

IPBeja

INSTITUTO POLITÉCNICO
DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura - Engenharia Informática

Relatório do Projeto de Base de Dados 2

Relatório do Projeto de Base de Dados 2

Tiago Alexandre Baptista Pacheco

Beja, 18 de Janeiro de 2023

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura - Engenharia Informática

Relatório do Projeto de Base de Dados 2

Relatório do Projeto de Base de Dados 2

Tiago Alexandre Baptista Pacheco

Orientado por :

Gonçalo Fontes
, IPBeja

Elsa Rodrigues, IPBeja

Relatório do Projeto de Base de Dados 2, no âmbito da criação de um sistema de aluguer
de habitações

Resumo

Relatório do Projeto de Base de Dados 2

Este relatório têm como objetivo demonstrar o desenvolvimento do Projeto de Base de Dados, que seria a criação de uma base de dados de raiz , sendo que o tema era ”Booking” e ”Airbnb”. Será exibido todo o desenvolvimento do trabalho, desde do processo de logicamente perceber e modular uma base de dados e depois como foi a sua implementação no *SQL-Server*, trabalhado em aula. O desenvolvimento do relatório terá a seguinte estrutura:

1. Criação da BD e dos seus elementos;
2. Método de preenchimento da BD;
3. *Stored Procedures e Triggers*;
4. Segurança;
5. Cópias de segurança (*Backups*);
6. Desempenho;
7. Manutenção e automatização do servidor;

Abstract

Relatório do Projeto de Base de Dados 2

This report aims to demonstrate the development of the Baseline Project Data, which would be the creation of a database from scratch, with the theme being "Booking" and "Airbnb". The entire development of the work will be displayed, from the logic process to understand and modulate a database and then how was its implementation in the SQL Server, worked in class. The development of the report will have the following structure:

1. Creation of the BD and its elements;
2. DB filling method;
3. *Stored Procedures and Triggers;*
4. Security;;
5. Security copies (Backups);
6. Performance;
7. Server maintenance and automation;

Keywords: *BD, Server ,SQL Server, database, Airbnb.*

Índice

Resumo	i
Abstract	iii
Índice	v
Índice de Figuras	vii
1 Introdução	1
2 Modelação da Base de Dados Projetada	3
2.1 Adaptação do Modelo Projetado	5
3 Criação da Base de Dados	7
3.1 Desempenho - Planeamento do Sistema de Discos	8
3.1.1 Criação dos Discos na <i>Virtual Machine (VirtualBox)</i>	9
3.1.2 Atribuição dos Discos na <i>VM</i>	10
3.1.3 Criação de Espaços de Armazenamento para a Criação de <i>Raids</i>	11
3.1.4 Espaços de Armazenamento Implementados	12
3.1.5 Espaços Implementados	12
3.1.6 Finalização da Criação dos <i>Raids</i>	13
3.2 Criação da Base de Dados Segmentada para um Maior Desempenho	14
3.2.1 <i>FileGroups</i>	15
3.2.2 Schemas	16
3.3 Criação da Base de Dados e Seus <i>FileGroups Scripting</i>	17
3.3.1 Verificação dos Ficheiros Criados para as Suas Pastas Correspondentes e <i>Raids</i>	18
3.3.2 Criação das Tabelas tendo em conta <i>Schemas</i>	19
3.3.3 Criação de Indexes nas tabelas Criadas	20
4 Método de preenchimento da <i>BD</i>	21
4.1 Inserção de Dados	22

ÍNDICE

5 Segurança Implementada na Base de Dados	23
5.1 Cópias de segurança (<i>Backups</i>)	24
5.1.1 Backups Cifrados	25
5.1.2 <i>Schedule de Backups</i>	26
5.2 Utilizadores e Permissões	28
5.2.1 Utilizadores Windows e Suas Permissões - Base de Dados	29
5.3 Criação dos Utilizadores SQL (<i>LOGINS - Security SQL Server</i>)	33
6 Stored Procedures e Triggers	37
6.1 O que é um <i>Stored Procedure</i> ?	37
6.2 <i>Triggers</i>	37
6.3 Criação de <i>Stored Procedures</i>	38
6.4 Criação de <i>Triggers</i>	39
7 Futuras Implementações Possíveis	47
8 Manutenção e automatização do servidor	49
9 Conclusão	51
Bibliografia	53
Anexos	55
I Script total	57
II Schedule Backups	71
III Utilizadores e <i>Logins</i>	75

Índice de Figuras

2.1	<i>Modelo de Base de Dados n:1</i>	4
2.2	<i>Modelo Base de Dados n:2</i>	5
3.1	<i>SQL Management Studio</i>	7
3.2	<i>Raids</i>	8
3.3	<i>Tipo de disco</i>	9
3.4	<i>Novos volumes</i>	10
3.5	<i>Espaços de Armazenamento</i>	11
3.6	<i>Raid 50</i>	12
3.7	<i>Raid 1</i>	12
3.8	<i>Raid Finalização</i>	13
3.9	<i>Modelo Schemas</i>	16
3.10	<i>FileGroups</i>	17
3.11	<i>Pastas FileGroups</i>	18
3.12	<i>ficheiros</i>	18
3.13	<i>Criação schemas</i>	19
3.14	<i>Criação tabelas</i>	19
3.15	<i>Indices</i>	20
4.1	<i>Mokarro</i>	21
4.2	<i>Dados para Inserção de Dados</i>	22
4.3	<i>Query Dados</i>	22
5.1	<i>Segurança</i>	23
5.2	<i>backups</i>	24
5.3	<i>Querys Backups normais e cifrados + certificado</i>	25
5.4	<i>Schedule backups</i>	27
5.5	<i>Schedule backups2</i>	27
5.6	<i>Users</i>	28
5.7	<i>Logins Windows</i>	30
5.8	<i>User Windows</i>	31
5.9	<i>Permissões user windows</i>	32

ÍNDICE DE FIGURAS

5.10 <i>Utilizador sql - cliente</i>	33
5.11 <i>criação user cliente</i>	34
5.12 <i>Permissões cliente</i>	35
5.13 <i>Pasta Users</i>	36
6.1 <i>Stored Procedures: 1</i>	38
6.2 <i>Stored Procedures: 2</i>	38
6.3 <i>Stored Procedures: 3</i>	38
6.4 <i>Triggers</i>	39
6.5 <i>Trigger1</i>	41
6.6 <i>Trigger2</i>	42
6.7 <i>Trigger3</i>	43
6.8 <i>Trigger4</i>	44
6.9 <i>Trigger5</i>	45
6.10 <i>Pastas Triggers</i>	46
6.11 <i>Pastas Triggers2</i>	46
7.1 <i>Futuras Implementações</i>	47
8.1 <i>Manutenção Base de Dados</i>	49
I.1 <i>Anexo 1 - 1</i>	57
I.2 <i>Anexo 1 - 2</i>	58
I.3 <i>Anexo 1 - 3</i>	59
I.4 <i>Anexo 1 - 4</i>	60
I.5 <i>Anexo 1 - 5</i>	61
I.6 <i>Anexo 1 - 6</i>	62
I.7 <i>Anexo 1 - 7</i>	63
I.8 <i>Anexo 1 - 8</i>	64
I.9 <i>Anexo 1 - 9</i>	65
I.10 <i>Anexo 1 - 10</i>	66
I.11 <i>Anexo 1 - 11</i>	67
I.12 <i>Anexo 1 - 12</i>	68
I.13 <i>Anexo 1 - 13</i>	68
I.14 <i>Anexo 1 - 14</i>	69
I.15 <i>Anexo 1 - 15</i>	69
I.16 <i>Anexo 1 - 16</i>	70
I.17 <i>Anexo 1 - 17</i>	70
II.1 <i>Anexo 2 - 1</i>	71
II.2 <i>Anexo 2 - 2</i>	72

Índice de Figuras

II.3 <i>Anexo 2 - 3</i>	72
II.4 <i>Anexo 2 - 4</i>	73
II.5 <i>Anexo 2 - 5</i>	73
II.6 <i>Anexo 2 - 6</i>	74
III.1 <i>Anexo 3 - 1</i>	75
III.2 <i>Anexo 3 - 2</i>	76

Capítulo 1

Introdução

Quando falamos de Base de Dados e como tudo funciona temos sempre ver que é necessário haver uma boa modulação e planeamento da mesma, tendo em conta o tema em que se tá a criar a base de dados. Este trabalho demonstra todo esse processo, e como tudo foi elaborado.

De seguida vamos começar por o mais importante que seria a modulação da base de dados e todo esse processo, onde será tudo explicado com clareza e objetividade.

Capítulo 2

Modelação da Base de Dados Projetada

Antes de qualquer criação de base de dados, foi planeado e projetada uma base de dados mais próxima do ideal tendo com conta tudo o que era pretendido pelo enunciado facultado pelos docentes. Sabendo que era uma base de dados de "Booking"/ "Airbnb", isto é , uma base de dados em que tinha como propósito reservas de alojamento para férias , em todas as épocas do ano, sendo a mesma uma *database* projetada para funcionar internacionalmente. Com isto teríamos de aceitar reservas e clientes provenientes de outras localidades e países que não fosse unicamente Portugal. O primeiro planeamento da base de dados tinha um total de 14 tabelas, todas elas normalizadas da melhor forma para estar de encontro do pretendido.

Todas as tabelas foram sempre "divididas", em sub-tabelas de modo a que a informação esteja mais repartida, e não haja tabelas muito extensas de atributos que nem sempre ajuda em termos de desempenho e usabilidade.

Neste modelo o objetivo era responder a tudo o que estava no enunciado, tendo uma característica a mais pensada , que seria cada cliente poder colocar mais que 1 apartamento, dentro da mesma Reserva, o que assim melhorava a questão de não haver multiplicas reservas por parte da mesma pessoa, podendo afetar o desempenho a longo prazo.

2. MODELAÇÃO DA BASE DE DADOS PROJETADA

A imagem a abaixo mostra a ideia projetada, sendo que a mesma foi avaliada pelos os docentes, e designada como uma boa ideia e funcional, para o que era pretendido.

Segue abaixo uma imagem referente à modelação planeado e entregue aos Docentes.

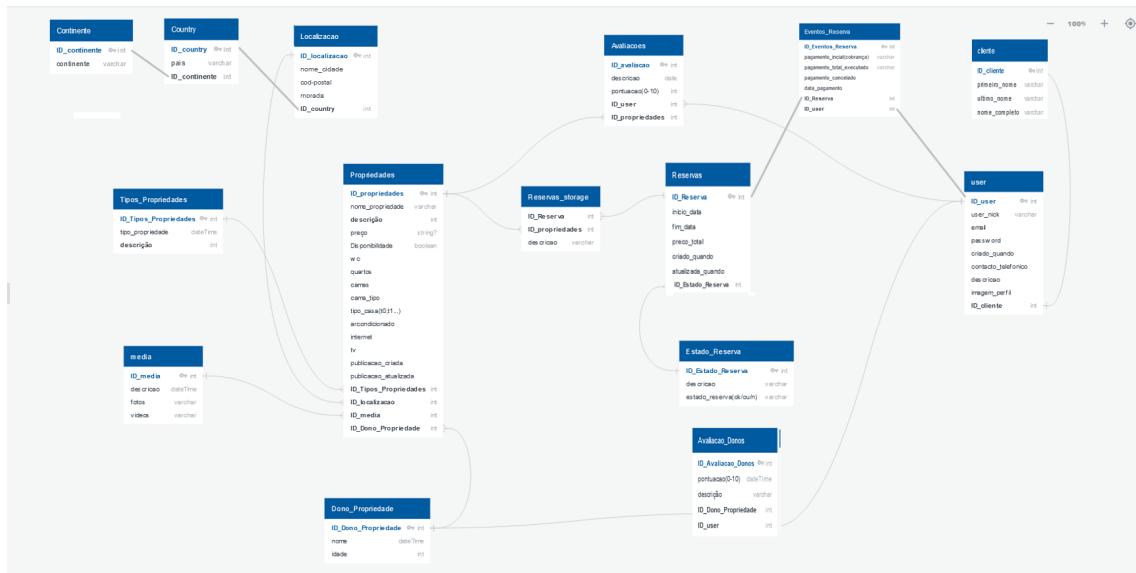


Figura 2.1: Modelo de Base de Dados n:1

2.1 Adaptação do Modelo Projeto

Com o processo de iniciação do trabalho e a realização de alguns testes, estes que são sempre bem-vindos antes de uma implementação completa, foi detetado alguma dificuldade na criação de *triggers*, que serão referenciado mais à frente no relatório, o que fez com que tivesse de haver uma alteração dos planos, de modo a facilitar me o trabalho, devido a não estar a perceber como poderia implementar o que tinha pensado.

