Testes Online

Bases de Dados P6 Grupo 2
Licenciatura em Engenharia Informática
Universidade de Aveiro

Autores

NMEC	Nome
97673	Pedro Daniel Lopes Duarte
98430	Paulo Pereira

Conteúdos

Análise de Requisitos

Este trabalho foi inspirado pelas funcionalidades de criar e responder a testes do sistema de e-learning da Universidade de Aveiro.

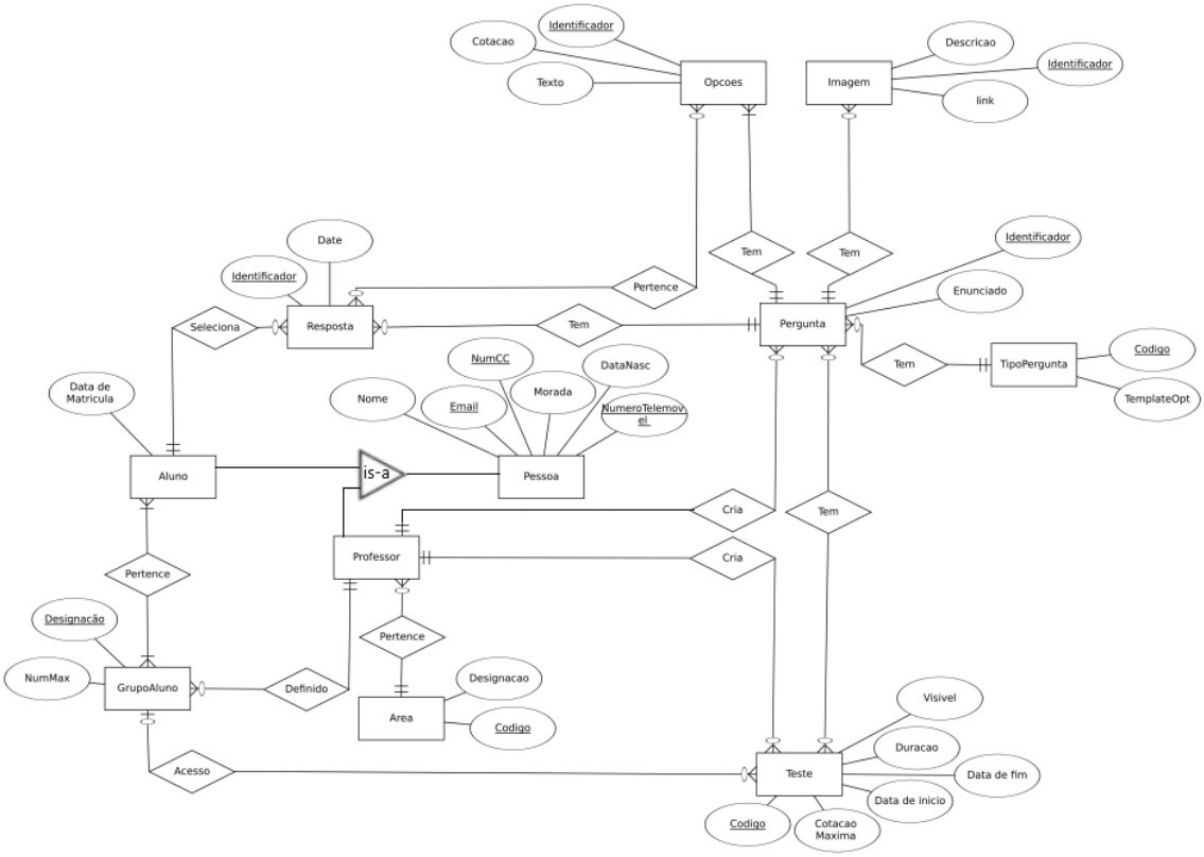
As duas entidades que iriam interagir com o sistema são **professores** e **alunos**, que podem ser ambos abstraídos numa entidade mais geral, **pessoa**, que pode guardar todos os dados que as tuas especificações têm em comum como nome, e-mail, nº de cartão de cidadão, morada, data de nascimento e nº de telemóvel. Um professor teria ainda uma área de trabalho e um aluno teria a sua data de matrícula. Esta especialização é sobreposta, nada impede que uma pessoa possa ser aluno e professor ao mesmo tempo.

