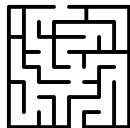




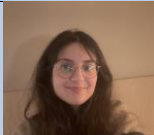

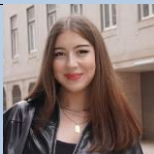



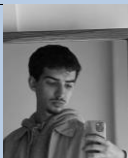
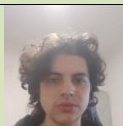


## PI de Sistemas de Informação Distribuídos (24/25)

Licenciaturas em Engenharia Informática e Informática e Gestão de Empresas



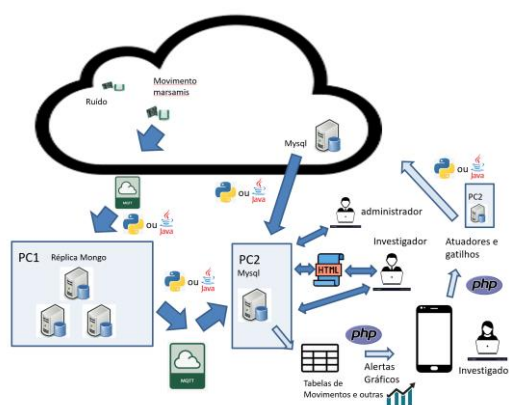
Grupo 16		Grupo 9	
112028 Matilde Glória mggas@iscte- iul.pt		110916 Davi de Mattos Balieiro dmbos@iscte- iul.pt	
111600 Inês Pedro Pinto ipflo@iscte- iul.pt		110937 Ji Hua Zhu jhzua@iscte- iul.pt	
112412 Rita Ferreira Dias rfdsa3@iscte- iul.pt		111121 Guilherme da Mota Castilho Caramelo Riço gmccr@iscte- iul.pt	
110749 Elisabete Kirikov ekvel@iscte- ul.pt		111213 Gonçalo Vieira Henriques gvhsa@iscte- iul.pt	
111188 Rodrigo Parente rjepe@iscte- ul.pt		111255 Rodrigo Miguel Cosme dos Santos rmcss1@iscte- ul.pt	
123456 Santiago Velooso srvol@iscte- ul.pt		111257 Ricardo Fernandes Paulo Isidro rfpio@iscte- ul.pt	



## Instruções

Estas instruções são de cumprimento obrigatório. Relatórios que não cumpram as indicações serão penalizados na nota final.

- Podem (e em várias situações será necessário) ser adicionadas novas páginas ao relatório, mas não podem ser removidas páginas. Se uma secção não for relevante, fica em branco, não pode ser removida;
- A paginação tem de ser sequencial e não ter falhas;
- O índice tem de estar atualizado.
- O grupo que inicia o documento (coluna à esquerda na folha de rosto) preenche apenas a parte inicial (até ao fim da secção secção 1). Este documento word vai ser colocado no moodle para que o outro grupo (à direita da folha de rosto) possa descarregar e continuar a preenche-lo (secção 2)





# Índice

1	Especificação	5
1.1	Da Nuvem para o Mongo	5
1.2	Descrição Geral do Procedimento de Mongo Para Mysql	8
1.4	Tratamento de dados anómalos (valores de sensores errados)	11
1.5	Tratamento de outliers de ruído	12
1.6	Tratamento de Alertas de ruído	14
1.7	Tratamento de número de marsamis numa sala (obter pontuação)	16
1.8	Especificação de Store Procedures SQL de apoio à migração e tratamento de dados	18
1.9	Especificação de Triggers de apoio à migração e tratamento de dados	19
1.10	Modelo Relacional	21
1.11	Utilizadores Base de Dados Mysql	23
1.12	Procedimentos Manutenção da Aplicação	25
1.13	Eventos de suporte à aplicação (caso existam)	26
1.14	Consulta por HTML/PHP	27
2	Implementação	28
2.1	Coleções a criar em cada uma das réplicas do Mongo	28
2.2	Descrição Geral do Procedimento de Mongo Para Mysql	30
2.3	Tratamento de dados anómalos (valores de sensores errados)	32
2.4	Tratamento de outliers de ruído	33
2.5	Tratamento de Alertas de Som	34
2.6	Tratamento de número de marsamis numa sala (obter pontuação)	35
2.7	Implementação de Stored Procedures SQL de apoio à migração e tratamento de dados	36
2.8	Implementação de Triggers	37
1.1	Modelo Relacional	38
1.2	Utilizadores Base de Dados Mysql	41
1.3	Procedimentos Manutenção da Aplicação	43
1.4	Eventos de suporte à aplicação (caso existam)	45
	<i>Trabalho de Projeto de Integração de Sistemas de Informação Distribuídos</i>	3



## PI de Sistemas de Informação Distribuídos, 23/24

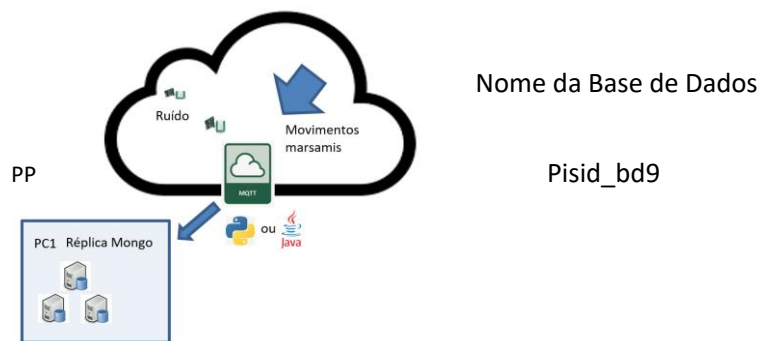
1.5	PrintScreen dos formulários HTML implementados	46
1.6	PrintScreen do formulários Android com dados	49
	Código de Triggers implementados	77
	Código Stored Procedures implementados	82



# 1 Especificação

Esta secção é onde o grupo que inicia o documento (coluna à esquerda na folha de rosto) coloca a especificação do que pretende implementar. Mais tarde pode implementar de outra maneira, mas aqui vão as primeiras ideias que serão avaliadas na primeira oral e que vão ser entregues a outro grupo para que analisem e vejam se aproveitam as vossas ideias.

## 1.1 Da Nuvem para o Mongo



Nome Coleção	O que armazena?
<b>sensores_movimento</b>	Armazena os movimentos dos marsamis entre as salas (lidos no tópico pisid_mazemov)
<b>sensores_ruído</b>	Armazena os níveis de ruídos captados no labirinto (lidos no tópico pisid_mazesound)
<b>configuracoes</b>	Armazena configurações iniciais do jogo para permitir confirmar se os dados que chegam dos sensores estão corretos.
<b>dados_invalidos</b>	Armazena os dados que são considerados inválidos na fase de processamento de dados do mongoDB para o mqtt

Para cada coleção exemplifica um documento

Coleção : sensores movimento

```
{
  "_id": {
    "$oid": "67d94a4a0519f895186e6cc1"
  }
}
```



## PI de Sistemas de Informação Distribuídos, 23/24

```
"payload": {  
  "Player": 16,  
  "Marsami": 15,  
  "RoomOrigin": 3,  
  "RoomDestiny": 2,  
  "Status": 1  
},  
"processed": true  
},  
"topic": "pisid_mazemov_16"  
}
```

Coleção : sensores ruido

```
{  
  "_id": {  
    "$oid": "67d85937e955024c65add6be"  
  },  
  "payload": {  
    "Player": 16,  
    "Hour": "2025-03-17 17:17:11.168077",  
    "Sound": 19.04  
  },  
  "topic": "pisid_mazesound_16"  
}
```



Coleção : dados invalidos

```
{  
  "tipo": "som" ou "movimento",  
  "payload": {  
    conteúdo original recebido via MQTT  
  },  
  "erro": "Descrição do erro",  
  "timestamp": ISODate("AAAA-MM-DDTHH:MM:SSZ")  
}
```

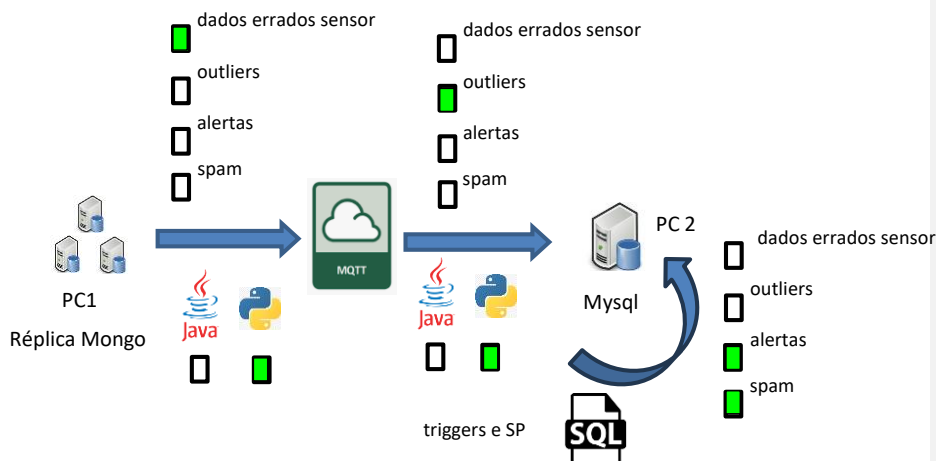
Coleção : game\_configs

```
{  
  "_id": ObjectId("..."),  
  "maxSoundLevel": 75.0,  
  "soundVariationLimit": 15.0,  
  "roomsConfig": [  
    {"roomId": 1, "connectedTo": [2, 3]},  
    {"roomId": 2, "connectedTo": [1, 4]},  
    {"roomId": 3, "connectedTo": [1, 5]},  
    {"roomId": 4, "connectedTo": [2, 5]},  
    {"roomId": 5, "connectedTo": [3, 4]}  
  ]  
}
```



## 1.2 Descrição Geral do Procedimento de Mongo Para Mysql

A passagem do Mongo para Mysql tem duas fases: a) enviar de Mongo para MQTT e b) receber de MQTT para Mysql. No diagrama deve ser indicado (nas check box) em qual das fases são programadas (em java e/ou python) as tarefas enumeradas (podem ser na fase a) b) ou ambas).



- Com que “periodicidade” (segundos) o programa vai buscar ao mongo: 2 em 2 segundos, de forma a receber os dados quase em “tempo real”, mas não sobrecarregando a base de dados com demasiadas consultas
- Como garantem que não enviam duas vezes o mesmo documento do Mongo para o Mysql / MQTT (com base em datas ou booleano ou etc.):

Vamos utilizar um campo "processed" em cada documento no MongoDB. Quando um documento é processado e enviado para o MQTT, este campo será atualizado para "true". Apenas documentos com "processed=false" serão considerados para processamento garantindo que os que foram enviados já não voltam a ser.

