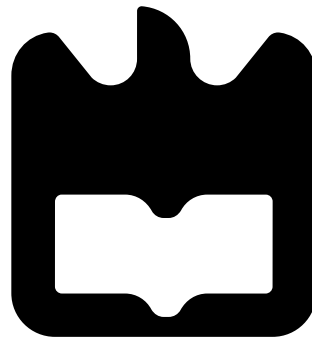




Bruno Manuel  
de Moura Ramos

Sistema de Recolha e Armazenamento Remoto  
de Informação Sensorial de um Processo  
Industrial usando Bases de Dados Múltiplas

# DOCUMENTO PROVISORIO





**o juri/the jury**

presidente/president

**ABC**

Professor Catedratico da Universidade de Aveiro (por delegacao da Reitora da Universidade de Aveiro)

vogais/examiners committee

**DEF**

Professor Catedratico da Universidade de Aveiro (orientador)

**GHI**

Professor associado da Universidade J (co-orientador)

**KLM**

Professor Catedratico da Universidade N



**agradecimentos /  
acknowledgements**

Um obrigado aos que me ajudaram. (Adicionar agradecimentos)



## Palavras-chave

Base de dados relacional; rede de bases de dados; monitorização; monitorização remota; aplicação; *web*.

## Resumo

Moldes de injeção têm uma vasta aplicabilidade no mundo industrial. Afim de melhorar a qualidade do produto final e reduzir falhas surgiu a necessidade de instrumentar e monitorizar moldes remotamente. Neste projeto desenvolveu-se uma rede de bases de dados relacionais e uma aplicação em ambiente *Web*. A primeira garante uma transferência de valores segura e permanente de forma a criar um histórico. A segunda permite ao utilizador interagir com as bases de dados desenvolvidas podendo editá-las e consultá-las de forma a gerar relatórios.

No desenvolvimento deste projeto utilizou-se *MySQL* e *C* para criar e definir a rede de bases de dados resultando numa solução simples e funcional a baixo nível. Utilizou-se *Apache*, *PHP*, e *HTML* para desenvolver e implementar a aplicação de forma a que esta seja multiplataforma e garanta um acesso remoto ao utilizador.

A solução proposta cumpre todos os objetivos definidos podendo ser já utilizada numa fase experimental apesar de necessitar alguns melhoramentos a nível de desempenho.





## **Abstract**

Nowadays, it is usual to evaluate a work . . .



# Conteúdo

<b>Conteúdo</b>	<b>i</b>
<b>Lista de Figuras</b>	<b>iii</b>
<b>Lista de Tabelas</b>	<b>v</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Estado de Arte</b>	<b>3</b>
2.1 Moldes de injeção . . . . .	3
2.2 Base de dados relacional . . . . .	4
2.2.1 Domínios, Atributos, Tuplos e Relações . . . . .	6
2.2.2 Análise de requisitos . . . . .	7
2.2.3 Diagrama Entidade Relação . . . . .	7
2.2.4 Esquema Relacional . . . . .	11
2.3 <i>Structured Query Language</i> . . . . .	12
2.4 <i>Software</i> utilizado . . . . .	13
2.4.1 <i>Ubuntu</i> . . . . .	13
2.4.2 <i>MySQL</i> . . . . .	14
2.4.3 <i>C</i> . . . . .	16
2.4.4 <i>Apache</i> . . . . .	17
2.4.5 <i>HTML, JS e PHP</i> . . . . .	17
<b>3 Proposta de Solução</b>	<b>19</b>
3.1 Infraestrutura de Dados . . . . .	19
3.2 Base de Dados . . . . .	20
3.2.1 Análise de Requisitos . . . . .	20
3.2.2 Desenho conceptual e esquema lógico . . . . .	23
3.2.3 Construção da base de dados . . . . .	24
3.2.4 Programa de transferência . . . . .	27
3.2.5 Gestão de <i>backups</i> . . . . .	28
3.2.6 Simulador . . . . .	31
3.2.7 Utilizadores . . . . .	31
<b>4 Aplicação de Gestão do Sistema</b>	<b>35</b>
4.1 Adaptação da infraestrutura . . . . .	35
4.2 Interface gráfica . . . . .	37

4.2.1	<i>Main</i> . . . . .	39
4.2.2	<i>Login</i> . . . . .	40
4.2.3	Consultas . . . . .	41
4.2.4	Administração Local . . . . .	44
	Administração . . . . .	45
	Conectar Local . . . . .	49
<b>5</b>	<b>Conclusões</b>	<b>53</b>
5.1	Comentários . . . . .	53
5.1.1	Infraestrutura de dados . . . . .	53
5.1.2	Aplicação . . . . .	53
5.2	Trabalhos Futuros . . . . .	54
	<b>Bibliografia</b>	<b>57</b>

# Lista de Figuras

1.1	esquema . . . . .	1
2.1	Molde de injeção aberto com a peça realizada[1] . . . . .	3
2.2	Esquema simplificado de um sistema de bases de dados . . . . .	5
2.3	Exemplo onde é possível observar a relação com os seus atributos e alguns tuplos	6
2.4	Representação de entidades, relações e atributos . . . . .	8
2.5	Representação de entidades e relações fortes e fracas . . . . .	8
2.6	Representação dos vários graus das relações (NECESSITA EDIÇÃO) . . . . .	9
2.7	Representação de múltiplas relações entre duas entidades . . . . .	9
2.8	Diagrama dos vários tipos de cardinalidade . . . . .	10
2.9	Notação da cardinalidade de Chen . . . . .	10
2.10	Representação da obrigatoriedade de participação . . . . .	11
2.11	Representação dos vários tipos de atributos. Os atributos derivados são re- presentados a tracejado, os compostos têm sub-atributos associados a si e os multi-valor são representados com linha dupla . . . . .	11
2.12	Representação de um esquema relacional com as ligações das chaves estrangeiras	12
2.13	Logótipo do <i>Ubuntu</i> . . . . .	13
2.14	Logótipo do <i>MySQL</i> . . . . .	14
2.15	Logótipo do <i>GNU</i> . . . . .	16
2.16	Logótipo do <i>Apache</i> . . . . .	17
2.17	Representação da ferramenta mais usada para desenvolver aplicações em ambi- ente <i>Web</i> . . . . .	17
3.1	Diagrama da infraestrutura com múltiplas bases de dados locais conectadas a uma base de dados central . . . . .	19
3.2	Diagrama do fluxo de informação. Os simuladores substituem provisoriamente sistemas de aquisição de dados . . . . .	20
3.3	Diagrama final da infraestrutura de dados. Os valores gerados nos simuladores são transferidos para a base de dados central. Estes valores são então proces- sados pelo gestor de <i>backups</i> que cria e guarda <i>backups</i> provisoriamente num repositório online no <i>GitHub</i> . . . . .	20
3.4	Diagrama Entidade/Relação realizado a partir da análise de requisitos . . . . .	23
3.5	Esquema Lógico obtido através da análise do diagrama entidade/relação onde as entidades e os seus atributos se traduzem em tabelas e as relações nas chaves estrangeiras destas . . . . .	24
3.6	fluxograma . . . . .	27

3.7	fluxograma . . . . .	29
4.1	Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local . . . . .	36
4.2	Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local . . . . .	36
4.3	Fluxograma 2 . . . . .	37
4.4	Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local . . . . .	37
4.5	Funcionalidades da página <i>Main</i> com e sem <i>login</i> . . . . .	39
4.6	Página de <i>Login</i> para iniciar sessão na base de dados central . . . . .	40
4.7	Exemplos de erros retornados quando introduzidas credenciais não válidas na página <i>Login</i> . . . . .	40
4.8	Página de Consultas e exemplo de resposta a uma consulta . . . . .	41
4.9	Exemplos de erros retornados na página de Resposta da Consulta . . . . .	43
4.10	Funcionalidades da página Administração com e sem conexão local . . . . .	44
4.11	Área de Gestão de Clientes sem conexão local . . . . .	46
4.12	Área de Gestão de Moldes . . . . .	47
4.13	Área de Gestão de Sensores . . . . .	47
4.14	Função do botão Validar, onde os valores da base de dados temporária local são transferidos de forma permanente para as bases de dados central e local . . . . .	48
4.15	Área Conectar Local onde se visualiza as bases de dados instaladas no sistema . . . . .	49
4.16	<i>Main</i> com conexão local . . . . .	50
4.17	Área Criar Local cria a base de dados para o cliente selecionado no sistema em que se acede a aplicação . . . . .	50
4.18	<i>Queries</i> geradas para completar a instalação da base de dados no sistema local. Consistem nas permissões para os utilizadores que só podem ser garantidas via <i>root</i> bem como informação para completar o sistema de dados . . . . .	51
4.19	Área Instalar <i>MySQL</i> para instalar o <i>software</i> com os comandos em <i>Linux</i> . . . . .	52
5.1	Área de Criar Local com criação via <i>root</i> em vez de gerar <i>queries</i> para o utilizador inserir manualmente no <i>MySQL</i> . . . . .	54
5.2	Exemplo de área de Registar para criar utilizadores com base no ID de trabalhador e email . . . . .	55

# Lista de Tabelas

3.1	Tabelas com os seus atributos e respetivos domínios e obrigatoriedades. Os atributos data e hora da entidade registos são fundidos no atributo r_data_hora graças ao tipo de dados datetime . . . . .	25
3.2	Tabelas com as permissões das credenciais criadas para as várias tabelas das bases de dados . . . . .	33





# Capítulo 1

## Introdução

Para satisfazer as necessidades de uma população cada vez maior, surge a necessidade de produzir artigos em grande escala. Uma das tecnologias que se salientou neste campo, especialmente na produção com plástico, foi a moldagem por injeção.

Os fabricantes de moldes de injeção procuram formas de melhorar as suas ferramentas, como a qualidade do processo de fabrico e a qualidade dos materiais usados na sua construção. No entanto, dada a complexidade do processo e a quantidade de variáveis envolvidas surge a necessidade de instrumentar e monitorizar o molde de forma a garantir um controlo de qualidade. Num esforço conjunto com outros projetos e dissertações procura-se dar mais um passo na monitorização dos moldes de injeção.

O comportamento destas ferramentas pode não se alterar de forma imediata mas alterar-se lentamente ao longo do seu tempo de vida. Para analisar esta alteração do comportamento procura-se uma forma de garantir a criação de um histórico da resposta térmica e integridade estrutural em pontos críticos de um molde de injeção. Dado que os moldes são produzidos para vários clientes pode tornar-se difícil coordenar os históricos desenvolvidos. Este projeto descreve uma solução capaz de coordenar e armazenar tais históricos mantendo a informação organizada e disponível a qualquer momento.

Propõe-se armazenar a informação gerada localmente nos clientes num sistema central situado na empresa que fabrica os moldes. Esta ligação entre os sistemas locais e central, representada na Figura 1.1, visa ser segura, atempada e sem interferência nos processos produtivos. Quanto ao método de armazenamento, propõem-se usar bases de dados relacionais que satisfaçam os requisitos propostos no projeto.

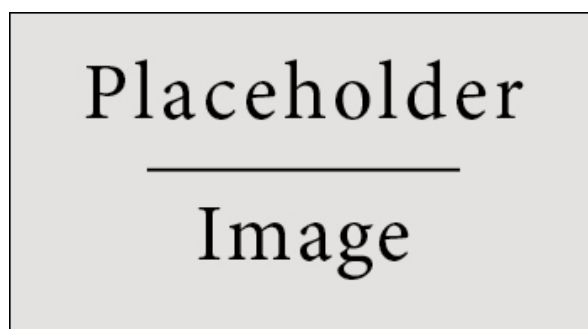


Figura 1.1: esquema



## Capítulo 2

# Estado de Arte

### 2.1 Moldes de injeção

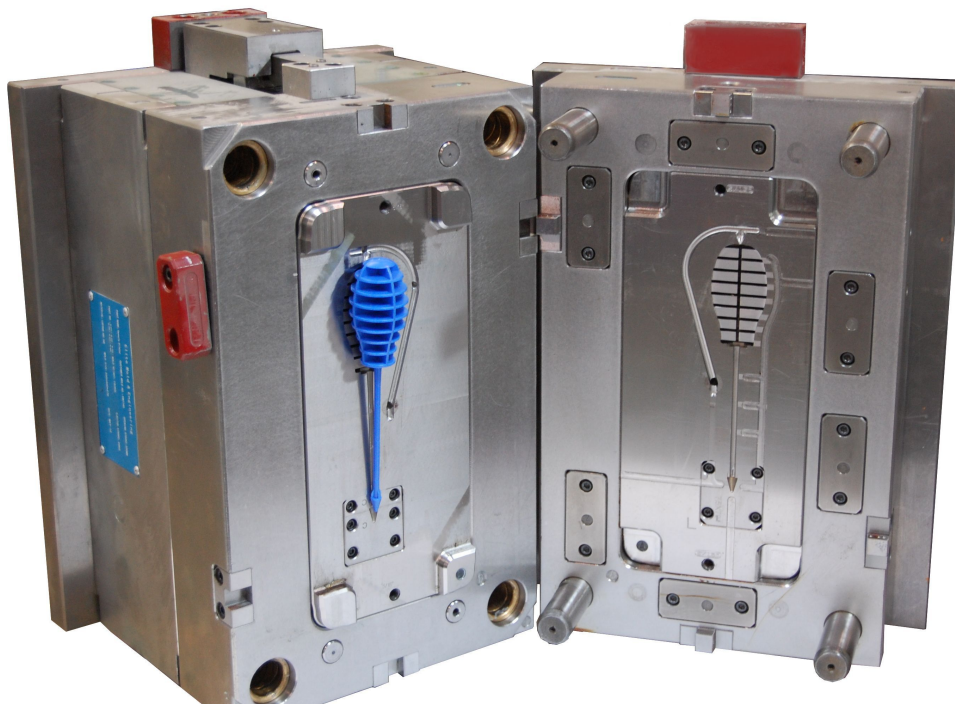


Figura 2.1: Molde de injeção aberto com a peça realizada[1]

Moldagem é o processo mecânico de dar forma a um material no estado líquido usando um molde[2, 3]. Um molde é uma ferramenta sólida oca desenvolvida para efeitos de fundição[4]. Esta é enchida com um material em estado líquido ou em pó como plástico, metal, cerâmica ou vidro[5, 6, 7, 8].

A moldagem por injeção é particularmente útil na produção em massa de peças de plástico com elevada complexidade geométrica[9, 10]. Estas podem ser encontradas em todas as áreas da indústria como por exemplo empacotamento, aviação, construção e eletrónica[11]. Neste

processo, plástico quente é forçado para dentro de um molde frio com a forma desejada. O material cobre todas as feições do molde e vai endurecendo sobre o efeito de altas pressões. O processo de injeção pode ser dividido em fases distintas[9]:

- Fecho do molde
- Enchimento
- Compactação
- Abertura do molde
- Extração

Em 1868, John Wesley Hyatt inventou uma maneira de fazer bolas de bilhar injetando celuloide para dentro de um molde[12, 13].

Em 1872 John e o seu irmão Isaiah patentearam a primeira máquina de moldes de injeção. Esta era relativamente simples comparada às que são usadas hoje em dia na indústria. Consistia de um pequeno embolo para injetar plástico num molde através de um cilindro quente[12, 14].

A indústria cresceu lentamente produzindo artigos de plástico como botões e pentes. Nos anos 40 a utilização de moldes de injeção cresceu por causa da Segunda Guerra Mundial que tinha uma grande procura de produtos baratos e produzidos em massa[12].

Em 1946, James Hendry construiu a primeira máquina de moldes de injeção com um sem fim, revolucionando a indústria dos plásticos com um design para substituir o embolo de Hyatt. Este sem fim é colocado dentro do cilindro e mistura o material a ser moldado antes de ser injetado no molde. Isto permitiu que cor ou plástico reciclado fossem adicionados à mistura[12, 15].

A qualidade de fabrico dos moldes e das peças produzidas evoluiu com o passar do tempo. A indústria investiu em técnicas sofisticadas no desenvolvimento de moldes, para que estes não contenham falhas, e no uso de materiais com qualidade elevada para evitar o desgaste da ferramenta. No entanto, dificilmente se atinge peças com a qualidade desejada unicamente através das ferramentas desenvolvidas. É necessário implementar também técnicas de monitorização de qualidade[16].

## 2.2 Base de dados relacional

Bases de dados e sistemas de gestão de bases de dados são uma componente essencial na vida da sociedade moderna: muitos de nós realizamos ações todos os dias que envolvem interações com bases de dados. Por exemplo, o ato de depositar e levantar dinheiro num banco, reservar estadias e voos ou comprar alguma coisa online podem envolver alguém ou um algum programa informático que acede a uma base de dados[17].

Esta tecnologia tem um impacto cada vez maior no uso dos computadores. É seguro afirmar que as bases de dados desempenham um papel crítico em quase todas as áreas onde são usados computadores[17].