Um **teste** é criado por um professor e tem um código próprio, uma cotação máxima (a escala em que o teste vai ser avaliado), data de ínicio, data de fim, opcionalmente uma duração, que faz com que um aluno possa apenas responder ao teste até determinado tempo depois de começar mesmo que ainda não tenha atingido a data de fim, e um estado de visibilidade que permita o professor responsável controlar se os alunos conseguem ver o teste ou não.

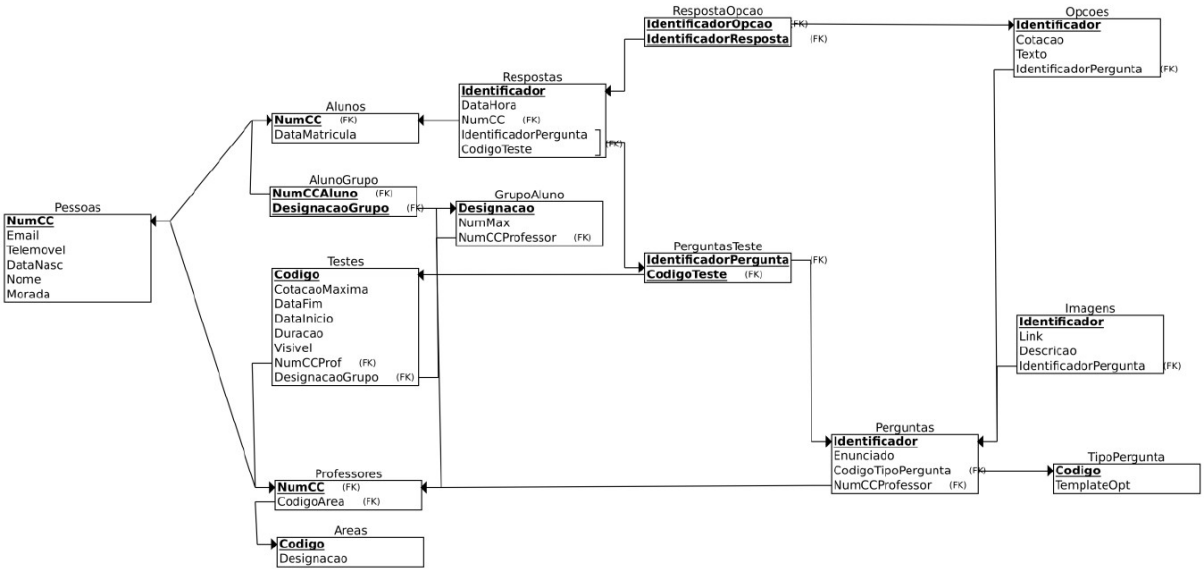
Um teste tem de ter **perguntas** que devem ter identificador, um tipo de pergunta associado (de resposta única ou múltipla por exemplo), enunciado e pode ainda ter associado **imagens**, que devem de ter além do link uma descrição, e opções, que têm o texto e a cotação associada, que pode ser positivo ou negativo caso a escolha dessa opção leve a penalização.

Um teste é disponibilizado a um **grupo** de alunos, que tem uma designação única e um nº máximo de participantes. Um aluno que seja parte do grupo com o qual o teste foi partilhado pode então submeter **respostas** ao mesmo, registando a data da resposta para deteção de situações de fraude. Uma resposta está associada a um único teste e a uma única pergunta mas pode ter mais que uma opção.