Com isso, foi então retirada a tabela , ”Reservas_Storage” onde eram reservados os dados múltiplos de cada reserva, e também acrescentada + 2 tabelas ao plano inicial, sendo elas: a tabela ”Taxas” e tabela ”Métodos de Pagamento”, de modo a melhorar a modulação e facilitação da criação de *procedures* e *triggers* para a base de dados. Com isto, foi então desenhada uma nova modulação, apresentada em baixo. O diagrama abaixo é criada já dentro do *sql*, numa forma de testes, como referido anteriormente.

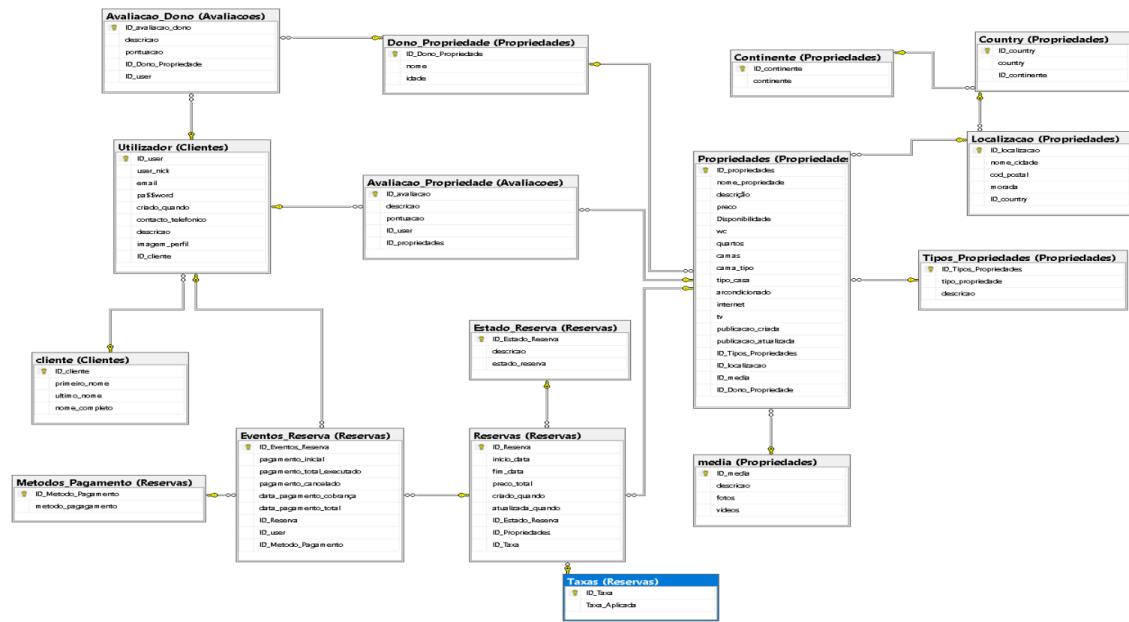


Figura 2.2: Modelo Base de Dados n:2

De Seguida será demonstrado a implementação da base de dados idealizada, no *software SQL Server / SQL Server Management Studio*.

Capítulo 3

Criação da Base de Dados

A criação da base de dados do trabalho foi realizada utilizando o *SQL Server - SQL Server Management Studio*, como já tinha sido referenciado, de modo a complementar tudo o que foi aprendido no conjunto das duas cadeiras de Base de Dados que o curso contém, sendo que para fazer este trabalho tinha de haver bom conhecimento dos conteúdos "antigos" apreendidos , pois no fim tudo se conjugava de uma forma direita e indireta. Inicialmente e com a modulação realizada, e eu estando confiante com a mesma, foi então pensado começado a pensar em alguns passos que pudessem ser introduzidos de uma forma positiva para que o seu desenvolvimento e desempenho , a curto e longo prazo fosse bastante eficiente. O primeiro deles foi pensar num sistema de discos / Sistema de *raids*, para implementar , já que é algo bastante importante, estando também referenciado no enunciado do trabalho que teríamos de ter no mínimo 6 discos para puder separar toda a informação/dados. De seguida serão apresentadas as ideias implementadas, começando primeiramente pelo que foi referenciado neste início de capítulo que seriam os Sistemas de Discos em Windows , com fundamento virado para *SQL* e segurança da informação.



Figura 3.1: *SQL Management Studio*

3. CRIAÇÃO DA BASE DE DADOS

3.1 Desempenho - Planeamento do Sistema de Discos

Existindo um mínimo de discos a usar, foi logo verificado então quais os melhores *raids* disponíveis e que normalmente se usam para sistemas de base de dados. O que foi verificado é que os mais usados seriam o *raid 5+0* e também o *raid 1 + 0*. Dentro de toda a pesquisa decidi usar o *raid 50* e crie-lo com 9 discos, sendo que assim existiria bastante redundância dos dados e também um bom desempenho, de modo a prevenir os dados em caso de catástrofe.

A ideia geral passava por utilizar o *raid 5+0* onde concatenar esses 9 discos, num só, onde iriam haver 8 pastas, correspondentes aos *filegroups*, índices e *backups* planeados. Tendo em conta que o *raid 50* funciona com *stripping com paridade (raid 5)* e *Stripping (raid 0)*, toda a informação estará bem distribuída contribuindo para um excelente desempenho. Por fim decidi também adicionar + 2 discos, onde estes servirão unicamente para *backups* da base de dados, onde a ideia passava por ter cópias da base de dados em 2 *raids* distintos. O meu pensamento passava por colocar estes 2 discos num *raid 1*, obtendo assim mais redundância. Estes dados estarem numa localização diferente permite que em caso de desastre dos 9 discos (*raid 50*), a base de dados estaria preservada, pois este *raid* na minha idealização não estaria junto dos demais discos. Assim a probabilidade de tudo se estragar ao mesmo tempo, é quase nula.

De seguida serão demonstrados os passos para a criação dos *raids* no Sistema Operativo.

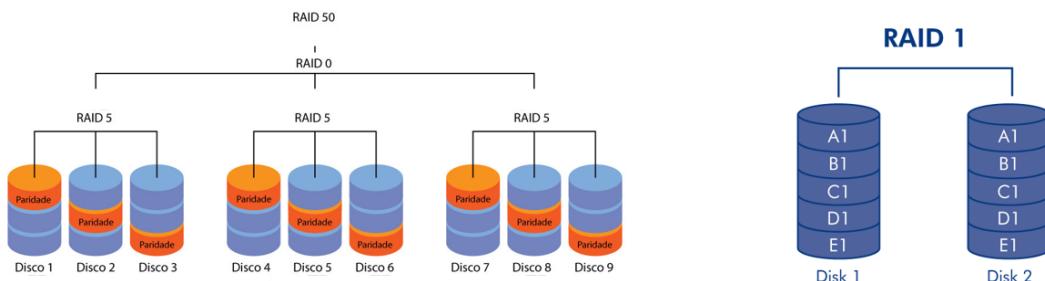


Figura 3.2: *Raids*

3.1.1 Criação dos Discos na *Virtual Machine (VirtualBox)*

O primeiro passo no desenvolvimento dos *Raids*, foi criar os discos virtuais necessários no *virtualBox*. Foram criados no total 11 discos virtuais de modo a corresponder ao que estava idealizado. Todos os discos foram criados com o tipo *VDI (VirtualBox Disk Image)*, sendo que cada um deles tinha o tamanho de 50GB, no seu total. Este valor é exagerado para o que era pretendido, mas achei que ficaria bem no sentido de pensar isto para produção (mundo do trabalho).

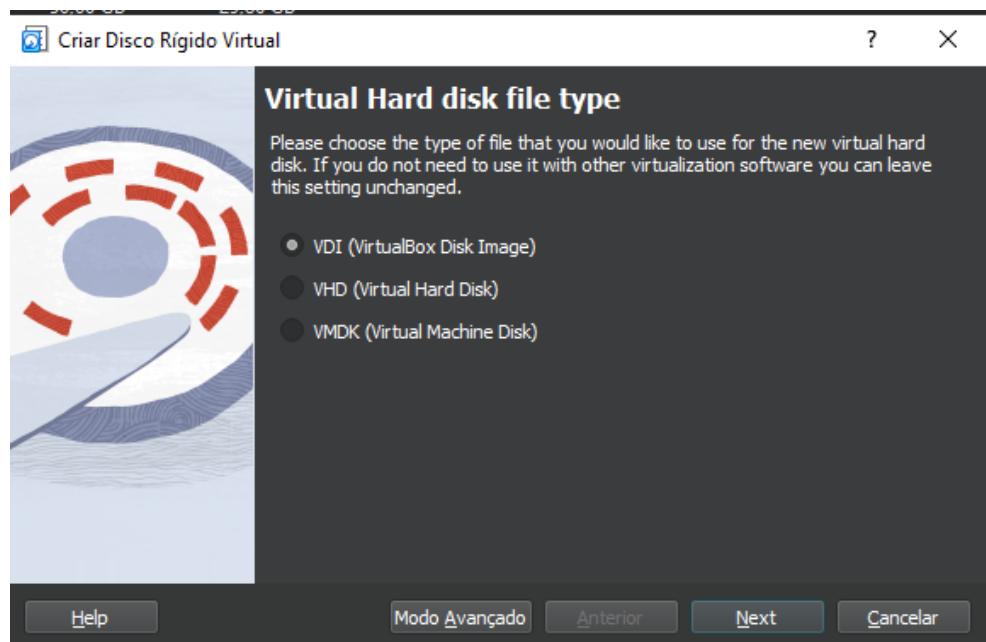


Figura 3.3: *Tipo de disco*

3. CRIAÇÃO DA BASE DE DADOS

3.1.2 Atribuição dos Discos na VM

Com os discos criados, foi necessário abrir a gestão de discos da *Virtual Machine*, e criar novos volumes , onde iria ficar com 11 discos , para puder fazer o que era necessária. Obviamente de seguida foram todos formatados numa forma de segurança e posteriormente foram introduzidos em espaços de armazenamento com o próximo sub-capítulo irá tratar.

Volume	Esquema	Tipo	Sistema de...	Estado	Capacidade	Espaco ...	% Livre
— (C)	Simples	Básico	NTFS	Bom Estado...	50.00 GB	15.73 GB	31 %
— Disco 0	Básico	50.00 GB	Online	(C)	50.00 GB NTFS	Bom Estado de Funcionamento (Sistema, Arranque, Ficheiro de paginação, Ativo, Imagem de Eros, Partição primária)	
— Disco 1	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 2	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 3	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 4	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 5	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 6	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
— Disco 7	Básico	49.88 GB	Online		49.87 GB	Não atribuído	
■ Não atribuído							

Figura 3.4: Novos volumes

3.1. Desempenho - Planeamento do Sistema de Discos

3.1.3 Criação de Espaços de Armazenamento para a Criação de *Raids*

Para a criação de *raids* no sistema operativo *Windows*, idealmente usa-se espaços de armazenamento , onde se torna muito fácil e rápido juntar vários discos, e facilita na escolha de como os dados vão ser separados etc.

