*1º Relatório*

Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informática

*Bases de Dados, 1º Semestre*

Base de Dados de uma

Biblioteca



**Trabalho realizado por:**

* **Ivo Correia nº 2008110814**
* **João Barbosa nº 2008111830**

Coimbra, 4 de Janeiro de 2011

# Índice

Contents

[Índice 1](#_Toc281766857)

[Introdução 2](#_Toc281766858)

[Definição do Produto 3](#_Toc281766859)

[Visão do Produto 3](#_Toc281766860)

[Relevância e Interesse 4](#_Toc281766861)

[Diagrama de Gant 5](#_Toc281766862)

[Especificação Funcional 6](#_Toc281766863)

[Protótipo Rápido 6](#_Toc281766864)

[Plano de Desenvolvimento do Código 10](#_Toc281766865)

[Tempo de execução: 15 de Novembro a 22 de Novembro 10](#_Toc281766866)

[Tempo de execução: 22 de Novembro a 6 de Dezembro 10](#_Toc281766867)

[Tempo de execução: 6 de Dezembro a 20 de Dezembro 10](#_Toc281766868)

[Tempo de execução: 20 de Dezembro a 31 de Dezembro 10](#_Toc281766869)

[Especificação Técnica 11](#_Toc281766870)

[Classes Java 11](#_Toc281766871)

[Book 11](#_Toc281766872)

[Publisher 11](#_Toc281766873)

[Person 11](#_Toc281766874)

[SQL 11](#_Toc281766875)

[ER 11](#_Toc281766876)

[Tabelas 11](#_Toc281766877)

[Sequências 12](#_Toc281766878)

[Utilizadores, Perfis e Sinónimos 12](#_Toc281766879)

[Programas do Servidor 14](#_Toc281766880)

[Conclusão 21](#_Toc281766881)

[Bibliografia 22](#_Toc281766882)

# Introdução

Depois de um semestre de trabalho, chegamos finalmente ao fim do projecto da disciplina de ‘Base de Dados’. O nosso produto foi sofrendo diversas alterações até chegar à sua versão final, muitas delas que não estavam inicialmente nos nossos planos. Este último relatório irá servir para estabelecer essas diferenças, detalhar todas as etapas de evolução do nosso projecto e por fim, expor cada componente e funcionamento da nossa aplicação, aplicação essa que simula a gestão de uma biblioteca.

A parte inicial, ‘Definição de Produto’ e ‘Especificação Funcional’, não são mais do que a agregação dos dois relatórios anteriores, o primeiro onde, tal como o nome indica, definimos no que consistiria o nosso projecto; o segundo faz um esboço de como será realizado o mesmo projecto, estabelecendo metas e prazos e já apresentando alguns planos para a interface gráfica.

Deste modo, o capítulo mais importante deste relatório acaba por ser a ‘Especificação Técnica’, já que é o único capítulo exclusivo deste relatório e no qual fazemos um apanhado de todos os detalhes da aplicação. Consequentemente, o capítulo é iniciado com uma análise das classes usadas

# Definição do Produto

## Visão do Produto

Tal como foi dito na introdução, vamos construir uma plataforma capaz de comportar a gestão de uma base de dados de suporte ao funcionamento de uma biblioteca.

Consequentemente, necessitamos de criar diversas entidades que através das suas relações, irão formar a base de dados:

* Publicação (que iria incluir livros, revistas, jornais, *papers,* documentos de opinião e artigos digitais);
* Funcionário;
* Leitores;
* Editoras;
* Prateleiras/Corredores;
* Emprestimo
* Autenticação

A nossa aplicação vai permitir registar novos funcionários, leitores e aquisições (englobando os livros, revistas, jornais e artigos).

Para além das entidades já mencionadas, haverá a existência de um administrador (super utilizador) que terá privilégios de acesso a todos os campos da base de dados e modificá-los consoante a sua necessidade.

Por seu lado, os funcionários apesar de poderem ter acesso a todos os registos, ser-lhes-á impedida a execução de acções relativas a si mesmas (i.e. modificar os registos dos funcionários já existentes ou adicionar novos registos de funcionários). Será permitido ainda aos funcionários pesquisar informações relativas aos seus colegas, nome ou identificação, para dar apenas alguns exemplos.

Só leitores registados no sistema poderão usufruir dos serviços de empréstimo da biblioteca. Deste modo, se um dado utilizador pretender levar um documento e não estiver na base de dados, o funcionário que está a despachar o pedido, terá de inserir diversas informações, como nome, morada, contacto telefónico e endereço electrónico, e então sim, efectuar a requisição do dado documento.

Aquando da requisição, é necessário primeiro verificar se é possível levantar o documento desejado e se o utilizador não está em falta com alguma aquisição que tenha ultrapassado a sua data de entrega ou se não possui já o número de documentos em simultâneo que este pode alugar. Se esta condição for satisfeita, registaremos a data de devolução, o funcionário que atendeu o utilizador e por fim o documento é levado.

Poderemos também efectuar pesquisas sobre os documentos segundo os vários parâmetros. Todos os documentos terão como atributos comuns um título; um número único identificativo do mesmo no sistema; uma categoria; um resumo do seu conteúdo; autores; data da publicação; número de exemplares; o número de páginas. Se necessário, podem ser pesquisados atributos como editora contudo, este não é obrigatório.

Os documentos serão divididos por prateleiras com uma determinada capacidade. As prateleiras são criadas conforme são necessárias. Contudo existe uma restrição. Os livros com vários duplicados têm de ficar sempre na mesma prateleira. Como tal, sempe que são adicionados e não existem prateleiras com o tamanho suficiente, é criada uma nova com uma capacidade adequada.

Novas editoras podem ser referenciadas no sistema, sendo que, para registar uma nova editora, o funcinário nada terá de fazer. Ao adicionar um novo livro que possua uma editora desconhecida ao sistema, esta será automaticamente adicionada.

Em relação a acessos concorrentes, havendo vários funcionários que poderão registar novos livros no sistema, proceder a empréstimos, temos de garantir que não há conflitos provocados por requisições simultâneas. Por seu lado, sendo permitido ao administrador efectuar qualquer operação, teremos de garantir que as interferências por parte este, não vai inviabilizar a veracidade dos dados obtidos pelos funcionários.