DER



Esquema Relacional



SQL DDL e DML

Foi escrito e executado na base de dados o DDL necessário para reproduzir o ER desenhado em SQL Server. Além da estrutura propriamente dita, a partir do DDL foi já possível definir **CHECK**s que garantem algumas regras básicas dos dados dentro de uma tabela, por exemplo, que dados podem ser **NULL** ou não e garantir que num teste a data de inicio deve ser depois da data de fim. Também do DDL foi possível definir já alguns triggers onde fazia sentido, associando as ações **SET NULL** e **CASCADE** a ações de delete e update. Para as tabelas em que não foi possível chegar a uma chave primária foi adicionado o campo *Identificador* (visivel no ER) que foi implementado utilizando a função **IDENTITY(1,1)** do T-SQL.

Foram também inserido com DML alguns exemplos de dados válidos.

O DDL e DML foram depois exportados usando as ferramentas do SQL Server Management Studio para os ficheiros *DDL.sql* e *DML.sql* respetivamente.

Normalização

Redundância de Dados

Não houve alterações, no desenho da base de dados não foi encontrado possível repetição de dados.

Redução dos NULLs

Valores NULL eram permitidos e possivelmente repetitivos na tabela Pessoas derivado de dados não obrigatórios, mas separar todos os camos em novas relações não pareceu ser adequado pelo que não houve alterações.

Junção de Relações com Base em PK e FK

Não houve alterações, as chaves estrangeiras criadas apontavam sempre para a chave primária da outra tabela.

1ª Forma Normal

Não houve alterações, o desenho da base de dados não supunha a existência de atributos compostos ou multivalor.

2ª Forma Normal

Não houve alterações, no desenho da base de dados todos os atributos tinham dependência total da chave primária, para fazer a identificação de qualquer campo de uma tabela era preciso a chave primária.

3ª Forma Normal

Não houve alterações, o desenho da base de dados não tinha colunas a depender de outras que não fossem parte da chave primária.

Forma Normal de Boyce Codd

Não houve alterações, o desenho da base de dados já mostrava que todos os valores de qualquer tabela apenas dependiam da chave primária.

Views

Foram criadas views mas não são usadas diretamente pelo programa propriamente dito, são seleções simples que servem apenas de auxilio a USPs e algumas foram criadas apenas para seguir o estado do sistema em fase de teste de forma a agilizar o debug. Segue-se um resumo de cada uma, o código encontra-se no ficheiro *Views.sql*.

projeto.**Random**

View que utiliza a função **NEWID** do SQL para permitir ordenação de perguntas e opções aleatóriamente.

projeto.**OpcoesInfo**

Lista as opções das perguntas tendo acesso á cotação de cada um e ao tipo de pergunta (de resposta única ou múltipla).

projeto.**CotacaoPergunta**

Usa a última view para calcular a melhor pontuação possível em cada pergunta.

projeto.**CotacaoTeste**

Usa a última view para calcular a melhor pontuação possível que um aluno pode ter em cada teste.

projeto.**RespostasOpcoes**

Listagem de todas as respostas dadas e as respetivas opções.

Índices

Foram analisadas as queries necessárias com vista a encontrar índices que se pudesse criar de forma a otimizar os tempos de resposta.

O primeiro caso analisado foi nas tabelas **Imagens** e **Opcoes** onde o interesse não estava em encontrar os elementos por si só mas sim associados á respetiva pergunta no campo **IdentificadorPergunta**. O número de inserções nestas tabelas podia ser enorme mas em príncipio os elementos vão sempre ser lidos muitas mais vezes e quando lidos em situação de teste o tempo é muito mais crítico do que quando o professor o está a criar, pelo que se decidiu criar o indíce **PerguntaIdx** em ambas as tabelas.

De seguida verificou-se que os procedimentos criados para fazer criação e atualização de dados relacionados com perguntas (tabelas **Perguntas**, **Imagens** e **Opcoes**) começavam sempre por verificar que a pessoa que estava a tentar fazer a tarefa era o professor que criou a pergunta, fazendo a procura na tabela pelo identificador da pergunta e pelo nº de cartão de cidadão do professor. Assim, criou-se um índice composto com este dois campos.