- * 3 Agrupamentos de Armazenamento (cada 1 junta 3 discos (*raid 5*));
- * 1 Agrupamento de Armazenamento (2 discos (*raid 1*)).



Figura 3.5: *Espaços de Armazenamento*

3. CRIAÇÃO DA BASE DE DADOS

3.1.4 Espaços de Armazenamento Implementados

Espaço Armazenamento referente ao Raid 50

- ★ 3 Discos de 50GB cada;
- ★ Tipo de *Raid*: *Stripping* com Paridade.

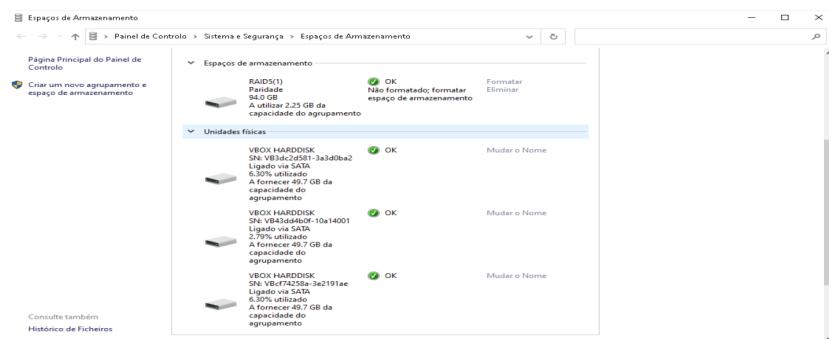


Figura 3.6: Raid 50

3.1.5 Espaços Implementados

Espaço Armazenamento referente ao *Raid 1*

- ★ 2 Discos de 50GB cada;
- ★ Tipo de *Raid*: espelhamento (Clone).

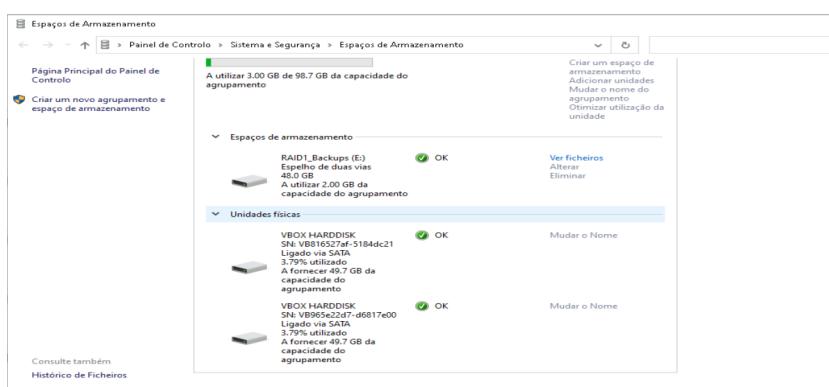


Figura 3.7: Raid 1

3.1.6 Finalização da Criação dos *Raids*

Com os espaços criados, foi então feito o ultimo passo na criação dos *raids* em definitivo, sendo este passo um pouco estranho, mas bastante funcional. Para o *raid 50*, no gestor de discos, aparecem então unicamente 3 discos. O que foi feito então foi eliminar esses volumes criados automaticamente ao fazer os espaços, e depois sim criar um novo volume alocando os 3 auto-criados em modo *raid 0*, utilizando o *stripping*, unicamente.

Com isto foi então criado perfeitamente o *raid 50*, como se pode ver na imagem em baixo referenciada. Para o *raid 1* foi elaborado o mesmo, sendo que assim tudo ficou a funcionar perfeitamente e sim agora poderia começar a criar a base de dados em si, pois já tinha a divisão de discos bem elaborada.

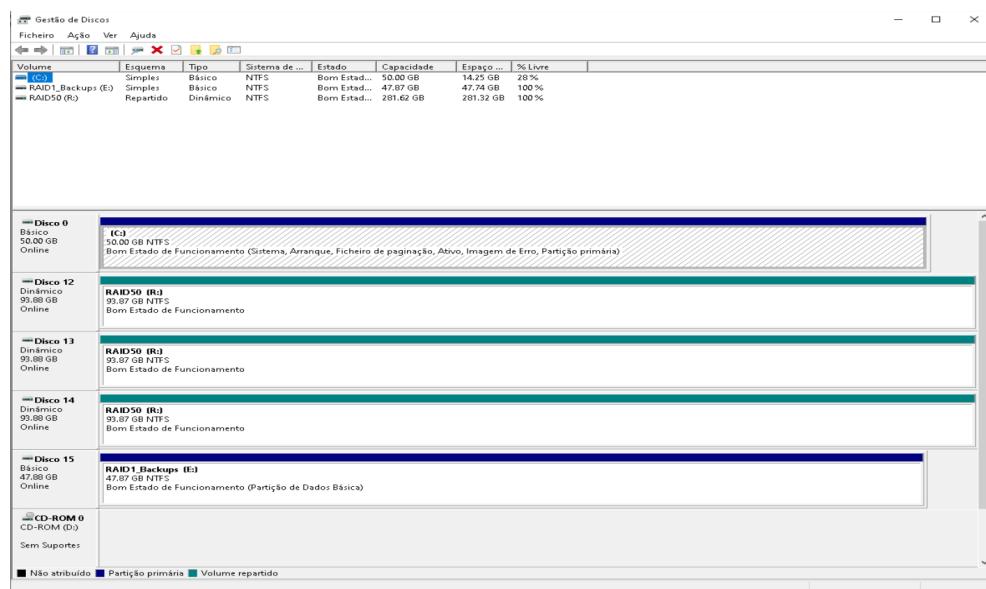


Figura 3.8: Raid Finalização

3.2 Criação da Base de Dados Segmentada para um Maior Desempenho

O plano para a base de dados, a nível de desempenho passava por fazer uma boa gestão dos discos, como foi mostrado anteriormente, no capítulo anterior, e de seguida poderia haver outras situações que se podiam elaborar, sendo uma delas, os ditos "*FileGroups*" que são conjuntos de ficheiros que são usados para simplificar o posicionamento de dados e a sua distribuição.

FileGroups no geral tem sempre pelo menos um *Master Data File*, extensão *.MDF*) e outro reservado para o *log* de transações (*Log Data File*, extensão *.LDF*). Por fim também existem os ficheiros secundários, com a extensão *.ndf*. Para este trabalho cada *file group* criado teria 2 ficheiros secundários (*.ndf*), sendo que o *filegroup default* denominado de *Primary*, eu continuei a usá-lo , sendo que esse mesmo teria o ficheiro *.mdf* e mais um ficheiro *.ndf* de modo a todo o conjunto da base de dados possa ser gravado lá. Por fim existiram também 2 ficheiros *.ldf*, sendo estes focados em *logs* da base de dados.

A outra seria os *Schemas*, sendo eles coleções de objetos dentro de uma determinada base de dados, que organizam vários aspectos e são importantes para segmentação da segurança, onde eu os usei para separar/distribuir as tabelas em departamentos de modo a ser mais fácil aceder a dados, e também facilitar nas permissões atribuídas para utilizadores.

3.2.1 *FileGroups*

Foram criados 4 "*FileGroups*", onde a ideia passava por cada *filegroup* estar diretamente ligado com as tabelas pretendidas. Os *file groups* criados foram os seguintes.

- ★ Reservas - Tabelas Correspondentes a informações relacionadas com Reservas;
- ★ Propriedades - Tabelas Correspondentes a informações relacionadas com Propriedades;
- ★ Avaliações - Tabelas Correspondentes a informações relacionadas com Avaliações;
- ★ Clientes - Tabelas Correspondentes a informações relacionadas com Cliente;
- ★ Índices - Tabelas Correspondentes a informações relacionadas com index (indexação de dados).

Cada ficheiro dos *filegroup* tinha um tamanho específico e foi definido também a expansão do mesmo. Segue-se as configurações delineadas para os *filegroups*.

- ★ *Size* : 20 MB / 10 MB, tendo em conta o nível de dados que poderia ter;
- ★ *FileGrowth*: 10 a 15% tendo em conta o nível de dados novamente.

3. CRIAÇÃO DA BASE DE DADOS

3.2.2 Schemas

Estes *Schemas* tinha como objetivo separar toda a Informação , e melhor uma melhor organização de todas as tabelas, sendo que iriam estar a trabalhar diretamente com os *Filegroups*, na Organização de tudo, o que depois irá facilitar toda organização e divisão de permissões para cada utilizador , sendo este o primeiro método que influenciará toda a segurança da base dados como será referenciado mais a frente.

Os *schemas* Criados Foram os Seguintes:

A imagem abaixo exemplifica e mostra claramente todas as divisões de *schemas* criados, contendo uma legenda de cores, correspondendo a cada "Departamento".

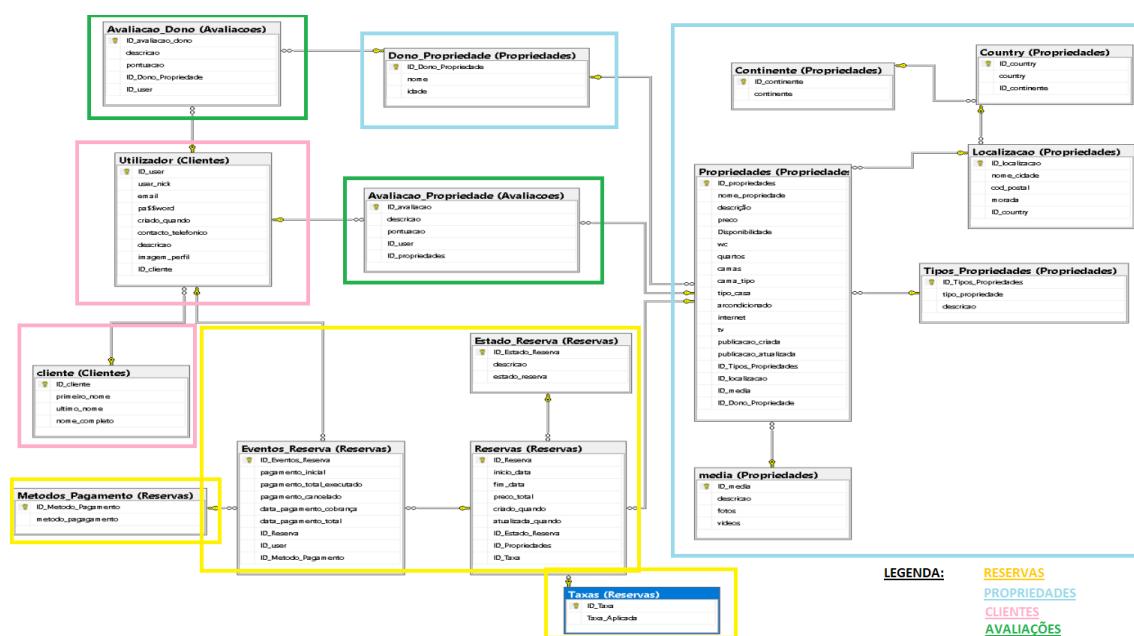


Figura 3.9: *Modelo Schemas*

3.3 Criação da Base de Dados e Seus *FileGroups Scripting*

Contendo a logística e tudo pronto , para iniciar o processo de *scripting* e criação da *query* necessária, foi então criada a base de dados regindo se por tudo aquilo que foi falado anteriormente. Criou se a base de dados dentro do ”*FileGroup*” primário e de seguida, criado os 4 *filegroups*.