Uma base de dados é uma coleção organizada de dados que estão relacionados e que podem ser partilhados por múltiplas aplicações[18]. São considerados dados factos reais que podem ser registados e têm um significado implícito, como por exemplo, nomes, moradas e números de telefone. Uma base de dados tem uma fonte de onde os dados são provenientes, algum grau de interação com eventos do mundo real e uma audiência que está ativamente interessada no seu conteúdo[17].

A implementação destas é garantida por parte de um sistema de gestão de base de dados. Este é um *software* de que facilita os processos de definir, construir, manipular e partilhar bases de dados entre vários utilizadores e aplicações. Definir uma base de dados envolve especificar o tipo de dados, estruturas e restrições dos dados a serem armazenados. Construir é o processo de guardar dados e armazená-los num local definido pelo sistema de gestão de bases de dados. Manipular inclui funções como realizar *queries* à base de dados e receber informação específica, alterá-la de acordo com mudanças nas variáveis e gerar relatórios a partir dos dados presentes. Partilhar permite que múltiplos utilizadores e programas acedam à base de dados simultaneamente[17].

Outras funções importantes fornecidas pelos sistemas de gestão de bases de dados incluem proteger e manter a base de dados durante um longo período de tempo. Proteger inclui proteção contra falhas de *hardware* e *software* e segurança contra acessos maliciosos ou não autorizados. Tipicamente uma grande base de dados pode ter um tempo de vida de vários anos, então o sistema de gestão de bases de dados tem de fornecer ferramentas para manter a base de dados de forma a permitir que o sistema evolua com novos requerimentos que surjam ao longo do tempo[17]. Para completar estas definições iniciais, chama-mos sistema de base de dados ao conjunto das bases de dados com o sistema de gestão de bases de dados como representado na Figura 2.2.

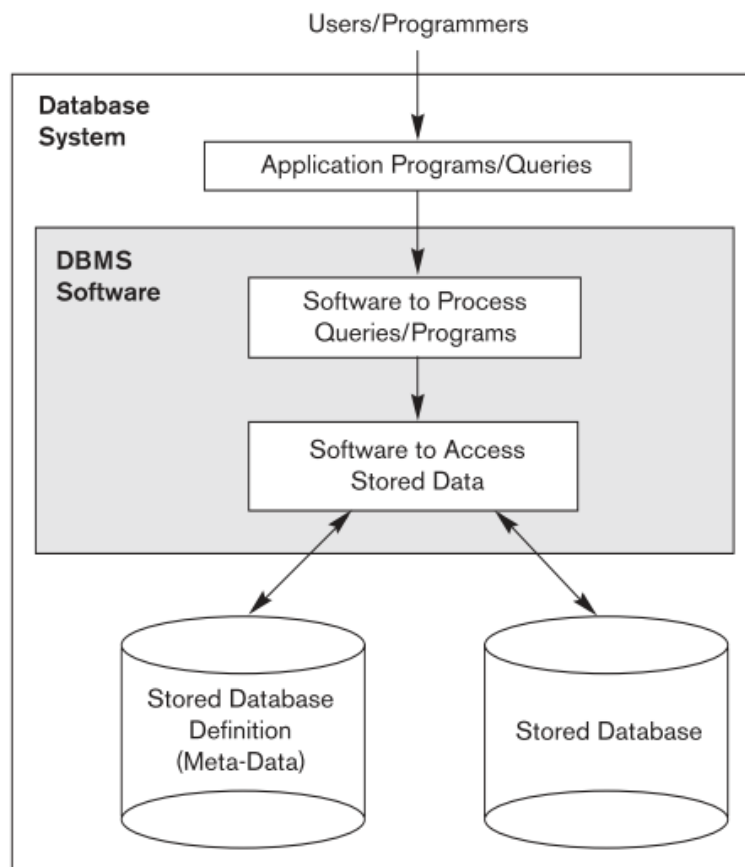


Figura 2.2: Esquema simplificado de um sistema de bases de dados

O modelo relacional foi primeiramente introduzido por Ted Codd da *IBM Research* em 1970[17, 19] e atraiu imediatamente atenção pela sua simplicidade e base matemática. O modelo utiliza o conceito matemático de "relação" representado por tabelas e é baseado na teoria dos conjuntos[17].

As primeiras implementações comerciais do modelo relacional ficaram disponíveis no início dos anos 80, como o *SQL/DS* no sistema operativo *MVS* do *IBM* e o sistema de gestão de base de dados *Oracle*. Desde aí, o modelo tem sido implementado num grande número de sistemas comerciais. Os sistemas de gestão de bases de dados relacionais populares atualmente incluem *DB2* e *Informix Dynamic Server* (do *IBM*), *Oracle* e *Rdb* (da *Oracle*), *Sybase* (da *Sybase*) e *SQLServer* e *Access* (da *Microsoft*). Existem também vários sistemas grátis como *MySQL* e *PostgreSQL*[17].

Modelos de dados que procederam o relacional incluem os modelos hierárquicos e de rede. Estes foram propostos nos anos 60 e foram implementados nos primeiros sistemas de gestão de bases de dados nos anos 60 e 70. Estes modelos tiveram bastante importância na história das bases de dados e são referidos atualmente como sistemas de bases de dados *legacy*[17].

### 2.2.1 Domínios, Atributos, Tuplos e Relações

Um domínio é um conjunto atómico de valores. Atómico significa que cada valor num dado domínio é único e indivisível. Um método comum de especificar um domínio é especificar o tipo de dados do mesmo. Além disto, também é útil dar nomes aos domínios. Por exemplo, a informação de números de telemóvel pode ser guardada num domínio onde o seu tipo de dados é um inteiro de nove dígitos[17].

Um esquema de relação é constituído por um nome de uma relação e uma lista de atributos. Cada atributo é o nome do papel desempenhado por domínio no esquema da relação. É possível múltiplos atributos terem o mesmo domínio[17].

A relação do esquema de relação é um grupo de múltiplos tuplos. Cada tuplo é um conjunto de valores ordenados que estão associados diretamente a um atributo.

A Figura 2.3 representa um exemplo do que foi dito anteriormente.

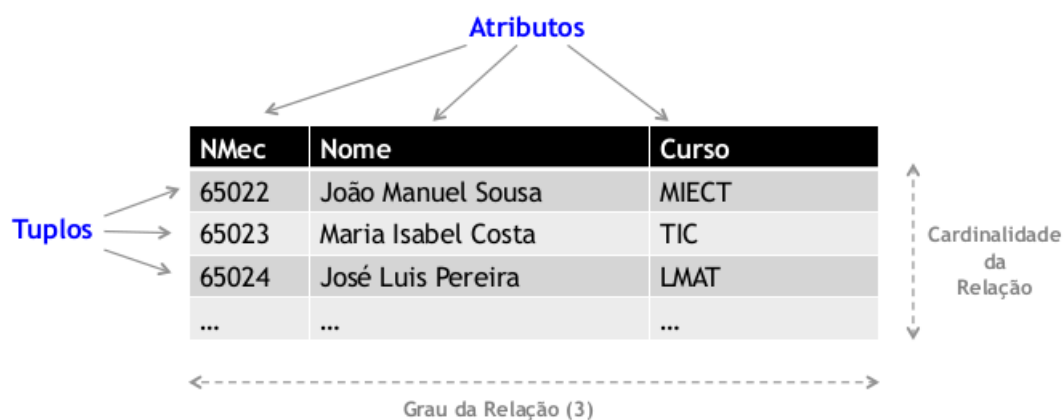


Figura 2.3: Exemplo onde é possível observar a relação com os seus atributos e alguns tuplos

### 2.2.2 Análise de requisitos

A análise de requisitos é o primeiro passo necessário para definir uma base de dados. Este processo envolve uma comunicação com a entidade que deseja adquirir a base de dados e pode ser sumariada nos seguintes passos:

1. Numa primeira fase realiza-se uma recolha detalhada de toda a informação referente ao problema do mundo real e retirar entidades, atributos, restrições, etc.
2. Filtrar a informação de forma a remover redundâncias e informação pouco relevante
3. Clarificar aspetos pouco claros
4. Completar o problema com informação adicional necessária
5. Distinguir dados de operações

Este processo é transversal ao modelo de dados escolhido para realizar a base de dados. Para uma base de dados relacional é necessário identificar as chaves de uma relação:

- Superchave - Conjunto de atributos que identificam de forma única dois tuplos distintos
- Chave candidata - são todas as superchaves que não podem ser mais simplificadas
- Chave primária - escolhida do conjunto de chaves candidatas e identifica de forma única o tuplo
- Chave única - restantes chaves candidatas que não foram escolhidas como chave primária
- Chave estrangeira - conjunto de atributos que é chave primária de outra relação

A escolha da chave primária pode ser feita de forma aleatória mas, priorizam-se chaves que identifiquem de forma natural um atributo. Por exemplo: uma pessoa pode ser identificada por um número de identificação, número de identificação fiscal ou pelo seu número de telefone dado que estes nunca se repetem. O número de identificação é mais natural para identificar uma pessoa do que o seu número de identificação fiscal e menos volátil do que o seu número de telefone, dado que este pode ser alterado.

Para realizar um Diagrama Entidade Relação são necessárias informações sobre a relação entre as entidades, a cardinalidade e a obrigatoriedade de participação na relação. Estas são deduzidas durante a análise de requisitos mas serão explicadas na Subseção 2.2.3.

### 2.2.3 Diagrama Entidade Relação

A representação lógica dos dados é uma componente importante na área das bases de dados. Existem vários modelos para esta representação antes da criação do modelo entidade-relação apresentado por P. P. Chen em 1976, sendo os mais conhecidos os modelos em rede, relacional e entidade. Estes modelos têm as suas vantagens e desvantagens. O modelo em rede permite uma vista mais natural dos dados dividindo-os em entidades e relações (até um determinado ponto), mas a sua capacidade de garantir independência dos dados foi superada. O modelo relacional consegue ter um grande grau de independência dos dados, mas pode perder alguma informação semântica importante sobre o mundo real. O modelo de entidade

também consegue um grande grau de independência dos dados, mas a forma visualizar os dados não é tão fácil para algumas pessoas[20].

O modelo entidade-relação adota uma vista mais natural em que o mundo real consiste de entidades e relações, incorpora alguma informação semântica importante sobre o mundo real e consegue ter um grande grau de independência de dados. O modelo entidade-relação é baseado na teoria das relações e foi desenvolvido com o objetivo de servir de base para um sistema de visualização de dados unificado[20].

O diagrama entidade relação é uma representação gráfica da análise de requisitos realizada anteriormente baseada no modelo entidade-relação. Este diagrama não é determinístico pois, para uma mesma análise, podem nascer diferentes diagramas que cumprem todos os requisitos.

Um diagrama é constituído elementos como entidades, atributos e relações. Uma entidade é algo que existe no mundo real como uma pessoa ou um carro. Um atributo é uma característica da entidade como uma pessoa tem um nome e um carro tem uma matricula. Uma relação é como uma ou mais entidades interagem entre si como uma pessoa tem um carro[20].

Passando à notação, as entidades são representadas por caixas retangulares, os atributos por caixas ovais e as relações por caixas em forma de losango como mostra a Figura 2.4. Os atributos sublinhados representam a chave primária da entidade.

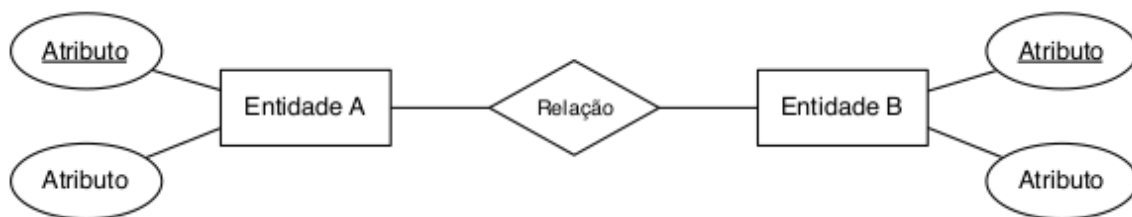


Figura 2.4: Representação de entidades, relações e atributos

As entidades e relações podem ser fortes e fracas. Uma entidade forte não depende de outras entidades, enquanto uma entidade fraca necessita de ser identificada em conjunto com uma entidade forte. As relações fortes definem a ou as entidades fortes que identificam uma entidade fraca entre as quais esta se relaciona. As entidades fracas são representadas por caixas retangulares com linha dupla e as relações fortes são identificadas por um losango de linha dupla como representado na Figura 2.5.



Figura 2.5: Representação de entidades e relações fortes e fracas



Para completar a relação entre entidades é necessário identificar também o grau, cardinalidade e obrigatoriedade de participação de uma relação. Quanto ao grau uma relação pode ser unária, binária ou ternária como representado na Figura 2.6. As relações ternárias podem ser decompostas em relações binárias.

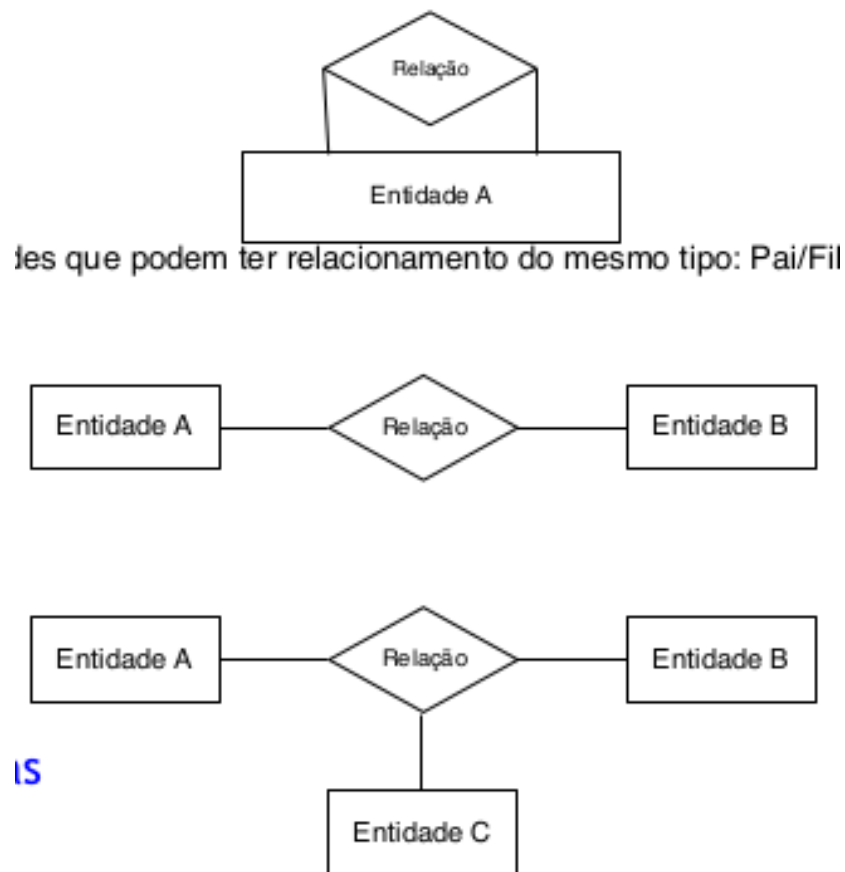


Figura 2.6: Representação dos vários graus das relações (NECESSITA EDIÇÃO)

As relações podem também ser múltiplas, ou seja, mais do que uma relação entre entidades como demonstrado na Figura 2.7.

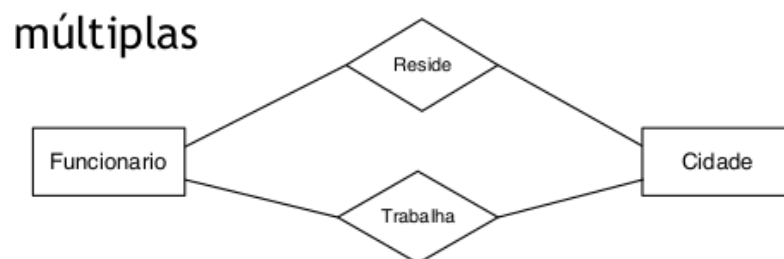


Figura 2.7: Representação de múltiplas relações entre duas entidades

Quanto à cardinalidade uma relação pode ser de três tipos:

- 1 para 1
- N para 1
- N para M

A Figura 2.8 apresenta uma representação visual destas cardinalidades e a Página 10 apresenta a notação da cardinalidade definida por Chen.