- Pensam usar threads do Mongo para MQTT?: nao e do MQTT para Mysql nao?





- Quantas threads e/ou quantos maim em cada um dos passos?:

- No transporte MQTT que QOS vão utilizar? Porquê? `Vamos usar QOS 1 (at least once) pois garante a entrega das mensagens, essencial para não perder dados cruciais do jogo, e oferece bom desempenho com baixa latência. Apesar da possibilidade de dados duplicados, já temos mecanismos para tratar destes casos (campo "processed").`



## PI de Sistemas de Informação Distribuídos, 23/24

Aqui podem desenvolver informação que considerem relevante relativo ao processo de migração, aspectos que não esteja refletido nas secções seguintes.



#### 1.4 Tratamento de dados anómalos (valores de sensores errados)

*Aqui devem explicar o que fazer caso se detectem valores "errados" (datas impossíveis, caracteres estranhos, etc.). Se recorrerem a triggers ou SP então indicam em secção mais adiante.*

*No caso de serem detectados valores "errados", como por exemplo datas impossíveis, mensagens incompletas ou mal formatadas, entre outros... devemos implementar verificações quando os dados são passados do MongoDB para o MQTT. Fazendo a limpeza de dados anómalos já nesta fase evita sobrecarga na passagem de dados que à partida já não serão mais analisados nem usados, melhorando assim a eficiência nas seguintes fases de limpeza.*

*Estes dados anómalos vão ser guardados numa coleção do MongoDB (dados\_invalidos) para poderem ser analisados futuramente.*



### 1.5 Tratamento de outliers de ruído

*Aqui devem explicar o que fazer caso se detectem "outliers" (valores fisicamente possíveis mas irrealistas, como por exemplo variações muito bruscas do ruído apenas num segundo). Se recorrerem a triggers ou SP então indicam em secção mais adiante.*

*Em relação aos outliers decidimos fazer o tratamento na fase de passagem dos dados do mqtt para o MySQL. Isto evita que entrem na BD do MySQL dados irrealistas e incoerentes mesmo que sejam possíveis e tenham passado na primeira fase de limpeza (do mongodb para o mqtt). Por exemplo, vamos ter definido um intervalo máximo de desvio entre dois ruídos consecutivos para que, no caso de haver uma variação muito brusca de ruído, este seja descartado.*

*Nesta fase decidimos também eliminar mensagens redundantes para evitar que a BD cresça desnecessariamente com valores desnecessários, por exemplo com dados de ruído que sejam idênticos em intervalos de tempo curtos. Estes valores são ignorados e não serão passados para a BD.*





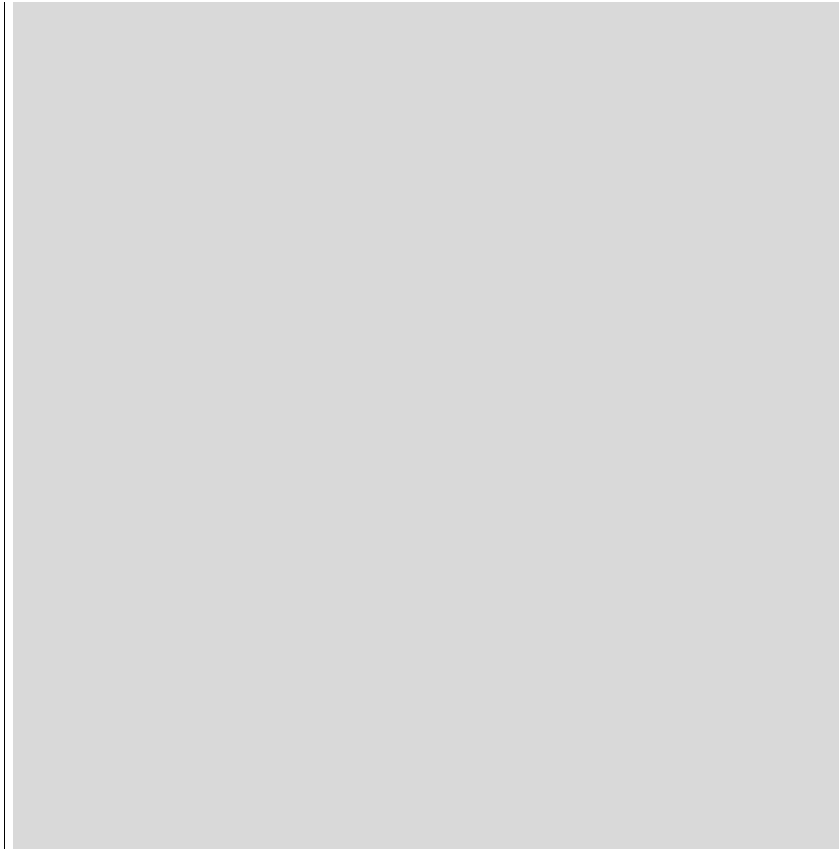
### 1.6 Tratamento de Alertas de ruído

*Aqui devem explicar o que fazer caso se detectem situações alarmantes relativos ao som. Se recorrerem a triggers ou SP então indicam em secção mais adiante. Qu situações despoletam os alertas, onde e como são armazenados. Explicar se existem mecanismos para evitar "spam" (demasiadas mensagens). É aconselhável recorrer a esquemas gráficos para explicar o mecanismo.*

Para os alertas de ruído optámos por gerir as mensagens com base em três níveis. Nível 1 (Atenção) - quando é atingido 70% do ruído máximo; Nível 2 (Crítico) - quando é atingido 85% do ruído máximo; Nível 3 (Limite Atingido) - quando o ruído ultrapassa o ruído máximo. O valor ruído máximo (ruídoMax) está guardado na nossa base de dados na tabela Jogo.

Mecanismo anti-spam: - Um alerta do mesmo nível só pode ser enviado a cada 10 segundos. Isto evita que o jogador receba muitas mensagens de alerta repetidas desnecessariamente.

Ao receber um ruído num desses 3 níveis é criada uma mensagem (tabela Mensagem) que vai ser enviada para o utilizador. Isto é feito recorrendo a um trigger (explicado mais à frente)





### 1.7 Tratamento de número de marsamis numa sala (obter pontuação)

*Aqui devem explicar como funciona o mecanismo que detecta que o número de marsamis odd é (ou vai ser) igual ao número de marsamis even. Como detecta, e o que desencadeia.*

*Para tratar o número de marsamis odd e even em cada sala optamos por quando uma mensagem do sensor de movimento chega à base de dados um trigger é responsável por alterar o número dos marsamis nas respectivas salas (origem e destino). Ao ser alterado o número de marsamis de uma sala é acionado um outro trigger que verifica se o número de marsamis even e odd ficou igual. Caso isso aconteça, é enviada uma mensagem ao jogador a informá-lo disso (tabela Mensagem).*







### 1.8 Especificação de Store Procedures SQL de apoio à migração e tratamento de dados

Nas secções anteriores foram descritos mecanismos que podem ou não necessitar de recorrer a Store Procedures. É nesta tabela que eles deverão ser listados. Na descrição apenas colocar informação que não seja óbvia.

Nome SP	Argumentos	Muito breve descrição
...	...	...



### 1.9 Especificação de Triggers de apoio à migração e tratamento de dados

Nas secções anteriores foram descritos mecanismos que podem ou não necessitar de recorrer a Triggers. É nesta tabela que eles deverão ser listados. Nas Notas apenas colocar informação que não seja óbvia.

Nome Trigger		de Operação (I,U,D)	(After, Before)	
<b>atualizar_ocupacao_movimento</b>	OcupacaoLabirinto	I	AFTER	atualiza automaticamente a contagem de odd e even na tabela OcupacaoLabirinto da sala origem e da sala destino, sempre que recebe uma medicao de passagem
<b>verificar_equilibrio_marsamis</b>	OcupacaoLabirinto	U	AFTER	Verifica se na tabela onde foi atualizado o valor de marsamis odd e even, estes numeros sao ou nao iguais e caso sejam, cria uma mensagem de alerta para o jogador
<b>alerta_ruido</b>	Sound	I	AFTER	Quando o nível de ruído ultrapassa certos limites (como explicado antes) cria-se uma mensagem com um alerta para o



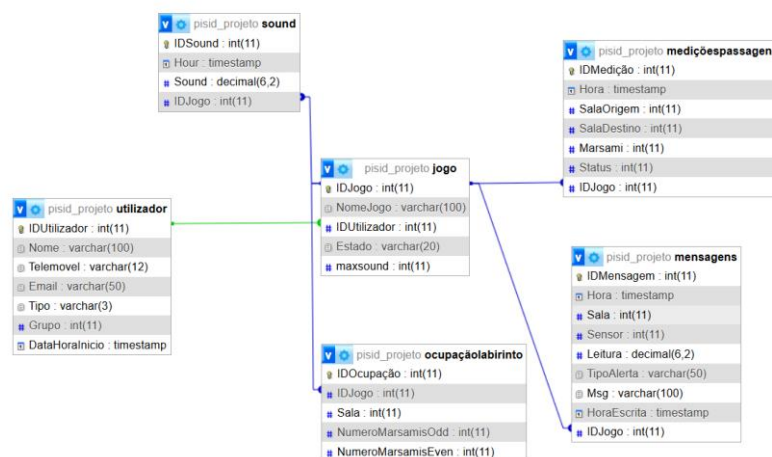
## PI de Sistemas de Informação Distribuídos, 23/24

				utilizador (na tabela Mensagem) para ser enviada
...				



### 1.10 Modelo Relacional

Diagrama relacional completo. Alterações à base de dados original terão de ser justificadas aqui. Caso seja pertinente podem ser adicionadas comentários a justificar opções pouco óbvias.



Na tabela jogo, decidimos incluir o campo "maxSound" que irá armazenar o valor do som máximo permitido em cada jogo. Optámos por adicionar diretamente esse valor para minimizar as consultas feitas à base de dados da nuvem. Desta forma, sempre que um novo jogo é iniciado, o valor do ruído máximo é registado na tabela, o que melhora também a eficiência das comparações realizadas durante o jogo.