A imagem em baixo exemplifica um pouco do ”código” usado, sendo que todo o *scripting* irá estar nos anexos, e será entregue posteriormente aos docentes.

```

CREATE DATABASE ProjetoBD2
ON PRIMARY
(
    NAME= BD2_Data1,
    FILENAME = 'R:\Primary\BD2_Data1.mdf',
    SIZE = 20MB,
    FILEGROWTH = 10%
),
(
    NAME = BD2_Data2,
    FILENAME = 'R:\Primary\BD2_Data2.ndf',
    SIZE = 10MB,
    FILEGROWTH = 10%
),
FILEGROUP Reservas
(
    NAME = BD2_Data3,
    FILENAME = 'R:\Reservas\BD2_Data3.ndf',
    SIZE = 20MB,
    FILEGROWTH = 10%
),
(
    NAME = BD2_Data4,
    FILENAME = 'R:\Reservas\BD2_Data4.ndf',
    SIZE = 10MB,
    FILEGROWTH = 10%
)

```

Figura 3.10: *FileGroups*

3. CRIAÇÃO DA BASE DE DADOS

3.3.1 Verificação dos Ficheiros Criados para as Suas Pastas Correspondentes e *Raids*

Ao executar o *script* para criar a base de dados, ainda sem conteúdos, foi feita uma verificação se tudo tinha corrido com o pretendido, onde bastou me ir verificar cada pasta no *raid 50*, se continha os ficheiros que foram planeados, e com os tamanhos que se desejava.

Verificação das Pastas:

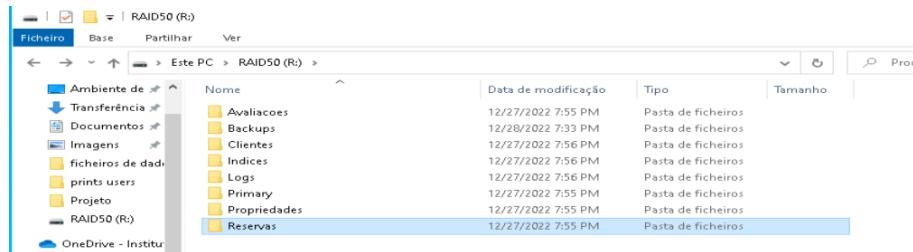


Figura 3.11: *Pastas FileGroups*

Verificação dos ficheiros (.mdf,.ndf,.ldf)

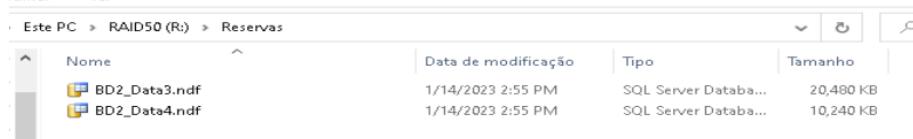


Figura 3.12: *ficheiros*

3.3.2 Criação das Tabelas tendo em conta *Schemas*

Partindo para a criação das tabelas e de todos os seus atributos, antes de mais teriam de ser criados os *schemas* definidos para a base de dados. No *script* bastou criar 4 linhas de código onde cada uma criava um *schema*. Com isso feito foi então criada as tabelas já associadas a cada *schema*, exemplo: ("Propriedades.Propriedades").

```
CREATE SCHEMA Propriedades
CREATE SCHEMA Reservas
CREATE SCHEMA Clientes
CREATE SCHEMA Avaliacoes
```

Figura 3.13: *Criação schemas*

Como se pode verificar cada tabela foi criada dentro de um dos *filegroups*, anteriormente criados de modo a separar os dados. No fim de cada *script* de "CREATE TABLE", era usado o "*ON*", isto é, a tabela seria criada numa localização escolhida pelo administrador.

```
use ProjetoBD2
GO

CREATE TABLE Reservas.Taxas (
    ID_Taxa smallint primary key,
    Taxa_APLICADA Decimal
)ON Reservas;

CREATE TABLE Reservas.Metodos_Pagamento (
    ID_Metodo_Pagamento smallint primary key,
    metodo_pagamento varchar(50)
)ON Reservas;

CREATE TABLE Propriedades.Continente (
    ID_continente smallint primary key,
    continente varchar(60)
)ON Propriedades;

CREATE TABLE Propriedades.Country (
    ID_country smallint primary key,
    country varchar(60),
    ID_continente SMALLINT NOT NULL,
    FOREIGN KEY (ID_continente)
    REFERENCES Propriedades.Continente (ID_continente)
    on update cascade
    on delete cascade
)ON Propriedades;
```

Figura 3.14: *Criação tabelas*

3. CRIAÇÃO DA BASE DE DADOS

3.3.3 Criação de Indexes nas tabelas Criadas

O uso de índices pode trazer grandes melhorias a nível de desempenho. Os registos são armazenados em páginas de dados, páginas estas que compõem o que chamamos de pilha, que por sua vez é uma coleção de páginas de dados que contém os registos de uma tabela. Cada página de dados tem seu tamanho definido em até 8 Kb, apresenta um cabeçalho, também conhecido como *header*, que contém pastas de *links* com outras páginas e identificadores (*hash*) que ocupam a nona parte do seu tamanho total (8 Kb) e o resto de sua área é destinada aos dados.

Para este trabalho decidi criar alguns indexes nas tabelas onde achei que poderiam haver mais inserções e haver mais movimentações de dados. Os Índices criados foram os seguintes:

- ★ Index não clusterizado no atributo ID_Cliente na tabela Clientes.Utilizador;
- ★ Index não clusterizado nos atributos primeiro_nome e ultimo_nome na tabela Clientes.Cliente;
- ★ Index não clusterizado no atributo ID_Propriedades na tabela Reservas.Reservas;
- ★ Index não clusterizado no atributo nome_propriedade e preco na tabela Propriedades.Propriedades.

```
create nonclustered INDEX ID_Cliente_Index ON Clientes.Utilizador (ID_cliente) on Indices
create nonclustered INDEX Nomes_Index ON Clientes.cliente (primeiro_nome,ultimo_nome) on Indices
create nonclustered INDEX Propriedades ON Reservas.Reservas (ID_Propriedades) on Indices
create nonclustered INDEX preco_nome ON Propriedades.Propriedades (nome_propriedade,preco) on Indices
```

Figura 3.15: Indices

Todos os Indexes criados para as tabelas foram guardados dentro do *filegroup* Índices, para não exercer "pressão" na tabela correspondendo, melhorando o desempenho.

Capítulo 4

Método de preenchimento da *BD*

O método utilizado para a inserção de dados foi um dos métodos opcionais que tínhamos em mãos, que neste caso passava por usar *softwares* que seriam geradores de informação para pudermos encher a nossa base de dados com os mais variados dados. Com isso poderíamos testar todas as funcionalidades com mais clareza.

O software que utilizei tem como nome ”*Mockaroo*”, que passa por ser ,como referi, um gerador de dados e uma ferramenta de simulação de *API* que ajuda na criação de um conjunto de dados *CSV*, *JSON*, *SQL* e *Excel* personalizados para testar e demonstrar o software.

Trata se de um software muito bem visto no ramo de base de dados e de fácil utilização evitando certo problemas que poderão haver na criação de um gerador de dados feito do zero.

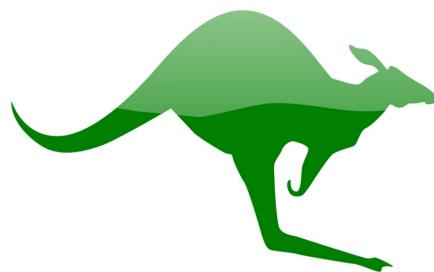


Figura 4.1: *Mokarro*

4. MÉTODO DE PREENCHIMENTO DA BD

4.1 Inserção de Dados

Para a inserção de dados, foram usados vários ficheiros *sql*, isto é, *querys* geradas pelo *mockaroo*, para colocar todos os dados. Com a inserção feita a maior preocupação seria verificar se todos as ligações entre as tabelas estavam de facto corretas, já que normalmente pode acontecer erros a introduzir dados, devido aos atributos estarem incorretos.

📄	Reservas.Eventos_Reserva(2).sql	✓	14/01/2023 14:53	Microsoft SQL Ser...	275 KB
📄	Reservas.Reservas(2).sql	✓	14/01/2023 14:42	Microsoft SQL Ser...	241 KB
📄	Reservas.Estado_Reserva.sql	✓	14/01/2023 14:00	Microsoft SQL Ser...	129 KB
📄	Propriedades.Propriedades.sql	✓	14/01/2023 13:59	Microsoft SQL Ser...	427 KB
📄	Propriedades.Localizacao.sql	✓	14/01/2023 13:36	Microsoft SQL Ser...	147 KB
📄	Propriedades.Country.sql	✓	14/01/2023 13:31	Microsoft SQL Ser...	3 KB
📄	Propriedades.Continente.sql	✓	14/01/2023 13:11	Microsoft SQL Ser...	1 KB
📄	Propriedades.media(2).sql	✓	14/01/2023 13:09	Microsoft SQL Ser...	173 KB
📄	Reservas.Metodos_Pagamento.sql	✓	14/01/2023 12:48	Microsoft SQL Ser...	1 KB
📄	Avaliacoes.Avaliacao_Dono.sql	✓	14/01/2023 12:43	Microsoft SQL Ser...	177 KB
📄	Propriedades.Dono_Propriedade(1).sql	✓	14/01/2023 12:38	Microsoft SQL Ser...	111 KB
📄	Cientes.Utilizador.sql	✓	14/01/2023 12:30	Microsoft SQL Ser...	294 KB
📄	Cientes.cliente.sql	✓	14/01/2023 12:25	Microsoft SQL Ser...	135 KB

Figura 4.2: Dados para Inserção de Dados

```
Cientes.cliente.sql...|AGO-PC\Tiago (60) ✘ X
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (1, 'Noemi', 'Commins', 'Noemi Commins');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (2, 'Elie', 'Demaine', 'Elie Demaine');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (3, 'Cassey', 'Carrol', 'Cassey Carrol');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (4, 'Flynn', 'Garrique', 'Flynn Garrique');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (5, 'Barby', 'Ellens', 'Barby Ellens');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (6, 'Gauthier', 'Birley', 'Gauthier Birley');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (7, 'Seana', 'Bolsover', 'Seana Bolsover');
INSERT INTO Clientes.cliente (ID_cliente, primeiro_nome, ultimo_nome, nome_completo) VALUES (8, 'Isidor', 'Gaymer', 'Isidor Gaymer');
```

Figura 4.3: Query Dados

Capítulo 5

Segurança Implementada na Base de Dados

A nível de implementações referentes a segurança foi abordado 2 temas, *Backups* e seus derivados, e também gestão de utilizadores com permissões repartidas de modo ao acesso à base de dados ser realizado de forma calculada e sem problemas de fuga de informação. Optei por realizar inicialmente backups de teste, *backups* não cifrados e posteriormente backups cifrados de modo a caso haja ”roubo” dos backups, seja mais difícil o interveniente ser encontrado e seja mais rapidamente resolvido o caso.

Quando à gestão de utilizadores foram criados vários utilizadores a nível de *Windows*, numa forma de divisão de cargos, assim como utilizadores *SQL*, sendo o mais importante, o utilizador de testes denominado de cliente, servindo este para testar a inserção de uma reserva.