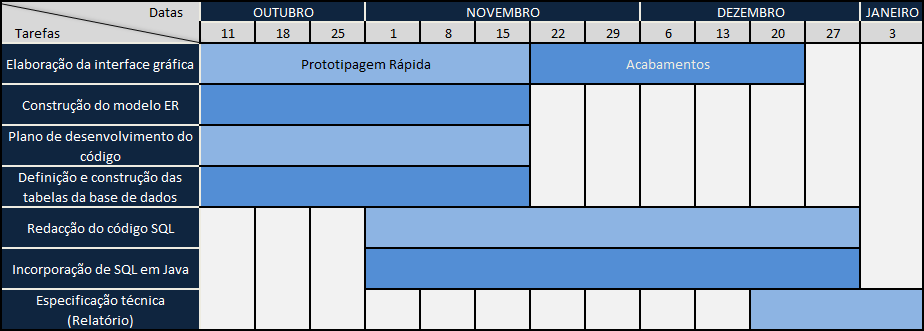
## Relevância e Interesse

Como já foi dito brevemente na introdução, uma base de dados é fulcral para o bom funcionamento de uma instituição como uma biblioteca e consequentemente, escolhemos este cenário como base para todo o projecto não só porque é um local que tradicionalmente está sempre ligado a um grande repositório de informação, o que nos dá liberdade suficiente não só para tornarmos o produto final complexo o suficiente, como não precisaremos de divagar para fora do contexto real para alcançarmos tal objectivo.

Por outro lado, dá-nos um campo viável e suficientemente extenso para enriquecermos os nossos conhecimentos sobre a disciplina da Base de Dados, como a aprendizagem de SQL, construção de modelos de entidade-relacionamento e elaboração de um projecto de média dimensão.

## Diagrama de Gant

Em baixo, encontra-se o diagrama de Gant construído com base nas datas previstas para a conclusão de cada meta do projecto:



1. ***Diagrama de Gant para o projecto***

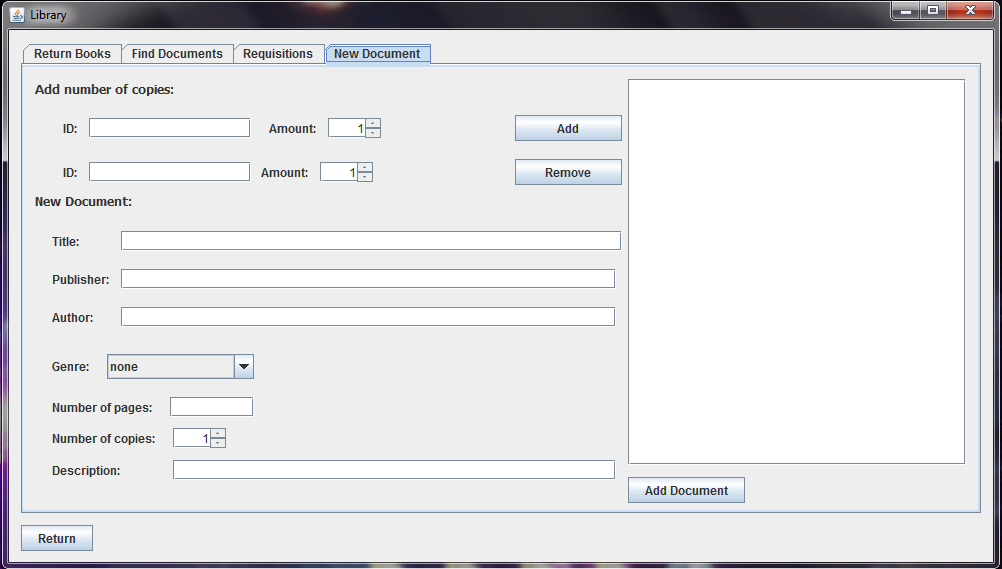
Este foi o diagrama inicialmente proposto contudo, é sempre bastante difícil de um cumprir rigorosamente. Ainda assim, os prazos foram compridos sempre com um desfazamento de não mais de uma semana em algumas situações pontuais.

# Especificação Funcional

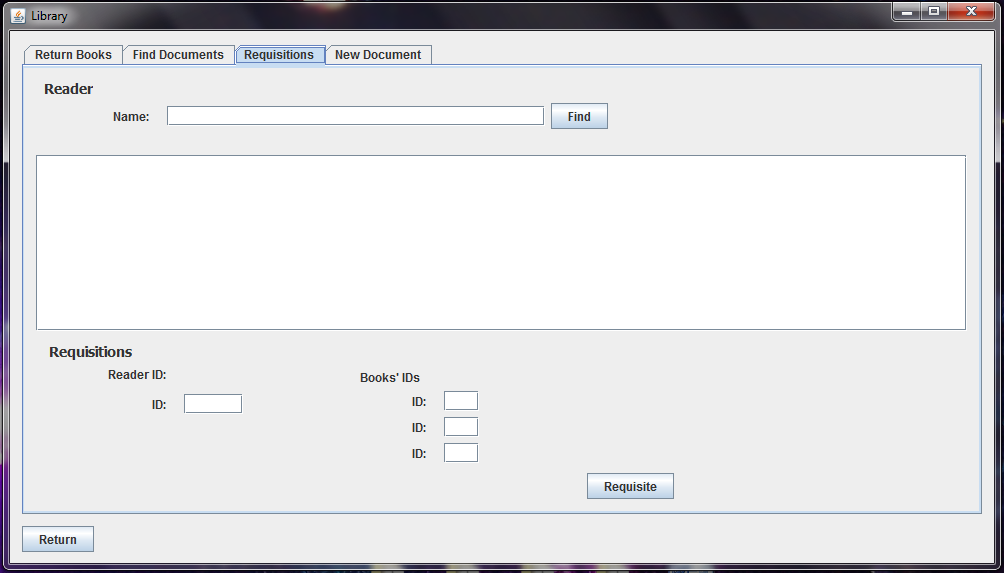
## Protótipo Rápido

Em primeiro lugar, convém esclarecer que apenas criámos uma única aplicação. Num sistema real, naturalmente, teríamos de ter duas aplicações, uma para ser utilizada pelos funcionários e outra para os clientes. Contudo, como os clientes apenas poderão efectuar pesquisas, decidimos eliminar a aplicação destinada ao cliente, pois apenas iríamos estar a replicar trabalho, já que também daremos a possibilidade de pesquisas de publicações para a aplicação dos funcionários.