(a HoraDataInicio na tabela utilizador é um erro, devia estar na tabela do jogo. Na tabela do utilizador a primary key é o email em vez de ter o IDUtilizador)





### 1.11 Utilizadores Base de Dados Mysql

Nesta secção deverá ser explicado de que forma deverá ser feita a manutenção de utilizadores Mysql. Nomeadamente deverá ser indicado, para cada tipo de utilizador, que privilégios ele tem sobre que tabelas e Stored Procedures (todos os SP usados na Aplicação

Tabela	Tipo de Utilizador		
	Administrador	Investigador	Sistema
Tabela Jogo	U/D	-	I
Tabela Utilizador	L	I	-
Tabela Sound	U/D/I/L	L	I
Tabela MedicoesPassagens	U/D/I/L	L	I
Tabela Mensagens	U/D/I/L	L	I
Tabela OcupacaoLabirinto	U/D/I/L	L	I
Stored Proc.			
Inserir movimento marsami	X	-	X
Atualizar Ocupacao das salas	X	X	-
Verificar equilibrio de marsamis	X	X	-
processar ruido e alertas	X	-	X



acionar gatilho de jogo	X	X	-
abrir porta	X	X	-
fechar porta	X	X	-
recuperar dados de migração	X	-	X
Criar_util izador	X	X	-
Remover_ut ilizador	X	X	-
Criar_jogo	X	X	-
Alterar_jo go	X	X	-

Em que U=Update, I Insert, D- Delete, L=Leitura, X=Executar SP e - sem permissões.

Tabela Formatada





### 1.12 Procedimentos Manutenção da Aplicação

Nesta secção deverão ser listados os SP para a manutenção de utilizadores e jogos (apenas os obrigatórios)

Nome SP	Argumentos	Muito breve descrição
Criar_utilizador	emailUtilizador , nome, telemovel, tipo, grupo	Cria um novo utilizador com os dados fornecidos. O email é o identificador único do utilizador.
Remover_utilizador	email	Remove um utilizador específico do sistema pelo seu email
Alterar_utilizador		
Criar_jogo	nomejogo, emailUtilizador,maxRuido, estado	Cria um novo jogo
Alterar_jogo	idJogo, emailUtilizador	Troca de jogo
...	...	...



### 1.13 Eventos de suporte à aplicação (caso existam)

Deverão ser indicados todos os eventos relevantes para o processo de migração, eventos do Windows e do Mysql. Por eventos entende-se as tarefas do Windows ou eventos do Mysql

Nome Evento	Local Execução (Mysql/Windows)	Muito breve descrição



### 1.14 Consulta por HTML/PHP

*Desenhar o layout dos formulários pretendidos, se relevante colocar texto a explicar a funcionalidade pretendida. Formulários:*

*Fazer login*

*Criar (ou seleccionar de uma lista de jogos um para alterar) um jogo) e, para esse jogo, editar os valores associados. Quando está a alterar não pode alterar valores de chaves estrangeiras e primárias.*

*Indicar para cada botão qual o SP que deverá ser executado*



## 2 Implementação

*Esta secção é para ser preenchida pelo grupo que recebeu o documento. Aqui vão falar da implementação que fizeram. A implementação é o "best of", ou seja, o que **agora** acham que é a melhor solução, com base em tudo o que aprenderam (com as vossas experiências, com as ideias do outro grupo, discussões com o professor, google, chatgpt, etc), no limite podem não seguir nada do que tinham especificado no documento que entregaram ao outro grupo.*

### 2.1 Coleções a criar em cada uma das réplicas do Mongo

Versão	Número Coleções
Especificação inicial	3
Recebida outro grupo	4
<b>Implementada</b>	4

#### Justificação da escolha

*Foi retirado a coleção "Atuador" porque esta não iria ter influência ou utilidade em parte alguma do trabalho.*

*Foram então, adicionadas as coleções "dados\_invalidos" e "game\_configs" recebidas do outro grupo. Julgamos necessário guardar os dados inválidos para fazer, posteriormente, verificação desses mesmos dados para encontrar possíveis problemas e resolvê-los, e utilizar o game\_configs para guardarmos as configurações iniciais do jogo para sabermos que dados estariam inválidos.*

Para cada coleção **implementada** exemplifica um documento

```
Coleção : _movement__ {  
  _id: ObjectId('67dbf608ad60fd16ebe719c0')  
  Player: 9  
  Marsami: 9  
  RoomOrigin: 1
```



RoomDestiny: 3

Status: 1

}

Coleção : \_sound {

id: ObjectId('67dbf612ad60fd16ebe71a1d')

Player: 9

Hour: "2025-03-20 11:03:16.498354"

Sound: 19.196864203740272

}

Coleção : dados\_invalidos {

"tipo": "som", "movimento" "outlier", "invalid sound"

"payload": {

conteúdo original recebido via MQTT

},

"erro": "Descrição do erro",

"timestamp": ISODate("AAAA-MM-DDTHH:MM:SSZ")

}

Coleção : game\_configs {

"\_id": ObjectId("..."),

"NormalNoise": 19,

"soundVariationLimit": 2.5,

"roomsConfig": [

{"roomId": 1, "connectedTo": [2,3]},

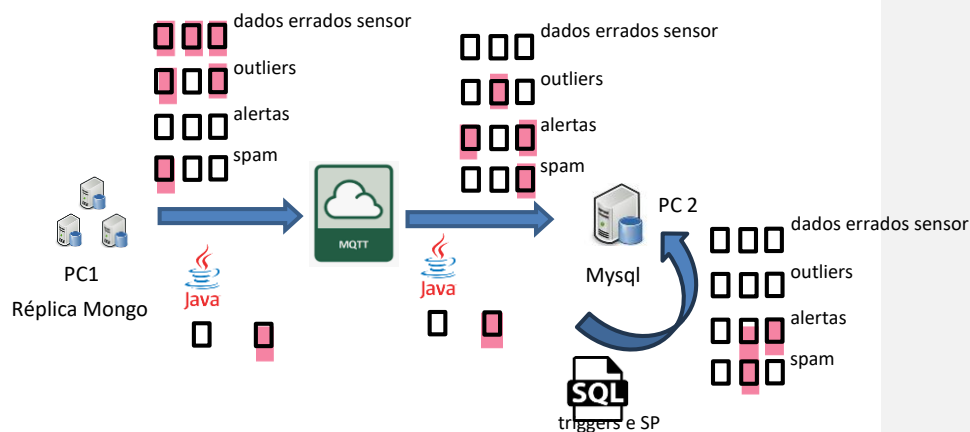


```
{"roomId": 2, "connectedTo": [1, 4]}, {...}]
```

```
}
```

## 2.2 Descrição Geral do Procedimento de Mongo Para Mysql

Na checkbox da esquerda indicam o que especificaram inicialmente, na do meio a especificação do outro grupo e na da direita a vossa implementação.



### Justificação da escolha

Tanto os dados errados do sensor como os outliers são tratados antes de chegarem ao mongo, ou seja, quando recebemos os dados do MQTT, antes de serem colocados dentro do mongo, isto pois assim tratamos logo dos dados anómalos e colocamos logo na coleção dos dados inválidos. Os alertas de ruído são tratados na secção de envio do MQTT para o MySQL, para inserirmos logo na tabela Mensagens sem atraso, e tratamos também do spam nessa secção, evitando que haja alertas de som com um intervalo de X segundos (flexível no código). Consideramos um alerta quando o som excede os 75%, e perigo quando passa os 90%. Os alertas de movimentos (nr par= nr ímpar numa certa sala) são tratados por um trigger no MySQL, e verificamos diretamente na tabela sala quando é que há um número igual de marsamis pares e ímpares.



Versão	Periodicidade vai buscar Mongo	Como evitam enviar duas vezes para MQTT	Número Threads	QOS
Especificação inicial	2	Ficheiro JSON que guarda id's do que já foi enviado	0	2
Recebida outro grupo	2	Booleano que valida se o documento já foi enviado ou não	0	1
<u>Implementada</u>	0	Booleano que valida se o documento já foi enviado ou não	0	2

#### Justificação da escolha

*Optamos por utilizar booleanos como forma de validação, visto que é mais escalável do que a proposta inicial e mais eficiente.*

*Tomamos a decisão de 0s na periodicidade com que se vai buscar ao mongo, pelo que facto de que assim não surgem atrasos na transferência de dados para o MySql.*



*Nas próximas quatro secções devem, na descrição, resumidamente descrever num texto escoreito e legível, a forma como foi implementada. Têm de ficar muito explicitamente indicado a o que não resultou da especificação inicial (cor preta), o que foi aproveitado da especificação que receberam de outro grupo (cor azul) e o que resultou de ideias posteriores vossas (cor verde). Na justificação sejam muito objectivos a explicar a razão de terem alterado a especificação inicial*

Por exemplo (não necessariamente correto), para deteção de valores anómalos:

### 2.3 Tratamento de dados anómalos (valores de sensores errados)

#### **1 Descrição**

Não tratávamos dados anómalos. **Classificamos dados resultantes de som negativo e também considerámos formatos de datas impossíveis** (por exemplo, meses com mais de 31 dias), mensagens com ruídos fora do intervalo [18,30], mensagens incompletas ou mal formatadas, como dados anómalos. Também foi adicionada a coleção “dados\_invalidos” para guardarmos o que foi anteriormente referido. **Os dados inválidos são então guardados nessa mesma coleção.**

#### **2 Justificação das alterações caso tenham havido**

Tínhamos optado apenas pela filtração dos dados anómalos, **desta maneira os dados em vez de serem filtrados (ou seja apagados), são guardados de modo a que possam ser consultados posteriormente.**





## 2.4 Tratamento de outliers de ruído

### **1 Descrição**

Não tratávamos os outliers por considerarmos serem possíveis e não querermos descartar essas hipóteses. Decidimos então implementar um limite de variação possível entre um som e outro para fazer a deteção dos outliers, que é inserido como argumento na execução do script, ou por omissão o seu valor é 10.

### **2 Justificação das alterações caso tenham havido**

Como a hipótese de o ruído mudar bruscamente pelo fechar/abrir de portas não é possível, decidimos então definir uma variação limite de som para descobrir os outliers.



## 2.5 Tratamento de Alertas de Som

### 1 Descrição

*Os alertas são despoletados quando os níveis de ruído / som ultrapassam thresholds de modo a avisar o jogador que está cada vez mais perto de perder o jogo.*

*Estes são armazenados na base de dados MySql, na tabela “Mensagens”.*

*Para evitar spam, implementámos o envio de mensagens de X em X segundos, comparando com o último alerta de som inserido da tabela.*

*Consideramos um alerta de aviso quando o som excede os 75%, e um de perigo quando passa os 90%. Se o último som foi um “alerta” e o próximo seja um “perigo”, mesmo que ainda não tenha passado os X segundos de controlo de spam, a mensagem é enviada na mesma pois mudou o “tipo de alerta”.*

*Tudo isto é tratado no transporte de dados MQTT -> MySQL.*

### 2 Justificação das alterações caso tenham havido

*Os alertas de ruído são tratados diretamente na secção de envio para o MySQL via MQTT, de forma a inseri-los imediatamente na tabela Mensagens sem qualquer atraso. Esta abordagem evita a necessidade de utilizar stored procedures ou triggers adicionais, garantindo assim que os dados estejam disponíveis em tempo real. Como a secção “Mensagens” na aplicação Android depende dessa informação com urgência para análise e visualização imediata, a prioridade aqui foi optar pela solução mais rápida e eficiente.*



## 2.6 Tratamento de número de marsamis numa sala (obter pontuação)

### 1 Descrição

Verificar se o nº de gatilhos dentro da sala é menor do que 3, e após essa verificação verificar se o nº de marsamis odd e even é igual, e caso seja, aciona um gatilho.

