universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Sistema de Gestão Escolar

Licenciatura em Engenharia Informática
Base de Dados
Docentes Carlos Costa (Teórica) e Joaquim Pinto (Prática)

Gonçalo Matos, 92972
Maria Rocha, 93320

Ano letivo 2019/2020

Índice

Introdução

Para o nosso projeto da unidade curricular de Base de Dados decidimos desenvolver um sistema de gestão escolar. Este seria utilizado num ambiente de ensino, como o nome sugere, para auxiliar a gestão do dia a dia, suportando não só a administração de todos os indivíduos e das suas funções, sejam eles alunos com aulas, professores com turmas ou não docentes com funções, como ainda integra um pequeno sistema de controlo de bibliotecas, do seu catálogo e respetivas requisições.

O seu desenvolvimento tem como objetivo a aplicação prática dos conhecimentos adquiridos nas aulas teóricas e práticas desta unidade curricular em toda a sua plenitude, desde o pensamento e desenho do modelo da base de dados à sua gestão e manipulação por sistemas de *software*.

Entregáveis

Em conjunto com este relatório entregamos vários ficheiros, descritos abaixo.

bdp4g2.sql

Script com o esquema da base de dados desenvolvida para o projeto.

CódigoSQL.zip

Pasta zipada com todo o código SQL desenvolvido no âmbito do projeto.

- **SQL_DDL.sql** Definição das entidades da base de dados
- **SQL_DML.sql** Inserção de dados nas relações criadas
- **/Triggers** Todos os *triggers* criados
- **/Functions** Todas as funções criadas
- **/StoredProcedures** Todos os *stored procedures* criados
- **/Views** Todas as *views* criadas

Interface.zip

Pasta zipada com as interface que criámos para administrar a base de dados.

Para alterar o utilizador utilizado no login, deve ser modificada a linha 32 do ficheiro Home.cs.