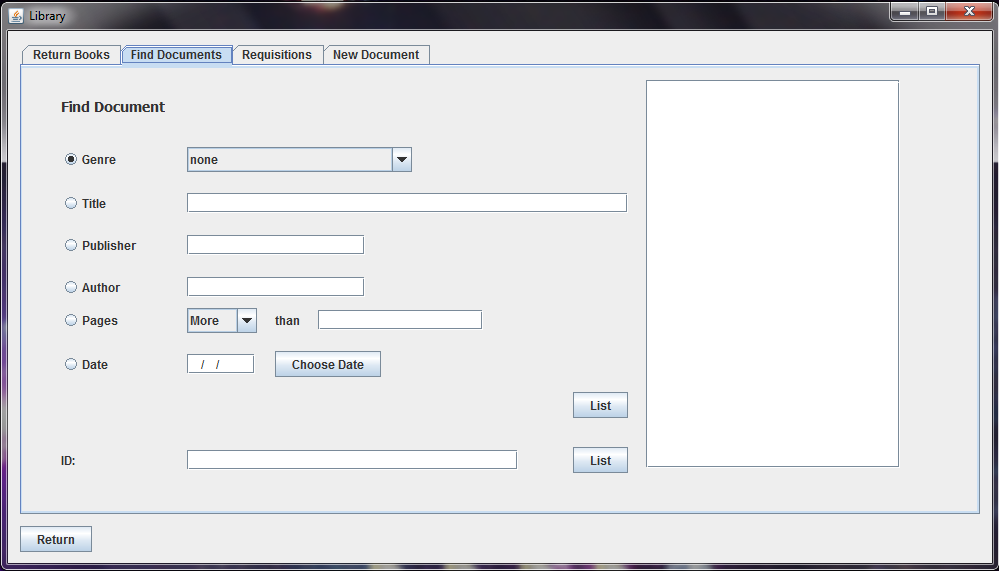
Assim, apresentámos um primeiro menu onde são feitos os registos de publicações que dão entrada na biblioteca. Depois, temos três menu semelhantes, que se destinam as pesquisas sobre publicações, aluguer de documentos e entrega dos mesmos. Por fim, são apresentados dois protótipos relativos à gestão de funcionários e clientes, onde estes podem ser adicionados, removidos e ser consultada informação relativa aos mesmos. É importante voltar a referir que as opções relativas ao despedimento de funcionários e alteração dos dados dos empregados, só estão disponíveis para o administrador. Todas as restantes são acessíveis por também pelos operadores.



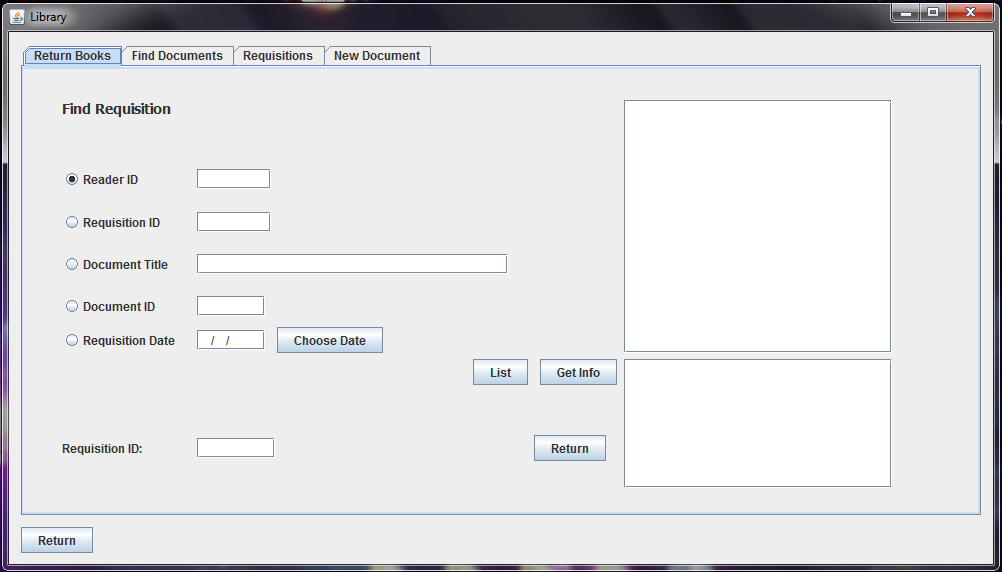
Menu 1 - Registo no sistema da biblioteca de uma nova publicação.