Para detectar estes acontecimentos temos de ter um algoritmo, e verificar em que salas os marsamis estão.

Tal algoritmo baseia-se na utilização de um Map que guarda um tuplo da quantidade de Marsamis pares e ímpares em cada sala.

### 2 Justificação das alterações caso tenham havido

Inicialmente utilizávamos dois Maps numa tentativa de adicionar alguma robustez, mas acabamos por retirar o segundo Map pois este acabava por ser uma verificação redundante que não contribuía para a robustez.



## 2.7 Implementação de Stored Procedures SQL de apoio à migração e tratamento de dados

É nesta tabela que deverão ser listados os SP que implementam mecanismos anteriores. Na quarta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

A(og) - Alterado com base em ideia de outro grupo

A(ni) - Alterado com base em novas ideias

N(og) - Novo com base em ideia de outro grupo

N(ni) - Novo com base em ideias novas

A - Alterado com base em novas ideias

Nome SP	Argumentos	Descrição	
...	...	...	

Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher



## 2.8 Implementação de Triggers

É nesta tabela que deverão ser listados os triggers Na sexta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

A(og) – Alterado com base em ideia de outro grupo

A(ni) – Alterado com base em novas ideias

N(og) - Novo com base em ideia de outro grupo

N(ni) – Novo com base em ideias novas

A – Alterado com base em novas ideias

Nome Trigger	Tabela	Tipo de Operação (I,U,D)	Evento (After , Before )	Notas
Criar_salas_apos_criar_jogo	jogo	I	After	Cria as salas após criação de um novo jogo, com 15 Marsamis odd e 15 even na sala 0
atualizar_ocupacao_movimento	movement	U,I	After	O trigger atualiza os contadores de Marsamis pares e ímpares nas salas sempre que há movimento, e cria na tabela mensagens um alerta se numa sala os números ficarem iguais.
verificar_movimentos_finalizados	movement	Insert	After	O trigger finaliza automaticamente um jogo na tabela jogo quando forem inseridos 30 movimentos com



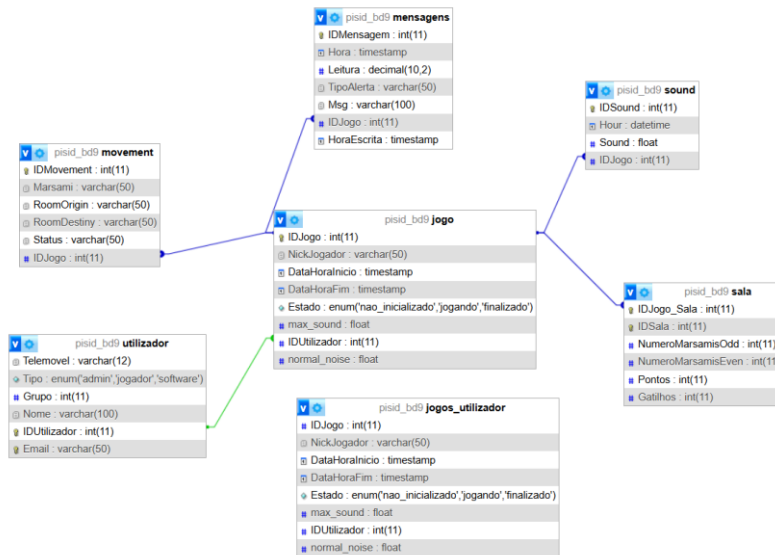
				Status = 2 para esse jogo, atualizando o campo Estado para 'finalizado' e registando a hora do fim no campo DataHoraFim.
--	--	--	--	--

*Com estes triggers, asseguramos que as salas são criadas automaticamente após a criação de um novo jogo e que são atualizadas sempre que há um novo movimento. Se o número de Marsamis ímpares for igual ao número de pares numa sala, é gerada uma mensagem de alerta indicando um possível ponto a marcar. Além disso, o jogo é finalizado automaticamente após receber 30 movimentos com Status = 2, o que marca o fim do jogo, atualizando o seu estado e hora de fim. Isto garante coerência nos dados, evita alterações manuais e proporciona alertas úteis durante o jogo.*

### 1.1 Modelo Relacional

Diagrama relacional completo implementado. Assinalar a azul alterações derivadas de outro grupo e a verde alterações novas.





Na tabela Jogo foram adicionados os campos `normal_noise` e `max_sound(normal_noise + noiservartoleration)`, que são inicializados quando é feita a leitura da tabela `SetUpMaze`, no script python.

Apesar de não estarem representados na implementação anterior a chave estrangeira de `IDUtilizador` vai para a tabela Jogo devido a relação de 1 para \*.

Na tabela Sala temos uma chave composta formada pelo `IDSala` e `IDJogo`.



*Na tabela Mensagens foi removido o atributo sensor, visto que só há um e não vemos necessidade para o propósito deste jogo.*

*Foi removida a ligação entre sala e mensagens visto que as mensagens não dependem de uma sala.*

*Foram adicionadas as associações de movement e mensagens à tabela jogo.*

*Esta troca deve-se ao facto de agora entendermos que os registos da tabela movement e sound não dependem da tabela mensagens, isto é, na nossa implementação anterior assumimos que cada registo de movement e sound teria um campo IDMessage, como chave estrangeira.*

*Nesta implementação vemos a tabela Mensagens como a tabela que apenas vai guardar os alertas que vão para o Android. Logo não faz sentido as associação simples entre movement e sound para com mensagens, então mudámos o relacional para que as três tabelas tenham uma associação simples com a tabela jogo, em que todas recebem a chave estrangeira de jogo(IDJogo).*

*Foi adicionada uma view para que os utilizadores tenham o mínimo de acesso possível à bd. De modo a evitar que os users possam fazer select's na tabela utilizador e aceder a informações de outros users.*





## 1.2 Utilizadores Base de Dados Mysql

Nesta secção deverão ser indicados os utilizadores e perfis implementados (têm de constar todos os SP usados). Assinalar a azul alterações derivadas de outro grupo e a verde alterações novas.

Tabela	Tipo de Utilizador		
	Administrador	Jogador	Software
mensagens	-	-	L/I
movement	-	-	-
sound	-	-	-
jogo	-	-	L
utilizador	I/D/L	-	-
sala	-	-	-
View: jogos_utilizador	-	L/U	-
SP (remover utilizador)	X	-	-
SP (alterar utilizador)	X	-	-
SP	-	X	-



(criar jogo)			
getIdJo go_IdUt ilizado r	-	x	-
get_jog os	-	x	-
get_mar sami_ro om	-	x	-
get_men sagens	-	x	-
get_sen sores	-	x	-
iniciar _jogo	-	x	-

Em que U=Update, I Insert, D- Delete, L=Leitura, X=Executar SP e - sem permissões.

*Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher*

*Foi necessário implementar SP's para manutenção do android e PHP. Deste modo foi implementados os anteriores.*



### 1.3 Procedimentos Manutenção da Aplicação

É nesta tabela que deverão ser listados os SP que implementam mecanismos anteriores. Na quarta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

A(og) - Alterado com base em ideia de outro grupo

A(ni) - Alterado com base em novas ideias

N(og) - Novo com base em ideia de outro grupo

N(ni) - Novo com base em ideias novas

A - Alterado com base em novas ideias

Nome SP	Argumentos	Descrição	
getIdJogo_IdUtilizador	-	Retorna o ID do jogo mais recente e o ID do utilizador	N(ni)
get_jogos	-	Retorna a lista de jogos não inicializados	N(ni)
get_marsami_roma	-	Retorna todos os dados da sala associada ao mais recente	N(ni)
get_mensagens	-	Retorna as mensagens enviadas na última hora	N(ni)
get_sensores	IDJogo	Retorna os registos de som dos últimos 10 segundos para um jogo específico	N(ni)



iniciar_jogo	IDJogo	Inicia um jogo	$N(ni)$
--------------	--------	----------------	---------

*Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher*

Ao longo do desenvolvimento do projeto, foi-se pensado em SP's para facilitar a manutenção da aplicação android e PHP. Cada SP facilita o que foi dito na descrição dos mesmos.



#### 1.4 Eventos de suporte à aplicação (caso existam)

É nesta tabela que eles deverão ser listados todos os eventos relevantes para o processo de migração Na quarta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

A(og) – Alterado com base em ideia de outro grupo

A(ni) – Alterado com base em novas ideias

N(og) - Novo com base em ideia de outro grupo

N(ni) – Novo com base em ideias novas

A – Alterado com base em novas ideias

Nome Evento	Local Execução (Mysql/Windows)	Muito breve descrição

Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher



### 1.5 PrintScreen dos formulários HTML implementados

The screenshot shows a login form with the following elements:

- Log in** (Title)
- Email do Jogador** (Label)
- (Input field)
- Senha** (Label)
- (Input field)
- (Submit button)