Figura 5.1: Segurança

5.1 Cópias de segurança (*Backups*)

Backup é uma cópia de segurança dos seus dados de um dispositivo de armazenamento ou sistema (aplicativos, softwares e jogos) para outro ambiente para que eles possam ser restaurados se você perdeu as informações originais, trocou de aparelho, entre outros casos. Em concreto nas base de dados é algo bastante importante haver para não haver perda de dados de clientes, que poderá comprometer a privacidade dos mesmos.

Como foi referido anteriormente fiz alguns testes de criação de cópias de segurança muito ao exemplo do que foi trabalho em aula e demonstrado pelos docentes, de modo a testar se tudo funcionava corretamente. Decidi testar o *backup* na minha VM, e depois fazer um *restore* da mesma num computador diferente a ver se estava tudo corretamente a funcionar.



Figura 5.2: *backups*

5.1.1 Backups Cifrados

Com alguns testes feitos, foram realizadas algumas pesquisas de modo a como se poderia cifrar estes mesmos *backups*, já que assim a segurança dos dados fica muito mais salvaguardada. Com isso criei uma *query* de modo a testar os *backups* cifrados, onde foram feitos alguns passos para o realizar, já que temos de respeitar alguns pré requisitos, dos quais eles:

- ★ Criação de uma chave mestre, utilizando uma password forte;
- ★ Criação de um certificado;
- ★ Criação do *Backup* utilizando um algoritmo forte e seguro (ex: *AES 256*).

```
|BACKUP DATABASE [ProjetoBD2] TO DISK = N'E:\Backups\ProjetoBD2_normal.bak'
WITH NOFORMAT, NOINIT, NAME = N'ProjetoBD2- Full Database Backup non encrypted',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

--encrypted backup
use master;
GO

|CREATE MASTER KEY
    ENCRYPTION BY PASSWORD = 'Pa$$w0rd'
GO

|CREATE CERTIFICATE ProjetoBD2Cert
    WITH SUBJECT = 'SQL-BD2 self-signed backup'
GO

|BACKUP DATABASE ProjetoBD2
    TO DISK = N'E:\Backups\ProjetoBD2_EncryptedV3.bak'
    with
        Encryption (
            Algorithm = AES_256,
            SERVER CERTIFICATE = ProjetoBD2Cert
        );
GO
```

Figura 5.3: *Querys Backups normais e cífrados + certificado*

5. SEGURANÇA IMPLEMENTADA NA BASE DE DADOS

5.1.2 *Schedule de Backups*

Para a criação de agendamento de backups, foi utilizados *jobs*, isto é, são "trabalhos" que são executados quando pretendido. Pode se agendar um pouco de tudo e tornar uma base de dados muito versátil. Neste caso agendei 3 backups realizados para 2 discos separados, discos nos quais foram referidos no capítulo 3.1.

Os passos para a criação do *schedule* foram:

- ★ Criação de um *job* e definir o seu nome;
- ★ Criar um *step*, isto é, um passo que o *job* irá realizar;
- ★ Criar o agendamento;
- ★ Ativar o *job* , tendo em conta o *step* criado.

O Pretendido era criar *backups* cifrados, como foi referido anteriormente, mas pelo o que percebi não é possível, por não optei por agendar backups não cifrados. Para os *raids* foram elaborados backups mais da base de dados cifrada e também *backups* do certificado criado. Tudo isso foi testado ao fazendo *restore* da base de dados num computador diferente. O que foi evidenciado é que os *backups* cifrados não eram possíveis de ser abertos devido a não conter o certificado, já que assim torna tudo muito mais seguro. Quanto aos *backups* normais tudo funcionou corretamente na nova máquina. O *script* com o desenvolvido encontra-se no anexo II, e a pesquisa acerca foi realizada em [SQL23] .

Plano de Backups

- ★ *Backup Full* - todos os dias às 3 da manhã;
- ★ *Backup diferencial* - todos os dias de 4 em 4 horas;
- ★ *Backup transacion log* - todos os dias de 15 em 15 minutos.

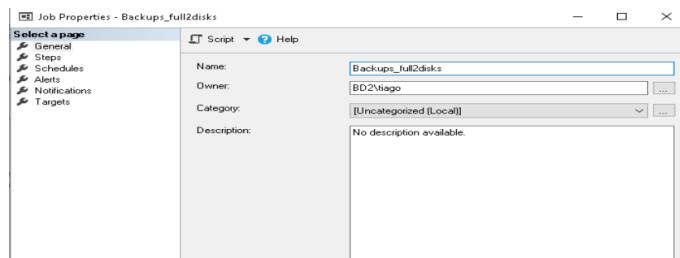


Figura 5.4: Schedule backups

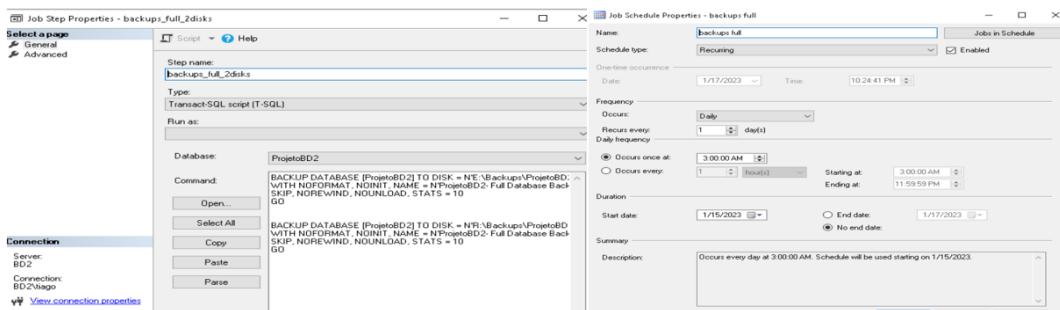


Figura 5.5: Schedule backups2

5.2 Utilizadores e Permissões

A nível de utilizadores foi decidido criar vários *users* para demonstrar o trabalho e também para tentar aproximar o trabalho o mais possível para o mundo de trabalho, onde havia uma diferença efetiva de *users*, como é demonstrado na tabela abaixo. Contendo uma boa separação dos dados e da informação tornou-se fácil atribuir as permissões desejadas, onde foi trabalhado sempre através de permissões a objetos específicos da base de dados, o que facilitou o trabalho numa forma geral.

- ★ *Admin Windows;*
- ★ *Select_ALL Windows;*
- ★ *Insert_ALL Windows;*
- ★ *Reservas Manager ;*
- ★ *Cliente Manager;*
- ★ *Propriedades Manager;*
- ★ *Cliente SQL.*



Figura 5.6: *Users*

5.2.1 Utilizadores *Windows* e Suas Permissões - Base de Dados

Os utilizadores *Windows* foram repartidos, sendo que cada tinha permissões, diferentes

- ★ *Admin Windows* - Permissões totais, já que se trata do administrador da base de dados, onde poderá fazer tudo o que deseja;
- ★ *Select_ALL Windows* - Permissões unicamente para ver os dados, neste caso unicamente permissões de *select*;
- ★ *Insert_ALL Windows*; Permissões unicamente para escrever e atualizar os dados de tabelas, neste caso unicamente permissões de *insert*;
- ★ *Reservas Manager* - Permissões ver e inserir dados nas tabelas representativas do *schema* reservas, e permissões de verificação de dados de clientes e propriedades;
- ★ *Cliente Manager*; - Permissões ver e inserir dados nas tabelas representativas do *schema* Cliente, e permissões de verificação de dados de reservas e propriedades;
- ★ *Propriedades Manager*; - Permissões ver e inserir dados nas tabelas representativas do *schema* propriedades, e permissões de verificação de dados de clientes e reservas;

De seguida serão demonstrados os passos para criação de *logins* e utilizadores.

5. SEGURANÇA IMPLEMENTADA NA BASE DE DADOS

Criação dos Utilizadores Windows (*LOGINS - Security SQL Server*)

Antes da criação dos utilizadores, é necessário criar os utilizadores de *login*, devido a que os utilizadores de uma base de dados são associados aos *logins* existentes. *Logins* servem para entrar em qualquer base de dados, isto é, o objetivo é termos vários logins para várias bases de dados, sendo que depois é feita a separação para as bases de dados que queremos.

Para a criação dos *logins* basta definir o *login* que se quer, sendo ele de *windows* ou *sql*. Depois defini a base de dados “*default*”, isto é, a base de dados a ser selecionada automaticamente ao fazer *login*, que neste caso era denominada de ”*ProjetoBD2*”.

Passos para a criação de Logins

- ★ Tipo de *Login*
- ★ Base de dados *default*
- ★ Permissões, que neste caso foram a *public*.

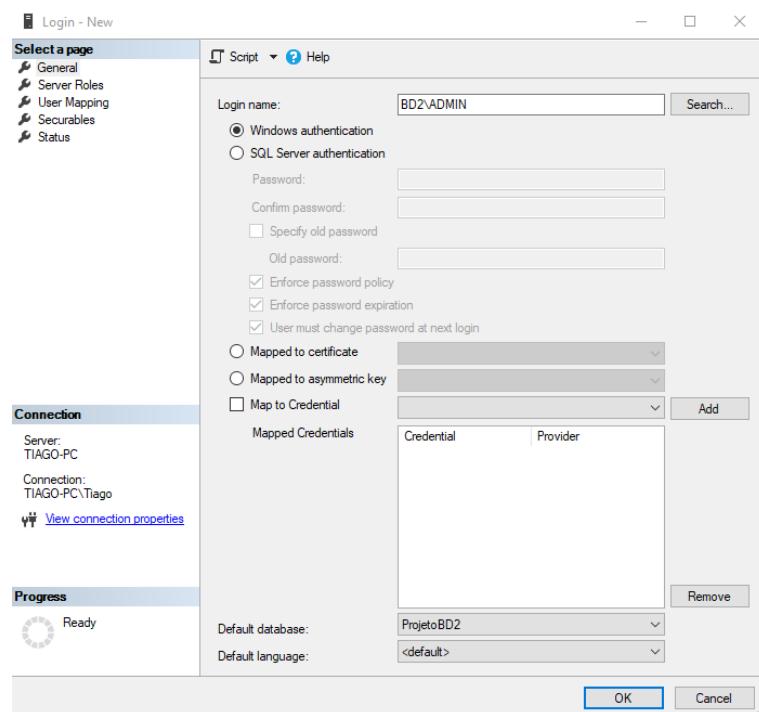


Figura 5.7: *Logins Windows*

Criação dos *Users Windows* na Base de Dados

Com os *logins* de no *sql server* criados foi então começado a criar os utilizadores pretendidos, neste caso começando pelos utilizadores *windows*. Como referido anteriormente cada um tinhos objetivos e permissões fixas, portanto trabalhei através de objetos, de modo a dar lhes as permissões corretas. Objetos e *schemas* é ideal para dar permissões específicas e de uma forma organizada. O primeiro passo é definir o tipo de utilizador, neste caso "*windows*", depois segue a definição do *username*, e por fim definir o *login* que se quer.

Por fim e de forma opcional foi criado um *schema* especificamente desse utilizador que poderia ter permissões específicas para esse utilizador.