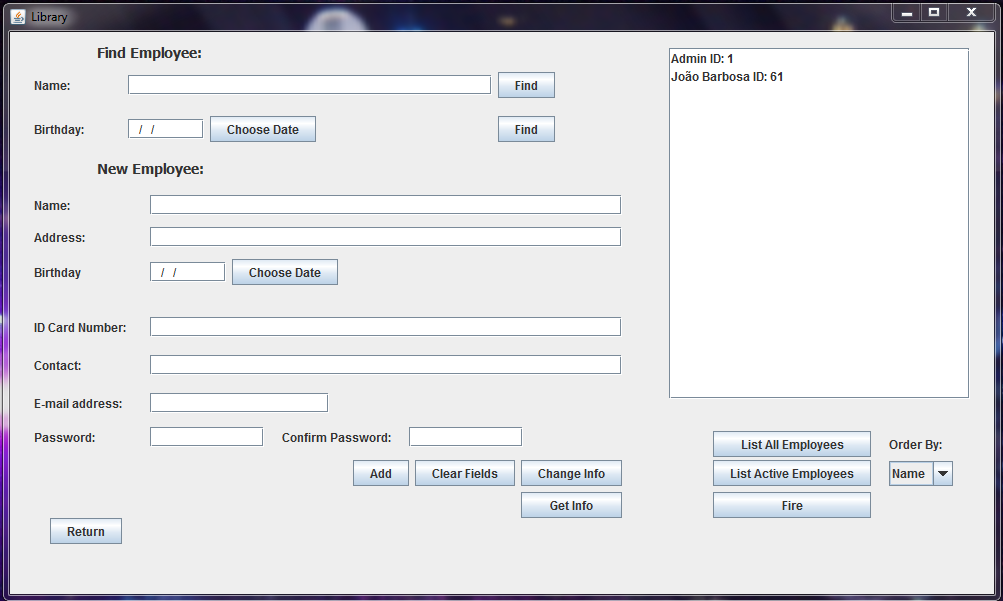
Menu 2 - Efectuar Requesição



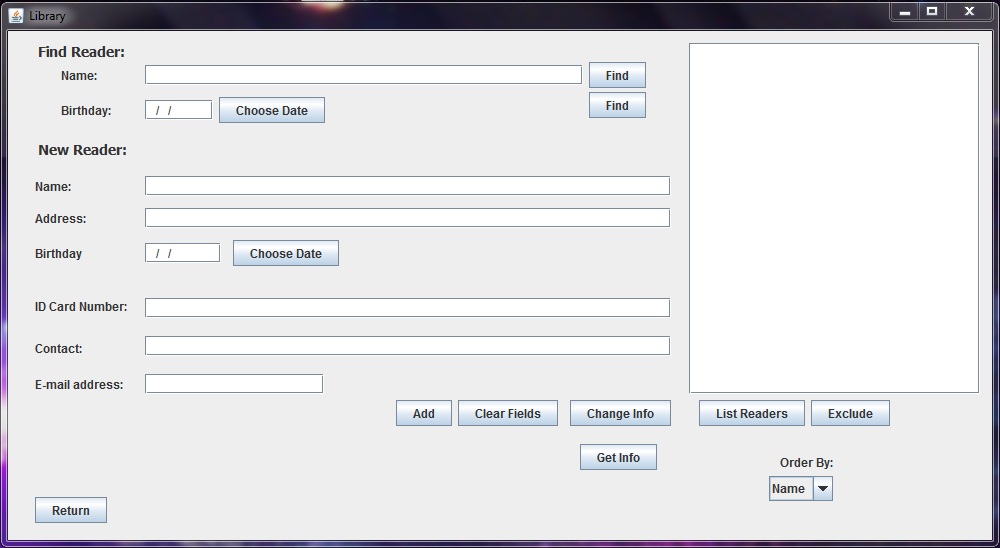
Menu 3 - Procurar Documento



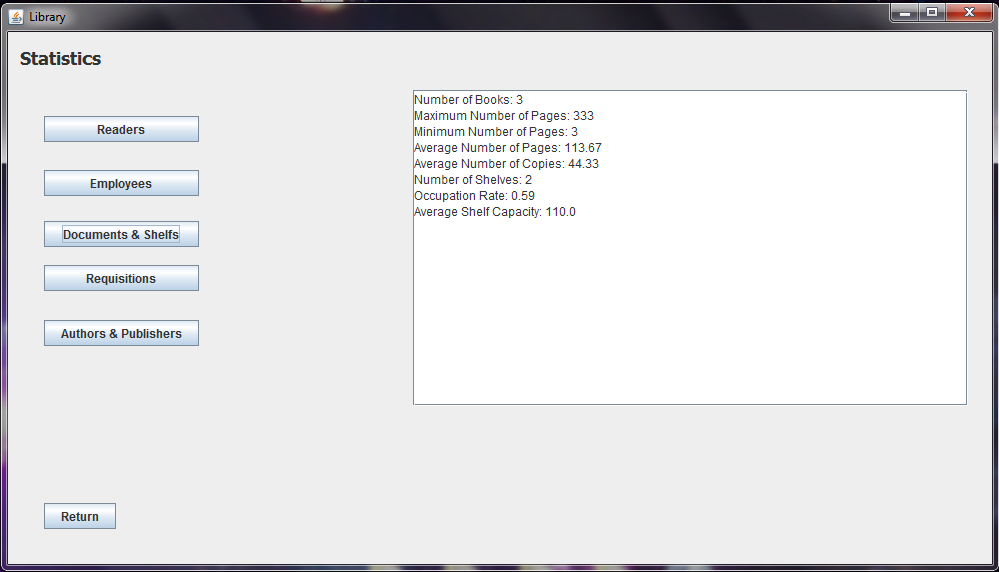
Menu 4 - Devolver Requesição



Menu 5 - Procurar e Inserir Empregados



Menu 6 - Procurar e Inserir Leitores



Menu 7 – Estatísticas

## Plano de Desenvolvimento do Código

### Tempo de execução: 15 de Novembro a 22 de Novembro

O primeiro passo, uma vez construído o diagrama ER, foi a criação das tabelas em SQL que deram suporte à nossa base de dados, seguindo as entidades definidas no diagrama.

Com as tabelas e restrições definidas, procedemos à inserção de alguns dados de modo a podermos verificar que não estão a ser feitos nenhuns erros e consequentemente, podermos validar a implementação.

### Tempo de execução: 22 de Novembro a 6 de Dezembro

Em paralelo e à medida que fomos dando forma à nossa interface gráfica, produzimos o código SQL que permitiu realizar pesquisas, as *queries*, sobre a base de dados, o que se foi estendendo pelas semanas seguintes.

Nestas duas primeiras semanas, concluímos os menus de pesquisas e registo tanto de funcionários como de leitores.

Assim, a 6 de Dezembro, esperámos que fosse possível validar *logins* de funcionários e permitir que estes pudessem registar novos clientes, assim como o administrador registaria funcionários. Contudo devido a atrasos, as permissões dos funcionários e *logins* foram adiadas para a próxima etapa.

### Tempo de execução: 6 de Dezembro a 20 de Dezembro

Para esta fase, trabalhámos no intuito de concluir todas as operações relativas às publicações e seu armazenamento, o que implica a interacção da entidade Publicação com as entidades *Stock* e prateleira e a conclusão da validação dos *logins.*

Assim, no fim deste período, tínhamos dois blocos isolados. De um lado, os clientes a interagir com os funcionários, enquanto do outro teremos as publicações a serem colocadas nos seus devidos sítios.

### Tempo de execução: 20 de Dezembro a 31 de Dezembro

Com os dois blocos definimos e operacionais, partiremos para o estabelecimento da ligação entre os mesmos. Consequentemente, planeámos estabelecer nesta semana as funcionalidades relativas as novas requisições e detecção de clientes faltosos.

A 31 de Dezembro finalizámos por completo a redacção de código, tendo apenas continuado em pequenos retoques pontuais e na elaboração do relatório final.

# Especificação Técnica

## Classes Java

O uso *queries* SQL e da base dados, permite-nos uma simplificação tanto a nível de código *Java* em termos de métodos, como também nos permite uma diminuição de estruturas de dados no nosso programa. Como tal, estas apenas foram criadas com o intuito de facilitar o armazenamento de dados temporário, para que estes sejam recolhidos da base de dados e sejam usados na nossa aplicação.