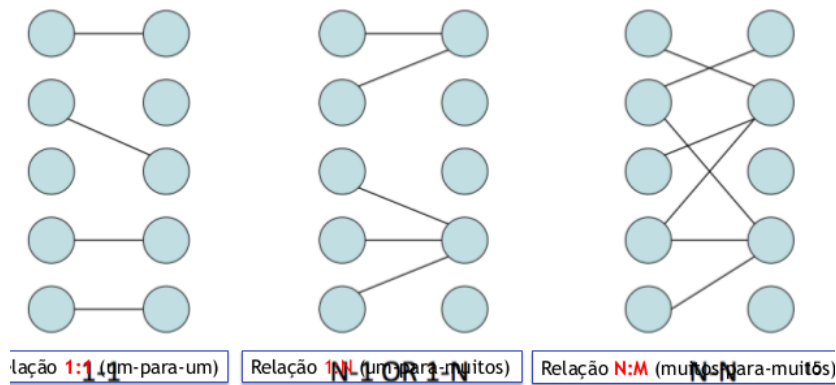


Figura 2.8: Diagrama dos vários tipos de cardinalidade



#### ▪ Exemplos



Um funcionário gere um departamento. Um departamento só tem um gestor.



Figura 2.9: Notação da cardinalidade de Chen

A obrigatoriedade de participação numa relação define se uma entidade tem de participar obrigatoriamente numa relação. Esta é representada por uma linha dupla na conexão com a relação como representado na Figura 2.10.



Figura 2.10: Representação da obrigatoriedade de participação

Os atributos também podem ser de tipos diferentes. Estes podem ser derivados, compostos ou multi-valor como mostra a Figura 2.11

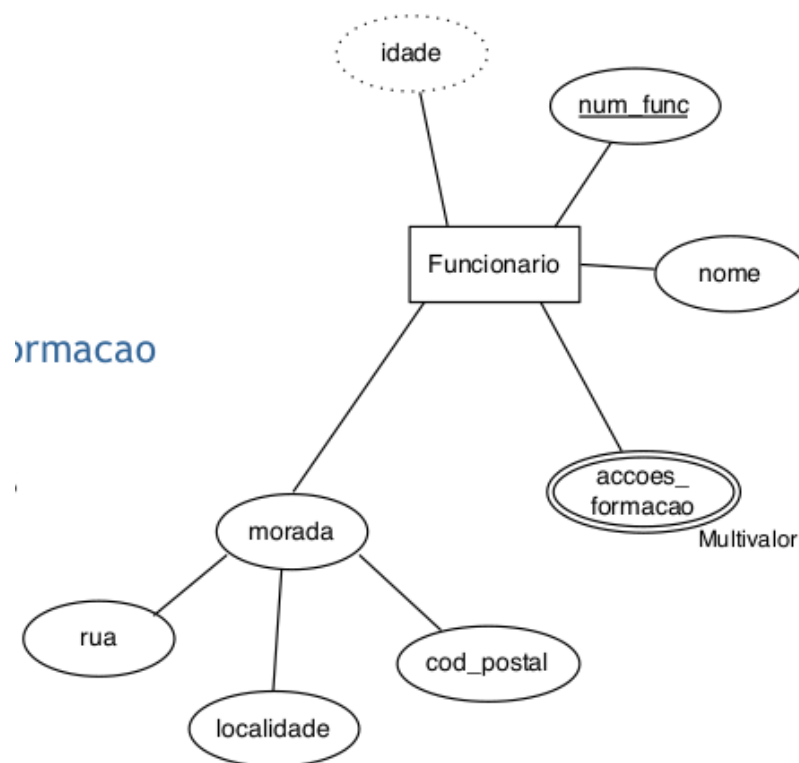


Figura 2.11: Representação dos vários tipos de atributos. Os atributos derivados são representados a tracejado, os compostos têm sub-atributos associados a si e os multi-valor são representados com linha dupla

## 2.2.4 Esquema Relacional

O esquema relacional inclui todas as relações que concretizam uma base de dados. Cada relação é representada pelos seus atributos, tendo os atributos chave sublinhados. Quando atributos de relações diferentes representam o mesmo elemento do mundo real, estes são conectados. Quando um elemento aparece em relações diferentes significa que numa destas, o elemento, é atributo chave da relação e assim sendo, os restantes atributos são considerados chaves estrangeiras. São representados com a ligação ao atributo que serve de chave primária como demonstrado na Figura 2.12[20].

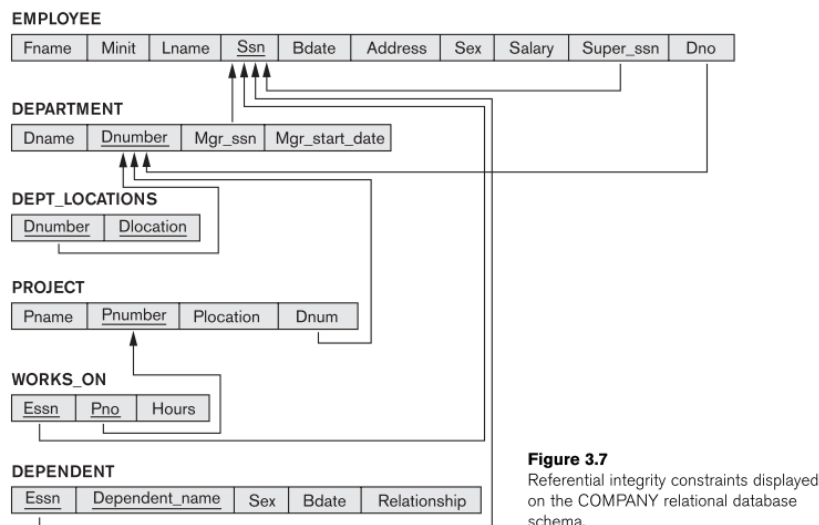


Figura 2.12: Representação de um esquema relacional com as ligações das chaves estrangeiras

Este esquema pode ser deduzido a partir do diagrama entidade relação onde as entidades se traduzem em relações e as relações do diagrama representam as chaves estrangeiras no esquema relacional.

## 2.3 Structured Query Language

A linguagem *SQL* é considerada uma das maiores razões para o sucesso comercial das bases de dados relacionais. Porque se tornou normal a sua utilização em bases de dados relacionais, os utilizadores tiveram menos preocupações a transferir as suas aplicações de outros tipos de bases de dados - por exemplo, bases de dados hierárquicas e em rede - para uma base de dados relacional. Isto porque se um sistema de gestão de bases de dados não cumprisse os requisitos definidos, não era problemático transferir a solução para outro sistema de gestão de bases de dados dado que ambos utilizam a mesma linguagem. Na realidade existem várias sub-versões desta linguagem dependentes de cada sistema de gestão de bases de dados. No entanto, se o utilizador se limitar apenas às funções standard desta linguagem a mudança de sistema é bastante simplificada. Outra vantagem de ter uma linguagem normalizada é que uma aplicação pode utilizar informação de bases de dados em sistemas de gestão de bases de dados diferentes ao mesmo tempo[17].

O nome *SQL* atualmente significa *Structured Query Language*. Originalmente era chamado *SQUEL* de *Structured English QUery Language*, foi desenhado e implementado no *IBM Research* como uma interface experimental para o sistema de bases de dados relacionais *SYSTEM R*. *SQL*. Um esforço conjunto entre o *American National Standards Institute (ANSI)* e a *International Standards Organization (ISO)* conduziu à normalização da versão do *SQL* chamado *SQL-86* ou *SQL1*. Surgiu depois uma versão expandida chamada *SQL-92* ou *SQL2*. A seguinte versão reconhecida é o *SQL:1999* ou *SQL3*. Dois *updates* de depois resultaram nas versões *SQL:2003* e *SQL:2006*[17]. Sendo a atualização mais recente em 2016.

O *SQL* é uma linguagem básica para realizar *queries* dividida em dois campos principais, *DDL* e *DML*. *DDL* ou *Data Definition Language* são as *queries* que permitem definir e construir

uma base de dados. *DML* ou *Data Manipulation Language* são as *queries* que permitem interagir com os dados[17]. Estas serão mais exploradas na Subseção 2.4.2.

## 2.4 *Software* utilizado

Para desenvolver as redes de bases de dados e aplicação escolheram-se os *softwares* e linguagens apresentados neste capítulo. Como a solução será futuramente implementada em ambiente empresarial, definiu-se que não se deve usar *softwares* que estejam em versão *beta*. Além disto sugere-se que sejam também gratuitos.

### 2.4.1 *Ubuntu*



Figura 2.13: Logótipo do *Ubuntu*

Escolheu-se o *Ubuntu* 16.04LTS como sistema operativo para os sistemas usados neste projeto, representado na Página 13. Este é uma distribuição do *Linux* baseado no *Debian*. É um sistema operativo grátis e *open source* desenvolvido pela *Canonical*. A liberdade deste sistema operativo na área da programação criou uma comunidade de utilizadores que ajudam a pesquisar e desenvolver este sistema operativo.[21].

A *Canonical* encarrega-se de garantir *updates* de segurança e performance de forma regular. Apesar de não ser infalível o *Linux* é um dos sistemas mais estáveis e menos provável de ser afetado por vírus, dado que a maior parte destes são desenhados para afetar sistemas operativos mais populares como o *Windows*.

### 2.4.2 *MySQL*



Figura 2.14: Logótipo do *MySQL*

Escolheu-se o *MySQL*, representado na Figura 2.14, como sistema de gestão de bases de dados relacionais para construir e utilizar as bases de dados usadas neste projeto, dada a sua simplicidade e velocidade de resposta. Existem outras ofertas gratuitas como o *SQLite* e o *PostgreSQL*. O *SQLite* funciona apenas localmente e não permite uma conexão remota que se procura como solução para o problema. O *PostgreSQL* é um sistema robusto que permite realizar tarefas que o *MySQL* não consegue realizar. No entanto, dada a simplicidade da base de dados prevista na fase de projeto, não é necessário usar uma ferramenta tão completa e robusta[22].

O *MySQL* é o mais popular e o sistema de gestão de bases de dados relacionais e o mais usado[22]. Garante um acesso múltiplo, rápido e robusto a uma base de dados no servidor. Foi desenvolvido para lidar com grandes quantidades de informação e *softwares* com utilização massiva[23].

O *MySQL* oferece soluções gratuitas e pagas, *Community* e *Enterprise*, respetivamente. A primeira recebe menos *updates* e correções que a solução paga. Tem também algumas funcionalidades bloqueadas e um limite na capacidade de sensivelmente 4Gb[23]. Estas limitações não afetam o projeto e como tal opta-se pela solução gratuita deste *software*.

Como referido anteriormente os vários sistemas de gestão de bases de dados possuem as suas próprias sub-versões da linguagem *SQL* e, o *MySQL*, não é exceção. Existem vários comandos presentes nas componentes *DDL* e *DML* desta linguagem, a seguinte lista enumera os comandos principais utilizados neste projeto:

- **SHOW DATABASES/TABLES** - mostra todas as bases de dados ou tabelas da base de dados (*MySQL*)
- **CREATE DATABASE/TABLE** - criar bases de dados ou tabelas
- **DROP DATABASE/TABLE** - eliminar bases de dados ou tabelas
- **SELECT FROM** - visualizar tuplos de uma tabela
- **INSERT** - inserir tuplos numa tabela

- **INTO** - se forem inseridos múltiplos tuplos e um não respeitar as restrições de integridade definidas, nenhum tuplo é inserido
- **IGNORE** - se forem inseridos múltiplos tuplos e um não respeitar as restrições de integridade, esse é descartado e os restantes são inseridos na tabela (*MySQL*)
- **DELETE** - elimina tuplos de uma tabela
- **UPDATE** - altera tuplos de uma tabela
- **GRANT** - dar privilégios a um utilizador

A *querie* do tipo **SELECT** permite adicionar condições do tipo:

- **WHERE** - definir parâmetros de busca de um atributo
- **GROUP BY** - permite agrupar informação permitindo operações matemáticas sobre valores, por exemplo: contagem, soma, média, etc.
- **ORDER BY** - escolher a ordem dos tuplos

As *queries* do tipo **DELETE** e **UPDATE** também permitem utilizar a condição **WHERE**.

Na criação das tabelas atribuem-se o nome dos atributos e o seu respetivo domínio. O *MySQL* oferece vários tipos de dados, dos quais foram usados:

- **VARCHAR(n)** - *string* com tamanho máximo n
- **INT** - inteiro entre -2147483648 e +2147483647
- **TINYINT** - inteiro entre -128 e +127
- **FLOAT** - numérico com casa decimais
- **DATETIME** - data e hora no formato AAAA-MM-DD HH:MM:SS

Define-se também as restrições de integridade:

- **PRIMARY KEY** - define chave primária
- **UNIQUE** - define chave única
- **FOREIGN KEY REFERENCES** - define a chave estrangeira e o atributo de referência

Estas restrições ou **CONSTRAINTS** podem e devem ser atribuídas um nome de forma a identifica-las em mensagens de erro. Além disto é possível definir o comportamento de uma restrição de uma chave estrangeira quando o atributo de referencia é alterado (**ON UPDATE**) ou eliminado (**ON DELETE**):

- **NO ACTION** - não deixa realizar a ação
- **CASCADE** - todas as chaves estrangeiras dependentes do atributo de referencia são alteradas ou eliminadas de acordo com este
- **SET NULL** - altera o valor da chave estrangeira para NULL

- **SET DEFAULT** - altera o valor da chave estrangeira para um valor predefinido

O valor **NULL** representa um valor de um atributo que não existe ou é desconhecido, adiciona-se **NOT NULL** a um atributo quando este não pode assumir este valor.

Estes são os comandos e conceitos principais que surgem ao longo deste documento. Como referido anteriormente e é possível observar aqui, o *SQL* é uma linguagem simples e auto-explicativa.

### 2.4.3 *C*



Figura 2.15: Logótipo do *GNU*

Escolheu-se utilizar *C* no desenvolvimento deste projeto devido à afinidade com esta linguagem e com o facto do *MySQL* disponibilizar protocolos oficiais e documentação de comunicação com esta linguagem[23].

O *C* é uma linguagem para todos os tipos de problemas e importante no mundo da programação. Originalmente desenvolvido por Dennis Ritchie entre 1969 e 1973 no *Bell Labs* e usado para re-implementar o sistema operativo *Unix*. Desde então esta tornou-se uma das linguagens de programação mais popular de sempre com uma oferta de vários compiladores existentes no mercado. O *C* foi então normalizado pelo *ANSI* em 1989 e consequentemente pelo *ISO*.

No projeto usa-se o *GNU Compiler Collection (GCC)*, representado na Figura 2.15, como compilador dos programas em *C*. Este foi desenvolvido primeiramente para o sistema operativo *GNU* e focou-se em ser gratuito e garantir liberdade de programação ao utilizador. Conta com uma comunidade ativa que desenvolvem constantemente novas soluções e *updates* regulares ao compilador de forma a que este não fique desatualizado[24].



#### 2.4.4 *Apache*



Figura 2.16: Logótipo do *Apache*

Escolheu-se utilizar *Apache HTTP Server*, representado na Figura 2.16 como servidor *HTTP* para a aplicação *Web*. Este foca-se em ser um servidor gratuito e garantir uma solução segura, eficiente e extensível[25].

Lançado em 1995, tornou-se no *software* mais utilizado pelas empresas para correr os seus *sites*. Atualmente na versão 2.4, realizam *updates* constantes ao programa para o manter estável e atualizado[25].

#### 2.4.5 *HTML, JS e PHP*



Figura 2.17: Representação da ferramenta mais usada para desenvolver aplicações em ambiente *Web*

Escolheu-se *HTML*, *JS* e *PHP* para desenvolver a aplicação *Web* deste projeto. Junto com *SQL* e *CSS* formam o conjunto de ferramentas mais usadas no desenvolvimento de aplicações *Web* no mercado, representado na Figura 2.17.

*HTML* ou *Hyper Text Markup Language* descreve a estrutura da página *Web* desenvolvida. Possui elementos que funcionam como blocos para as páginas permitindo a criação de uma interface com um utilizador. *CSS* ou *Cascading Style Sheets* descreve como os elementos *HTML* devem ser dispostos na página, permitindo o desenvolvimento de uma interface mais apelativa. *JS* ou *JavaScript* permite interagir e alterar o código *HTML* e *CSS* criando uma interface mais interativa. Permite também desenvolver códigos e funções que são executadas no browser do cliente. *PHP* ou *PHP:Hypertext Preprocessor* permite desenvolver funções para aplicação que executam do lado do servidor. *JS* e *PHP* realizam ambas funções diferindo apenas no local onde são executadas. Sem contar com a interação com o *HTML* e *CSS*, o *PHP* pode substituir o *JS* como linguagem para criação de funções do sistema. No entanto, isto pode causar problemas de desempenho dado que o servidor terá de correr funções de todos os utilizadores. Assim sendo define-se que o uso do *PHP* deve ser limitado a funções de sistema mais importantes, como conectar a uma base de dados em *SQL*, e o *JS* deve ser usado no desenvolvimento de funções de aplicação mais triviais.

## Capítulo 3

# Proposta de Solução

A infraestrutura de dados desenvolvida tem como objetivo ser simples, funcional a baixo nível e sem impor restrições na instrumentação dos moldes. Este capítulo descreve o processo de desenvolvimento das bases de dados, o método de transferência de medições, a gestão de *backups*, as permissões dos utilizadores e um simples simulador de valores.