**Selecione o jogo que  
pretende alterar:**

Selecione um jogo ▼

Criar Novo Jogo

Logout

**Novo Jogo**

**Nick do Jogador:**

Insira o Nick do jogador

Criar

Voltar



**Selecione o jogo que pretende alterar:**

flash ▼

Alterar Jogo

Iniciar Jogo

Criar Novo Jogo

Logout

**Nick Jogador Atual: teste**

**Novo Nick do Jogador:**

Insira o Novo Nick do Jogador

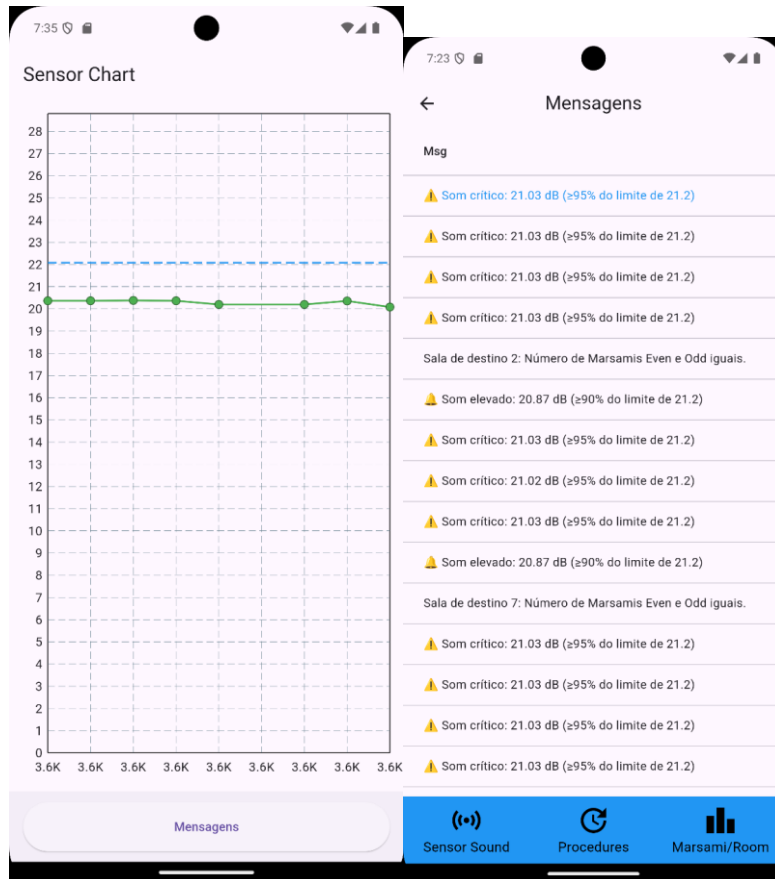
Alterar

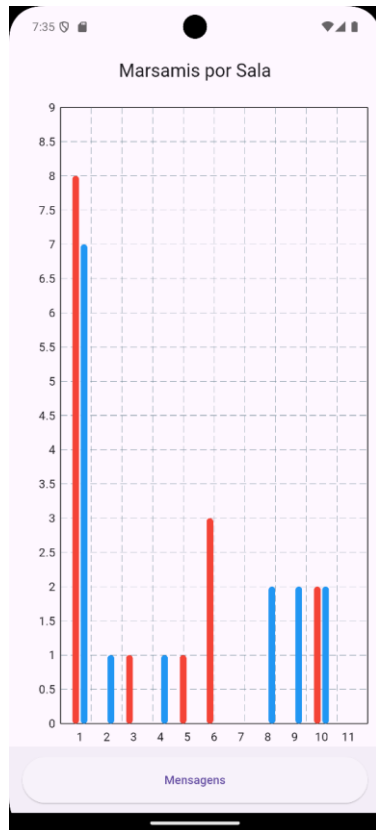
Voltar Iniciar





### *1.6 PrintScreen do formulários Android com dados*





## Anexo Código SQL

```
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Tempo de geração: 11-Maio-2025 às 00:37
-- Versão do servidor: 10.4.32-MariaDB
-- versão do PHP: 8.2.12
```



```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";

START TRANSACTION;

SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;


--
-- Banco de dados: `psid_bd9`
--

DELIMITER $$

--
-- Procedimentos
--

CREATE DEFINER=`root`@`localhost` PROCEDURE `alterar_jogo` (IN `p_idjogo` INT, IN
`p_NickJogador` VARCHAR(50)) BEGIN

    DECLARE v_id_utilizador INT;

    DECLARE v_email VARCHAR(50);

    SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

    SELECT IDUtilizador INTO v_id_utilizador

    FROM utilizador
```



```
WHERE Email = v_email;

IF EXISTS (

    SELECT 1 FROM jogo

    WHERE IDJogo = p_idjogo

    AND Estado != 'jogando'

    AND IDUtilizador = v_id_utilizador

) THEN

    UPDATE jogo

    SET NickJogador = p_NickJogador

    WHERE IDJogo = p_idjogo;

ELSE

    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Você não tem permissão para alterar
este jogo.';

END IF;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `alterar_utilizador` (IN `p_emailAntigo`
VARCHAR(50), IN `p_emailNovo` VARCHAR(50), IN `p_NomeNovo` VARCHAR(100), IN
`p_TelemoveINovo` VARCHAR(12), IN `p_GrupoNovo` INT, IN `p_SenhaNovaOpcional`
VARCHAR(100)) BEGIN

    DECLARE v_email_antigo VARCHAR(50);

    DECLARE v_IDUtilizador INT;

    DECLARE sql_query VARCHAR(1000);

    SET v_email_antigo = p_emailAntigo;

    SELECT IDUtilizador INTO v_IDUtilizador
```



```
FROM utilizador

WHERE Email = v_email_antigo;

IF p_emailNovo IS NOT NULL AND p_emailNovo != "" AND TRIM(p_emailNovo) !=
TRIM(v_email_antigo) THEN

    -- Atualiza dados e email

    UPDATE utilizador

    SET Telemovel = p_TelemovelNovo,

        Grupo = p_GrupoNovo,

        Nome = p_NomeNovo,

        Email = p_emailNovo

    WHERE IDUtilizador = v_IDUtilizador;

    -- Renomeia o utilizador do sistema

    SET sql_query = CONCAT('RENAME USER "', v_email_antigo, '"@"localhost" TO "',
p_emailNovo, '"@"localhost";');

    PREPARE stmt FROM sql_query;

    EXECUTE stmt;

    DEALLOCATE PREPARE stmt;

    -- Atualiza email antigo para o novo

    SET v_email_antigo = p_emailNovo;

ELSE

    -- Só atualiza os restantes dados (sem mexer no email)

    UPDATE utilizador

    SET Telemovel = p_TelemovelNovo,

        Grupo = p_GrupoNovo,
```



```
Nome = p_NomeNovo

WHERE IDUtilizador = v_IDUtilizador;

END IF;

-- Se a senha foi passada, altera também
IF p_SenhaNovaOpcional IS NOT NULL AND p_SenhaNovaOpcional != '' THEN

    SET sql_query = CONCAT('ALTER USER ', v_email_antigo, '"@"localhost" IDENTIFIED BY "',
p_SenhaNovaOpcional, '";');

    PREPARE stmt FROM sql_query;

    EXECUTE stmt;

    DEALLOCATE PREPARE stmt;

END IF;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `criar_jogo` (IN `p_nick_jogador`
VARCHAR(50)) BEGIN

    DECLARE v_id_utilizador INT;

    DECLARE v_email VARCHAR(50);

    SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

    -- Tenta obter o ID do utilizador

    SELECT IDUtilizador INTO v_id_utilizador

    FROM utilizador

    WHERE Email = v_email;

    -- Se não encontrou nenhum ID, dá erro (mas tratamos o erro com CONTINUE HANDLER)
```



## PI de Sistemas de Informação Distribuídos, 23/24

```
IF v_id_utilizador IS NULL THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'Utilizador não encontrado com o email do utilizador atual.';

ELSE

    -- Inserir o novo jogo

    INSERT INTO jogo (

        NickJogador,

        Estado,

        IDUtilizador

    )

    VALUES (

        p_nick_jogador,

        'nao_inicializado',

        v_id_utilizador

    );

    -- Mensagem de sucesso

    SELECT 'Jogo criado com sucesso!' AS Mensagem;

END IF;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `criar_utilizador` (IN `p_email` VARCHAR(50),
IN `p_nome` VARCHAR(100), IN `p_telemovei` VARCHAR(12), IN `p_tipo` VARCHAR(20), IN
`p_grupo` INT, IN `p_password` VARCHAR(100)) BEGIN

    DECLARE EXIT HANDLER FOR SQLEXCEPTION

    BEGIN

        -- Em caso de erro, desfaz qualquer mudança (opcional: ROLLBACK)
```





```
SELECT 'Erro ao criar utilizador.' AS MensagemErro;

END;

-- 1. Verifica se o utilizador já existe

IF NOT EXISTS (SELECT 1 FROM utilizador WHERE Email = p_email) THEN

-- 2. Insere na tabela

INSERT INTO utilizador (Email, Nome, Telemovel, Tipo, Grupo)

VALUES (p_email, p_nome, p_telemovei, p_tipo, p_grupo);

-- 3. Cria o utilizador MySQL

SET @sql_create = CONCAT(

    'CREATE USER "', p_email, '"@"localhost" IDENTIFIED BY "', p_password, '",'

);

PREPARE stmt1 FROM @sql_create;

EXECUTE stmt1;

DEALLOCATE PREPARE stmt1;

-- 4. Concede a role (pré-criada) ao utilizador

SET @sql_grant = CONCAT('GRANT ', p_tipo, ' TO "', p_email, '"@"localhost";');

PREPARE stmt2 FROM @sql_grant;

EXECUTE stmt2;

DEALLOCATE PREPARE stmt2;

-- 4.1 Define como role padrão

SET @sql_default = CONCAT('SET DEFAULT ROLE ', p_tipo, ' TO "', p_email, '"@"localhost";');

PREPARE stmt3 FROM @sql_default;
```



```
EXECUTE stmt3;

DEALLOCATE PREPARE stmt3;

-- 5. Mensagem de sucesso

SELECT 'Utilizador criado com sucesso!' AS Mensagem;

ELSE

    SELECT 'Utilizador já existe.' AS Mensagem;

END IF;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `getIdJogo_IdUtilizador`() BEGIN

DECLARE v_id_utilizador INT;

DECLARE v_email VARCHAR(50);

-- Extrai o email do utilizador atual (até o '@')

SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

-- Tenta obter o ID do utilizador

SELECT IDUtilizador INTO v_id_utilizador

FROM utilizador

WHERE Email = v_email;

-- Retorna o ID do jogo mais recente criado por esse utilizador

SELECT IDJogo, v_id_utilizador AS IDUtilizador

FROM jogo

WHERE IDUtilizador = v_id_utilizador AND Estado = 'jogando'
```



```
ORDER BY IDJogo DESC

LIMIT 1;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `get_jogos` () BEGIN
    DECLARE email_atual VARCHAR(100);

    -- Extrai apenas o nome de utilizador da função USER() (antes do @)
    SET email_atual = SUBSTRING_INDEX(USER(), '@', 2);

    -- Retorna os jogos do utilizador autenticado
    SELECT j.IDJogo, j.NickJogador
    FROM jogo j
    JOIN utilizador u ON j.IDUtilizador = u.IDUtilizador
    WHERE u.Email = email_atual
    AND j.Estado = 'nao_inicializado';

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `get_marsami_room` () BEGIN
    SELECT *
    FROM sala
    WHERE IDJogo_Sala = (SELECT MAX(IDJogo_Sala) FROM sala)
    ORDER BY IDSala;

END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `get_mensagens` () BEGIN
    SELECT Msg, Leitura, TipoAlerta, Hora, HoraEscrita
```