### Book

Esta estrutura guarda todos os atributos que um documento pode ter: título, género, descrição, autor, editora, número de páginas, data da publicação, ID do documento, número de cópias existente na biblioteca e respectivas cópias disponíveis e finalmente prateleira.

### Publisher

Apenas tem como função armazenar o ID da editora e respectivo nome.

### Person

Á semelhança da entidade anterior, também esta estrutura armazena o ID e o nome de uma determinada pessoa. Esta estutura é principalmente usada para fazer uma listagem de empregados ou clientes extretamente simples, para posteriormente obter toda a informação dessa mesma pessoa.

## SQL e Estruturas de Dados

### ER

### Tabelas

O nosso programa engloba, tal como é sugerido pelo nosso diagrama ‘Entidade-Relação’, nove diferentes tabelas, a saber:

* AUTOR;
* EDITORA;
* EMPRESTIMO;
* FUNCIONARIO;
* LEITOR;
* PESSOA;
* PRATELEIRA;
* PUBLICACAO;
* AUTENTICACAO;

Pensamos que os nomes da tabelas são suficientemente sugestivos e por isso não entramos em demasiadas explicações sobre a sua função, deixando apenas algumas notas de maior relevância.

Em primeiro lugar, notar que as tabelas AUTOR e EDITORA não contêm muita informação, apenas um identificador e um nome. Poder-se-ia argumentar então seria mais simples inserir directamente, por exemplo, o nome do autor no devido campo de um publicação. Contudo, optámos por esta solução por uma questão de criar um maior nível de normalização nas tabelas e por outro lado, permitir, se tal fosse preciso numa aplicação real, inserir novos campos nestas tabelas (como contactos da editora ou prémios do autor) sem que fosse necessário proceder a alterações de fundo nas tabelas.

A razão por não termos inserido mais campos foi porque seriam irrelevantes no contexto deste trabalho puramente académico, sendo que iríamos complicar a aplicação desnecessariamente. Apesar disso, tivemos o cuidado de projectar o trabalho de uma forma cuidada e regulada, sendo por isso necessário criar tabelas para estas duas entidades.

Para a tabela PRATELEIRA, devemos referir o facto de que cada prateleira apenas contem livros de um dado género, sendo que sempre que uma prateleira fica cheia, torna-se obrigatório criar uma nova prateleira, e daí termos usado, para alem do identificador GENERO, a chave primária ID\_PRATELEIRA.

As tabelas FUNCIONARIO e LEITOR derivam da tabela PESSOA, sendo que são usadas chaves primárias e forasteiras para identificar os registos entre as diferentes tabelas. Notar que a tabela FUNCIONARIO, devido à sua estrutura, não permite que, por exemplo, um empregado seja despedido (sendo então preenchido o campo DATA\_SAIDA) e volte a ser readmitido. A solução para este problema seria criar uma nova tabela à parte que iria identificar os diferentes períodos de trabalho de um dado empregado. Não optámos por este caminho porque assim, iríamos sair dos parâmetros definidos no enunciado tanto para a complexidade da aplicação como para o número de tabelas.

### Sequências

As sequências possuem um papel importante na base de dados da livraria. Graças a estas, foi possível criar e assegurar ID’s únicos para cada entidade que os necessite. No caso da livraria foi usada a *seq\_id\_pessoa* para a entidade pessoa, *seq\_id\_document* para os documentos, *seq\_id\_aluguer* para os alugueres, *seq\_id\_publisher* para as editoras, *seq\_id\_author* para os autores e finalmente *seq\_id\_shelf* para as prateleiras.

### Utilizadores, Perfis e Sinónimos

A estrutura do nosso programa foi construída com o pressuposto de que haveria um conjunto de utilizadores normais, constituídos pelos funcionários que trabalham no edifício no seu dia-a-dia, e um administrador central que teria todos os privilégios da base de dados.

Deste modo, tornou-se importante separar as duas identidades e construir mecanismos que preservassem esta integridade hierárquica, isto é, apenas o administrador tem permissão para adicionar novos registos de funcionários e alterar os já existentes. Por seu lado, o funcionário regular apenas tem permissão de visualizar estes registos e eventualmente, modificar o seu próprio. Deste modo, desenvolvemos protecções de dois níveis: a nível da aplicação em si mesmo e a nível da base de dados.

Para a base de dados, recorremos à criação de perfis e utilizadores específicos. Quando inicialmente se liga à base de dados, o utilizador anónimo (porque neste momento, nem sequer sabemos se é um funcionário com as credenciais correctas) acede à tabela de autenticação recorrendo ao perfil normal de funcionário, descrito pelo seguinte código:

--Creates the profile.

CREATE PROFILE perfil\_funcionario LIMIT

Sessions\_per\_user UNLIMITED

Logical\_reads\_per\_session UNLIMITED

Logical\_reads\_per\_call 100

Idle\_time 30;

-- Now, creates the user.

CREATE USER employee

IDENTIFIED BY employee

DEFAULT TABLESPACE users

QUOTA 500M ON users

PROFILE perfil\_funcionario;

Ou seja, enquanto o administrador corre a aplicação usando um utilizador com privilégios de DBA, o acesso do funcionário normal apresentar algumas restrições, recorrendo a uma conta cujos privilégios são limitados. A este utilizador, demos as permissões de aceder sem restrições a todas as tabelas, exceptuando duas, nomeadamente a de autenticação e a de funcionário:

GRANT SELECT, UPDATE ON FUNCIONARIO TO employee;

GRANT SELECT, UPDATE ON AUTENTICACAO TO employee;

Assim, garantimos através da base de dados que um dado funcionário somente conseguirá efectuar as acções descritas anteriormente. De modo ao acesso às diferentes estruturas por parte dos diferentes utilizadores ser mais uniforme, também definimos sinónimos públicos, tanto para as diversas tabelas, como para o diferentes procedimentos.

Contudo, neste momento, levanta-se um questão. Se permitimos o UPDATE das tabelas a um funcionário (pois como dissemos, um funcionário pode actualizar a sua própria informação), como é que se garante que este não irá usar este privilégio para adulterar registos de outros funcionários?