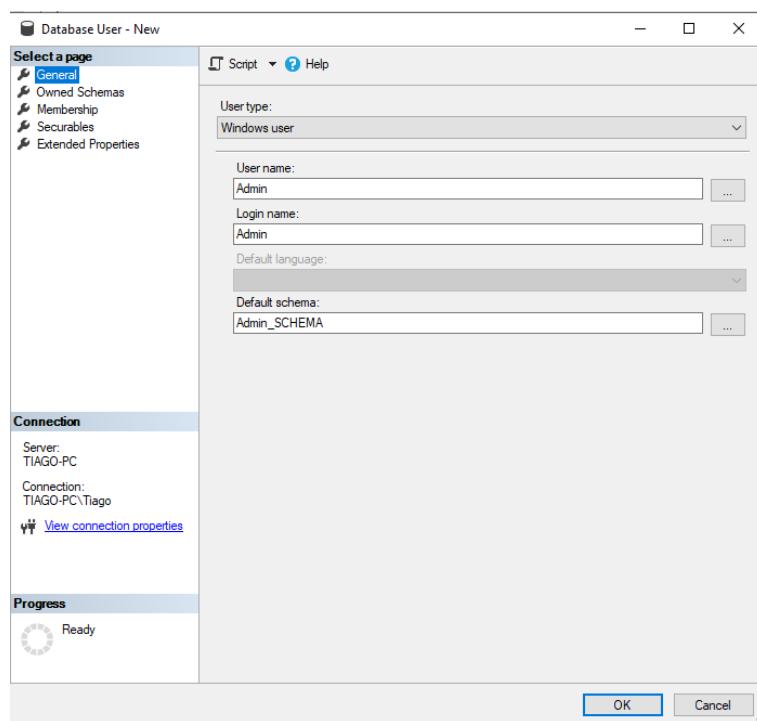


Figura 5.8: *User Windows*

5. SEGURANÇA IMPLEMENTADA NA BASE DE DADOS

Permissões dos *Users Windows* na Base de Dados

A nível de permissões, foi utilizada a partição de *database membership*, sendo esta editada unicamente para o utilizador *Windows*, que seria o administrador da base de dados, contendo as permissões ”*preset*”, sendo elas : *db_datareader, db_datawriter, db_dlladmin*. Quanto aos restantes utilizadores foi utilizado a partição dos ”*Securables*”, nas quais definíamos os objetos que queria que houvesse ou não permissão.

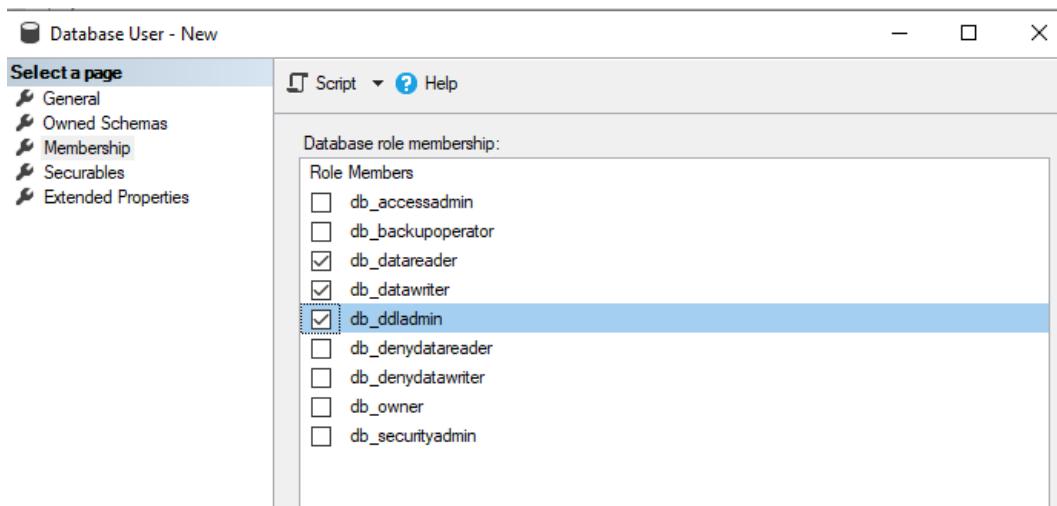


Figura 5.9: Permissões user windows

5.3 Criação dos Utilizadores SQL (*LOGINS - Security SQL Server*)

O processo para a criação do *login* para o utilizador cliente foi quase o mesmo que para os utilizadores *windows*. A única grande diferença é termos de definir uma password para o *login*, já que o utilizador é criado mesmo dentro do *sql* e não no exterior. Quanto a opções colocadas foi , para forçar uma política de passwords segura, isto é, unicamente funcionar *passwords* fortes com caracteres especiais, e não deixar criar *passwords* simples. Foi definido também que a password seria expirada tendo em conta algum tempo, o que faz com que não existam *passwords* estagnadas, querendo isto dizer, palavras *pass* a serem as mesmas durante muito tempo.

Por fim foi definido novamente a base de dados *default* e deixado tudo o resto com o seu *preset*. A *password* usada foi ”*Pa\$\$w0rd*”.

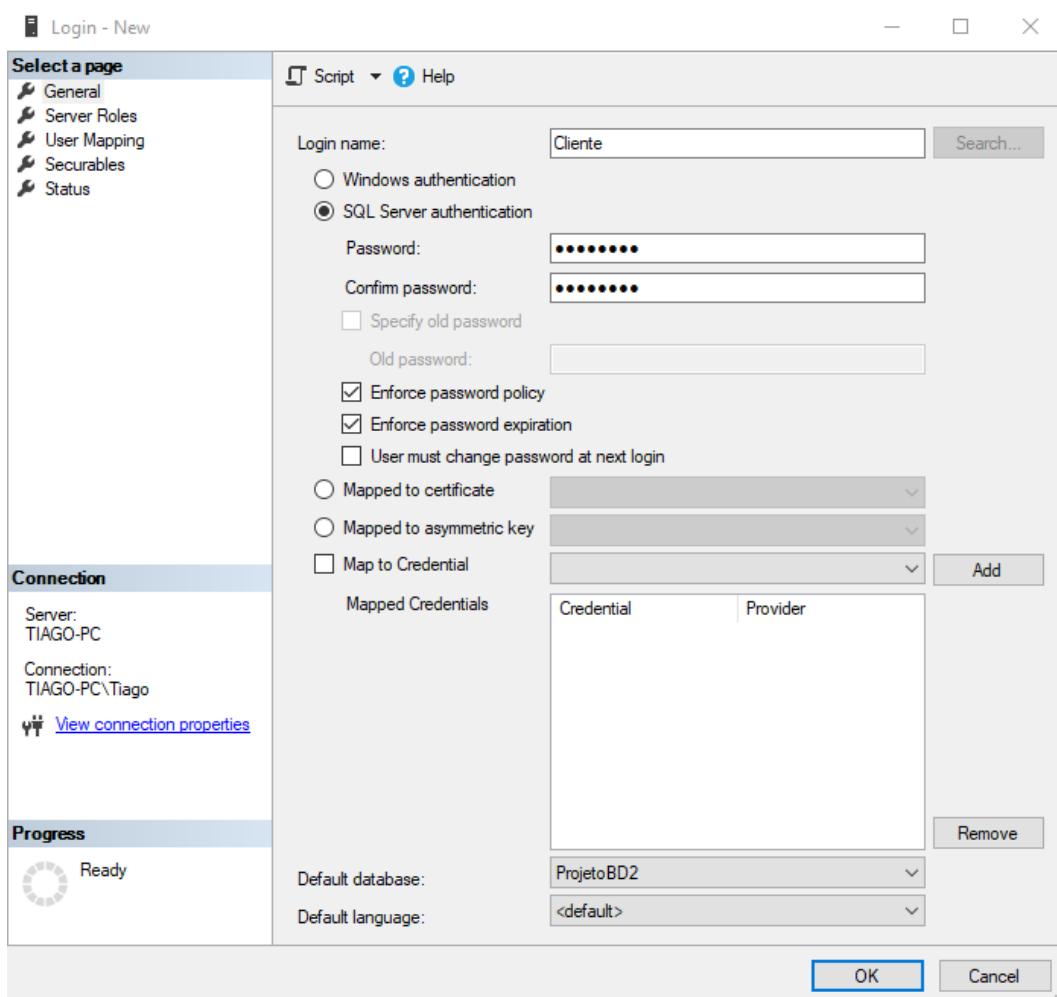


Figura 5.10: Utilizador sql - cliente

5. SEGURANÇA IMPLEMENTADA NA BASE DE DADOS

Criação dos *Users SQL* na Base de Dados

Com os *logins* de no *sql server* criados e os utilizadores "windows" foi então começado por criar o utilizador cliente de modo a testar um exemplo de cliente para a minha base de dados. Passou muito por definir o tipo de utilizador, neste caso "Utilizador SQL com Login", depois seguiu se a definição do *username* que iria aparecer na pasta *users* da base de dados e escolha do *login* correto que se quer.

Por fim e de forma opcional foi criado um *schema* especificamente para este utilizador que poderia ter permissões específicas e únicas dele próprio. Foram também criados utilizadores *sql*, sendo estes "réplicas" dos criados para o *SO windows*, de modo a haver utilizadores para tudo e onde se queira.

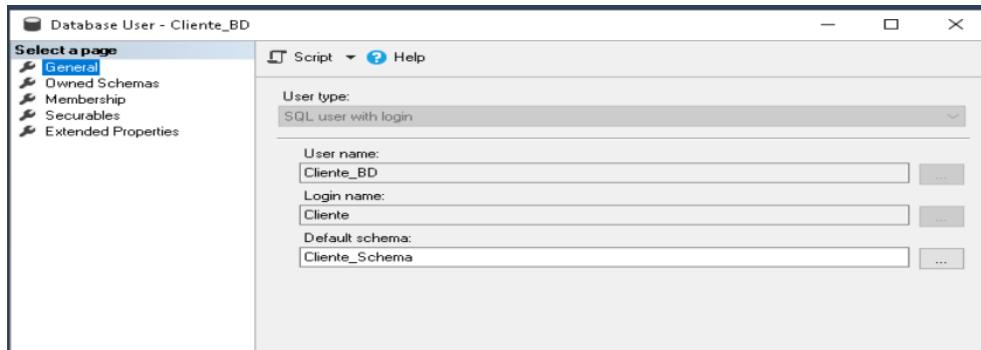


Figura 5.11: criação user cliente

Permissões do *User Cliente* na Base de Dados

A nível de permissões, foram realizadas as permissões muito ao exemplo do que foi feito anteriormente. Foi só necessário trabalhar com os "Securables"(Objetos), onde foram selecionadas as permissões necessárias.

Permissões Atribuídas ao Utilizador Cliente:

- ★ Permissões de leitura e inserção na tabela Reservas;
- ★ Permissões de leitura e inserção na tabela Clientes;
- ★ Permissões de leitura e inserção na tabela Utilizador;
- ★ Permissões de leitura na tabela Propriedades.
- ★ Permissões de leitura na tabela Avaliação_Dono.
- ★ Permissões de leitura na tabela Avaliação_Propriedade.

