
    BEGIN TRY
        INSERT INTO projeto.Login (Utilizador, PasswordHash, Salt, Fname, Lname)
        VALUES(@pLogin, HASHBYTES('SHA2_512', @pPassword+CAST(@salt AS NVARCHAR(36))), @salt, @pFirstName, @pLastName)
       SET @responseMessage='Success'
    END TRY
    BEGIN CATCH
        SET @responseMessage=ERROR_MESSAGE()
    END CATCH
END
```

Posteriomente, necessitámos de conseguir logar na aplicação para autenticar o usuário usando a senha criptografada com o salt.

```
DROP PROCEDURE projeto.logar
CREATE PROCEDURE projeto.logar
    @pLoginName NVARCHAR(254),
    @pPassword NVARCHAR(50),
    @responseMessage NVARCHAR(250)='' OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @userID INT
    IF EXISTS (SELECT TOP 1 UserID FROM projeto.Login WHERE Utilizador=@pLoginName)
    BEGIN
        SET @userID=(SELECT UserID FROM projeto.Login WHERE Utilizador=@pLoginName AND PasswordHash=HASHBYTES('SHA2_512',
        @pPassword+CAST(Salt AS NVARCHAR(36))))

       IF(@userID IS NULL)
           SET @responseMessage='Incorrect password'
       ELSE
           SET @responseMessage='User successfully logged in'
    END
    ELSE
       SET @responseMessage='Invalid login'
END
```

- Criar um aluno e um Professor
- Adicionar um instrumento
- Fazer a atualização da informação de um aluno ou professor
- Algumas condições necessárias na remoção
- Criar um evento
- Associar uma turma ao aluno e um disciplina ao professor
- Adicionar um professor à direção
- Atualizar o salário de um professor na adição de novos alunos

4.5 User Defined Functions (UDF)

As *UDF's* foram útil no sentido que permitiu criar uma função auxiliar ao *Stored Procedure* na consulta do salário de um determinado professor.

- Consulta do salário de um determinado professor

```
DROP FUNCTION projeto.getCountAlunosProf
CREATE FUNCTION projeto.getCountAlunosProf(@PROFESSOR_Codigo SMALLINT)
RETURNS INT
AS
BEGIN
    DECLARE @returnValue INT;
        SET @returnValue = (SELECT COUNT (ALUNO_Codigo) from projeto.alunInst
        WHERE INTRUMENTO_Nome = (SELECT INTRUMENTO_Nome FROM projeto.profInst WHERE PROFESSOR_Codigo = @PROFESSOR_Codigo))
    RETURN @returnValue
END
```

4.6 Indices

Utilizamos dois índices para facilitar a busca de informações nas tabelas com o menor número possível de operações de leituras, tornando assim a busca mais rápida e eficiente.

5. Interação com a base de dados

A nossa interface está dividida por por 7 tópicos gerais, divididos alguns desses por subtópicos para melhorar utilização da interface. São esses tópicos: “Login”, “Aluno”, “Professor”, “Grupos”, “Turma”, “Direção” e “Evento”.

1. Register_Form : Necessitamos de garantir que o utilizador já não existia, e o uso da store procedure “projeto.adicionarRegisto” para conseguir registar esses dados na base de dados.
2. Login_Form: Store procedure projeto.Logar, para conseguir “entrar” na nossa base de dados com o utilizador e a respetiva password codificada.
3. Aluno

a. AdicionarAluno

Utilizamos várias procedures para dar a inserção do aluno “projeto.criarAluno”, para fazer o update “projeto.updateAluno” utilizada numa outra form e “projeto.deleteAluno” para remover o respetivo aluno.

b. alunosLista

Utilizada várias views para obter informações.

c. UpdateDelete

É utilizada uma função de outra form, para dar update, e faz-se também uma procura por determinado aluno/professor

4. Professor

a. ListadeAlunosProfessor

Possivel obter o número de alunos que tocam o mesmo instrumento.

. que o professor.

b. professoresLista

É possivel observar os dados todos dos professor, tanto com o salário atualizado devido à UDF “projeto.getCountAlunosProf “ com a stored procedure “projeto.atualizarSalario”

5. Grupos

a. Grupos

É possivel criar um grupo, “projeto.criarGrupos” para posteriormente adicionar um aluno a ele mesmo.

b. Form1

A partir dos nomes dos grupos e dos ID criados, tanto como professor, tanto com aluno é possível adicioná-lo a um grupo para fazer parte dele.

6. Turma

a. Turma

É possivel criar uma turma, “projeto.criarTurma” para posteriormente adicionar um aluno a ela mesmo.

b. adicionarAlunoTurma

É possivel, a partir dos ID criados da turma, colocar um aluno numa respetiva turma criada.

7. Direção

a. AdicionarMembroDirecao

A partir do ID do professor, é possível defini-lo como um dos cargos atribuidos na base de dados, “projeto.addProfessorDirecao” inserindo tambem o ano em que se encontra

8. Eventos

a. Event

Nesta form pode-se criar um evento, e também ver se foi aceite ou não.

b. ListEvent

É possivel ver os eventos criados.

c. EventEdit

Por cada evento criado, o professor pode aceitar esse evento criado ou pode abdicar dele.

Em todos os forms faz-se a conexão com a base de dados, daí será necessário modificar em cada form a SqlConnection.

```
public partial class ListEvent : Form
{
    1 reference
    public ListEvent()
    {
        InitializeComponent();

    }

    private SqlConnection cn;

    3 references
    private SqlConnection getSGBDConnection()
    {
        return new SqlConnection("Data Source = tcp:mednat.ieeta.pt\\SQLSERVER, 8101; Initial Catalog = p7g2; uid = p7g2;" + "password = BaseDeDados123");

    }
    0 references
    private bool verifySGBDConnection()
    {
        if (cn == null)
            cn = getSGBDConnection();

        if (cn.State != ConnectionState.Open)
            cn.Open();

        return cn.State == ConnectionState.Open;
    }
```

6- Conclusões e análise crítica

Neste documento descrevemos as diversas fases de desenvolvimento do projeto desenvolvido na disciplina de Base de Dados

Este trabalho permitiu-nos pôr em prática os conhecimentos adquiridos durante as aulas de Bases de Dados, aplicados a um contexto específico do mundo real.