É aqui que entra as protecções a nível da aplicação. Sempre que é pressionado o comando de alteração de informação, a aplicação verifica se o número de funcionário sobre o qual se pretendem realizar modificações coincide com o número de *login*. Se forem iguais, então o funcionário apenas quer alterar a sua informação e o programa prossegue normalmente. Caso contrário, é detectada uma quebra no protocolo e a aplicação aborta a operação, informando o utilizador que este não tem permissão para efectuar tal operação.

O administrador, naturalmente, tem autorização para efectuar toda e qualquer operação, sendo que por isso, é identificado sempre como o indivíduo com o ID\_PESSOA (da tabela PESSOA) número 1.

Apesar de não muito complexa e elaborada, procurámos com esta solução explorar um pouco mais a utilização de perfis, criação de utilizadores e manutenção de privilégios, matéria apenas dada nas aulas teóricas.

### Programas do Servidor

#### Procedures

Esta secção irá ser dividida de acordo com as tabelas/zona em que os procedimentos operam. Desta forma, iremos começar pelos Leitores seguindo de imediato para os Funcionários, dado que ambos herdam da entidade Pessoa.

##### PROCEDURE addEmployee

( nomePessoa IN Pessoa.nome\_pessoa%type, morada IN Pessoa.morada%type, bi IN NUMBER, data in date ,telefone IN NUMBER, email IN Pessoa.e\_mail%type, password IN Autenticacao.password%type, returnValue OUT INTEGER)

##### PROCEDURE addReader

( nomePessoa IN Pessoa.nome\_pessoa%type, morada IN Pessoa.morada%type, bi IN NUMBER, data IN date ,telefone IN NUMBER, email IN Pessoa.e\_mail%type, returnValue OUT INTEGER)

Os procedimentos apresentados em cima, são responsáveis pelo adicionar de leitores e funcionários à base de dados. Ambos recebem como parâmetros os atributos da Pessoa: nome da pessoam, morada, número de bi, data de nascimento, telefone e e-mail. Contudo, como no caso dos funcionários estes necessitam de uma *password* para lhes conceder acesso ao sistema, o procedimento addEmployee necessita também de receber este parâmetro.

Em ambas as situações, caso ocorra algum impedimento no decorrer dos procedimentos, através da variável do tipo de saída *returnValue*, os procedimentos podem devolver -1 ao ser levantada a excepção *DUP\_VAL\_ON\_INDEX* por já existir uma pessoa com esse mesmo número de BI ou podem devolver -2 no caso de surgirem outros problemas inexperados.

##### PROCEDURE updateReader

(nomePessoa IN Pessoa.nome\_pessoa%type, pmorada IN Pessoa.morada%type, pbi IN NUMBER, varData IN DATE ,ptelefone IN NUMBER, email IN Pessoa.e\_mail%type, returnValue OUT INTEGER)

##### PROCEDURE updateEmployee

(nomePessoa IN Pessoa.nome\_pessoa%type, pmorada IN Pessoa.morada%type, pbi IN NUMBER, varData IN DATE ,ptelefone IN NUMBER, email IN Pessoa.e\_mail%type, returnValue OUT INTEGER)

Os procedimentos apresentados são bastante idênticos entre si e são também idênticos aos respectivos procedimentos acima descritos para adicionar funcionários e leitores. O funcionamento destes passo por simplesmente verificar se existe a pessoa através do número de BI e fazer o update aos campos respectivos. Apesar da sua semelhança existem pequenas diferenças no que toca a valores de retorno e nos parâmetros. Começando por estes últimos, no caso dos funcionários estes necessitam de um parâmetro extra para receber um possível alteração de password. Relativamente aos valores de retorno, devolvem -1 se não tiverem sido encontradas pessoas com o ID fornecido ao levantarem a excepção NO\_DATA\_FOUND e devolvem -2 se surgirem outros imprevistos.

##### PROCEDURE fireEmployee

(biFuncionario IN Pessoa.id\_pessoa%type, retVal OUT INTEGER)

Para finalizar os procedimentos que operatam directamente sobre a entidade *pessoa* (Funcionário e Leitor), existe ainda um procedimento responsável pelo despedimento dos empregados. Um despedimento na verdade, é o funcionário possuir uma data de saída. Desta forma, o procedimento começa por verificar novamente se existe um funcionário com aquele ID. Se não existir é levantada uma excepção *NO\_DATA\_FOUND* e é devolvido -1. Caso existe mas a data de saída do funcionário não se encontrar vazia, então significa que este já foi despedido e como tá devolve -3. Por fim o procedimento devolve -2 para reportar outro tipo de erros.

Finda a análise dos procedimentos directamente relacionados com a entidade Pessoa, passemos então aos documentos.

##### PROCEDURE addDocument

( Aut IN AUTOR.NOME\_AUTOR%type, Edi IN EDITORA.NOME\_EDITORA%type, gen IN PRATELEIRA.GENERO%type, pages IN PUBLICACAO.PAGINAS%type, descri IN PUBLICACAO.DESCRICAO%type, varData IN PUBLICACAO.DATA%type, nome IN PUBLICACAO.NOME\_DOC%type, total IN PUBLICACAO.TOTAL%type, retVal OUT INTEGER)

O procedimento responsável pelo adicionar de novos documentos é algo complexo, razão pela qual iremos por partes.

Numa primeira fase, analisamos se o autor do documento introduzido já existe na nossa base de dados. Se por alguma razão esta pesquisa devolver mais do que um resultado, significa que algo de errado aconteceu na base de dados pois não podem existir autores com nomes iguais. Numa situação deste tipo é devolvido -1 pela variável de saída *retVal.* Se não existir nenhum autor, inserimos um novo com o nome fornecido por parâmetro e associamos-lhe um ID recorrendo à sequência *seq\_id\_author*.

Tendo registado o autor passamos à segunda fase relativa ao armazenamento do documento. Recorrendo à função *getPrateleira* é feita uma análise que nos indica se existem prateleiras para aquele género e com espaço suficiente contíguo para o número de cópias introduzidas. Se não existirem prateleiras, é criada uma nova prateleira com um tamanho mínimo de cem espaços ou então se ainda não for suficiente é criada uma prateleira especial com o número de cópias do documento mais um espaço de manobra de dez elementos. Estas novas prateleiras ficam associadas a um novo ID obtido pela sequência seq\_id\_shelf.