## PI de Sistemas de Informação Distribuídos, 23/24

```
FROM mensagens

WHERE HoraEscrita >= NOW() - INTERVAL 60 MINUTE

ORDER BY HoraEscrita DESC;

END$$
```

```
CREATE DEFINER='root'@'localhost' PROCEDURE `get_sensores` (IN `p_idjogo`
INT(50)) BEGIN

SELECT s.Hour, s.Sound, j.normal_noise

FROM sound s

JOIN jogo j ON j.IDJogo = p_idJogo

WHERE s.IDJogo = p_idJogo AND s.Hour >= NOW() - INTERVAL 10 SECOND

ORDER BY s.Hour DESC;

END$$
```

```
CREATE DEFINER='root'@'localhost' PROCEDURE `iniciar_jogo` (IN `p_idjogo` INT) BEGIN

DECLARE v_id_utilizador INT;

DECLARE v_email VARCHAR(50);

DECLARE timenow timestamp;

set timenow = CURRENT_TIMESTAMP;

SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

SELECT IDUtilizador INTO v_id_utilizador

FROM utilizador

WHERE Email = v_email;
```



```
IF EXISTS (
    SELECT * FROM jogo
    WHERE IDJogo = p_idjogo
    AND Estado = 'nao_inicializado'
    AND IDUtilizador = v_id_utilizador
) THEN
    UPDATE jogo
    SET Estado = 'jogando', DataHoralInicio = timenow
    WHERE IDJogo = p_idjogo;
ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Você não tem permissão para alterar
este jogo.';
END IF;
END$$

CREATE DEFINER='root'@'localhost' PROCEDURE `remover_utilizador` (IN `p_email`
VARCHAR(50)) BEGIN
    DECLARE v_IDUtilizador INT(100);

    -- 1. Verifica se o utilizador existe e obtém o email
    SELECT IDUtilizador INTO v_IDUtilizador
    FROM utilizador
    WHERE Email = p_email;

    -- 2. Se encontrou o email, continua
    IF p_email IS NOT NULL THEN
        -- 3. Remove da tabela pisode.utilizador
```



```
DELETE FROM utilizador WHERE IDUtilizador = v_IDUtilizador;

-- 4. Remove o utilizador do MySQL

SET @sql_drop = CONCAT(
    'DROP USER IF EXISTS "', p_email, '"@"localhost"'
);

PREPARE stmt1 FROM @sql_drop;

EXECUTE stmt1;

DEALLOCATE PREPARE stmt1;

END IF;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `validar_login` (IN `p_email` VARCHAR(100),
IN `p_senha` VARCHAR(100)) BEGIN

    SELECT IDUtilizador, Nome, Email

    FROM utilizador

    WHERE Email = p_email; -- idealmente, Senha deve estar encriptada

END$$

DELIMITER ;

-----

--

-- Estrutura da tabela `jogo`

--
```



```
CREATE TABLE `jogo` (  
  `IDJogo` int(11) NOT NULL,  
  `NickJogador` varchar(50) DEFAULT NULL,  
  `DataHoraInicio` timestamp NULL DEFAULT NULL,  
  `DataHoraFim` timestamp NULL DEFAULT NULL,  
  `Estado` enum('nao_inicializado','jogando','finalizado') DEFAULT NULL,  
  `max_sound` float DEFAULT NULL,  
  `IDUtilizador` int(11) NOT NULL,  
  `normal_noise` float DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
--  
  
-- Extraindo dados da tabela `jogo`  
  
--  
  
INSERT INTO `jogo` (`IDJogo`, `NickJogador`, `DataHoraInicio`, `DataHoraFim`, `Estado`,  
  `max_sound`, `IDUtilizador`, `normal_noise`) VALUES  
(1, 'Mufasa', NULL, NULL, 'nao_inicializado', NULL, 6, NULL);  
  
--  
  
-- Acionadores `jogo`  
  
--  
  
DELIMITER $$  
  
CREATE TRIGGER `criar_salas_apos_criar_jogo` AFTER INSERT ON `jogo` FOR EACH ROW BEGIN  
  DECLARE i INT DEFAULT 0;  
  
  -- Criação das 11 salas (de 0 a 10) para o novo jogo
```



```
WHILE i <= 10 DO

  -- Verifica se a combinação de IDJogo_Sala e IDSala já existe na tabela

  IF NOT EXISTS (SELECT 1 FROM sala WHERE IDJogo_Sala = NEW.IDJogo AND IDSala = i) THEN

    -- Se for a sala 0, coloca valores específicos para MarsamiOdd e MarsamiEven

    IF i = 0 THEN

      INSERT INTO sala (IDJogo_Sala, IDSala, NumeroMarsamisOdd, NumeroMarsamisEven,
Pontos, Gatilhos)

        VALUES (NEW.IDJogo, i, 15, 15, 0, 0);

    ELSE

      INSERT INTO sala (IDJogo_Sala, IDSala, NumeroMarsamisOdd, NumeroMarsamisEven,
Pontos, Gatilhos)

        VALUES (NEW.IDJogo, i, 0, 0, 0, 0);

    END IF;

  END IF;

  SET i = i + 1;

END WHILE;

END

$$

DELIMITER ;

-- -----

--

-- Estrutura stand-in para vista `jogos_utilizador`

-- (Veja abaixo para a view atual)

--
```





```
CREATE TABLE `jogos_utilizador` (  
  `IDJogo` int(11)  
  , `NickJogador` varchar(50)  
  , `DataHoraInicio` timestamp  
  , `DataHoraFim` timestamp  
  , `Estado` enum('nao_inicializado','jogando','finalizado')  
  , `max_sound` float  
  , `IDUtilizador` int(11)  
  , `normal_noise` float  
);  
  
-----  
  
--  
  
-- Estrutura da tabela `mensagens`  
--  
  
CREATE TABLE `mensagens` (  
  `IDMensagem` int(11) NOT NULL,  
  `Hora` timestamp NULL DEFAULT NULL,  
  `Leitura` decimal(10,2) DEFAULT NULL,  
  `TipoAlerta` varchar(50) DEFAULT NULL,  
  `Msg` varchar(100) DEFAULT NULL,  
  `IDJogo` int(11) NOT NULL,  
  `HoraEscrita` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE  
current_timestamp()  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```



```
-- -----  
  
--  
-- Estrutura da tabela `movement`  
--  
  
CREATE TABLE `movement` (  
  `IDMovement` int(11) NOT NULL,  
  `Marsami` varchar(50) DEFAULT NULL,  
  `RoomOrigin` varchar(50) DEFAULT NULL,  
  `RoomDestiny` varchar(50) DEFAULT NULL,  
  `Status` varchar(50) DEFAULT NULL,  
  `IDJogo` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
--  
-- Acionadores `movement`  
--  
DELIMITER $$  
  
CREATE TRIGGER `atualizar_ocupacao_movimento` AFTER INSERT ON `movement` FOR EACH  
ROW BEGIN  
  
  DECLARE marsami_num INT;  
  
  SET marsami_num = CAST(NEW.Marsami AS UNSIGNED);  
  
  -- Atualiza sala de origem  
  
  IF (marsami_num % 2 = 0) THEN
```



```
UPDATE sala

SET NumeroMarsamisEven = IFNULL(NumeroMarsamisEven, 0) - 1

WHERE IDSala = NEW.RoomOrigin;

ELSE

UPDATE sala

SET NumeroMarsamisOdd = IFNULL(NumeroMarsamisOdd, 0) - 1

WHERE IDSala = NEW.RoomOrigin;

END IF;

-- Atualiza sala de destino

IF (marsami_num % 2 = 0) THEN

UPDATE sala

SET NumeroMarsamisEven = IFNULL(NumeroMarsamisEven, 0) + 1

WHERE IDSala = NEW.RoomDestiny;

ELSE

UPDATE sala

SET NumeroMarsamisOdd = IFNULL(NumeroMarsamisOdd, 0) + 1

WHERE IDSala = NEW.RoomDestiny;

END IF;

-- Verifica a sala de origem

IF EXISTS (

SELECT 1 FROM sala

WHERE IDSala = NEW.RoomOrigin AND NumeroMarsamisEven > 1

AND IFNULL(NumeroMarsamisEven,0) = IFNULL(NumeroMarsamisOdd,0)

) THEN

INSERT INTO mensagens (IDJogo, Hora,Leitura, TipoAlerta, Msg, HoraEscrita)
```



## PI de Sistemas de Informação Distribuídos, 23/24

```
VALUES (NEW.IDJogo, null, 0, 'Alerta Igualdade',

        CONCAT('Sala de origem ', NEW.RoomOrigin, ': Número de Marsamis Even e Odd
iguais.'),

        NOW());

END IF;

-- Verifica a sala de destino

IF EXISTS (

    SELECT 1 FROM sala

    WHERE IDSala = NEW.RoomDestiny AND NumeroMarsamisEven > 1

    AND IFNULL(NumeroMarsamisEven,0) = IFNULL(NumeroMarsamisOdd,0)

) THEN

    INSERT INTO mensagens (IDJogo, Hora, Leitura, TipoAlerta, Msg, HoraEscrita)

    VALUES (NEW.IDJogo, null, 0, 'Alerta Igualdade',

            CONCAT('Sala de destino ', NEW.RoomDestiny, ': Número de Marsamis Even e Odd
iguais.'),

            NOW());

END IF;

END

$$

DELIMITER ;

-- -----

--

-- Estrutura da tabela `sala`
```



--

```
CREATE TABLE `sala` (  
  `IDJogo_Sala` int(11) NOT NULL,  
  `IDSala` int(11) NOT NULL,  
  `NumeroMarsamisOdd` int(11) DEFAULT NULL,  
  `NumeroMarsamisEven` int(11) DEFAULT NULL,  
  `Pontos` int(11) DEFAULT NULL,  
  `Gatilhos` int(11) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

--

-- Extraindo dados da tabela `sala`

--

```
INSERT INTO `sala` (`IDJogo_Sala`, `IDSala`, `NumeroMarsamisOdd`, `NumeroMarsamisEven`,  
  `Pontos`, `Gatilhos`) VALUES  
  
(1, 0, 15, 15, 0, 0),  
  
(1, 1, 0, 0, 5, 0),  
  
(1, 2, 0, 0, 0, 0),  
  
(1, 3, 0, 0, 0, 0),  
  
(1, 4, 0, 0, 0, 0),  
  
(1, 5, 0, 0, 2, 0),  
  
(1, 6, 0, 0, 0, 0),  
  
(1, 7, 0, 0, 0, 0),  
  
(1, 8, 0, 0, 0, 0),  
  
(1, 9, 0, 0, 0, 0),
```



(1, 10, 0, 0, 0, 0);

-----

--

-- Estrutura da tabela `sound`

--

```
CREATE TABLE `sound` (  
  `IDSound` int(11) NOT NULL,  
  `Hour` datetime DEFAULT NULL,  
  `Sound` float DEFAULT NULL,  
  `IDJogo` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

-----

--

-- Estrutura da tabela `utilizador`

--