SistemaGestaoEscolar.pptx

Slides utilizados na apresentação na aula prática do dia 9 de junho

https://bit.ly/bdp4g2v1

Vídeo da interface apresentado na aula prática do dia 9 de junho

https://bit.ly/bdp4g2v2

Vídeo da interface terminada e entregue no dia 12 de junho

Alterações realizadas depois da apresentação

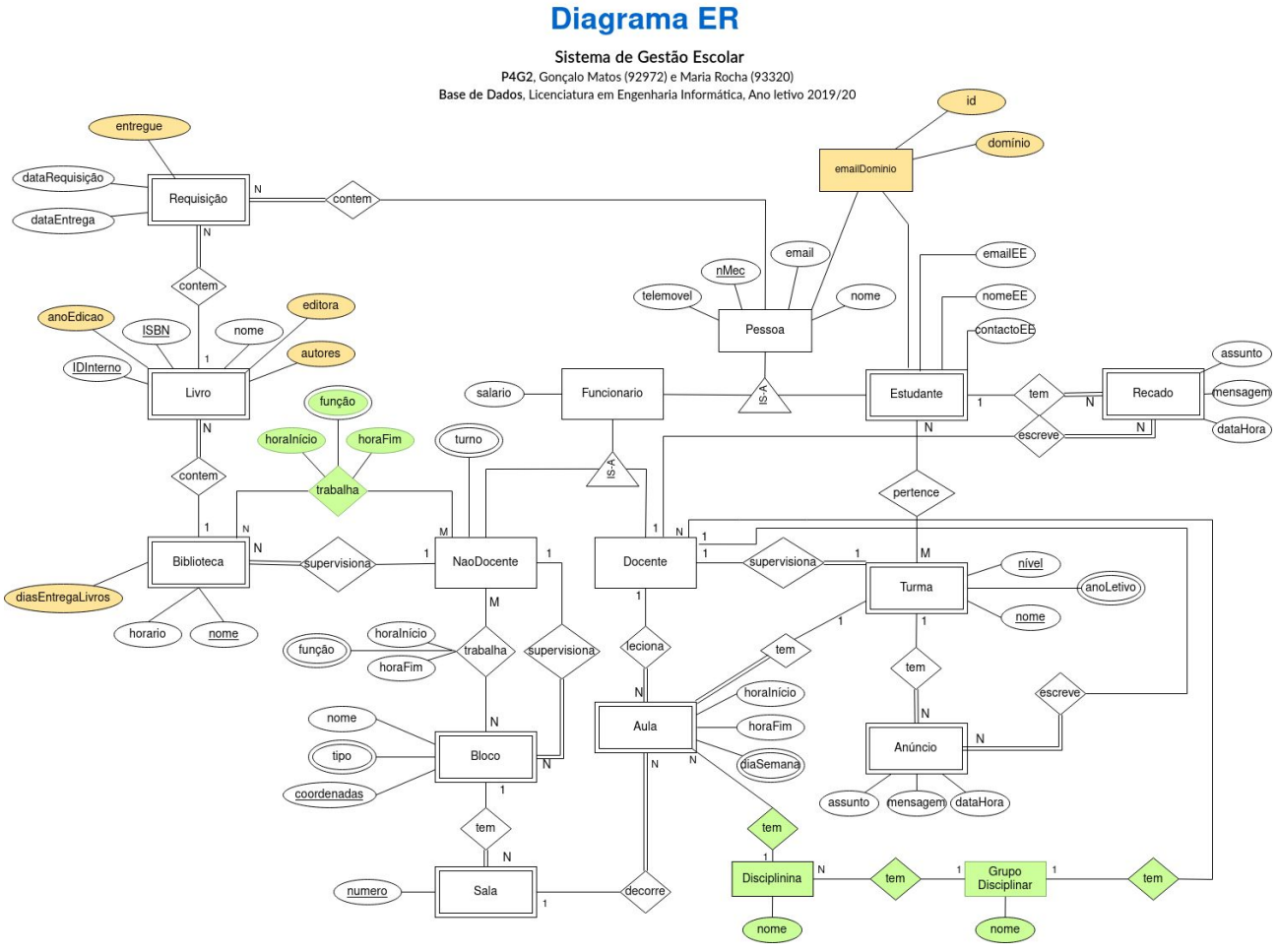
Depois da apresentação do dia 9 de junho, implementámos na nossa base de dados a maioria dos *triggers*, *stored procedures*, *views* e funções. Para demonstrar a sua utilização criámos um segundo vídeo disponível em https://bit.ly/bdp4g2v2.

Integrámos ainda as interfaces da Biblioteca e melhorámos as dos Estudantes e das Turmas que não foram mostradas nesse dia por ainda não estarem finalizadas.

Desenho da base de dados

Desde o início do desenvolvimento do nosso projeto que o desenho da base de dados que implementámos tem vindo a sofrer alterações no sentido de a tornar mais robusta, menos suscetível a erros e uma melhor representação da realidade a que se propõe.

Na próxima figura apresentamos o esquema que desenhámos para este planeamento, indicando num sublinhado verde as alterações realizadas por sugestão do docente Joaquim Pinto na primeira apresentação do trabalho e a laranja as que foram introduzidas no desenvolvimento, enquanto criávamos as entidades na base de dados, por nos termos apercebido de atributos adicionais necessários ou alterações na topologia dos existentes.