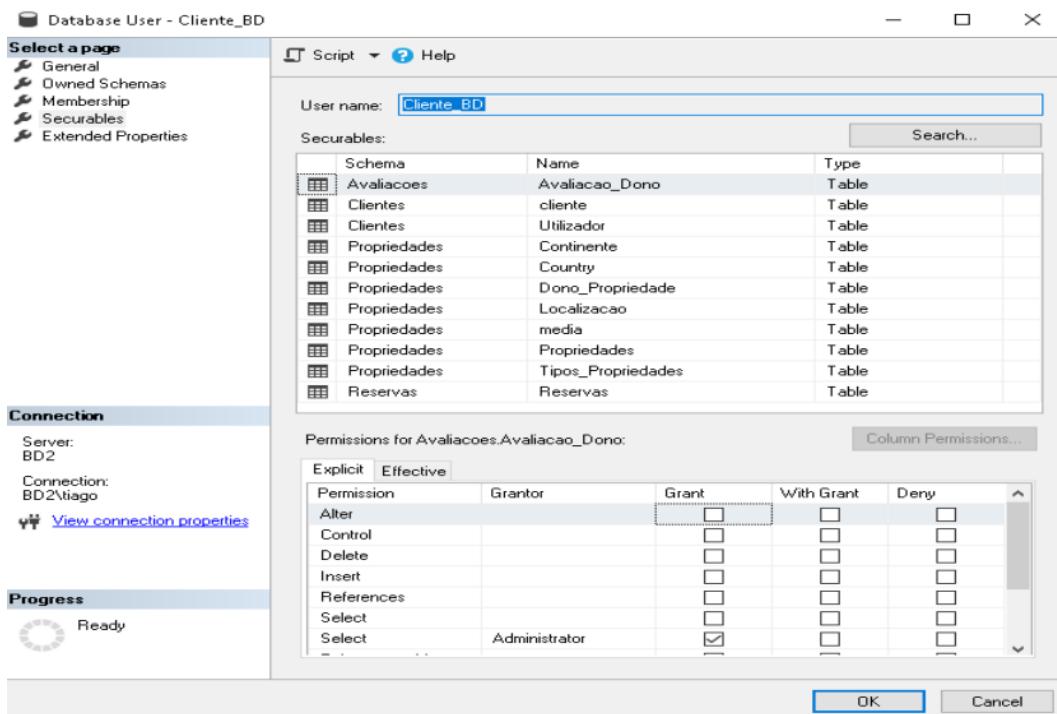


Figura 5.12: Permissões cliente

5. SEGURANÇA IMPLEMENTADA NA BASE DE DADOS

Verificação da Pasta Users

Segue a abaixo a verificação dos utilizadores na sua pasta específica.

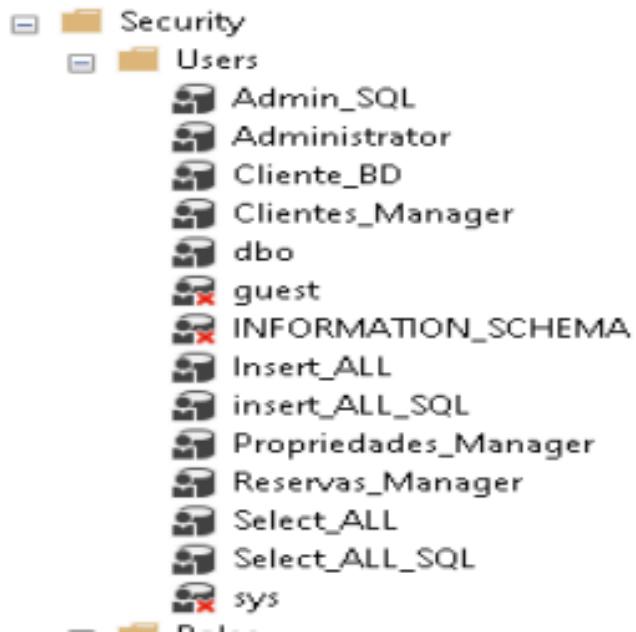


Figura 5.13: Pasta Users

O *script* com a criação dos utilizadores + *logins* encontram se no anexo III. Todo o desenvolvimento de permissões estará disponibilizado as *querys sql* junto da entrega do relatório.

Capítulo 6

Stored Procedures e Triggers

Um tema bastante importante foi a criação de *stored procedures* e especialmente *triggers*, onde dispus muito do meu tempo. Estes dois procedimentos tem como o seu maior objetivo tornar a base de dados dinâmica e mais automatizada, isto é, ela própria realiza pesquisas ou atualizações sem ter de ser os utilizadores a fazer tudo à mão. Numa base de dados de grande escala é bastante importante ter todos estes componentes a funcionar corretamente.

6.1 O que é um *Stored Procedure*?

Stored Procedure, que traduzido significa Procedimento Armazenado, é uma conjunto de comandos em *SQL* que podem ser executados de uma só vez, como em uma função. Ele armazena tarefas repetitivas e aceita parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual. Um *Stored Procedure* pode reduzir o tráfego na rede, melhorar a performance de um sistema de base de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento, etc.

6.2 *Triggers*

Um *Trigger* é um procedimento armazenado no banco de dados que é chamado automaticamente sempre que ocorre um evento especial no banco de dados. Por exemplo, um acionador pode ser chamado quando uma linha é inserida em uma tabela especificada ou quando determinadas colunas da tabela estão sendo atualizadas. Geralmente essas ações que acionam os *triggers* são alterações nas tabelas por meio de operações de inserção, exclusão e atualização de dados (*insert*, *delete* e *update*).

6.3 Criação de *Stored Procedures*

A nível de planeamento de *stores procedures* foi decidido criar uns procedimentos simples nos quais ajudavam o administrador e mesmo os utilizadores com permissões puderem fazer pesquisas à base de dados de uma forma mais rápida, sem terem de criar e escrever o código manualmente. Inicialmente foram criados 3 procedimentos para aceder aos dados respetivos, das tabelas principais : Clientes, Reservas e Propriedades. De seguida foram realizados procedimentos do género para as restantes tabelas, sendo estes procedimentos não tão importantes. Por fim foram elaborados alguns procedimentos tendo em conta a época do ano : Verão e Inverno, de modo a verificar os dados e posteriormente puder ser feitas extrações dos dados numa forma estatística.

As imagens abaixo mostras os procedimentos mais importantes criados para este trabalho.

```

CREATE PROCEDURE All_Clientes
AS (SELECT * FROM Clientes.Cliente)
GO

CREATE PROCEDURE All_Reservas
AS (SELECT * FROM Reservas.Reservas)

CREATE PROCEDURE All_Propriedades
AS (SELECT * FROM Propriedades.Propriedades)

EXEC All_Clientes
EXEC All_Propriedades
EXEC All_Reservas

```

Figura 6.1: *Stored Procedures:* 1

```

CREATE PROCEDURE All_Eventos_Cobranca
AS (SELECT * FROM Reservas.Eventos_Pagamento)
USE ProjetoBD2
GO

CREATE PROCEDURE All_Reservas_Verao
AS (SELECT * FROM Reservas.Reservas WHERE inicio_data BETWEEN '2023/06/01' AND '2023/10/01')

CREATE PROCEDURE All_Reservas_Inverno
AS (SELECT * FROM Reservas.Reservas WHERE inicio_data BETWEEN '2022/06/01' AND '2023/05/31')

EXEC All_Clientes
EXEC All_Propriedades
EXEC All_Reservas
EXEC All_Eventos_Cobranca
EXEC All_Reservas_Verao
EXEC All_Reservas_Inverno

```

Figura 6.2: *Stored Procedures:* 2

```

CREATE PROCEDURE All_Reservas_Verao
AS (SELECT * FROM Reservas.Reservas WHERE inicio_data BETWEEN '2023/06/01' AND '2023/10/01')
|

exec All_Clientes
exec All_Propriedades
exec All_Reservas
exec All_Eventos_Cobranca
4 Pausa Faturacao
| Result Messages
| ID_Reserva | inicio_data | periodo_total | preco_total | ID_Estado_Reserva | ID_Propriedades | ID_Taxa
| 1 | 2023-06-28 00:00:00.000 | 2023-06-30 00:00:00.000 | 312,00 | 2002-01-15 00:00:00.000 | 2002-01-16 00:00:00.000 | 1 | 1 | 1
| 1001 | 2023-06-28 00:00:00.000 | 2023-06-30 00:00:00.000 | 312,00 | 2002-01-15 00:00:00.000 | 2002-01-16 00:00:00.000 | 1 | 1 | 1
| 1002 | 2023-06-28 00:00:00.000 | 2023-06-30 00:00:00.000 | 312,00 | 2002-01-15 00:00:00.000 | 2002-01-16 00:00:00.000 | 1 | 1 | 1
| 1003 | 2023-06-28 00:00:00.000 | 2023-06-30 00:00:00.000 | 312,00 | 2002-01-15 00:00:00.000 | 2002-01-16 00:00:00.000 | 1 | 1 | 1
| 1004 | 2023-06-28 00:00:00.000 | 2023-06-30 00:00:00.000 | 312,00 | 2002-01-15 00:00:00.000 | 2002-01-16 00:00:00.000 | 1 | 1 | 1

```

Figura 6.3: *Stored Procedures:* 3

6.4 Criação de *Triggers*

A criação de *triggers* foi logo desde cedo, algo que eu queria muito fazer e já tinha muito bem planeado, tendo em conta a estrutura da minha base de dados. A ideia passou por criar alguns *triggers* na tabela de reservas onde a ideia passa pelo o seguinte: Tendo em conta que o cliente iria saber o preço da casa que desejava, e não saberia o preço total na primeira instância da reserva, tiveram de ser criados alguns *triggers*. A ideia principal era o cliente fazer a reserva e no fim da mesma estar realizada, o preço total da mesma ser atualizando tendo em conta época do ano em que a fez , se é reserva para o dia e quantos dias irá ser a estadia. Um *trigger* que foi criado também foi na tabela onde estará e será realizado o processo de pagamento, onde foi definido que a pessoa teria sempre de pagar um valor inicial que seria 10% do valor total. Com isso prevenia haver reservas canceladas, ou reservas sem fundamento.

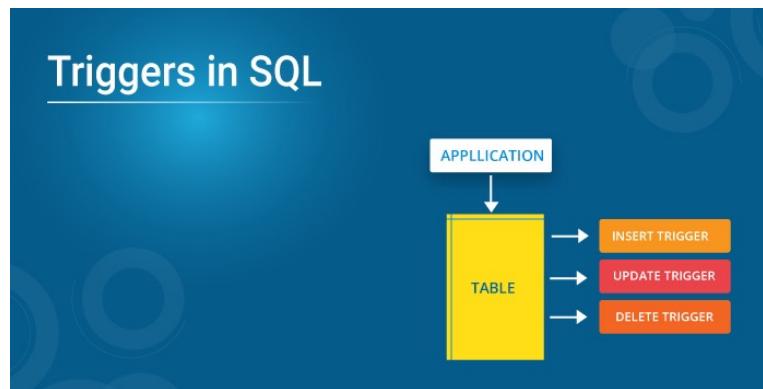


Figura 6.4: Triggers

1: Trigger

Este *trigger* é o *trigger* mais importante da base de dados e consequentemente o mais extenso onde atualiza o valor do preço total da reserva automaticamente. Este *trigger* funciona após qualquer inserção de dados na tabela Reservas.Reservas onde verifica um conjunto de condições, sendo elas:

- ★ Reserva do dia;
- ★ Reserva com uma data superior ao dia atual em que se está fazer a mesma;
- ★ Reservas tendo em conta época: caso época de verão o valor encarece, na época de inverno é aplicado a taxa normal de reserva e um valor mais baixo consequentemente.

Para a sua realização inicialmente criei algumas variáveis usando o ”*declare*” e defini o tipo de cada uma delas. Devido a este modelo de base de dados trabalhar com taxas, foram também criadas variáveis de taxas.

Sets utilizados para as variáveis:

- ★ Variáveis @Taxa (1..4) - Foi usado o *select* de modo a ir buscar os valores de cada taxa presente na tabela Taxas;
- ★ Variável @UltimoID_valor - Foi usado a função *MAX* para encontrar o ultimo *id* das reservas para só essa reserva ser atualizada;
- ★ Variável @Data_Inicial - Selecionada a data inicial tendo em conta a ultima reserva;
- ★ Variável @Convert_data_sistema - Usa as funções *GETDATE()* e depois converte para tipo ”*date*”;
- ★ Variável @return_countDays_entre2datas - Utiliza a função *DATEDIFF()*, de modo a entrar quantos dias de estadia seria. O cálculo seria feito utilizando os atributos inicio_data e fim_data, que foram coletados da ultima reserva inserida.