```
CREATE TABLE `utilizador` (  
  `Telemovel` varchar(12) DEFAULT NULL,  
  `Tipo` enum('admin','jogador','software') NOT NULL,  
  `Grupo` int(11) DEFAULT NULL,  
  `Nome` varchar(100) DEFAULT NULL,  
  `IDUtilizador` int(11) NOT NULL,
```



```
`Email` varchar(50) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--

-- Extraindo dados da tabela `utilizador`

--

INSERT INTO `utilizador` (`Telemovel`, `Tipo`, `Grupo`, `Nome`, `IDUtilizador`, `Email`) VALUES
('111222333', 'jogador', 9, 'jogador', 6, 'jogador@gmail.com'),
('444555666', 'admin', 9, 'admin', 7, 'admin@gmail.com'),
('777888999', 'software', 9, 'software', 8, 'software@gmail.com');

-----

--

-- Estrutura para vista `jogos_utilizador`

--

DROP TABLE IF EXISTS `jogos_utilizador`;

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`jogos_utilizador` AS SELECT `j`.`IDJogo` AS `IDJogo`, `j`.`NickJogador` AS `NickJogador`,
`j`.`DataHoraInicio` AS `DataHoraInicio`, `j`.`DataHoraFim` AS `DataHoraFim`, `j`.`Estado` AS
`Estado`, `j`.`max_sound` AS `max_sound`, `j`.`IDUtilizador` AS `IDUtilizador`, `j`.`normal_noise`
AS `normal_noise` FROM (`jogo` `j` join `utilizador` `u` on(`j`.`IDUtilizador` = `u`.`IDUtilizador`))
WHERE `u`.`Email` = substring_index(user(), '@', 2);

--

-- Índices para tabelas despejadas

--
```



```
--  
-- Índices para tabela `jogo`  
--  
ALTER TABLE `jogo`  
  ADD PRIMARY KEY (`IDJogo`),  
  ADD KEY `IDUtilizador_Jogo` (`IDUtilizador`) USING BTREE;  
  
--  
-- Índices para tabela `mensagens`  
--  
ALTER TABLE `mensagens`  
  ADD PRIMARY KEY (`IDMensagem`) USING BTREE,  
  ADD KEY `IDJogo_Mensagens` (`IDJogo`) USING BTREE;  
  
--  
-- Índices para tabela `movement`  
--  
ALTER TABLE `movement`  
  ADD PRIMARY KEY (`IDMovement`),  
  ADD KEY `IDJogo_Movement` (`IDJogo`) USING BTREE;  
  
--  
-- Índices para tabela `sala`  
--  
ALTER TABLE `sala`  
  ADD PRIMARY KEY (`IDSala`, `IDJogo_Sala`) USING BTREE,
```





```
ADD KEY `IDJogo_Sala` (`IDJogo_Sala`) USING BTREE;

--

-- Índices para tabela `sound`
--

ALTER TABLE `sound`

  ADD PRIMARY KEY (`IDSound`),

  ADD KEY `IDJogo_Sound` (`IDJogo`) USING BTREE;

--

-- Índices para tabela `utilizador`
--

ALTER TABLE `utilizador`

  ADD PRIMARY KEY (`IDUtilizador`),

  ADD UNIQUE KEY `Email` (`Email`);

--

-- AUTO_INCREMENT de tabelas despejadas
--

--

-- AUTO_INCREMENT de tabela `jogo`
--

ALTER TABLE `jogo`

  MODIFY `IDJogo` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;

--
```



```
-- AUTO_INCREMENT de tabela `mensagens`  
  
--  
  
ALTER TABLE `mensagens`  
  
  MODIFY `IDMensagem` int(11) NOT NULL AUTO_INCREMENT;  
  
--  
  
-- AUTO_INCREMENT de tabela `movement`  
  
--  
  
ALTER TABLE `movement`  
  
  MODIFY `IDMovement` int(11) NOT NULL AUTO_INCREMENT;  
  
--  
  
-- AUTO_INCREMENT de tabela `sound`  
  
--  
  
ALTER TABLE `sound`  
  
  MODIFY `IDSound` int(11) NOT NULL AUTO_INCREMENT;  
  
--  
  
-- AUTO_INCREMENT de tabela `utilizador`  
  
--  
  
ALTER TABLE `utilizador`  
  
  MODIFY `IDUtilizador` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;  
  
--  
  
-- Restrições para despejos de tabelas  
  
--
```



```
--  
  
-- Limitadores para a tabela `jogo`  
  
--  
  
ALTER TABLE `jogo`  
  
  ADD CONSTRAINT `IDUtilizador_Jogo` FOREIGN KEY (`IDUtilizador`) REFERENCES `utilizador`  
  (`IDUtilizador`) ON DELETE CASCADE ON UPDATE CASCADE;  
  
--  
  
-- Limitadores para a tabela `mensagens`  
  
--  
  
ALTER TABLE `mensagens`  
  
  ADD CONSTRAINT `IDJogo_Mensagens` FOREIGN KEY (`IDJogo`) REFERENCES `jogo` (`IDJogo`)  
  ON DELETE CASCADE ON UPDATE CASCADE;  
  
--  
  
-- Limitadores para a tabela `movement`  
  
--  
  
ALTER TABLE `movement`  
  
  ADD CONSTRAINT `IDJogo_Movement` FOREIGN KEY (`IDJogo`) REFERENCES `jogo` (`IDJogo`)  
  ON DELETE CASCADE ON UPDATE CASCADE;  
  
--  
  
-- Limitadores para a tabela `sala`  
  
--  
  
ALTER TABLE `sala`  
  
  ADD CONSTRAINT `IDJogo_Sala` FOREIGN KEY (`IDJogo_Sala`) REFERENCES `jogo` (`IDJogo`)  
  ON DELETE CASCADE ON UPDATE CASCADE;
```



```
--  
  
-- Limitadores para a tabela `sound`  
  
--  
  
ALTER TABLE `sound`  
  
  ADD CONSTRAINT `IDJogo_Sound` FOREIGN KEY (`IDJogo`) REFERENCES `jogo` (`IDJogo`) ON  
  DELETE CASCADE ON UPDATE CASCADE;  
  
COMMIT;  
  
  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```



#### Código de Triggers implementados

```
1. Nome Trigger: criar_salas_apos_criar_jogo

Cria as salas após criação de um novo jogo, com 15
Marsamis odd e 15 even na sala 0

CREATE TRIGGER `criar_salas_apos_criar_jogo` AFTER INSERT
ON `jogo` FOR EACH ROW BEGIN

    DECLARE i INT DEFAULT 0;

    -- Criação das 11 salas (de 0 a 10) para o novo jogo
    WHILE i <= 10 DO

        -- Verifica se a combinação de IDJogo_Sala e IDSala
        já existe na tabela

        IF NOT EXISTS (SELECT 1 FROM sala WHERE IDJogo_Sala =
NEW.IDJogo AND IDSala = i) THEN

            -- Se for a sala 0, coloca valores específicos para
MarsamiOdd e MarsamiEven

            IF i = 0 THEN

                INSERT INTO sala (IDJogo_Sala, IDSala,
NumeroMarsamisOdd, NumeroMarsamisEven, Pontos, Gatilhos)

                VALUES (NEW.IDJogo, i, 15, 15, 0, 0);

            ELSE

                INSERT INTO sala (IDJogo_Sala, IDSala,
NumeroMarsamisOdd, NumeroMarsamisEven, Pontos, Gatilhos)

                VALUES (NEW.IDJogo, i, 0, 0, 0, 0);

            END IF;

        END IF;

    END IF;
```



```
        SET i = i + 1;

    END WHILE;

END

2. Nome Trigger: atualizar_ocupacao_movimento
// Breve Descrição
O trigger atualiza os contadores de Marsamis pares e
ímpares nas salas sempre que há movimento, e cria um
alerta se numa sala os números ficarem iguais.

CREATE TRIGGER `atualizar_ocupacao_movimento` AFTER
INSERT ON `movement` FOR EACH ROW BEGIN

    DECLARE marsami_num INT;

    SET marsami_num = CAST(NEW.Marsami AS UNSIGNED);

    -- Atualiza sala de origem

    IF (marsami_num % 2 = 0) THEN

        UPDATE sala

            SET NumeroMarsamisEven =
IFNULL(NumeroMarsamisEven, 0) - 1

            WHERE IDSala = NEW.RoomOrigin;

    ELSE

        UPDATE sala

            SET NumeroMarsamisOdd = IFNULL(NumeroMarsamisOdd,
0) - 1

            WHERE IDSala = NEW.RoomOrigin;
```



```
END IF;

-- Atualiza sala de destino

IF (marsami_num % 2 = 0) THEN

    UPDATE sala

    SET NumeroMarsamisEven =
IFNULL(NumeroMarsamisEven, 0) + 1

    WHERE IDSala = NEW.RoomDestiny;

ELSE

    UPDATE sala

    SET NumeroMarsamisOdd = IFNULL(NumeroMarsamisOdd,
0) + 1

    WHERE IDSala = NEW.RoomDestiny;

END IF;

-- Verifica a sala de origem

IF EXISTS (

    SELECT 1 FROM sala

    WHERE IDSala = NEW.RoomOrigin

    AND IFNULL(NumeroMarsamisEven,0) =
IFNULL(NumeroMarsamisOdd,0)

) THEN

    INSERT INTO mensagens (IDJogo, Hora,Leitura,
TipoAlerta, Msg, HoraEscrita)

    VALUES (NEW.IDJogo, null, 0, 'Alerta Igualdade',

            CONCAT('Sala de origem ', NEW.RoomOrigin,
': Número de Marsamis Even e Odd iguais.'),

            NOW());
```



```
END IF;

-- Verifica a sala de destino

IF EXISTS (

    SELECT 1 FROM sala

    WHERE IDSala = NEW.RoomDestiny

    AND IFNULL(NúmeroMarsamisEven,0) =
IFNULL(NúmeroMarsamisOdd,0)

) THEN

    INSERT INTO mensagens (IDJogo, Hora, Leitura,
TipoAlerta, Msg, HoraEscrita)

    VALUES (NEW.IDJogo, null, 0, 'Alerta Igualdade',

            CONCAT('Sala de destino ',
NEW.RoomDestiny, ': Número de Marsamis Even e Odd
iguais.'),

            NOW());

END IF;

END
```