As restantes queries assentavam principalmente em chaves primárias pelo que o processo ficou por aqui. O código utilizado pode ser encontrado no ficheiro *Indices.sql*.

Triggers

Foram usados triggers **ON UPDATE** e **ON DELETE**, **CASCADE** e **SET NULL** como já referido, e não foi identificada a necessidade de criar outros para essas duas situações.

Já para a inserção de dados utilizou-se principalmente Stored Procedures, analisadas seguidamente, pois era mais cómodo na criação de uma row poder passar logo dados de outras tabelas e criar todos em conjunto, como é o caso de criar um aluno em que se pode passar logo os dados para a linha da tabela pessoa correspondente. No entanto surgiu a necessidade de garantir que na tabela AlunoGrupo o professor de um grupo nunca pudesse ser também um aluno no mesmo e que se respeitava a lotação máxima de alunos representada pela coluna NumMax. Criou-se então o trigger **CriarAlunoGrupo AFTER INSERT, UPDATE** que pode ser encontrada no ficheiro *Triggers.sql*.

Types

Foi criado um tipo, **ListaPerguntaIds**, de forma a permitir passar uma tabela com identificadores de perguntas como argumento de um stored procedure (CriarTeste, explicado na próxima página). O SQL para este tipo pode ser encontrado no ficheiro *Types.sql*.

Stored Procedures

Foram criados SPs para a maioria das inserções e atualizações de dados. A ideia era passar também o nº de cc da pessoa a tentar fazer a operação de forma a garantir que tinha acesso. Segue-se uma curta descrição de cada SP criado, o código pode ser encontrado no ficheiro *SP.sql*.

projeto.**createPessoa**

@NumCC INT, @Email VARCHAR(45), @Telemovel INT, @DataNasc DATE, @Nome VARCHAR(45), @Morada VARCHAR(250), @DataMatricula DATE, @CodigoArea CHAR(2)

Cria os registos na tabela Pessoa, Professor e Aluno, podendo passar @DataMatricula NULL ou @CodigoArea NULL para criar apenas professor ou aluno respetivamente. Garante-se aqui que a pessoa vai ser pelo menos um deles.

projeto.**createResposta**

@NumCC INT, @CodigoTeste VARCHAR(25), @IdentificadorPergunta INT, @IdentificadorOpcao INT

Cria uma resposta e a opção associada se necessário, controla todos os campos para garantir que a resposta pode ser inserida, validando por exemplo a validade da hora de inserção comparativamente ao inicio e fim do teste. Controla ainda a lógica de respostas a perguntas de resposta única e múltipla.

projeto.**CriarTeste**

@NumCCProfessor INT, @DesignacaoGrupo VARCHAR(25), @Codigo VARCHAR(25), @CotacaoMax INT, @DataInicio DATETIME, @DataFim DATETIME, @Duracao TIME, @Visivel BIT, @Perguntas projeto.ListaPerguntaIds READONLY

Cria um teste e as perguntas associadas. Usa-se transactions para garantir que ou tudo é inserido com sucesso ou nada é e usa-se cursores para iterar sobre a tabela @Perguntas.

projeto.**CriarPergunta**

@Enunciado VARCHAR(250), @CodigoTipoPergunta VARCHAR(15), @NumCCProfessor INT

Criado apenas para abstrair a criação de perguntas.

projeto.**ToggleTesteVisibilidade**

@CodigoTeste VARCHAR(25), @NumCCProfessor INT

Muda o estado de visibilidade de um teste entre visivel e escondido para os alunos.