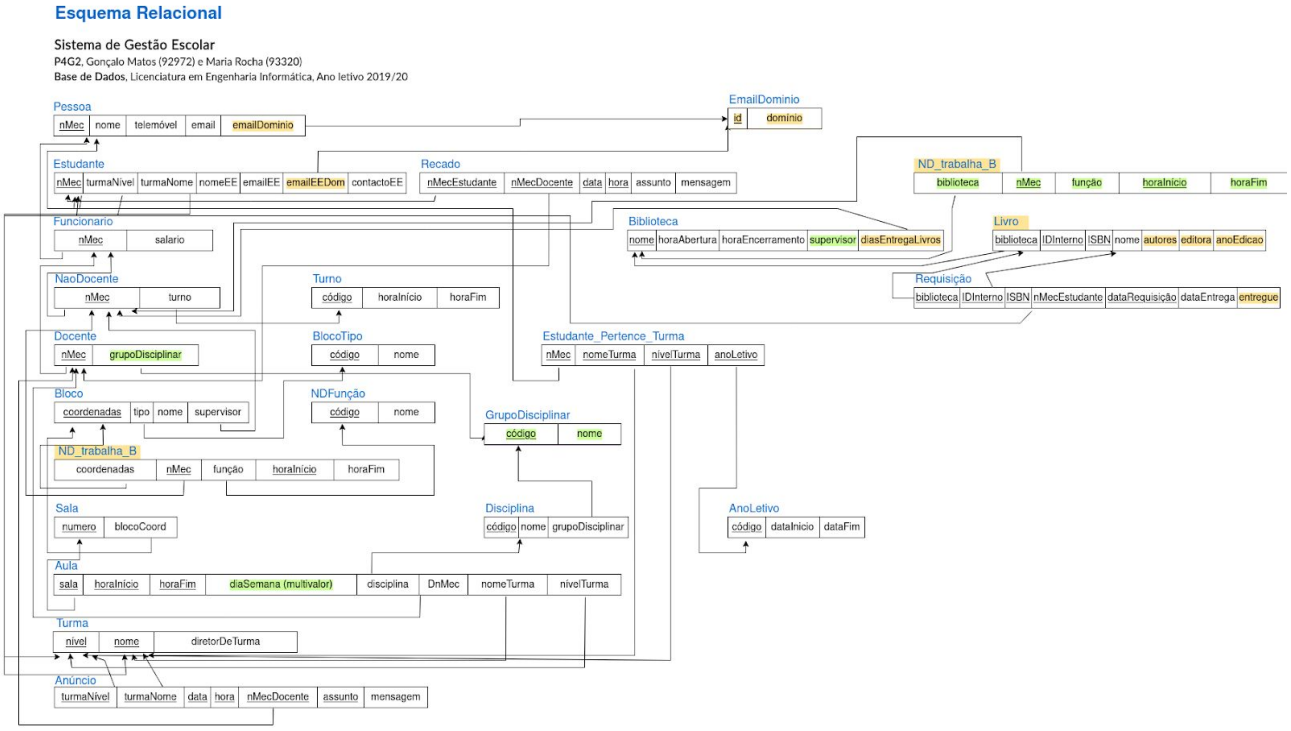
As alterações introduzidas no início do desenvolvimento foram:

- Em todas as entidades que armazenam um endereço de *email* (**Pessoa** e **Estudante**), separámos o domínio do prefixo, sendo o primeiro armazenado na

relação **emailDominio**, uma vez que é geralmente comum a vários endereços e pode ocupar até 255 caracteres de acordo com as normas;

- Na **Biblioteca** decidimos acrescentar o número de dias de entrega dos livros após a requisição, uma vez que este geralmente varia de biblioteca para biblioteca;
- No **Livro** alguns atributos com informação adicional;
- Na **Requisição** um atributo que permita saber se o livro requisitado foi ou não entregue, o que nos permite a sinalização dos livros com entrega em atraso.

Nos mesmos moldes do gráfico anterior, apresentamos agora o esquema entidade-relação.