Após termos a questão do armazenamento e do autor tratadas é necessário verificar se a editora introduzida já existe. Para isso recorre-se à função *getEditora*. Novamente, se a editora não existir procedemos da mesma forma que para o autor. Criamos uma nova associada também a um ID proveniente da sequência *seq\_id\_publisher*.

Estando finalmente reunidos quase todos os elementos exigidos, é obtido um ID para associar ao documento da sequência *seq\_id\_document,* e este é inserido na tabela PUBLICACAO e é feito o *update* à ocupação da prateleira obtida anteriormente.

Com a existência de várias inserções relativas aos documentos, editoras e prateleiras, existe a segurança de que é retornado também o valor -1 em caso de ser levantada a excepção *DUP\_VAL\_ON\_INDEX*, e -2 para os restantes erros.

##### PROCEDURE addCopyDocument

(idDoc IN PUBLICACAO.ID\_DOC%type, novos IN PUBLICACAO.TOTAL%type, retVal OUT NUMBER)

Numa biblioteca, é normal que determinados livros sejam requesitados mais frequentemente. Como tal, poderá ser necessário reforçar o seu *stock*. É neste âmbito que surge o procedimento responsável por adicionar cópias. Embora este seja extretamente simples, o procedimento está seguro pelo *trigger* *checkShelfs* responsável pela organização das prateleiras. Desta forma, este procedimento é apenas responsável por fazer o update das quantidades totais e disponíveis da tabela PUBLICACAO, de acordo o número de cópias passadas como parâmetro.

Em caso de as cópias não corresponderem a nenhum livro existente na base de dados ocorre a excepção *NO\_DATA\_FOUND* e é devolvido o valor -1. Para outros erros é devolvido -2 e em caso de sucesso 0.

##### PROCEDURE removeCopyDocument

(idDoc IN PUBLICACAO.ID\_DOC%type, noRem IN PUBLICACAO.TOTAL%type, retVal OUT NUMBER)

Assim como existe a possibilidade de adicionar livros, através deste procedimento é possível removê-los. Na verdade, e assim como o adicionar de cópias se traduz num incrementar do número de livros disponíveis e total, também o remover se traduz pelo decrementar do número de livros. O procedimento *removeCopyDocument* começa por procurar documentos com o ID passado por parâmetro. Se encontrar, então procede à procura da sua estante e remove as cópias necessárias da estante diminuindo a sua ocupação e diminuindo também o número de livros totais se estes não se encontrarem de momento alugados. Se estiverem alugados é devolvido de imediato 0. Em caso de sucesso é devolvido -1. Da mesma forma é devolvido -1 se o livro não existir na base de dados e -2 para os restantes erros.

##### PROCEDURE newRequisition

(book\_id IN Publicacao.id\_doc%type, reader\_id IN Pessoa.id\_pessoa%type, employee\_id IN Pessoa.id\_pessoa%type, returnValue OUT INTEGER)

Uma das principais funcionalidades de uma biblioteca são os pedidos/alugueres/requisitos de determinados documentos. Este procedimento é então o responsável por criar um novo empréstimo e associá-lo a um livro, a um leitor e a um funcionário.

Numa fase inicial, este procedimento verifica a existência da publicacação em questão. Se não existir então devolve pela excepção *NO\_DATA\_FOUND* o valor -2. Caso contrário prossegue para verificar se o leitor possui os três empréstimos máximos. Se tiver, então não pode prosseguir e devolve -5. Se estudo estiver dentro das normas então este procedimento passa para a verificação seguinte relacionada com os atrasos. Se o utilizador possuir empréstimos em atraso, então não pode alugar mais nada pelo que o procedimento retorna o valor -6. Por fim basta verificar se existem cópias suficientes para que se possa proceder ao empréstimo. Em caso afirmativo, recorre-se à sequência *seq\_id\_aluguer* para obter um novo ID para este empréstimo, e insere-se o novo empréstimo na tabela EMPRESTIMO com uma data de entrega com mais uma semana da data actual devolvendo 0. Ao efectuar esta inserção, é disparado o *trigger updateReqsAndCopies* para actualizar as cópias disponíveis da publicação e o número de empréstimos do leitor. Em caso não existirem seque cópias suficientes para alugar então é devolvido o valor -1 ou -3. Para os restantes erros está associado um valor de retorno de -4.

##### PROCEDURE returnRequisition

( req\_id IN Emprestimo.id\_emprestimo%type, returnValue OUT INTEGER)

Ao existir a funcionalidade de efectuar requisitos, surge naturalmente a necessidade de criar um procedimento para efectuar as entregas ou devoluções. Este procedimento foi criado com esse mesmo intuito.

Em primeiro tudo é necessário verificar se o requisito com o ID fornecido é válido e se existe. Para tal, é realizada uma pesquisa e guardada a data de entrega. Se nada tiver sido encontrado, então surge a excepção *NO\_DATA\_FOUND* e é devolvido de imediato o valor -2. Se for encontrada, o procedimento continua a executar e verifica se a data de entrega possui alguma data válida. Se possuir, significa que este empréstimo já foi entregue, devolvendo -1, se não é feito o update de encomendas do leitor e é incrementado o número de cópias disponíveis na PUBLICACAO. Nesta situação, tudo correu de forma correcta devolvendo no fim o valor 0. Para outros erros é sempre devolvido o valor -3.

##### PROCEDURE login

(username IN AUTENTICACAO.ID\_EMPREGADO%type, pw IN AUTENTICACAO.PASSWORD%type, returnValue OUT INTEGER)

Neste momento, possuimos todos os procedimentos para a gestão da base de dados contudo, falta-nos o procedimento responsável pela autenticação. O procedimento *login,* apenas consulta a tabela AUTENTICACAO e verifica se o *username* e *password* passados como parâmetros estão correctos dando permissão à aplicação retornando 1. Em caso de insucesso retorna -1.

##### Procedimentos estatísticos

Estes procedimentos não necessitam de um revisão aprofundada. Existem apenas para recolher dados efectuando apenas operações de leitura. São compostos apenas por parâmetros de saída para providenciar os valores estatísticos correspondentes. Os comandos SQL são apenas SELECT’s responsáveis pela recolha dos dados e seus tratamentos estatísticos relativos a médias, arredondamentos e contagens.