Os próximos foram criados para abstrair a criação de perguntas e garantir que @NumCCProfessor pode realizar a operação ao ser o responsável pela pergunta.

projeto.**CriarPerguntaImagem**

@IdentificadorPergunta INT, @Descricao VARCHAR(50), @Link VARCHAR(250), @NumCCProfessor INT

projeto.**CriarPerguntaOpcao**

@IdentificadorPergunta INT, @Texto VARCHAR(250), @Cotacao INT, @NumCCProfessor INT

projeto.**AtualizarPergunta**

@IdentificadorPergunta INT, @NumCCProfessor INT, @Enunciado VARCHAR(250), @CodigoTipoPergunta VARCHAR(15)

projeto.**AtualizarPerguntaImagem**

@IdentificadorPergunta INT, @NumCCProfessor INT, @IdentificadorImagem INT, @Descricao VARCHAR(50), @Link VARCHAR(250)

projeto.**AtualizarPerguntaOpcao**

@IdentificadorPergunta INT, @NumCCProfessor INT, @IdentificadorOpcao INT, @Texto VARCHAR(250), @Cotacao INT

User Defined Functions

Foram criados UDFs em substituição de Views quando envolviam operações mais complexas. O código está no ficheiro *UDF.sql*.

projeto.**ListagemAlunos**

@DesignacaoGrupo VARCHAR(25)

Lista os alunos de um grupo.

projeto.**PautaTeste**

@CodigoTeste VARCHAR(25)

Lista de alunos e notas respetivas no teste.

projeto.**ListagemGrupos**

@NumCC INT

Lista de todos os grupos de uma pessoa, seja aluno ou professor do mesmo.

projeto.**TestesInfo**

@NumCC INT, @DesignacaoGrupo VARCHAR(25)

Lista de todos os testes de um grupo, devolve também se disponível a nota que o aluno obteve no mesmo.

projeto.**ProximaPergunta**

@NumCC INT, @CodigoTeste VARCHAR(25)

Retorna uma pergunta do teste aleatória que o aluno não tenha respondido.

projeto.**OpcoesDesordenadas**

@IdentificadorPergunta INT

Devolve a lista de opções de uma pergunta por ordem aleatória.

projeto.**ProfPerguntas**

@NumCC INT

Devolve a lista de perguntas do professor.

projeto.**PerguntaImagens** projeto.**PerguntaOpcoes**

@IdentificadorPergunta INT

Devolve a lista de imagens/opções de determinada pergunta.

projeto.**PessoaInfo**

@NumCC INT

Devolve informações de uma Pessoa, dados pessoais, de aluno e/ou professor.

projeto.**pontuacaoTeste**

@NumCC INT, @CodigoTeste VARCHAR(25)

Devolve a nota do aluno no teste.

Implementação

A aplicação foi implementada em C# com Windows Forms, a gestão da conexão com a base de dados é feita na classe **DBAccess** onde pode também ser encontrada e alterada a string de conexão.

Em geral no decorrer do projeto nunca houve preocupação com segurança de contas de utilizador por considerarmos estar fora do âmbito da cadeira e não acrescentar complexidade ao nível da base de dados, pelo que o login foi simplificado a inserir o número de cartão de cidadão. Após o login é criado um objeto **Pessoa** com todos os dados do utilizador carregados da base de dados.

Após login ou registo é apresentada a página com listagem de grupos e no caso do professor esta página vai permitir aceder ao pop up de gerir as suas perguntas no lado esquerdo, criar um grupo e adicionar um aluno a um grupo.

Ao abrir um grupo aparece a listagem de testes e a cotação se disponível. Se possível esta página permite ao aluno fazer um teste e ao professor criar um teste e gerir o seu estado de visibilidade.

As opções num teste podem ser selecionadas com CheckBox ou RadioButton dependendo do tipo de pergunta. Esse tipo é controlado diretamente pela base de dados a partir do campo **TemplateOpt** da tabela **TipoPergunta** usando C# Reflection.

Ao fazer um teste as escolhas são enviadas no momento em que o aluno seleciona uma opção, ficando sempre registado o estado mais atual da resposta, a ordem das perguntas aparecerem é aleatória.

As notas são apenas disponibilizadas quando o a data de fim do teste é atingida, mesmo que se tente modificar a aplicação a base de dados nunca vai entragar essa informação antes do estipulado.