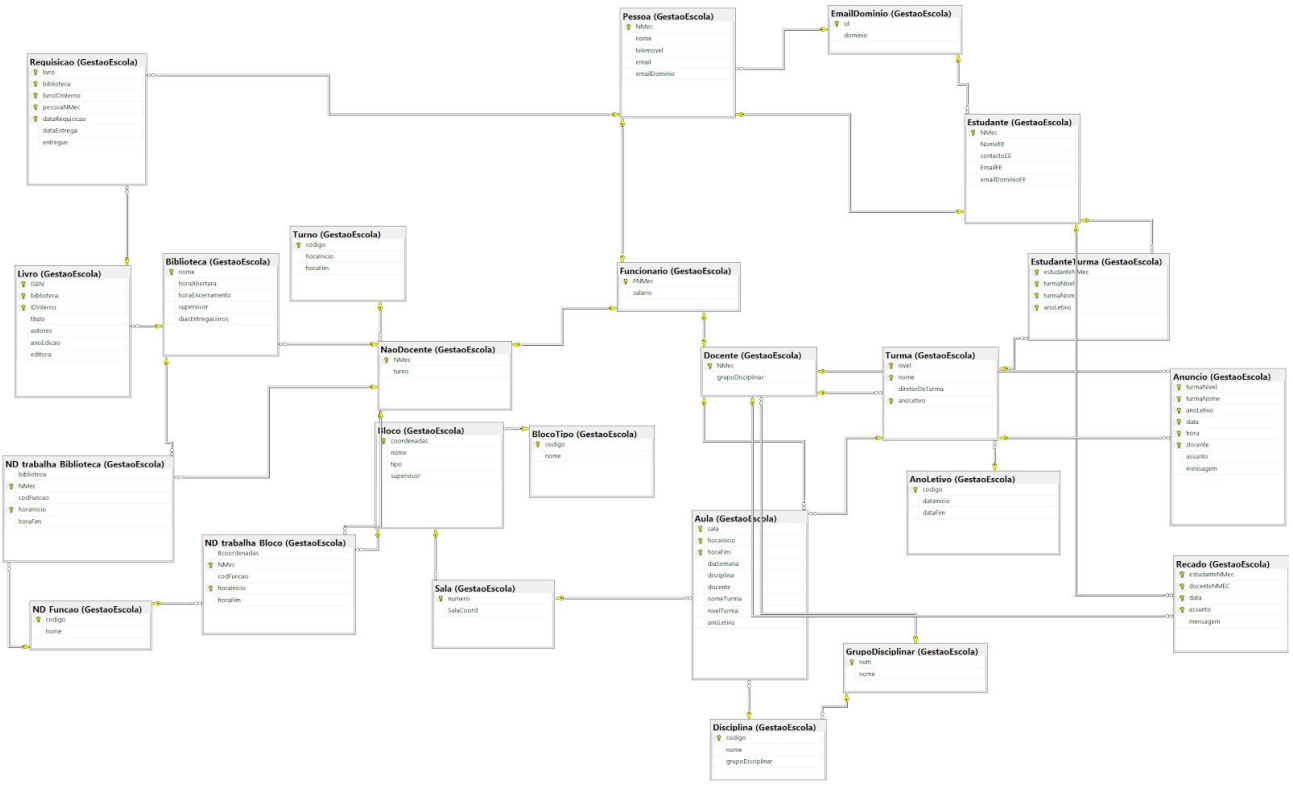


Para além das modificações já explicadas no âmbito do diagrama anterior, são agora visíveis alterações realizadas no âmbito das restrições de integridade, que consistiram em:

- Nas relações **ND_trabalha_Bloco** e **ND_trabalha_Biblioteca**, que armazenam os horários de trabalho dos funcionários, respetivamente, nos blocos e bibliotecas do estabelecimento de ensino, decidimos alterar a *primary key* de coordenadas/biblioteca+nmec para nmec+horaInicio, pois um funcionário pode trabalhar mais do que uma vez no mesmo bloco/biblioteca em janelas horárias distintas, mas nunca na mesma janela horária em dois locais diferentes;

- Na relação **Livro**, removemos o atributo referência inicialmente previsto para identificar cada livro e introduzimos o IDInterno, que a par do ISBN e da biblioteca compõem agora a *primary key* da tabela.

Por fim apresentamos o diagrama da base de dados já implementada gerado pelo nosso SGBD.



Interfaces

Para desenvolver as interfaces do nosso projeto recorremos aos Windows Forms da plataforma .NET, tendo por isso desenvolvido todo o código em C#.

De forma a melhorar a experiência de visualização e manipulação de grandes quantidades de dados, procurámos desde o início permitir ao utilizador fazer pesquisas, filtragens e ordenação de acordo com os vários atributos dos tuplos nas tabelas. No entanto, devido às dificuldades sentidas da integração de todas estas funcionalidades nas *ListView*, acabámos por descobrir uma alternativa que as encapsula, fornecendo ainda todas as características adicionais que procurávamos e ainda a categorização de acordo com o atributo: as *ObjectListView*.

Devido à sua integração, é possível em todas as tabelas ordenar por um atributo ao carregar no cabeçalho respetivo, o que vai fazer com que os dados sejam também agrupados por este campo. Nas interfaces principais é disponibilizado no canto superior direito um campo de pesquisa por atributo.

Na manipulação dos dados, os erros são sinalizados através de janelas *MessageBox*, sendo a maioria identificado através de uma mensagem que especifica o que correu mal. No entanto e para efeitos de *debugging*, é também apresentado abaixo desta mensagem a totalidade da mensagem de erro devolvida pela base de dados. Em produção esta seria excluída.