### 3.1 Infraestrutura de Dados

A instrumentação dos moldes geram valores localmente nos clientes e estes devem ser guardados numa base de dados central. A transferência é realizada usando protocolos TCP/IP. A fim de minimizar a perda de valores em caso de falha de conexão, coloca-se uma base de dados local em cada cliente, como representado na Figura 3.1.

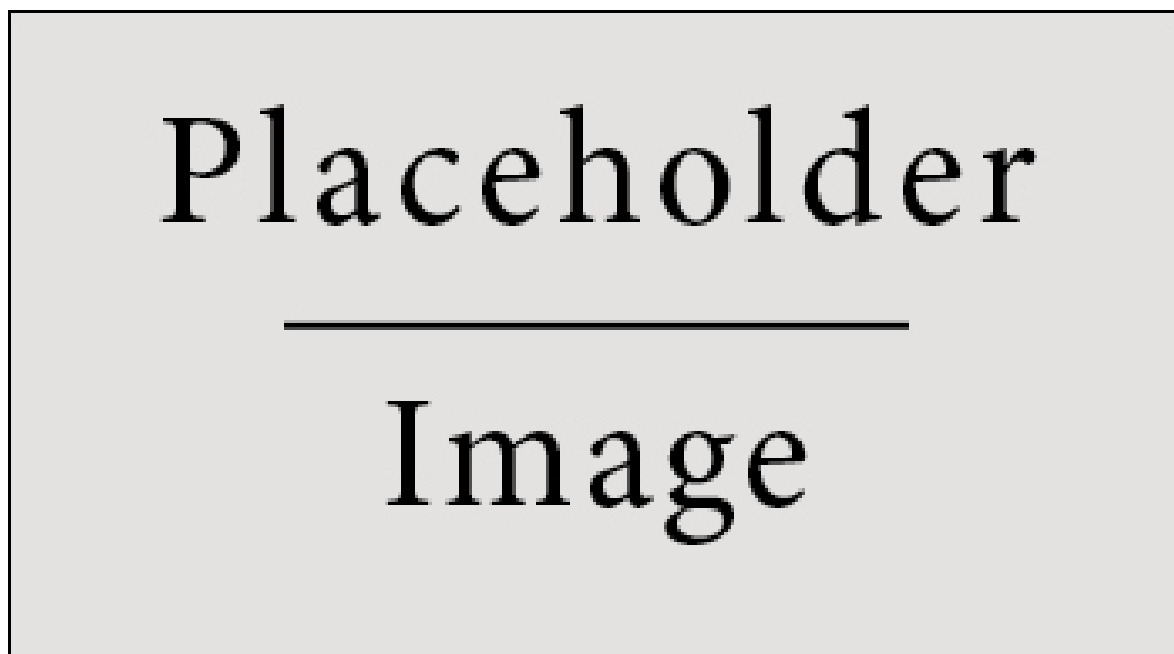


Figura 3.1: Diagrama da infraestrutura com múltiplas bases de dados locais conectadas a uma base de dados central

Para estabelecer o fluxo de informação desenvolveu-se um programa de transferência e um simulador, ambos desenvolvidos em *C*. O programa de transferência envia *queries* para as bases de dados locais e recebe respostas com valores que regista na base de dados central. O simulador gera valores para popular as bases de dados locais. Para simular múltiplos clientes usam-se simuladores independentes entre si, como representado na Figura 3.2.



Figura 3.2: Diagrama do fluxo de informação. Os simuladores substituem provisoriamente sistemas de aquisição de dados

Para melhorar o desempenho da base de dados central e prevenir falhas críticas do sistema, desenvolveu-se um programa de gestão de *backups*, também em *C*. Este programa limpa e cria *backups* da base de dados e guarda-os num repositório *online*. De forma a não interferir diretamente com a informação da base de dados central criam-se duas bases de dados para a gestão dos *backups*. Com esta adição, o esquema da infraestrutura final está representado na Figura 3.3.



Figura 3.3: Diagrama final da infraestrutura de dados. Os valores gerados nos simuladores são transferidos para a base de dados central. Estes valores são então processados pelo gestor de *backups* que cria e guarda *backups* provisoriamente num repositório online no *GitHub*

## 3.2 Base de Dados

### 3.2.1 Análise de Requisitos

Deseja-se, de uma forma geral, um sistema capaz de guardar remotamente medições realizadas em moldes instrumentados. Esta informação não é suficiente para se deduzir uma base

de dados. Assim sendo, analisando e completando a informação inicial chega-se ao seguinte enunciado:

*Um cliente, caracterizado por um ID, nome, morada, IP e port, tem moldes. Estes moldes são identificados por um ID, nome e descrição. Os moldes têm sensores com um número referente ao molde, tipo (termómetro, dinamómetro, extensómetro, etc.), nome e descrição. Estes sensores geram registos onde guardam a fase do processo (fecho do molde, enchimento, compactação, abertura e extração), data, hora, milissegundos e o valor num determinado momento.*

Analisando este enunciado identificam-se as seguintes entidades com os seus respetivos atributos:

- Tipo
  - ID
  - Nome
- Fase
  - ID
  - Nome
- Clientes
  - ID de cliente único
  - Nome
  - Morada
  - IP
  - Port
- Moldes
  - ID do cliente associado
  - ID de molde único
  - Nome
  - Descrição
- Sensores
  - ID do molde associado
  - Número do sensor
  - ID do tipo de sensor
  - Nome
  - Descrição
- Registos
  - ID do molde associado
  - Número do sensor associado
  - ID da fase do processo
  - Data
  - Hora
  - Milissegundos
  - Valor

Analisando os atributos identificam-se as seguintes chaves primárias, únicas e estrangeiras:

- Tipo
  - Chave primária: ID
  - Chaves únicas:
    1. Nome
- Fase
  - Chave primária: ID
  - Chaves únicas:
    1. Nome
- Clientes
  - Chave primária: ID de cliente
- Moldes
  - Chave primária: ID de molde
  - Chaves estrangeiras:
    1. ID do cliente associado
- Sensores
  - Chave primária: ID do molde associado, Número do sensor
  - Chaves estrangeiras:
    1. ID do molde associado
    2. ID do tipo de sensor
- Registos
  - Chave primária: ID do molde associado, Número do sensor associado, Data, Hora, Milissegundos
  - Chaves estrangeiras:
    1. ID do molde associado, Número do sensor associado
    2. ID da fase do processo

O tipo de sensor e a fase do processo são dicionários ou seja, são entidades que contêm informação predefinida de forma a evitar a adição de dados incorretos nas entidades relacionadas. Os seus dados iniciais são:

- Tipo
  - Termómetro
  - Dinamómetro
  - Extensómetro
  - Vibrómetro
  - Pressão
  - Acelerómetro X
  - Acelerómetro Y
  - Acelerómetro Z
- Fase
  - Fecho do molde

- Enchimento
- Compactação
- Abertura de molde
- Extração

Ligando as várias entidades entre si identificam-se as seguintes relações com a respetiva cardinalidade:

- 1 cliente tem N moldes
- 1 molde tem N sensores
- 1 sensor tem N registos
- 1 tipo define N sensores
- 1 fase define N registos

Quanto à obrigatoriedade de participação das entidades nas relações anteriores afirma-se:

- Não pode existir moldes sem clientes
- Não pode existir sensores sem moldes
- Não pode existir registos sem sensores
- Não pode existir sensores sem tipo
- Não pode existir registos sem fase

### 3.2.2 Desenho conceptual e esquema lógico

Concatenando a informação descrita na análise de requisitos, obtém-se o diagrama entidade/relação representado na Figura 3.4.



Figura 3.4: Diagrama Entidade/Relação realizado a partir da análise de requisitos

Traduzindo as entidades do diagrama anterior para tabelas e as relações para chaves estrangeiras, obtém-se o esquema lógico da base de dados representado na Figura 3.5.

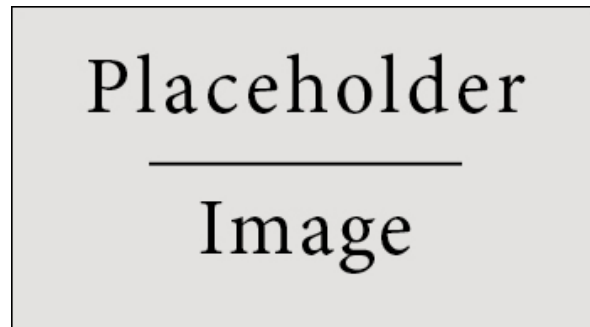


Figura 3.5: Esquema Lógico obtido através da análise do diagrama entidade/relação onde as entidades e os seus atributos se traduzem em tabelas e as relações nas chaves estrangeiras destas

### 3.2.3 Construção da base de dados

Avançando para a criação das tabelas em *MySQL*, a fim de evitar ambiguidades no nome dos atributos estes são alterados para campos semânticos específicos a cada entidade a que pertencem. Para cada atributo define-se o domínio e a obrigatoriedade como representado na Tabela 3.1.



tipo					
tipo_ID	tipo_nome				
int	varchar(50)				
not null	not null				
fase					
fase_ID	fase_nome				
int	varchar(50)				
not null	not null				
clientes					
cl_ID	cl_nome	cl_morada	cl_IP	cl_port	
int	varchar(50)	varchar(100)	varchar(50)	int	
not null	not null		not null	not null	
moldes					
m_IDCliente	m_ID	m_nome	m_descricao		
int	int	varchar(30)	varchar(200)		
not null	not null				
sensores					
s_IDMolde	s_num	s_tipo	s_nome	s_descricao	
int	int	int	varchar(30)	varchar(200)	
not null	not null	not null			
registos					
r_IDMolde	r_numSensor	r_fase	r_data_hora	r_milissegundos	r_valor
int	int	int	datetime	tinyint	float
not null	not null	not null	not null	not null	

Tabela 3.1: Tabelas com os seus atributos e respetivos domínios e obrigatoriedades. Os atributos data e hora da entidade registos são fundidos no atributo r\_data\_hora graças ao tipo de dados datetime

Define-se então nomes para as restrições de integridade definidas pelas chaves das entidades:

- Tipo
  - Restrição chave primária: REPETIDO\_ID\_TIPO
  - Restrição chaves únicas:
    1. REPETIDO\_NOME\_TIPO
- Fase
  - Restrição chave primária: REPETIDO\_ID\_FASE
  - Restrição chaves únicas:
    1. REPETIDO\_NOME\_FASE
- Clientes
  - Restrição chave primária: REPETIDO\_ID\_CLIENTE
- Moldes
  - Restrição chave primária: REPETIDO\_ID\_MOLDE

- Restrição chaves estrangeiras:
  1. MOLDE\_MAU\_ID\_CLIENTE
- Sensores
  - Restrição chave primária: REPETIDO\_NUM\_SENSOR
  - Restrição chaves estrangeiras:
    1. SENSOR\_MAU\_ID\_MOLDE
    2. SENSOR\_MAU\_ID\_TIPO
- Registos
  - Restrição chave primária: REPETIDO\_REGISTO
  - Restrição chaves estrangeiras:
    1. REGISTOS\_MAU\_ID\_MOLDE\_SENSOR
    2. REGISTOS\_MAU\_ID\_FASE

Em relação ao comportamento das restrições das chaves estrangeiras identificam-se as seguintes opções:

- ON DELETE NO ACTION ON UPDATE CASCADE
  - MOLDE\_MAU\_ID\_CLIENTE
  - SENSOR\_MAU\_ID\_TIPO
  - REGISTOS\_MAU\_ID\_FASE
- ON DELETE CASCADE ON UPDATE CASCADE
  - SENSOR\_MAU\_ID\_MOLDE
  - REGISTOS\_MAU\_ID\_MOLDE\_SENSOR

Estes comportamentos definem que quando se atualiza a chave primária de um tuplo (*UPDATE*) os restantes tuplos dependentes desta informação devem devidamente atualizados. No entanto, quando se elimina um tuplo são definidos dois comportamentos. Como existe muita informação associada a cada cliente, como segurança, não se permite a eliminação de um tuplo desta tabela se este tiver informação associada a si noutras tabelas. O mesmo se aplica aos dicionários, se se tentar eliminar um tuplo, a informação associada não deve ser comprometida. No caso dos registos, dado que estão associados diretamente a um molde, se um tuplo da tabela moldes for eliminado, os dados dos seus sensores e respetivos registos devem ser também eliminados para manter a base de dados limpa e sem informação desnecessária.

Reverendo as restrições de integridade criadas pelas chaves estrangeiras, como não se pode criar uma tabela que dependa de outra que não tenha sido ainda criada, define-se uma ordem para a criação das tabelas:

1. Tipo
2. Fase
3. Clientes
4. Moldes
5. Sensores
6. Registos

Este modelo de dados aplica-se a todas as bases de dados usadas neste projeto com a exceção da base de dados temporária local e notificações que serão abordadas na Seção 4.1.

### 3.2.4 Programa de transferência

Este programa prioriza a conservação dos dados e a sua taxa de transferência. O código foi realizado com vista à portabilidade para outras linguagens de programação. Consiste em várias ideias simples que podem ser observadas na Figura 3.6.



Figura 3.6: fluxograma

Seguindo a estrutura do programa, primeiramente realiza-se uma conexão à base de dados central, seguida de uma *query* para se saber o número de clientes existentes:

```
SELECT *  
FROM clientes;
```

Guarda-se o número de tuplos retornados numa variável e termina-se a conexão à base de dados central. De seguida replica-se o programa múltiplas vezes até se ter um subprograma para cada cliente com recurso à função:

```
fork();
```

Atribui-se a cada subprograma o número do cliente a que está associado. Cada um realiza uma conexão à base de dados central e obtém os respetivos *IPs* e *ports*, necessários para realizar a conexão à base de dados local de cada cliente:

```
SELECT cl_ID, cl_IP, cl_port  
FROM clientes  
ORDER BY cl_ID;
```

Com as conexões central e local estabelecidas inicia-se um ciclo infinito para a transferência de valores. Com a *query* à base de dados local:

```
SELECT CURRENT TIMESTAMP;
```

Recebe-se a data e hora atual do sistema e guarda-se numa variável @datahora\_lim. De seguida obtém-se todos os registos anteriores à data e hora do sistema:

```
SELECT *  
FROM registos  
WHERE r_data_hora < @datahora_lim  
ORDER BY r_data_hora, r_milissegundos,  
r_numSensor, r_IDMolde;
```

Esta organização pela data e hora garante uma transferência uniforme de valores. Por predefinição, o sistema retorna os registos um sensor de cada vez, tornando a transferência de valores menos eficiente para efeitos de análise. Por outras palavras, prefere-se saber os registos de todos os sensores num dado instante do que todos os registos de um sensor até ao momento. Os tuplos retornados são guardados numa *string* e inseridos na base de dados central com uma *query* do tipo:

```
INSERT IGNORE registos VALUES
(tuplo1),
(tuplo2),
(tuplo3),
...,
(tuploN);
```

Inserir múltiplos tuplos permite uma maior taxa de transferência comparativamente a uma inserção individual. A *string* que guarda os tuplos tem uma dimensão limitada, se a resposta proveniente da base de dados local for superior ao tamanho da *string*, o programa divide-a em grupos de informação mais pequenos que respeitem a dimensão desta *string*. A escolha da opção **IGNORE** garante que não são perdidos valores no caso de serem gerados registos repetidos ou não válidos. Após a transferência ser concluída, os tuplos transferidos da base de dados local são eliminados:

```
DELETE FROM registos
WHERE r_data_hora < @datahora_lim;
```

Numa utilização a baixo nível, sempre que se desejar um subprograma para um cliente novo, o programa de transferência tem de ser reiniciado manualmente. Reinicia-lo termina o ciclo infinito de transferência de valores, encerra as conexões com as bases de dados central e local e volta a iniciar todo o processo descrito nesta secção.

### 3.2.5 Gestão de *backups*

À semelhança do programa anterior, o programa de gestão de *backups* foi realizado com vista à portabilidade para outras linguagens de programação. De forma a organizar a informação armazenada, em vez de se gerar um *backup* de toda a base de dados, gera-se um *backup* específico para cada molde, que inclui a informação dos seus sensores, registos e cliente a que está associado. Além destes, realiza-se um outro *backup* com a informação dos clientes, moldes e sensores mais atual. Assim, em caso de falha crítica, simplifica-se a recuperação do sistema.

Ao contrário do programa anterior, este programa não executa de forma contínua. Define-se um temporizador para executar o código principal. A Figura 3.7 representa a estrutura do programa sempre que o temporizador chega ao fim.