### 3. Nome Trigger: verificar\_movimentos\_finalizados

// Breve Descrição

O trigger finaliza automaticamente um jogo na tabela jogo quando forem inseridos 30 movimentos com Status = 2 para esse jogo, atualizando o campo Estado para 'finalizado' e registrando a hora do fim no campo DataHoraFim.





```
BEGIN

    DECLARE total_movs INT;

    -- Só processa se o novo status for 2
    IF NEW.Status = 2 THEN

        -- Conta quantos movimentos desse jogo têm status
= 2

        SELECT COUNT(*) INTO total_movs
        FROM movement
        WHERE IDJogo = NEW.IDJogo AND Status = 2;

        -- Se forem 30 ou mais, atualiza o estado do jogo
e define a hora de fim
        IF total_movs >= 30 THEN

            UPDATE jogo

            SET Estado = 'finalizado',

                DataHoraFim = NOW()

            WHERE IDJogo = NEW.IDJogo;

        END IF;

    END IF;

END
```



Código Stored Procedures implementados

```
1. Nome SP: CriarUtilizador

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
    `criar_utilizador`(IN `p_email` VARCHAR(50), IN
    `p_nome` VARCHAR(100), IN `p_telemovel` VARCHAR(12),
    IN `p_tipo` VARCHAR(20), IN `p_grupo` INT, IN
    `p_password` VARCHAR(100))

BEGIN

    DECLARE EXIT HANDLER FOR SQLEXCEPTION

    BEGIN

        -- Em caso de erro, desfaz qualquer mudança
        (opcional: ROLLBACK)

        SELECT 'Erro ao criar utilizador.' AS
        MensagemErro;

    END;

    -- 1. Verifica se o utilizador já existe

    IF NOT EXISTS (SELECT 1 FROM utilizador WHERE Email =
    p_email) THEN

        -- 2. Insere na tabela

        INSERT INTO utilizador (Email, Nome, Telemovel,
        Tipo, Grupo)

        VALUES (p_email, p_nome, p_telemovel, p_tipo,
        p_grupo);

        -- 3. Cria o utilizador MySQL
```



```
SET @sql_create = CONCAT(
    'CREATE USER ''', p_email, '''@''localhost''
    IDENTIFIED BY ''', p_password, ''';'
);

PREPARE stmt1 FROM @sql_create;

EXECUTE stmt1;

DEALLOCATE PREPARE stmt1;

-- 4. Concede a role (pré-criada) ao utilizador

SET @sql_grant = CONCAT('GRANT ', p_tipo, ' TO ''',
p_email, '''@''localhost'';');

PREPARE stmt2 FROM @sql_grant;

EXECUTE stmt2;

DEALLOCATE PREPARE stmt2;

-- 4.1 Define como role padrão

SET @sql_default = CONCAT('SET DEFAULT ROLE ',
p_tipo, ' TO ''', p_email, '''@''localhost'';');

PREPARE stmt3 FROM @sql_default;

EXECUTE stmt3;

DEALLOCATE PREPARE stmt3;

-- 5. Mensagem de sucesso

SELECT 'Utilizador criado com sucesso!' AS
Mensagem;

ELSE

SELECT 'Utilizador já existe.' AS Mensagem;
```



```
END IF;

END$$

DELIMITER ;

2. Nome SP: AlterarUtilizador

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`alterar_utilizador`(IN `p_emailAntigo` VARCHAR(50), IN
`p_emailNovo` VARCHAR(50), IN `p_NomeNovo` VARCHAR(100),
IN `p_TelemoveiNovo` VARCHAR(12), IN `p_GrupoNovo` INT,
IN `p_SenhaNovaOpcional` VARCHAR(100))
BEGIN

    DECLARE v_email_antigo VARCHAR(50);

    DECLARE v_IDUtilizador INT;

    DECLARE sql_query VARCHAR(1000);

    SET v_email_antigo = p_emailAntigo;

    SELECT IDUtilizador INTO v_IDUtilizador
    FROM utilizador
    WHERE Email = v_email_antigo;

    IF p_emailNovo IS NOT NULL AND p_emailNovo != '' AND
    TRIM(p_emailNovo) != TRIM(v_email_antigo) THEN

        -- Atualiza dados e email

        UPDATE utilizador
```



```
SET Telemovel = p_TelemovelNovo,
    Grupo = p_GrupoNovo,
    Nome = p_NomeNovo,
    Email = p_emailNovo
WHERE IDUtilizador = v_IDUtilizador;

-- Renomeia o utilizador do sistema

SET sql_query = CONCAT('RENAME USER ''',
v_email_antigo, ''''@''localhost'' TO ''', p_emailNovo,
''''@''localhost'';');

PREPARE stmt FROM sql_query;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;

-- Atualiza email antigo para o novo

SET v_email_antigo = p_emailNovo;

ELSE

-- Só atualiza os restantes dados (sem mexer no
email)

UPDATE utilizador

SET Telemovel = p_TelemovelNovo,
    Grupo = p_GrupoNovo,
    Nome = p_NomeNovo

WHERE IDUtilizador = v_IDUtilizador;

END IF;

-- Se a senha foi passada, altera também
```



```
IF p_SenhaNovaOpcional IS NOT NULL AND
p_SenhaNovaOpcional != '' THEN

    SET sql_query = CONCAT('ALTER USER ',
v_email_antigo, '@'localhost' IDENTIFIED BY ',
p_SenhaNovaOpcional, ');

    PREPARE stmt FROM sql_query;

    EXECUTE stmt;

    DEALLOCATE PREPARE stmt;

END IF;

END$$

DELIMITER ;

3.Nome do SP: Remover_utilizador

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`remover_utilizador`(IN `p_email` VARCHAR(50))
BEGIN

    DECLARE v_IDUtilizador INT(100);

    -- 1. Verifica se o utilizador existe e obtém o email
    SELECT IDUtilizador INTO v_IDUtilizador
    FROM utilizador
    WHERE Email = p_email;

    -- 2. Se encontrou o email, continua
```



```
IF p_email IS NOT NULL THEN
    -- 3. Remove da tabela pisisd.utilizador
    DELETE FROM utilizador WHERE IDUtilizador =
v_IDUtilizador;

    -- 4. Remove o utilizador do MySQL
    SET @sql_drop = CONCAT(
        'DROP USER IF EXISTS ''', p_email,
        ''@''localhost''
    );
    PREPARE stmt1 FROM @sql_drop;
    EXECUTE stmt1;
    DEALLOCATE PREPARE stmt1;
END IF;
END$$
DELIMITER ;

4.Nome do SP: criar_jogo

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`criar_jogo`(IN `p_nick_jogador` VARCHAR(50))
BEGIN
    DECLARE v_id_utilizador INT;
    DECLARE v_email VARCHAR(50);
```



```
SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

-- Tenta obter o ID do utilizador
SELECT IDUtilizador INTO v_id_utilizador
FROM utilizador
WHERE Email = v_email;

-- Se não encontrou nenhum ID, dá erro (mas tratamos
o erro com CONTINUE HANDLER)
IF v_id_utilizador IS NULL THEN
    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'Utilizador não encontrado com
o email do utilizador atual.';
ELSE
    -- Inserir o novo jogo
    INSERT INTO jogo (
        NickJogador,
        Estado,
        IDUtilizador
    )
    VALUES (
        p_nick_jogador,
        'nao_inicializado',
        v_id_utilizador
    );

    -- Mensagem de sucesso
```





```
        SELECT 'Jogo criado com sucesso!' AS Mensagem;

    END IF;

END$$

DELIMITER ;

5.Nome do SP: alterar_jogo

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`alterar_jogo`(IN `p_idjogo` INT, IN `p_NickJogador`
VARCHAR(50))
BEGIN

    DECLARE v_id_utilizador INT;

    DECLARE v_email VARCHAR(50);

    SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

    SELECT IDUtilizador INTO v_id_utilizador
    FROM utilizador
    WHERE Email = v_email;

    IF EXISTS (

        SELECT 1 FROM jogo

        WHERE IDJogo = p_idjogo

        AND Estado != 'jogando'

        AND IDUtilizador = v_id_utilizador

    ) THEN
```



```
        UPDATE jogo
        SET NickJogador = p_NickJogador
        WHERE IDJogo = p_idjogo;

    ELSE

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Você
não tem permissão para alterar este jogo.';

    END IF;

END$$

DELIMITER ;

6.Nome do SP: iniciar_jogo

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`iniciar_jogo`(IN `p_idjogo` INT)
BEGIN

    DECLARE v_id_utilizador INT;

    DECLARE v_email VARCHAR(50);

    DECLARE timenow timestamp;

    set timenow = CURRENT_TIMESTAMP;

    SET v_email = SUBSTRING_INDEX(USER(), '@', 2);

    SELECT IDUtilizador INTO v_id_utilizador
    FROM utilizador
    WHERE Email = v_email;
```



```
IF EXISTS (
    SELECT * FROM jogo
    WHERE IDJogo = p_idjogo
    AND Estado = 'nao_inicializado'
    AND IDUtilizador = v_id_utilizador
) THEN
    UPDATE jogo
    SET Estado = 'jogando', DataHoraInicio = timenow
    WHERE IDJogo = p_idjogo;
ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Você
não tem permissão para alterar este jogo.';
END IF;
END$$
DELIMITER ;

7.Nome do SP: getIdJogo_IdUtilizador

8.Nome do SP: get_jogos
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_jogos`()
BEGIN
    DECLARE email_atual VARCHAR(100);

    -- Extrai apenas o nome de utilizador da função
    USER() (antes do @)

    SET email_atual = SUBSTRING_INDEX(USER(), '@', 2);
```



```
-- Retorna os jogos do utilizador autenticado
SELECT j.IDJogo, j.NickJogador
FROM jogo j
JOIN utilizador u ON j.IDUtilizador = u.IDUtilizador
WHERE u.Email = email_atual
      AND j.Estado = 'nao_inicializado';
END$$
DELIMITER ;

9.Nome do SP: get_marsami_room
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_marsami_room`()
BEGIN
    SELECT *
    FROM sala
    WHERE IDJogo_Sala = (SELECT MAX(IDJogo_Sala) FROM
sala)
    ORDER BY IDSala;
END$$
DELIMITER ;

10.Nome do SP: get_mensagens
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_mensagens`()
BEGIN
```



```
SELECT Msg, Leitura, TipoAlerta, Hora, HoraEscrita
FROM mensagens
WHERE Hora >= NOW() - INTERVAL 60 MINUTE
ORDER BY Hora DESC;
END$$
DELIMITER ;

11.Nome do SP: get_sensores

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_sensores`(IN `p_idjogo` INT(50))
BEGIN
    SELECT s.Hour, s.Sound, j.normal_noise
    FROM sound s
    JOIN jogo j ON j.IDJogo = p_idJogo
    WHERE s.IDJogo = p_idJogo AND s.Hour >= NOW() -
INTERVAL 10 SECOND
    ORDER BY s.Hour DESC;
END$$
DELIMITER ;
```