Neste capítulo vamos apresentar as várias interfaces que desenvolvemos, cuja totalidade permite fazer uma manipulação de todas as relações da base de dados que desenvolvemos. Em cada uma iremos fazer uma breve descrição das funcionalidades, mas o destaque irá para a interação da interface com a base de dados e para o código SQL que desenvolvemos de forma a permiti-la.

Docentes e Não Docentes

Estas interfaces, acessíveis através do menu principal, permitem a gestão dos docentes/não docentes do estabelecimento de ensino, nomeadamente da sua inserção, edição e eliminação.

Views

Tanto os dados de cada docente/não docente resultam da junção da informação que consta nas tabelas **Pessoa**, **Funcionário** e **Docente/NaoDocente** e para facilitar a sua consulta por parte das interfaces que iria implicar a execução de uma *query* algo extensa e complexa, decidimos criar uma **view** para cada uma destas entidades, às quais demos o nome, respetivamente de **vw_Docentes** e **vw_NaoDocentes**. Para além da junção, fazem ainda a filtragem dos funcionários "apagados" (mais informações sobre esta particularidade na secção dos *triggers*).

Stored Procedures

Devido à dispersão dos dados destas entidades em 3 tabelas, decidimos criar *stored procedures* para a sua inserção e edição, que recebem todos os atributos, validam e distribuem pelas relações, abstraindo toda a complexidade associada a esta tarefa do código da interface: **pr_Docentes** e **pr_NaoDocentes**.

Libertam ainda a interface de identificar o ID do domínio do *email*, que como explicado no capítulo anterior, é armazenado numa relação distinta. De forma a sinalizar ao utilizador diferentes tipos de erro retornam um número inteiro (1 para sucesso e negativo entre -1 e -4 para diferentes situações de erro).

Triggers

De forma a cumprir com as obrigações legais de proteção dos dados dos cidadãos, integrámos a possibilidade de um profissional ser esquecido no nosso sistema. No entanto, na impossibilidade de apagar todos os seus registos associados por motivos históricos e administrativos, criámos *triggers* que previnem a eliminação dos elementos das das tabelas Docente e NaoDocente e em vez disso limitam-se a "apagar" (mudar o valor para NULL) os atributos que armazenam informações pessoais, nomeadamente o nome, *email* e telemóvel: **tr_DocentesDelete** e **tr_NaoDocentesDelete**.

No caso dos não docentes é ainda feita a verificação se o funcionário a "apagar" é supervisor de algum bloco ou biblioteca, sendo a operação interrompida neste caso.

Gerir funções (Não Docentes)

Esta interface, acessível através da interface dos não docentes, permite a gestão das funções dos funcionários, que se caracterizam por uma janela horária de trabalho num determinado local com uma determinada função.

Destaca-se a complexidade associada à gestão simultânea de duas relações nesta interface, nomeadamente as **ND_trabalha_Bloco** e **ND_trabalha_Biblioteca**. No entanto, o utilizador é abstraído desta distinção, sendo todas as funções apresentadas em simultâneo e sem qualquer diferença.

Triggers

Para garantir a integridade da nossa base de dados na gestão das duas relações, como não pode ser possível um funcionário ter duas funções atribuídas na mesma janela horária (em ambas as relações) ou funções atribuídas fora do seu turno laboral (o que relaciona com a relação **NaoDocente**) e dada a complexidade destas validações, criámos um *trigger* para a inserção de dados em cada uma das tabelas: **tr_NDTrabalhaBlocoInsert** e **tr_NDTrabalhaBibliotecaInsert**.

Funções + Cursores

Para auxiliar as validações necessárias e dada a sua semelhança em ambos os *triggers*, para evitarmos a duplicação de código criámos uma função chamada **GestaoEscola.NDFuncaoScheduleValid**. Esta recebe o número mecanográfico do funcionário e as horas de início e fim do horário proposto e faz as seguintes verificações:

- Consulta o turno do funcionário na tabela NaoDocente e verifica se o horário se encontra dentro do primeiro;
- Consulta os horários atribuídos ao funcionário em ambas as relações relativas ao trabalho em blocos e bibliotecas e verifica se existe colisão em algum deste com a janela horária proposta.