Figura 3.7: fluxograma

Seguindo a estrutura do programa, realizam-se as conexões às bases de dados central, *backups* e *backups* temporária. Com a *query*:

```
SELECT CURRENT TIMESTAMP;
```

Recebe-se a data e hora atual do sistema e guarda-se numa variável @datahora\_lim. Para proteger os dados presentes na base de dados central, realiza-se a cópia dos registos para a base de dados *backups*:

```
INSERT IGNORE backups.clientes  
SELECT *  
FROM central.clientes;  
INSERT IGNORE backups.moldes  
SELECT *  
FROM central.moldes;  
INSERT IGNORE backups.sensores  
SELECT *  
FROM central.sensores;  
INSERT IGNORE backups.registos  
SELECT *  
FROM central.registos  
WHERE r_data_hora < @datahora_lim;
```

De forma a gerar um *backup* para cada molde filtra-se a informação com base no seu ID:

```
SELECT m_IDCliente, m_ID  
FROM backups.moldes;
```

As respostas são guardadas nas variáveis @IDCliente e @IDMolde. De forma cíclica copia-se a informação para a base de dados *backups* temporária:

```
INSERT IGNORE backups_temp.clientes  
SELECT *  
FROM backups.clientes  
WHERE cl_ID = @IDCliente;  
INSERT IGNORE backups_temp.moldes  
SELECT *  
FROM backups.moldes
```

```

WHERE m_ID = @IDMolde;
INSERT IGNORE backups_temp.sensores
SELECT *
FROM backups.sensores
WHERE m_ID = @IDMolde;
INSERT IGNORE backups_temp.registos
SELECT *
FROM backups.registos
WHERE m_ID = @IDMolde;

```

Após os valores copiados retiram-se informações sobre a data e hora mais antiga e mais recente:

```

SELECT r_data_hora
FROM backups_temp.registos
ORDER BY r_data_hora
LIMIT 1;
SELECT r_data_hora
FROM backups_temp.registos
ORDER BY r_data_hora DESC
LIMIT 1;

```

Os valores retornados são guardados nas variáveis @dataIni e @dataFim. Com estes valores e com a função:

```
FILE *popen(const char *command, const char *mode);
```

Envia-se para o terminal o comando que gera os *backups* num local predefinido:

```
mysqldump -u backupmanager -pbackup1234 backups_temp >
~/Backups/backup_@IDCliente_@IDMolde_@dataIni_@dataFim.sql
```

Depois do ficheiro ser criado com sucesso elimina-se das bases de dados *backups* e *backups* temporária a informação relativa ao molde:

```

DELETE FROM backups_temp.moldes;
DELETE FROM backups_temp.clientes;
DELETE FROM backups.registos
WHERE r_IDMolde = @IDMolde;

```

Quando todos os moldes tiverem sido armazenados, a tabela registos da base de dados *backups* deve estar vazia. Realiza-se então o *backup* da restante informação dos clientes, moldes, sensores, tipo e fase para efeitos de recuperação do sistema:

```
mysqldump -u backupmanager -pbackup1234 backups_temp >
~/Backups/backup_geral.sql
```

Definiu-se um nome estático para este *backup*. Este será rescrito cada vez que o programa executa enquanto os *backups* individuais dos moldes vão acumulando com o tempo. Termina-se o processo apagando os valores armazenados na base de dados central:

```

DELETE FROM central.registos
WHERE r_data_hora < @datahora_lim;

```

Os *backups* são então transferidos para o repositório *online* no *GitHub*. Esta solução serve apenas para demonstrar a capacidade de mover os *backups* para um sistema diferente. Do

ponto de vista prático, não se recomenda guardar informação potencialmente sensível num servidor público.

Na eventualidade de surgir uma falha crítica no sistema central, e este necessitar de ser formatado ou substituído, os registos cessarão de ser transferidos dos sistemas locais. Isto não significa que a informação esteja comprometida, apenas não está a ser transferida. Após re-instalar utiliza-se manualmente no terminal o comando:

```
mysql -u backupmanager -pbackup1234 central <
~/Backups/backup_geral.sql
```

Isto faz com que se recupere a informação dos clientes, moldes e sensores presentes no último *backup* de forma a acelerar o processo de restauro comparativamente a uma inserção manual dos dados.

### 3.2.6 Simulador

Para popular as bases de dados locais foi desenvolvido um simples programa que gera valores. Estes seguem o perfil de uma onda sinusoidal com a expressão:

$$v = O + A \sin\left(\frac{2\pi x}{t} + \gamma\right) \quad (3.1)$$

Onde  $v$  é o valor calculado,  $O$  o *offset* da onda,  $A$  a amplitude,  $t$  o período em segundos,  $x$  a hora atual em segundos e  $\gamma$  o desfasamento. Inicia-se uma conexão à base de dados local e calcula-se a hora atual com recurso às funções:

```
struct tm *localtime(const time_t *timer);
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Os valores das horas, minutos, segundos e milissegundos são guardados nas variáveis @hora, @min, @seg e @mseg respetivamente. Aplicando a expressão:

$$x = @hora \times 3600 + @min \times 60 + @seg + @mseg \times 0.001 \quad (3.2)$$

Obtém-se a hora atual do sistema em segundos. Com as expressões 3.1 e 3.2 gera-se um valor para cada sensor. Para cada valor gerado atribui-se uma fase do processo. Esta atribuição realiza-se de forma cíclica entre as várias opções disponíveis. Com a informação de cada sensor estabelecida insere-se na base de dados local com uma *query* do estilo:

```
INSERT INTO registos VALUES
(molde1, sensor1, fase, NOW(), @mseg, valor11),
(molde1, sensor2, fase, NOW(), @mseg, valor12),
(molde2, sensor1, fase, NOW(), @mseg, valor21),
...,
(moldeI, sensorN, fase, NOW(), @mseg, valorIN);
```

### 3.2.7 Utilizadores

Para minimizar o risco de erros por parte dos utilizadores que interagem com o sistema, são definidas credenciais com permissões específicas para cada base de dados:

- user, password

- transferencia, transferencia1234
- sensores, sensores1234
- backupmanager, backup1234

*user* representa o utilizador padrão, ou seja, aquele que interage com as bases de dados para gerir os clientes, moldes e sensores bem como consultar a tabela registos. As credenciais *transferencia* utilizam-se no programa de transferência de valores para realizar as conexões com as bases de dados central e locais, *sensores* utiliza-se no simulador para popular as bases de dados locais e, no futuro, pelo sistema de aquisição de dados que substituirá este simulador, *backupmanager* utiliza-se no programa de gestão de *backups*.

Cada conjunto de credenciais serve um objetivo, a Tabela 3.2 contém as permissões destas definidas em cada base de dados.



<b>central</b>						
clientes/tipo/fase/ moldes/sensores	CREATE	DROP	SELECT	INSERT	DELETE	UPDATE
user			X	X	X	X
transferencia						
sensores						
backupmanager			X			
registos	CREATE	DROP	SELECT	INSERT	DELETE	UPDATE
user			X		X	
transferencia				X		
sensores						
backupmanager			X		X	
<b>local</b>						
clientes/tipo/fase/ moldes/sensores	CREATE	DROP	SELECT	INSERT	DELETE	UPDATE
user			X	X	X	X
transferencia						
sensores						
backupmanager						
registos	CREATE	DROP	SELECT	INSERT	DELETE	UPDATE
user						
transferencia			X		X	
sensores				X		
backupmanager						
<b>backups e backups temporária</b>						
clientes/tipo/ fase/moldes/ sensores/registos	CREATE	DROP	SELECT	INSERT	DELETE	UPDATE
user						
transferencia						
sensores						
backupmanager			X	X	X	

Tabela 3.2: Tabelas com as permissões das credenciais criadas para as várias tabelas das bases de dados

Estas permissões estabelecem que nenhuma das credenciais estabelecidas pode fazer DROP às bases de dados o que eliminaria com um comando toda a estrutura e a informação nela contida. Além disto, nenhuma tem permissão INSERT e DELETE simultânea na tabela registos das bases de dados central e local protegendo o sistema de adulteração dos valores presentes nesta tabela.



## Capítulo 4

# Aplicação de Gestão do Sistema

A aplicação foi desenvolvida em ambiente *Web* com o objetivo de ser multiplataforma e permitir acesso remoto e sem recorrer a instalação de *softwares* nos dispositivos dos utilizadores. Corre num servidor *Apache* e foi desenvolvida com *PHP*, *JS* e *HTML*. Este capítulo descreve a adaptação da infraestrutura desenvolvida e as várias funcionalidades da aplicação.

### 4.1 Adaptação da infraestrutura

A infraestrutura desenvolvida no Capítulo 3 visa uma utilização a baixo a nível. Ainda que funcional, pode haver margem para erros e incoerência dos dados introduzidos manualmente. A aplicação minimiza estas incoerências através da instalação de uma nova base de dados temporária local no servidor local. Aqui os utilizadores têm a liberdade para adicionar, alterar e apagar informação sem consequências no sistema antes desta ser introduzida nas bases de dados central e local como representado na Figura 4.1. Como referido anteriormente, esta base de dados difere das restantes, não contendo em si as tabelas fase e registos.

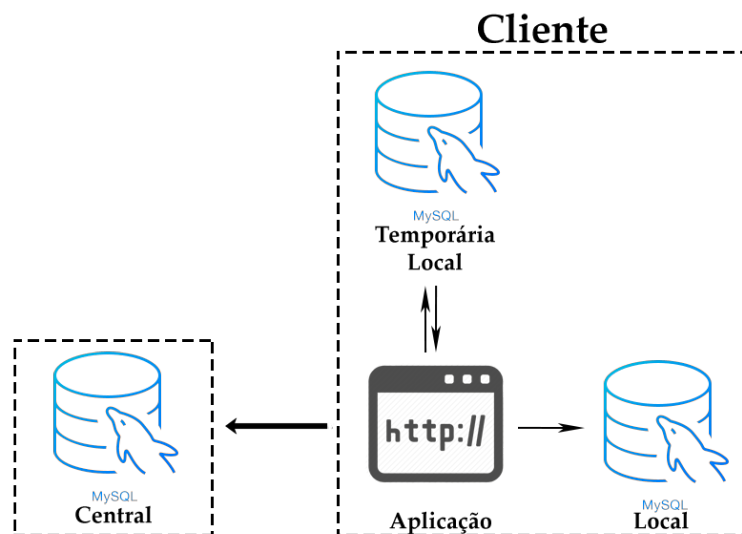


Figura 4.1: Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local

Relembrando, o programa de transferência de valores, numa utilização a baixo nível, tem de ser reiniciado manualmente. No entanto, como a aplicação gere informação dos clientes, moldes e sensores surge a necessidade que este programa possa ser reinicializado pela aplicação. Para este efeito cria-se uma nova base de dados notificações no sistema central. Tem como objetivo conter variáveis globais importantes para a comunicação entre os vários programas. Nesta fase do projeto serve apenas como ponto de ligação entre a aplicação e o programa de transferência. Esta tem uma tabela transferência com um campo atualizar, como demonstrado na Figura 4.2, que pode assumir os valores 0 e 1.



Figura 4.2: Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local

Adapta-se também o programa de transferência de valores de acordo com a Figura 4.3.



Figura 4.3: Fluxograma 2

Define-se um temporizador nos subprogramas de cada cliente para consultarem esta nova base de dados. Se atualizar for igual a 1 os subprogramas terminam as suas rotinas e as conexões às bases de dados. Quando todos tiverem terminado, o programa principal reinicia o processo de criação de novos subprogramas e retorna o parâmetro atualizar para 0. A Figura 4.4 representa a infraestrutura completa numa utilização com a aplicação.



Figura 4.4: Esquema ligação Aplicação-Bases de Dados. A aplicação comunica com a base de dados temporária local e depois regista os seus valores nas bases de dados central e local

## 4.2 Interface gráfica

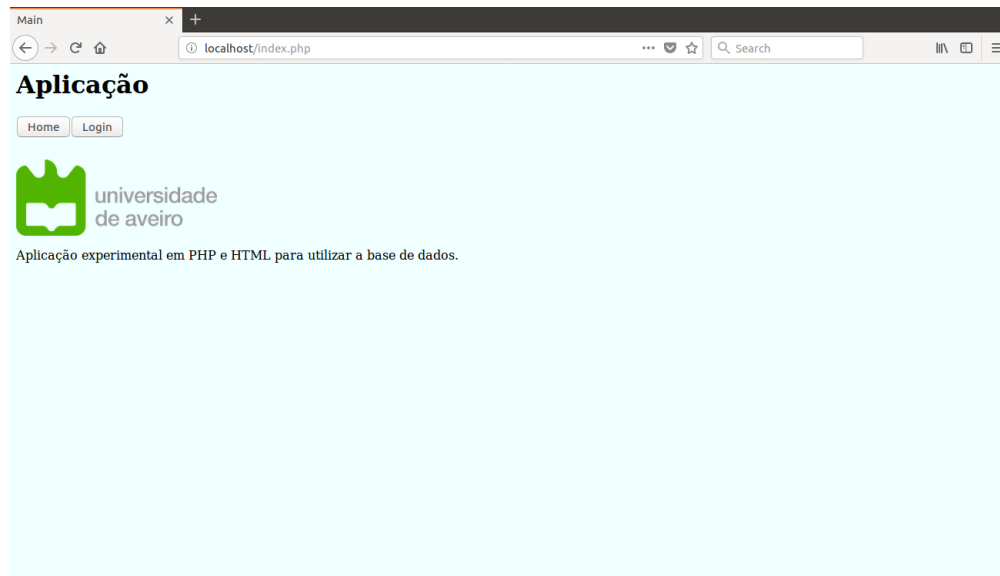
A aplicação divide-se em quatro partes distintas:

- *Main* - Página principal
- *Login* - Página de acesso
- Consultas
- Administração Local

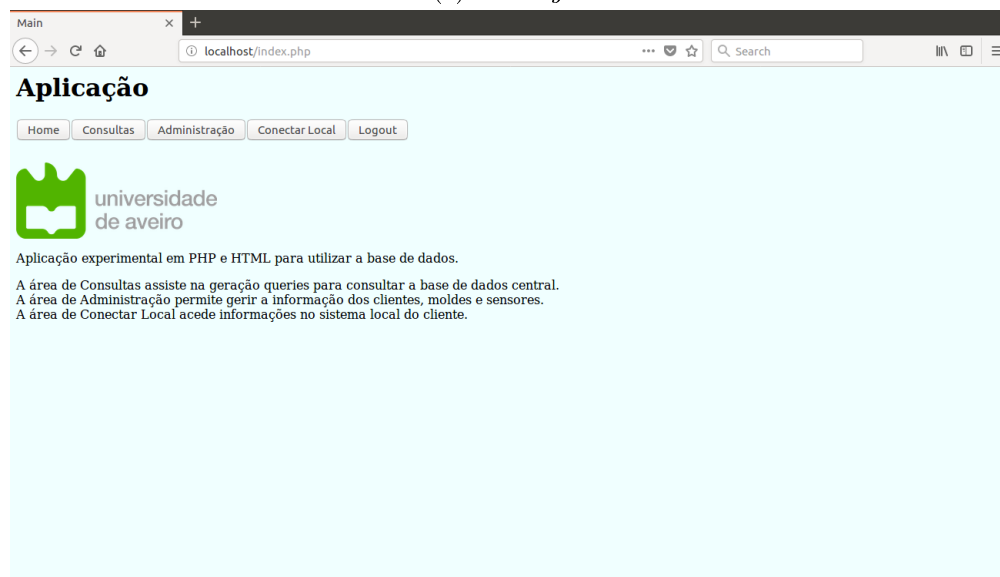
As páginas *Main*, *Login* e Consultas realizadas para uma utilização geral. A página Administração Local foi realizada para uma utilização local. A primeira visa um uso a partir de qualquer dispositivo e acessível a qualquer momento e a segunda foca-se num acesso local com o objetivo de configurar e definir a informação no servidor local. Por outras palavras, para o utilizador usar as funcionalidades desta página tem de aceder à aplicação num *browser* no sistema local que se situa no cliente.

Instalar um molde é o culminar de um projeto de elevada responsabilidade e esta ideia junto com a criação da base de dados temporária local serve para melhorar a qualidade da informação introduzida no sistema e diminuir as falhas no processo de instalação deste.

### 4.2.1 *Main*



(a) Sem *login*



(b) Com *login*

Figura 4.5: Funcionalidades da página *Main* com e sem *login*

*Main* serve como página principal da aplicação. Se não houver sessão iniciada, todas as restantes páginas redirecionam o utilizador para aqui. Contém apenas algumas informações gerais sobre a aplicação.

Iniciar sessão na página de *Login* desbloqueia funcionalidades na aplicação, como demonstrado nas Figuras 4.5a e 4.5b. Depois de iniciada a sessão, navega-se com os botões para as páginas de Consultas e Administração Local.

### 4.2.2 *Login*

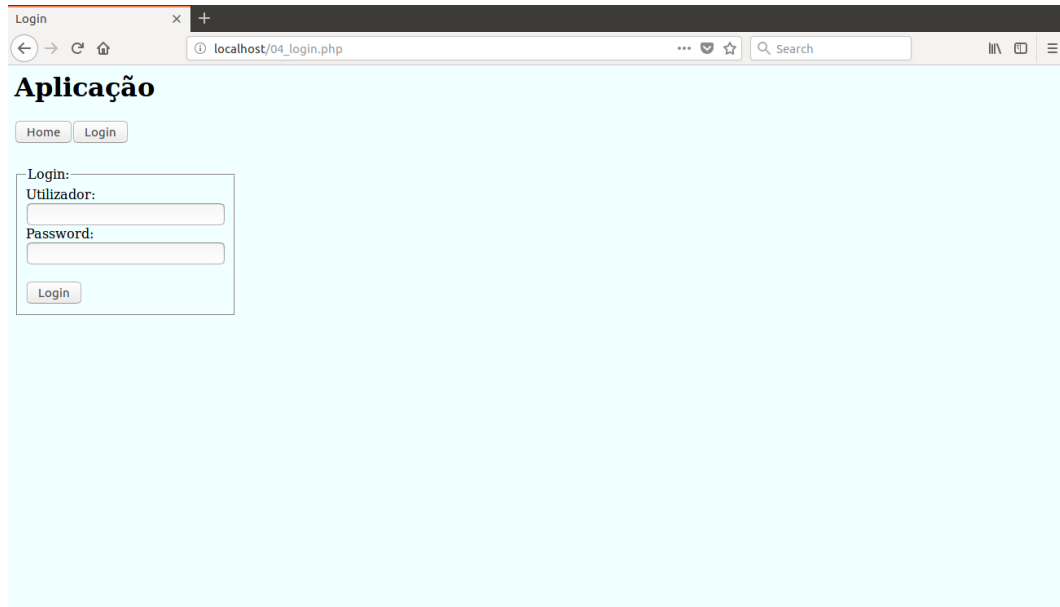
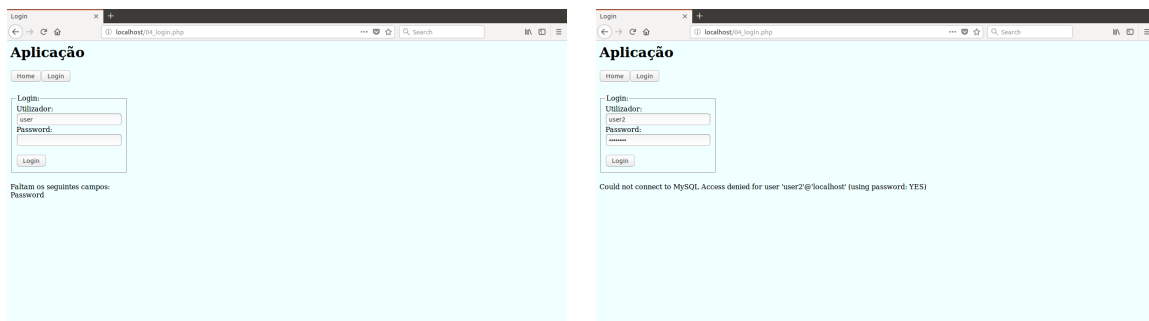


Figura 4.6: Página de *Login* para iniciar sessão na base de dados central

A página de *Login* consiste num simples formulário constituído por duas caixas de texto e um botão, como demonstrado na Figura 4.6. O botão *Login* lê as credenciais introduzidas e realiza uma conexão de teste à base de dados central validando-as diretamente com *MySQL*. Se as credenciais forem validadas com sucesso redireciona-se o utilizador para a página principal e altera-se o botão de *Login* para *Logout*. Se as credenciais introduzidas não forem suficientes ou válidas são retornados erros de forma a informar o utilizador como demonstrado nas Figuras 4.7a e 4.7b.

Quando se acede à página com *Logout* termina-se a sessão e redireciona-se o utilizador para a página principal.



(a) Exemplo de erro de falta de informação

(b) Exemplo de erro *MySQL*

Figura 4.7: Exemplos de erros retornados quando introduzidas credenciais não válidas na página *Login*



### 4.2.3 Consultas

Consultas

Home Consultas Administração Conectar Local Logout

Gerar Query

Escolher Atributos:

Clientes	Moldes	Sensores	Registos
<input type="checkbox"/> cl_ID	<input type="checkbox"/> m_ID	<input type="checkbox"/> s_num	<input type="checkbox"/> fase_nome
<input type="checkbox"/> cl_nome	<input type="checkbox"/> m_nome	<input type="checkbox"/> tipo_nome	<input type="checkbox"/> r_data_hora
<input type="checkbox"/> cl_morada	<input type="checkbox"/> m_descricao	<input type="checkbox"/> s_nome	<input type="checkbox"/> r_milissegundos
<input type="checkbox"/> cl_IP		<input type="checkbox"/> s_descricao	<input type="checkbox"/> r_valor
<input type="checkbox"/> cl_port			

Adicionar Condições:

Filtros:

Ordenar:

Ou:

Query:

Gerar Query

(a) Página de Consultas

Consultas Resultado

localhost/resultado\_consulta.php?cl\_ID=cl\_ID&m\_ID=m\_ID&s\_num=s\_num&tipo=tipo\_nome

Query anterior: SELECT cl\_ID, m\_ID, s\_num, tipo\_nome FROM clientes INNER JOIN moldes ON cl\_ID = m\_IDCliente INNER JOIN sensores ON m\_ID = s\_IDMolde INNER JOIN tipo ON s\_tipo = tipo\_ID WHERE m\_ID = 9001 OR m\_ID = 9009

cl_ID	m_ID	s_num	tipo_nome
10001	9001	1	Termómetro
10001	9001	2	Dinamómetro
10001	9001	3	Dinamómetro
10001	9001	4	Dinamómetro
10001	9001	5	Dinamómetro
10004	9009	1	Termómetro
10004	9009	2	Termómetro
10004	9009	3	Dinamómetro
10004	9009	4	Dinamómetro
10004	9009	5	Extensómetro
10004	9009	6	Extensómetro
10004	9009	7	Vibrómetro
10004	9009	8	Vibrómetro
10004	9009	9	Pressão
10004	9009	10	Pressão

(b) Exemplo de resposta de consulta

Figura 4.8: Página de Consultas e exemplo de resposta a uma consulta

A página de Consultas assiste os utilizadores sem conhecimentos de *SQL* a criarem *queries* para consultar a base de dados central. Na Figura 4.8a observa-se várias *checkboxes* e três caixas de texto. As *checkboxes* permitem selecionar os atributos que se desejam consultar na base de dados; estes atributos são guardados numa variável `@atributos`.

Quando se prime o botão *Query* gera-se uma das seguintes quatro *queries*:

- 1- **SELECT** @atributos  
**FROM** clientes;
- 2- **SELECT** @atributos  
**FROM** clientes  
**INNER JOIN** moldes **ON** cl\_ID = m\_IDCliente;
- 3- **SELECT** @atributos  
**FROM** clientes  
**INNER JOIN** moldes **ON** cl\_ID = m\_IDCliente  
**INNER JOIN** sensores **ON** m\_ID = s\_IDMolde  
**INNER JOIN** tipo **ON** s\_tipo = tipo\_ID;
- 4- **SELECT** @atributos **FROM** clientes  
**INNER JOIN** moldes **ON** cl\_ID = m\_IDCliente  
**INNER JOIN** sensores **ON** m\_ID = s\_IDMolde  
**INNER JOIN** tipo **ON** s\_tipo = tipo\_ID  
**INNER JOIN** registros **ON** s\_IDMolde = r\_IDMolde  
**AND** s\_num = r\_numSensor  
**INNER JOIN** fase **ON** r\_fase = fase\_ID;

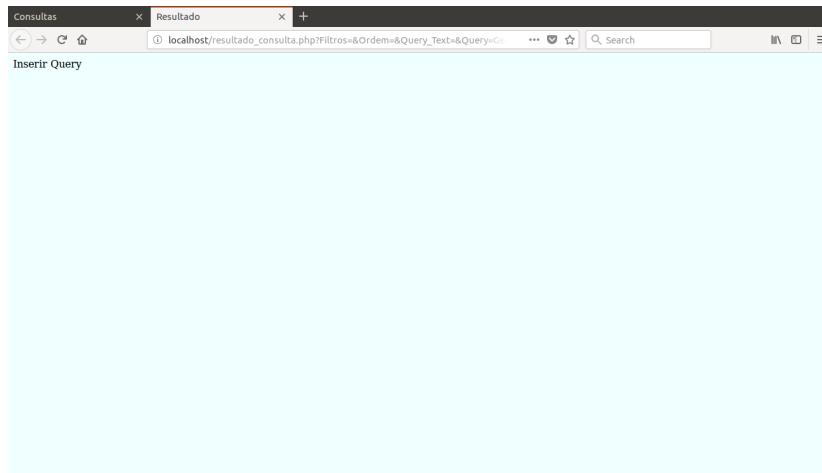
A seleção é feita com base na coluna mais à esquerda a que os atributos pertencem. Explicando melhor com um exemplo: se o utilizador desejar consultar o cl\_ID e o cl\_nome da tabela clientes gera-se a primeira *query* no entanto, se o utilizador desejar consultar os atributos cl\_ID, m\_ID e s\_num gera-se a terceira *query*.

Além destas, foram adicionadas três *queries* específicas quando os atributos tipo\_nome, fase\_nome e r\_data\_hora são selecionados sozinhos. As primeiras duas permitem consultar as opções disponíveis nos dicionários e a terceira devolve as datas e horas entre o primeiro e último registros.

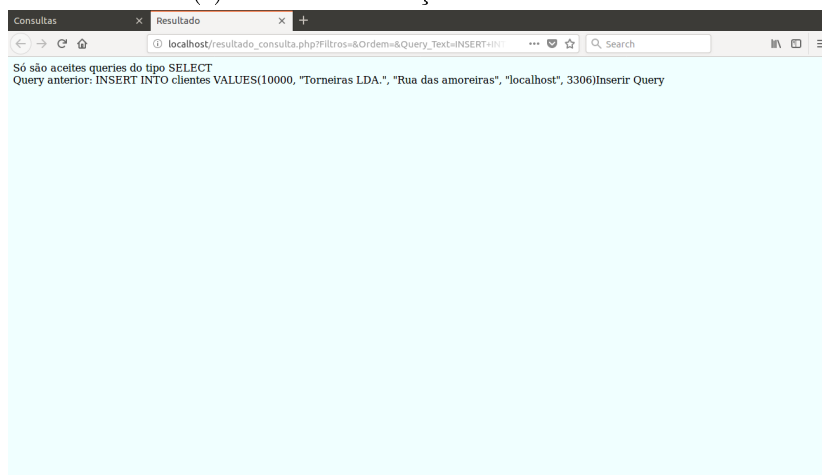
As caixas de texto Filtros e Ordem na Figura 4.8a permitem adicionar às *queries* geradas as cláusulas WHERE e ORDER BY, respetivamente. Para os utilizadores com conhecimentos em *SQL* está disponibilizada a caixa de texto *Query* que permite a criação direta de uma *query*. Este campo está limitado apenas para *queries* do tipo SELECT.

Depois da *query* ser gerada retorna-se uma resposta num novo separador como demonstrado na Figura 4.8b. O *link* desta resposta contém toda a informação da *query* gerada. Este pode ser arquivado ou enviado para outro utilizador sem ser necessário gerar a *query* novamente, isto é útil para *queries* com muitas cláusulas.

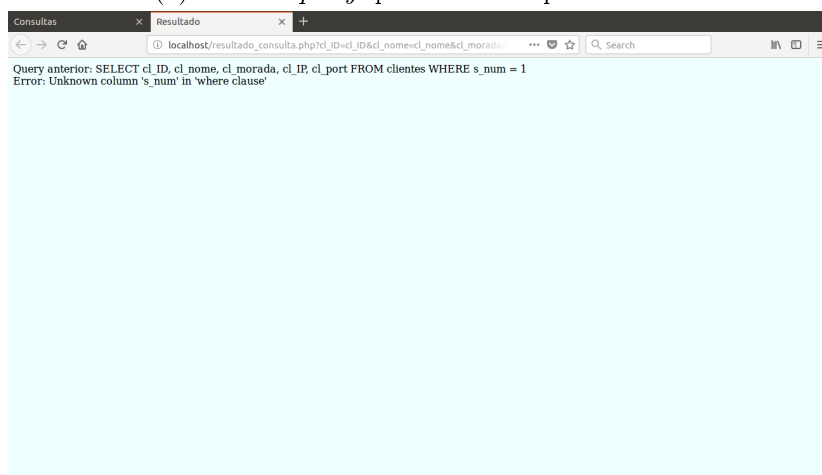
Se a *query* não for válida retorna-se um erro de forma a informar o utilizador, como demonstrado nas Figuras 4.9a, 4.9b e 4.9c.



(a) Erro de informação não introduzida



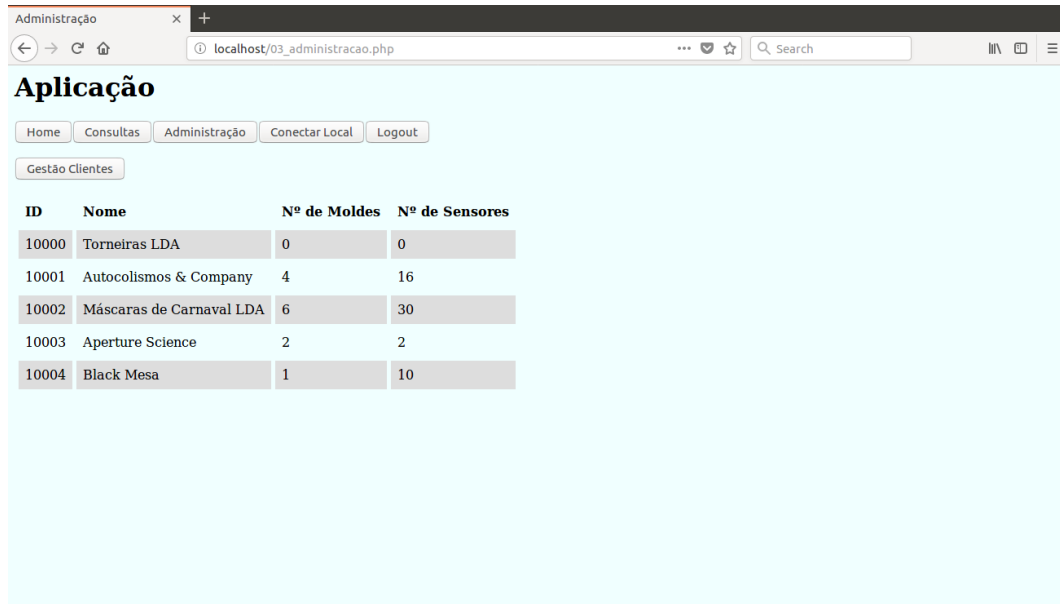
(b) Erro de *query* que não é do tipo SELECT



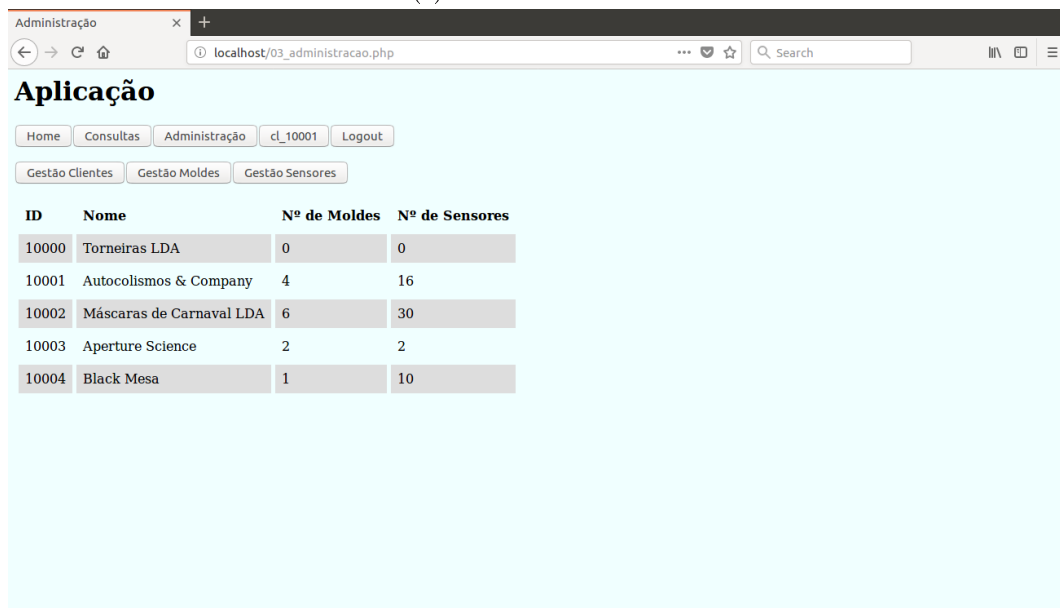
(c) Exemplo de erro *MySQL*

Figura 4.9: Exemplos de erros retornados na página de Resposta da Consulta

## 4.2.4 Administração Local



(a) Sem conexão local



(b) Com conexão local

Figura 4.10: Funcionalidades da página Administração com e sem conexão local

Esta área está dividida em duas componentes distintas. A componente de Administração permite ao utilizador alterar informações sobre os clientes, moldes e sensores. A componente de Conectar Local permite realizar uma conexão à base de dados local no servidor do cliente. A componente de Administração só pode ser acedida quando for efetuada uma conexão local. Após a conexão ser bem sucedida observa-se informação do cliente com a *query*:

```
SELECT cl_ID, cl_nome, COUNT(DISTINCT m_ID),  
COUNT(DISTINCT s_IDMolde, s_num)  
FROM clientes LEFT OUTER JOIN moldes ON cl_ID = m_IDCliente  
LEFT OUTER JOIN sensores ON m_ID = s_IDMolde  
GROUP BY cl_ID;
```

## Administração

Esta componente está dividida em três áreas: Gestão de Clientes, Gestão de Moldes e Gestão de Sensores como demonstrado na Figura 4.10a. Quando se acede a esta componente redireciona-se o utilizador para a área de Gestão de Clientes. Aqui a informação dos clientes pode ser alterada com o formulário demonstrado na Figura 4.11. Os botões Adicionar Cliente, Alterar Cliente e Eliminar Cliente executam *queries* do tipo INSERT, UPDATE e DELETE, respetivamente. Quando toda a informação tiver sido inserida e registos tiverem a ser gerados deve-se reiniciar o programa de transferência de valores. O botão Atualizar conecta à base de dados notificações e altera o parâmetro atualizar da tabela transferência para 1.

Administração

← → ↻ 🏠

localhost/031\_admin\_cliente.php

🔍 Search

## Aplicação

Home Consultas Administração Conectar Local Logout

Gestão Clientes

Cliente:

ID:

Nome:

Morada:

IP:

Port:

Adicionar Cliente Alterar Cliente Eliminar Cliente

Ver Clientes

ID	Nome	Morada	IP	Port	Nº de Moldes	Nº de Sensores
10000	Torneiras LDA	Rua das amoreiras	127.0.0.1	3306	0	0
10001	Autocolismos & Company	Rua das sanita, nº 2	127.0.0.1	3306	4	16
10002	Máscaras de Carnaval LDA	Rua de Elm street, nº 13	127.0.0.1	3306	6	30
10003	Aperture Science	Travessa de bolo, nº 404	127.0.0.1	3306	2	2
10004	Black Mesa	Área 51	127.0.0.1	3306	1	10

Figura 4.11: Área de Gestão de Clientes sem conexão local

As áreas de Gestão de Moldes e Gestão de Sensores demonstradas nas Figuras 4.12 e 4.13, permitem ao utilizador criar e apagar moldes e sensores, respetivamente. Estes dados são inseridos na base de dados temporária local, onde o utilizador pode criar e apagar moldes e sensores sem afetar o sistema. Desta forma confirma-se a informação introduzida antes de a inserir no sistema. Os botões de Criar e Apagar nestes formulários realizam *queries* do tipo INSERT e DELETE, respetivamente.

**Aplicação**

Home Consultas Administração cl\_10001 Logout

Gestão Clientes Gestão Moldes Gestão Sensores

**Temporária:**

ID Cliente	ID Molde	Nome	Descrição	Nº de Sensores
10001	9000	NULL	NULL	5

Validar

Molde:

ID Cliente:

ID Molde:

Nome:

Descrição:

Adicionar Molde Eliminar Molde

Ver Clientes Ver Moldes Ver Sensores

Figura 4.12: Área de Gestão de Moldes

**Aplicação**

Home Consultas Administração cl\_10001 Logout

Gestão Clientes Gestão Moldes Gestão Sensores

**Temporária:**

ID Cliente	ID Molde	Número do Sensor	Tipo de Sensor	Nome	Descrição
------------	----------	------------------	----------------	------	-----------

Validar

Sensor:

ID Molde:

Número do Sensor:

Tipo:

Nome:

Descrição:

Adicionar Sensor Eliminar Sensor

Ver Clientes Ver Moldes Ver Sensores

Figura 4.13: Área de Gestão de Sensores

Quando a informação dos moldes e sensores estiver completa, o botão Validar tenta registar os valores presentes na base de dados temporária local nas bases de dados central e local. Se a ação não executar com sucesso é retornado um erro *MySQL* de forma a informar o utilizador. Se a ação executar com sucesso a base de dados temporária local é limpa e os valores são

registados permanentemente nas bases de dados central e local, como representado nas Figuras 4.14a e 4.14b.

Depois de inseridos, moldes e sensores não podem ser eliminados via aplicação. Esta opção foi removida da aplicação para evitar erros, dado que apagar um molde em funcionamento faz com que se percam novos registos.

Aplicação

Home Consultas Administração d\_10000 Logout

Gestão Clientes Gestão Moldes Gestão Sensores

Temperaturas:

ID Cliente	ID Molde	Número do Sensor	Tipo de Sensor	Nome	Descrição
10000	5000	1	Termómetro	NULL	Temperatura ponto crítico 1
10000	5000	2	Pressão	NULL	Pressão ponto crítico 2

Validar

Sensor:

ID Molde:

Número do Sensor:

Tipo:

Nome:

Descrição:

Adicionar Sensor Eliminar Sensor

(a) Dados antes de serem validados

Aplicação

Home Consultas Administração d\_10000 Logout

Gestão Clientes Gestão Moldes Gestão Sensores

Temperaturas:

ID Cliente	ID Molde	Nome	Descrição	Nº de Sensores
10000	5000	NULL	Temperatura ponto crítico 1	2
10000	5000	NULL	Pressão ponto crítico 2	2

Validar

Molde:

ID Cliente:

ID Molde:

Nome:

Descrição:

Adicionar Molde Eliminar Molde

Ver Clientes Ver Moldes Ver Sensores

ID Cliente ID Molde Nome Descrição Nº de Sensores

10000 5000 NULL Torreta A 2

(b) Dados após serem validados

Figura 4.14: Função do botão Validar, onde os valores da base de dados temporária local são transferidos de forma permanente para as bases de dados central e local

Nas várias áreas de gestão existem os botões Ver Clientes, Ver Moldes e Ver Sensores que executam respetivamente as *queries*:

- 1- 

```
SELECT cl_ID, cl_nome, cl_morada, cl_IP, cl_port,
COUNT(DISTINCT m_ID), COUNT(DISTINCT s_IDMolde, s_num)
FROM clientes
LEFT OUTER JOIN moldes ON cl_ID = m_IDCliente
LEFT OUTER JOIN sensores ON m_ID = s_IDMolde
GROUP BY cl_ID
ORDER BY cl_ID
```
- 2- 

```
SELECT m_IDCliente, m_ID, m_nome, m_descricao,
COUNT(DISTINCT s_IDMolde, s_num)
FROM clientes
INNER JOIN moldes ON cl_ID = m_IDCliente
LEFT OUTER JOIN sensores ON m_ID = s_IDMolde
GROUP BY m_ID
ORDER BY m_IDCliente, m_ID
```
- 3- 

```
SELECT m_IDCliente, s_IDMolde, s_num, tipo_nome,
s_nome, s_descricao
FROM moldes
INNER JOIN sensores ON m_ID = s_IDMolde
INNER JOIN tipo ON s_tipo = tipo_id
ORDER BY m_IDCliente, s_IDMolde, s_num
```

Estas *queries* fornecem algumas informações contextuais para facilitar a navegação do utilizador.



## Conectar Local

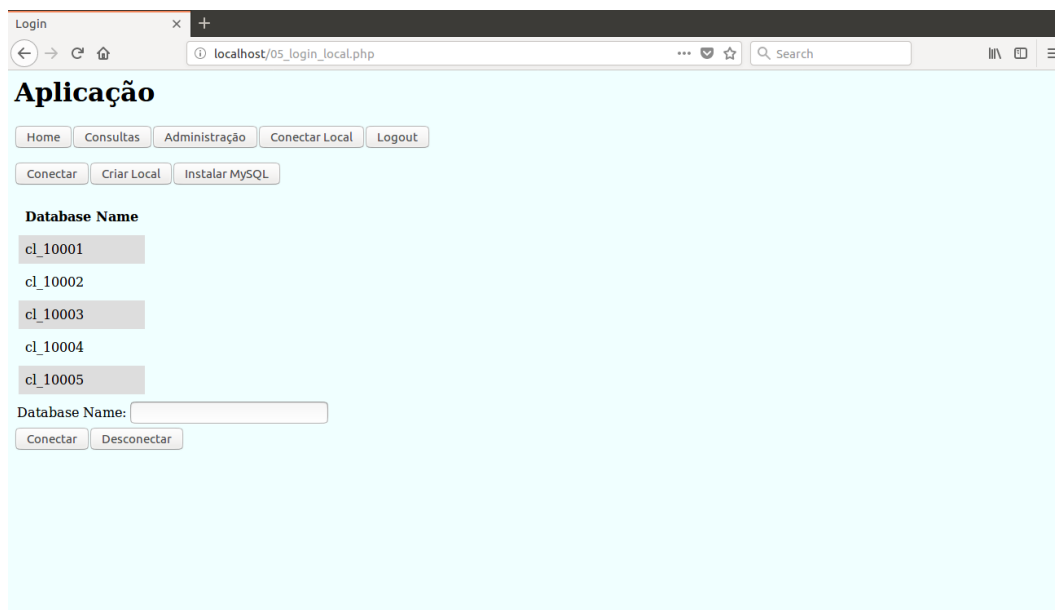


Figura 4.15: Área Conectar Local onde se visualiza as bases de dados instaladas no sistema

A área de Conectar Local na Figura 4.15 permite realizar uma conexão à base de dados local no servidor do cliente. Com recurso à *query*:

```
SHOW DATABASES
```

Obtém-se todas as bases de dados instaladas no servidor local. Do ponto de vista prático, cada cliente só terá uma base de dados local mas, para efeitos de desenvolvimento do projeto adotou-se esta vertente. O botão Conectar inicia sessão na base de dados local escolhida e redireciona o utilizador para a página de Administração como se observar na Figura 4.16. O botão Desconectar termina esta sessão e redireciona o utilizador também para a página de Administração.

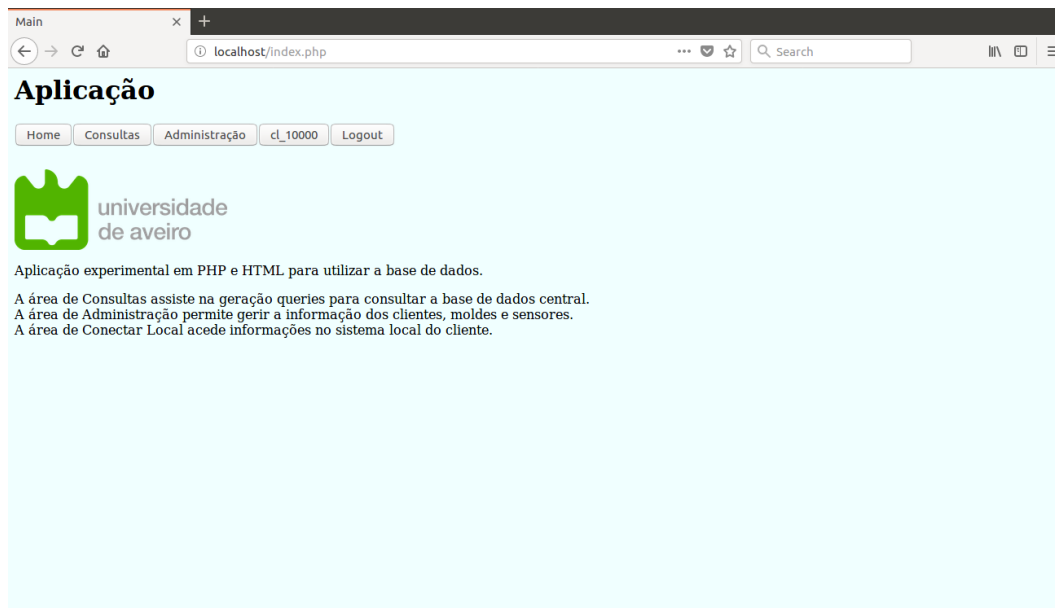


Figura 4.16: *Main* com conexão local

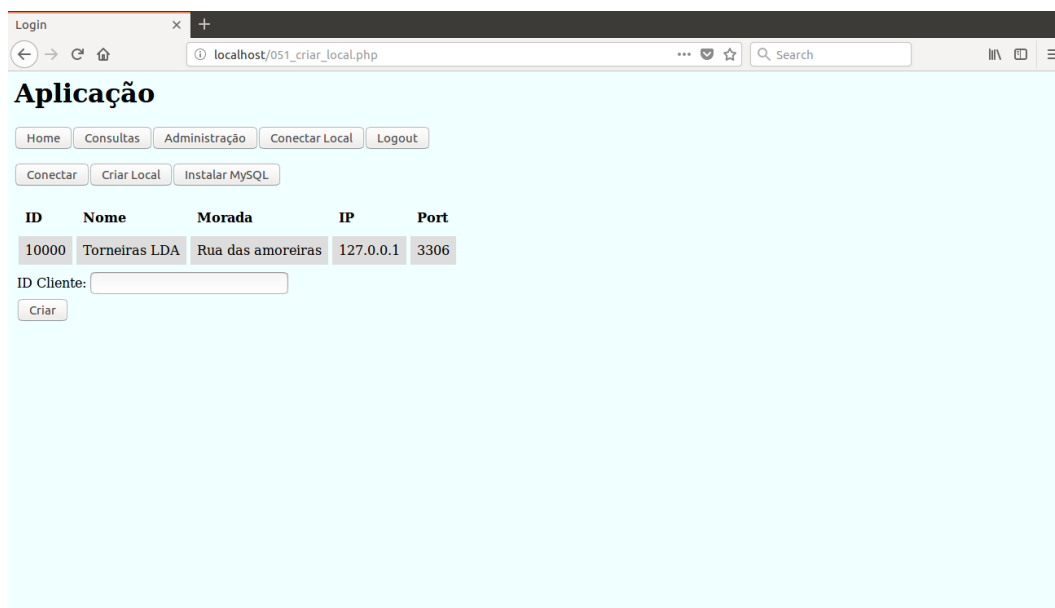


Figura 4.17: Área Criar Local cria a base de dados para o cliente seleccionado no sistema em que se acede a aplicação

A área Criar Local na Figura 4.17 permite instalar uma base de dados para um novo cliente. São considerados novos clientes todos os que não tenham moldes associados a si, esta informação obtém-se com a *query*:

```

SELECT cl_ID, cl_nome, cl_morada, cl_IP, cl_port
FROM
    (SELECT cl_ID, cl_nome, cl_morada, cl_IP, cl_port,
    COUNT(DISTINCT m_ID) AS n_moldes
    FROM clientes
    LEFT OUTER JOIN moldes ON cl_ID = m_IDCliente
    GROUP BY cl_ID) AS contagem
WHERE n_moldes = 0

```

Escolhendo um cliente válido o botão Criar gera *queries* para efetuar a instalação da base de dados local e definir as respetivas permissões. Estas *queries* não são visíveis ao utilizador, assim sendo, o botão Copiar Query copia-as para a área de transferência e estas devem ser introduzidas no *MySQL* com o comando colar, como representado na Figura 4.18.

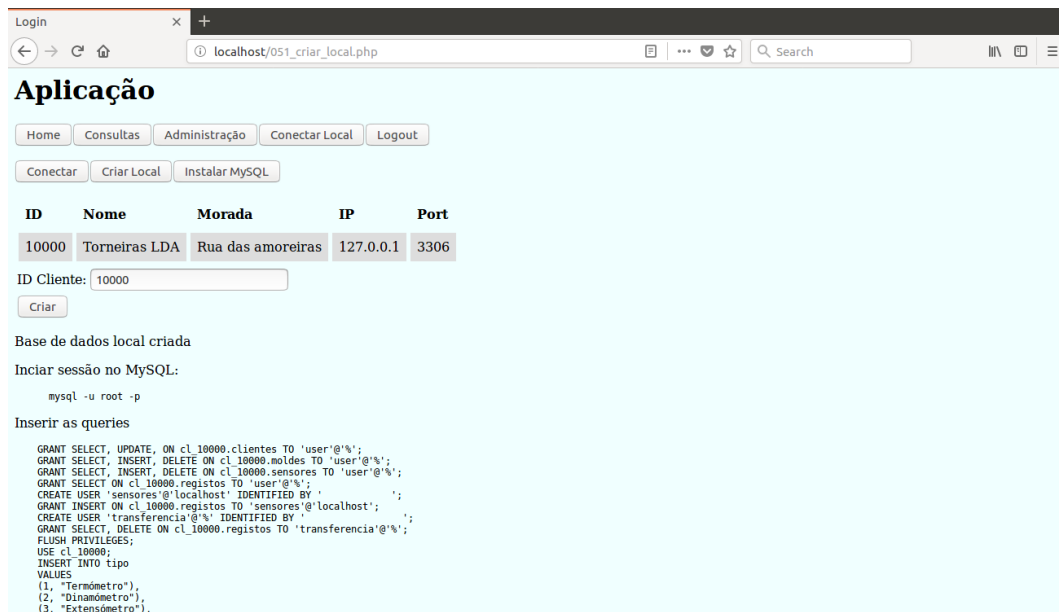


Figura 4.18: *Queries* geradas para completar a instalação da base de dados no sistema local. Consistem nas permissões para os utilizadores que só podem ser garantidas via *root* bem como informação para completar o sistema de dados

Terminado a análise das funcionalidades da aplicação com a área de Instalar *MySQL* na Figura 4.19 que contém os passos para instalar o *MySQL* num sistema *Linux*.

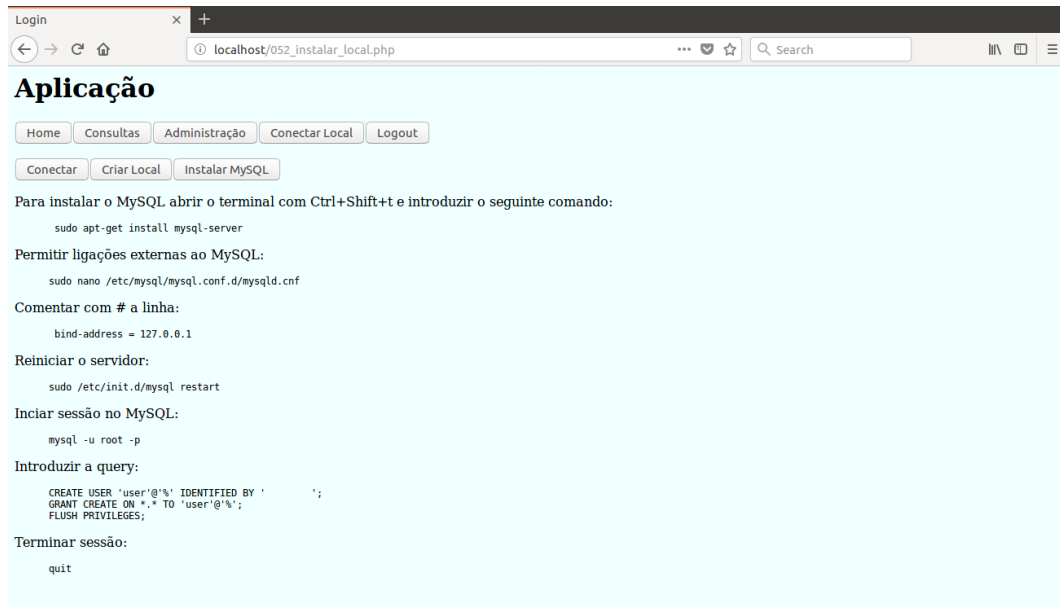


Figura 4.19: Área Instalar *MySQL* para instalar o *software* com os comandos em *Linux*

## Capítulo 5

# Conclusões

O desafio proposto consiste em monitorizar sensores remotamente. Para isto definiu-se como objetivos principais desenvolver uma rede de bases de dados e uma aplicação que permitisse interagir com estas. Estes objetivos foram concluídos com sucesso. Este capítulo contém comentários sobre o desempenho da solução desenvolvida e propostas para trabalhos futuros.

### 5.1 Comentários

#### 5.1.1 Infraestrutura de dados

Em relação à infraestrutura proposta, esta cumpre todos os requisitos propostos de garantir uma transferência segura, confidencial e permanente de valores, criando um histórico dos moldes monitorizados. Os programas desenvolvidos em *C* são simples e funcionais. No entanto, quando estão em funcionamento, o sistema executa-os sem restrições. Isto resulta num elevado consumo do processador e consequente perda de performance do sistema central. Esta perda de performance pode ser por causa do *hardware* utilizado no projeto e a utilização de um sistema devidamente dimensionado para a tarefa em mão pode resolver este problema. Outra solução seria limitar a velocidade de processamento da execução destes programas. Como definido nos objetivos esta não impõe restrições na instrumentação dos moldes. Para introduzir dados nas bases de dados locais pode ser utilizado qualquer sistema operativo e linguagem de programação desde que esta tenha protocolos de comunicação com *MySQL* e gere *queries* do tipo:

```
INSERT INTO registos
VALUES
(molde, sensor, fase, data_hora, milissegundos, valor);
```

Na realidade esta infraestrutura pode ser utilizada em qualquer contexto de monitorização remota de sensores desde que seja desenvolvido um modelo de dados apropriado.

#### 5.1.2 Aplicação

Em relação à aplicação, esta cumpre os objetivos propostos de ser multiplataforma e garantir um acesso remoto à base de dados central, bem como gerir as informações dos clientes, moldes e sensores. No entanto, o facto da aplicação ter sido desenvolvida na totalidade com *PHP* e *HTML*, causa uma perda de performance no servidor central. Isto aconteceu porque,

durante o desenvolvimento do projeto, não foi percebido na totalidade o conceito de lado do servidor e lado do cliente. De forma a melhorar o desempenho sugere-se a alteração de algumas funcionalidades desenvolvidas em *PHP* para *JavaScript*, como por exemplo, as conexões às bases de dados que são particularmente pesadas no servidor. A alteração das conexões de *PHP* para *JavaScript* permite também alterar o método de instalação da base de dados local mencionado na Seção 4.2.4. Em vez de serem gerados comandos para o utilizador executar no *MySQL* estes podem ser enviados diretamente pela aplicação desde que seja fornecida a *password* para a *root* do sistema como sugerido na Figura 5.1.

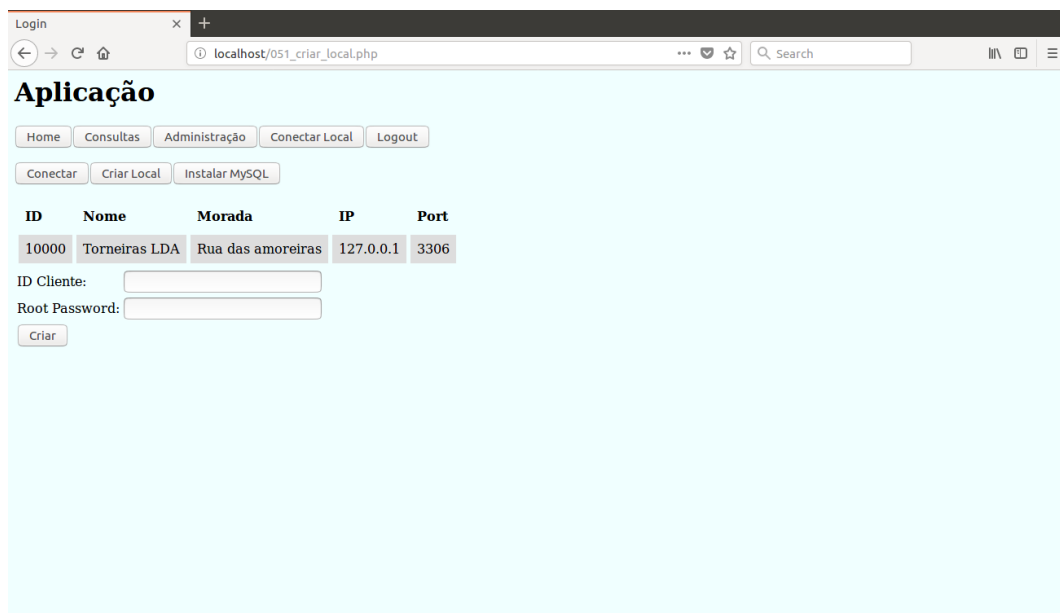


Figura 5.1: Área de Criar Local com criação via *root* em vez de gerar *queries* para o utilizador inserir manualmente no *MySQL*

Além disto é necessário realizar uma revisão de segurança à aplicação. Por exemplo, o botão Atualizar na área Gestão de Clientes que reinicia o programa de transferência de valores através de comandos no terminal. Se as permissões garantidas ao servidor *Apache* não forem bem definidas, pode constituir uma quebra de segurança. Um programador com intenções maliciosas pode acessar o sistema pela aplicação e realizar comandos no terminal de forma a comprometer o sistema. Para isto não acontecer é necessário garantir que o servidor *Apache* só tem permissões sobre o programa de transferência ou então definir um sistema de notificações entre a aplicação e o programa em *C*.

Apesar de serem necessários alguns ajustes de forma a melhorar performance, a solução proposta da infraestrutura e aplicação é completamente funcional e pode ser já implementada numa fase experimental.

## 5.2 Trabalhos Futuros

Quanto à infraestrutura dos dados não foi definido como os sensores dos moldes serão ligados ao servidor local. No desenvolvimento deste projeto assumiu-se que todos os moldes estão

ligados diretamente ao sistema local. Se esta proposta não se demonstrar viável é possível, em vez de se ter um servidor local por cliente, ter um servidor local por molde. Esta adaptação irá criar uma maior quantidade de bases de dados locais mas, isto não é problemático, se o modelo de dados e o programa de transferência forem adaptados para o efeito.

Quanto à aplicação, sugere-se que após uma apresentação inicial desta à empresa promotora, seja iniciado um processo iterativo de desenvolvimento para escolher e desenvolver novas funcionalidades que possam ser úteis e que não tenham sido abrangidas neste projeto, como por exemplo, a criação de utilizadores baseado no ID de trabalhador demonstrado na Figura 5.2. Culminando numa estilização da aplicação para que esta tenha um aspeto mais amigável ao utilizador.

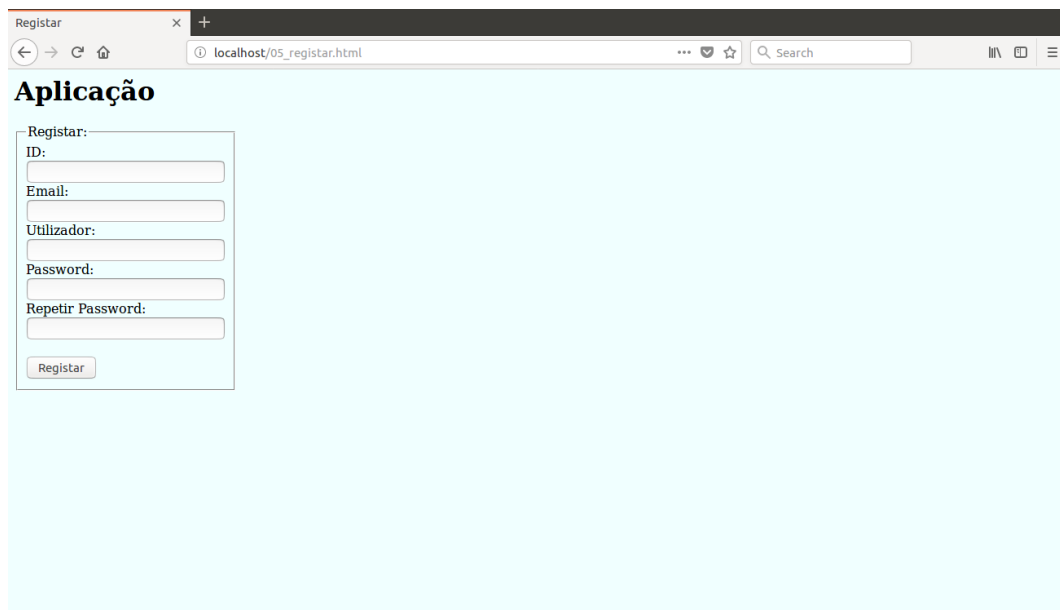
The image shows a web browser window with a single tab titled 'Registrar'. The address bar shows 'localhost/05\_registar.html'. The page has a light blue background. At the top, the word 'Aplicação' is written in bold black text. Below it, there is a registration form with a light blue border. The form contains the following fields: 'ID:' with a text input, 'Email:' with a text input, 'Utilizador:' with a text input, 'Password:' with a text input, and 'Repetir Password:' with a text input. At the bottom of the form is a button labeled 'Registrar'.

Figura 5.2: Exemplo de área de Registrar para criar utilizadores com base no ID de trabalhador e email

Além destas alterações sugere-se a criação de um sistema de notificações e monitorização automático dos moldes. É impossível para um utilizador analisar manualmente milhões de registos de um molde e concluir se este está a funcionar corretamente. Para este efeito criar um programa capaz de correr algoritmos que analisem o comportamento dos moldes. Este programa pode ser desenvolvido em *softwares* mais sofisticados, como por exemplo *MATLAB*, desde que estes tenham protocolos de comunicação com *MySQL*.

A gestão dos *backups* descrita na Subseção 3.2.5 onde se separa os registos dos moldes em vez de se realizar um *backup* geral foi realizada com este sistema de notificações em mente. Se for necessário, para efeitos de cálculo, que o programa carregue os registos de um molde armazenados em *backups*, este só necessita de carregar a informação do molde que está a ser analisado em vez de ter de carregar a informação de todos os moldes. Este programa deverá correr automaticamente no sistema de forma permanente ou com um temporizador e, no caso de ser necessário, notificar o utilizador via aplicação ou via email.





# Bibliografia

- [1] colored-part-large-mold-with-part.
- [2] moldagem in Dicionário infopédia da Língua Portuguesa [em linha]. Porto: Porto Editora, 2003-2018. [consult. 2018-05-22 18:20:35]. Disponível na Internet: <https://www.infopedia.pt/dicionarios/lingua-portuguesa/moldagem>.
- [3] moldar in Dicionário infopédia da Língua Portuguesa [em linha]. Porto: Porto Editora, 2003-2018. [consult. 2018-05-22 18:21:22]. Disponível na Internet: <https://www.infopedia.pt/dicionarios/lingua-portuguesa/moldar>.
- [4] molde in Dicionário infopédia da Língua Portuguesa [em linha]. Porto: Porto Editora, 2003-2018. [consult. 2018-05-22 16:51:33]. Disponível na Internet: <https://www.infopedia.pt/dicionarios/lingua-portuguesa/molde>.
- [5] G. Williams and H. A. Lord. Mold-filling studies for the injection molding of thermoplastic materials. Part I: The flow of plastic materials in hot- and cold-walled circular channels. *Polymer Engineering and Science*, 15(8):553–568, aug 1975.
- [6] Michael Trovant and Stavros A Argyropoulos. The implementation of a mathematical model to characterize mold metal interface effects in metal casting. *Canadian Metallurgical Quarterly*, 37(3-4):185–196, jun 1998.
- [7] TN) Janney, Mark A. (Knoxville and NG) Omatete, Ogbemi O. (Lagos. Method for molding ceramic powders using a water-based gel casting, jul 1991.
- [8] Jiwang Yan, Takashi Oowada, Tianfeng Zhou, and Tsunemoto Kuriyagawa. Precision machining of microstructures on electroless-plated NiP surface for molding glass components. *Journal of Materials Processing Technology*, 209(10):4802–4808, jun 2009.
- [9] Changyu Shen, Lixia Wang, and Qian Li. Optimization of injection molding process parameters using combination of artificial neural network and genetic algorithm method. *Journal of Materials Processing Technology*, 183(2-3):412–418, mar 2007.
- [10] K. Shelesh-Nezhad and E. Siores. An intelligent system for plastic injection molding process design. *Journal of Materials Processing Technology*, 63(1-3):458–462, jan 1997.
- [11] Babur Ozcelik and Ibrahim Sonat. Warpage and structural analysis of thin shell plastic in the plastic injection molding. *Materials & Design*, 30(2):367–375, feb 2009.
- [12] About Injection Molding | Xcentric Mold & Engineering.

- [13] *Polymer Process Engineering* - R. Griskey - Google Livros.
- [14] hyatt. nov 1872.
- [15] Injection molding machine. may 1949.
- [16] Suzanne L. B. Woll and Douglas J. Cooper. Pattern-based closed-loop quality control for the injection molding process. *Polymer Engineering & Science*, 37(5):801–812, may 1997.
- [17] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010.
- [18] base de dados in Dicionário infopédia da Língua Portuguesa [em linha]. Porto: Porto Editora, 2003-2018. [consult. 2018-05-31 18:24:13]. Disponível na Internet: [https://www.infopedia.pt/dicionarios/lingua-portuguesa/base de dados](https://www.infopedia.pt/dicionarios/lingua-portuguesa/base-de-dados).
- [19] Edgar Frank Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [20] Peter Pin-Shan Chen. The Entity-Relationship Unified View of Data Model-Toward a. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [21] Community | Ubuntu.
- [22] SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems | DigitalOcean.
- [23] MySQL :: MySQL 8.0 Reference Manual.
- [24] GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF).
- [25] Welcome! - The Apache HTTP Server Project.