De seguida serão apresentados então os procedimentos estatísticos com breves referências sobre quem eles operam e sobre os dados que recolhem

###### PROCEDURE employeesStats

(no\_entries OUT INTEGER, fired\_employees OUT INTEGER, avg\_working\_time OUT FLOAT)

Este procedimento permite-nos obter o número de empregados no sistema diferenciando os empregados activos dos despedidos e o tempo médio de trabalho.

###### PROCEDURE readersStats

(no\_entries OUT INTEGER, readers\_with\_books OUT INTEGER, faulty\_readers OUT INTEGER)

Para dados relativos aos leitores, existe o procedimento *readersStats* que devolve o número total de leitores no sistema, o número de leitores com empréstimos e em falta para com alguma entrega.

###### PROCEDURE booksAndShelvesStats

(no\_books OUT INTEGER, max\_pages OUT INTEGER, min\_pages OUT INTEGER, avg\_pages OUT FLOAT, avg\_copies OUT FLOAT, no\_shelves OUT INTEGER, occupation OUT FLOAT, avg\_capacity OUT FLOAT)

*bookAndShelvesStats* é um procedimento um pouco maior, mas nem por isso mais complex. Este é responsável por devolve o número total de livros, o número máximo e mínimo de páginas, uma média do número de páginas dos livros e de cópias, o número de prateleiras, a ocupação e finalmente a média da capacidade da nossa biblioteca.

###### PROCEDURE requisitionsStats

(no\_entries OUT INTEGER, on\_going\_reqs OUT INTEGER, finished\_reqs OUT INTEGER, no\_faulty\_reqs OUT INTEGER, avg\_reqs\_per\_day OUT FLOAT, no\_days\_with\_reqs OUT INTEGER)

Para obter informações estatísticas sobre as requesições, surge este procedimento. Após ser executado é possível retirar a quantidade de empréstimos no sistema, empréstimos ainda não entregues, já entregues e em falta, a média de empréstimos por dia e o número total de dias em que existiram empréstimos.

###### PROCEDURE authorsAndPublishersStats

(no\_authors OUT INTEGER, avg\_doc\_per\_author OUT FLOAT, no\_publishers OUT INTEGER, avg\_doc\_per\_publisher OUT FLOAT)

Por fim existe ainda este procedimento para obter informações sobre os autores e editoras. Após a sua execução será obtido o número de autores e de editoras no sistema, a média de documentos por autor e por editora.

#### Functions

##### getEditora

Esta função tem como objectivo a devolução do ID\_EDITORA para uma dada editora com o nome que é fornecido como parâmetro de entrada. Em caso de não encontrar devolve -1.

##### getPrateleira

À semelhança da função anterior, esta tem como objectivo a devolução do ID\_ ID\_PRATELEIRApara uma dada prateleira cujo género é igual ao fornecido como parâmetro de entrada. Em caso de não encontrar devolve -1.

#### Triggers

##### TRIGGER checkShelf

AFTER UPDATE OF TOTAL ON PUBLICACAO FOR EACH ROW WHEN (new.total != old.total)

Na construção da nossa livraria, apenas foi criado um trigger*,* dado que a complexidade do sistema não exige mais. Por definição um *trigger* é a última linha defensiva de uma base dados. Neste caso passa-se o mesmo. Quando é feito um *update* no campo total da tabela *PUBLICACAO,* e o novo campo é maior que o anterior, então significa que foram adicionadas novas cópias a um determinado livro que já existia numa determinada prateleira. Nesta caso, surge então um possível problema. Se a prateleira não tiver espaço suficiente para a nova remessa de livros de forma a que todos os livros iguais fiquem no mesmo local, então é necessário procurar uma prateleira disponível, ou se não existir, criar uma nova e mover todos os livros para esse novo local. Este é o papel do nosso trigger *checkShelf*.

#### Diagramas de classes

O diagrama de classes da nossa aplicação, revela a simplicidade da aplicação graças à integração da base dados.

Sendo assim existe a classe *DatabaseHandler* responsável pelo método que estabelece a ligação com a base de dados (*connect*) e por todos os métodos que contém as chamadas a procedimentos do servidor da base dados ou que contém simples *queries SQL* que apenas realizam pesquisas.

Na prática, as acções provocadas pelos botões e caixas de texto da interface gráfica, provocam eventos que chamam os simples métodos pertencentes à classe *databaseHandler* responsáveis por devolver os dados prontos a ser utilizados.

A classe *Validation* é meramente um local para guardar um conjunto de funções extretamente úteis e frequentemente usadas que permitem formatar determinado tipo de dados, principalmente datas e verificar se determinadas *Strings* são valores numérios.

Relativamente ás estruturas de dados da aplicação, estas já foram explicadas na seccção estruturas de dados, mais a cima neste relatório.

Por fim a classe *Constants* é responsável por guardar as referências para os vários menus, de forma a que se possa realizar uma navegação entre menus simples e eficaz.

A leitura desta descriçãoi não dispensa a consulta do diagrama de classes que segue em anexos.

#### Códigos de erros

# Conclusão

Na maioria das vezes, quando são usadas ferramentas que nos facilitam o trabalho de forma extremamente eficaz, significam que escondem dentro de si uma enorme complexidade e um enorme investimento. Este é sem dúvida, o caso da base de dados Oracle.

A nossa aplicação pretende gerir de uma forma académica uma livraria. Ao observarmos o diagrama de classes, é possível observar a simples estrutura do código e até mesmo, o reduzido tamanho deste se olharmos para os ficheiros fonte. Estes factos, reflectem a importância do uso de base dados para aplicações bem mais complexas e em maior escala no mundo real. Esta permite-nos um relativo fácil controlo da integridade de dados ao estabelecermos regras (*constraints*), segurança ao atribuirmos permissões/perfis a utilizadores de forma a que só um determinado conjunto possua determinados acessos e principalmente escalabilidade e aumento do *throughput* de pedidos graças aos mecanismos de *lock* da Oracle que permitem trancar apenas determinados registos em vez de bloquear o acesso a toda a tabela.

Concluindo, este trabalho permitiu uma familiarização com a programação de *queries sql* e com uma nova estrutura dos programas que não necessitam do desenvolvimento de novas e grandes estruturas de dados nem dos seus respectivmos mecanismos de concurrência que previnem a corrupção de dados.

# Bibliografia

1. Microsoft Word e Excel
2. Enunciado do Projecto de Base de Dados
3. Imagens: **Da capa:**
   1. http://www.estantevirtual.com.br/