Para a última tarefa fizemos uso de um **cursor**.

A função retorna um número inteiro que permite ao *trigger* saber se o horário é válido e caso não seja o porquê (1 para válido e entre -2 e 0 para vários tipos de erros).

Consultar horário (Não Docentes)

Esta interface, acessível através da interface dos não docentes, é estática e limita-se a apresentar o horário do funcionário em blocos de 2 horas, apresentando em cada um o horário das funções atribuídas, o local e o tipo de trabalho a desempenhar.

Biblioteca Escolar

Esta interface disponível no menu principal permite ao utilizador gerir todas as tabelas relacionadas com a rede de bibliotecas do estabelecimento de ensino, nomeadamente da sua criação, eliminação e manutenção, tal como dos respetivos catálogos e requisições.

Views

Para facilitar a gestão do catálogo quisemos apresentar ao utilizador nesta tabela o estado de cada livro, o que implica uma relação das tabelas Livro e Requisicao. Para tal, criámos uma *view* que aos atributos de cada livro acrescenta um denominado estado e que consiste num inteiro a 0 para disponível ou -1 para emprestado: **vw_LivrosComEstadoSimples**

Devido à impossibilidade de fazer o agrupamento de atributos do tipo TEXT, tivémos de criar uma segunda *view* que agrega à primeira o atributo autores: **vw_LivrosComEstadoCompletos**

Triggers + Cursores

Para garantir a integridade da relação Requisicao é necessário validar que ao registar uma requisição (seja atualização ou inserção) o livro não se encontra requisitado: **tr_RequisicaInsertUpdate**.

Este faz uso de um **cursor** para percorrer todas as requisições do livro para o qual se quer registar uma nova requisição e verifica se todas estas foram entregues.

Para permitir a correção de erros do utilizador, mas de forma a garantir o histórico das operações, decidimos ainda restringir a eliminação de uma requisição ao dia em que foi criada. Para esta validação criámos o *trigger* **tr_RequisicaoDelete**.

Interfaces do menu secundário

Algumas das tabelas que criamos servem apenas de suporte ao sistema e por isso a manipulação iria incidir na instalação do sistema, sendo depois apenas referenciados por outras tabelas e apenas alterados pontualmente. Por isto decidimos criar um menu secundário ao qual chamámos *backoffice*, que em *deploy* seria incluído apenas na interface do administrador do sistema no estabelecimento de ensino. Pelas particularidades já mencionadas e ainda devido ao menor volume de dados que cada uma comporta, decidimos não implementar a pesquisa nestas interfaces. Estas apresentam-se ainda como quase réplicas umas das outras, apenas com pequenas validações distintas, motivo pelo qual abordamos todas neste único capítulo.

As interfaces a que nos referimos são: Turnos, Funções ND, Blocos, Tipos Bloco e Grupos Disciplinares, que permitem a manipulação direta, respetivamente, das relações **Turno**, **ND_Funcao**, **Bloco**, **BlocoTipo** e **GrupoDisciplinar**.

Stored Procedures

Para testar a implementação de *stored procedures* decidimos criar para cada uma das relações um *stored procedure* bastante simples para o processo de eliminação dos tuplos, que se limita a tentar realizá-la e a "captar" uma exceção no caso da sua ocorrência, libertando o sistema de fazer esta gestão de erros. Os SP criados são **pr_TurnosDELETE**, **pr_NDFuncaoDELETE**, **pr_BlocosDELETE**, **pr_BlocoTiposDELETE** e **pr_GruposDisciplinaresDELETE**.

Todos estes retornam um inteiro que a 1 significa operação bem sucedida e a -1 erro, ao qual se soma o OUTPUT de uma mensagem de *feedback*.

Segurança

Para garantir a segurança da base de dados e prevenir a execução de ***SQL Injection*** por parte do utilizador, recorremos à parametrização de todas as instâncias criadas da classe **SqlCommand**.

Notas finais

Devido à facilidade de manipulação dos dados introduzida pela utilização da *ObjectListView* acabámos por nunca realizar pesquisas na base de dados. Assim, em cada interface limitamo-nos a carregar os dados “em bruto” sem qualquer filtragem e por este motivo não implementámos nenhum ***index***.