Legenda Taxas por parte do Administrador da BD:

- ★ TAXA 1 - Reserva dia Atual;
- ★ TAXA 2 - Reserva Verão;
- ★ TAXA 3 - Reserva Inverno;
- ★ TAXA 4 - Reserva dia superior ao dia atual.

Com isto foi feita uma estruturação usando o statement *IF, Else IF, ELSE*.

- ★ Caso fosse uma reserva feita para um dia superior ao atual e ser na época de inverno aplica a taxa 4 e taxa 3, sendo tudo multiplicado por o numero de dias que seriam a estadia;
- ★ Caso fosse uma reserva feita para um dia superior ao atual e ser na época de verão aplica a taxa 4 e taxa 2, sendo tudo multiplicado por o numero de dias que seriam a estadia;
- ★ Caso fosse uma reserva feita para o dia atual e ser na época de inverno aplica a taxa 1 e taxa 3, sendo tudo multiplicado por o numero de dias que seriam a estadia;
- ★ Caso fosse uma reserva feita para o dia atual e ser na época de verão aplica a taxa 1 e taxa 2, sendo tudo multiplicado por o numero de dias que seriam a estadia;

```

CREATE TRIGGER teste_FINALS -- RESERVA DIA DE HOJE
ON Reservas_Reservas
AFTER INSERT AS
declare @data_inicial date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint
declare @return_countDays_entre2datas int
declare @taxa1 decimal(19,4)
declare @taxa3 decimal(19,4)
declare @taxa2 decimal(19,4)
declare @taxa4 decimal(19,4)

set @taxa2 = (Select Taxa_Aplicada from Taxes where ID_Taxa=2)
set @taxa1 = (Select Taxa_Aplicada from Taxes where ID_Taxa=1)
set @taxa3 = (Select Taxa_Aplicada from Taxes where ID_Taxa=3)
set @taxa4 = (Select Taxa_Aplicada from Taxes where ID_Taxa=4)
set @ultimoID_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @data_inicial = (SELECT inicio_data FROM Reservas where ID_Reserva=@ultimoID_reservas)
set @data_inicial = (SELECT convert(date, GETDATE()))
set @converte_data_sistema = (select convert(date, @data_inicial))
set @return_countDays_entre2datas = (select DATEDIFF(day, inicio_data, fim_data) from Reservas where ID_Reserva= @ultimoID_reservas)

IF @data_inicial > @converte_data_sistema AND (@data_inicial < '2023-06-01' OR @data_inicial > '2023-10-01')
    BEGIN
        UPDATE Reservas
        SET preco_total = (((select preco from Propriedades_Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa4) * @taxa3) * @return_countDays_entre2datas
        WHERE Reservas.ID_Reserva = @ultimoID_reservas
    END;
ELSE IF @data_inicial > @converte_data_sistema AND (@data_inicial > '2023-06-01' OR @data_inicial < '2023-10-01')
    BEGIN
        UPDATE Reservas
        SET preco_total = (((select preco from Propriedades_Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa4) * @taxa2) * @return_countDays_entre2datas
        WHERE Reservas.ID_Reserva = @ultimoID_reservas
    END;
ELSE IF @data_inicial = @converte_data_sistema AND (@data_inicial < '2023-06-01' OR @data_inicial > '2023-10-01')
    BEGIN
        UPDATE Reservas
        SET preco_total = (((select preco from Propriedades_Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa1) * @taxa3) * @return_countDays_entre2datas
        WHERE Reservas.ID_Reserva = @ultimoID_reservas
    END;
ELSE
    BEGIN
        UPDATE Reservas
        SET preco_total = (((select preco from Propriedades_Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa2) * @return_countDays_entre2datas
        WHERE Reservas.ID_Reserva = @ultimoID_reservas
    END;

```

Figura 6.5: Trigger1

6. Stored Procedures e Triggers

2: Trigger

Este *trigger* é um dos *trigger* que trabalhará em conjunto com o *trigger* anteriormente mencionado, onde tem o objetivo de evitar inserções de dados na tabela Reservas.Reservas, com datas inferiores ao dia atual da sua realização, evitando a ocupação de espaço e perca de desempenho. Este *trigger* funciona após qualquer inserção de dados na tabela Reservas.Reservas onde verifica se a ultima reserva tem uma data inferior ao do sistema como referido anteriormente.

Para a sua realização inicialmente criei algumas variáveis usando o ”*declare*” e defini o tipo de cada uma delas.

Sets utilizados para as variáveis:

- ★ Variável @Data_Inicial - Selecionada a data inicial tendo em conta a ultima reserva;
- ★ Variável @Convert_data_sistema - Usa as funções *GETDATE()* e depois converte para tipo ”*date*”;
- ★
- ★ Variável @UltimoID_valor - Foi usado a função *MAX* para encontrar o ultimo *id* das reservas para só essa reserva ser atualizada;

Com isto foi feita uma estruturação usando o *statement IF*

- ★ Caso a variável @data_inicial fosse inferior à data do sistema, não permite introduzir a reserva, realizando os ditos *rollback statements*;

```
CREATE_TRIGGER Reservas_Incorreta -- RESERVA_N_PODE_SER_FEITA
ON Reservas.Reservas
AFTER INSERT AS
declare @data_inicial date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint

set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @data_inicial = (SELECT inicio_data from Reservas where ID_Reserva=@ultimoID_valor_reservas)
set @converte_data_sistema = (select convert(date, GETDATE()))

IF @data_inicial < @converte_data_sistema
    BEGIN
        Print 'A data introduzida esta incorreta'
        rollback transaction
    END;
```

Figura 6.6: Trigger2

3: Trigger

Este *trigger* é um dos *triggers* que trabalhará em conjunto com o *triggers* anteriormente mencionados, onde tem o objetivo de evitar inserções de dados na tabela Reservas.Reservas, com a data de termino da reserva ser inferior ao dia da sua realização, evitando a ocupação de espaço e perca de desempenho. Este *trigger* funciona após qualquer inserção de dados na tabela Reservas.Reservas onde verifica se a ultima reserva tem uma data de termino da reserva inferior ao do sistema como referido anteriormente.

Para a sua realização inicialmente criei algumas variáveis usando o ”*declare*” e defini o tipo de cada uma delas.

Sets utilizados para as variáveis:

- ★ Variável @fim_data - Selecionada a data de termino tendo em conta a ultima reserva;
- ★ Variável @Convert_data_sistema - Usa as funções *GETDATE()* e depois converte para tipo ”*date*”;
- ★ Variável @UltimoID_valor - Foi usado a função *MAX* para encontrar o ultimo *id* das reservas para só essa reserva ser atualizada;

Com isto foi feita uma estruturação usando o *statement IF*

- ★ Caso a variável @fim_data fosse inferior à data do sistema, não permite introduzir a reserva, realizando os ditos *rollback statements*;

```

CREATE TRIGGER Reserva_Data_Incorretav2 -- RESERVA DIA DE HJ
ON Reservas.Reservas
AFTER INSERT AS
declare @fim_dataaa date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint

set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @fim_dataaa = (SELECT fim_data from Reservas where ID_Reserva=@ultimoID_valor_reservas)
set @converte_data_sistema = (select convert(date, GETDATE()))

IF @fim_dataaa < @converte_data_sistema
BEGIN
    Print 'A data introduzida esta incorreta'
    rollback transaction
END;

```

Figura 6.7: Trigger3

4: Trigger

Este *trigger* é o ultimo para a tabela de Reservas que trabalhará novamente em conjunto com os *triggers* anteriormente mencionados, onde este tem o objetivo de evitar inserções de dados na tabela Reservas.Reservas, com a data de termino da reserva ser inferior ao inicio da reserva, evitando a ocupação de espaço e perca de desempenho.

Para a sua realização inicialmente criei algumas variáveis usando o ”*declare*” e defini o tipo de cada uma delas.

Sets utilizados para as variáveis:

- ★ Variável @Data_Inicial - Selecionada a data inicial tendo em conta a ultima reserva;
- ★ Variável @fim_data - Selecionada a data de termino tendo em conta a ultima reserva;
- ★ Variável @Convert_data_sistema - Usa as funções *GETDATE()* e depois converte para tipo ”*date*”;
- ★ Variável @UltimoID_valor - Foi usado a função *MAX* para encontrar o ultimo *id* das reservas para só essa reserva ser atualizada;

Com isto foi feita uma estruturação usando o *statement IF*

- ★ Caso a variável @fim_data fosse inferior à variável @data_inicial inferior à data do sistema, não permite introduzir a reserva, realizando os ditos *rollback statements*;

```
CREATE TRIGGER Reserva_Data_IncorretaV3
ON Reservas.Reservas
AFTER INSERT AS
declare @fim_dataa date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint
declare @data_inicial date
set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @fim_dataa = (SELECT fim_data from Reservas where ID_Reserva=@ultimoID_valor_reservas)
set @converte_data_sistema = (select convert(date, GETDATE()))
set @data_inicial = (SELECT inicio_data from Reservas where ID_Reserva=@ultimoID_valor_reservas)

IF @fim_dataa < @data_inicial
    BEGIN
        Print 'A data introduzida esta incorreta'
        rollback transaction
    END;
```

Figura 6.8: Trigger4

5: Trigger

Este *trigger* é referente à tabela Eventos_Reserva que tem uma simples função que é atualizar o atributo pagamento_inicial de uma reserva sendo 10% do valor total dessa mesma reserva. Sendo que o ID_Reserva será igual à chave primária da tabela Eventos_Reserva torna o processo mais fácil de ser executado.

Tendo em conta o que foi referido foi definido um set, iria atualizar o atributo em questão

- ★ Definir o atributo pagamento_inicial com o valor que vai ser coletado fazendo uma pesquisa à tabela reservas onde o ID_Reserva é igual ao ID_Eventos_Reservas. Obtendo esse valor multiplica por 0.10 (10%);

```
CREATE TRIGGER Eventos_Reserva_cobranca
ON Reservas_Eventos_Reserva
AFTER INSERT AS

UPDATE Reservas_Eventos_Reserva
SET pagamento_inicial = (select preco_total from Reservas_Reservas WHERE ID_Reserva = Eventos_Reserva.ID_Reserva) * 0.10
WHERE Eventos_Reserva.ID_Eventos_Reserva = ID_Reserva
print 'O pagamento inicial da reserva foi atualizado';
```

Figura 6.9: Trigger5

6. Stored Procedures e Triggers

Trigger Verificações nas Tabelas

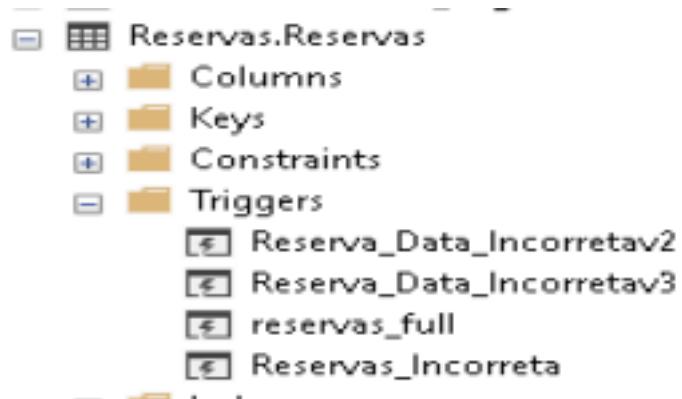


Figura 6.10: Pastas Triggers



Figura 6.11: Pastas Triggers2

Capítulo 7

Futuras Implementações Possíveis

As implementações que aqui posso citar foram *triggers* que achei que fariam sentido haver mas que não correram como planeado, pois foi tentado implementar cada uma delas.

O primeiro que pensei seria que as propriedades ao serem introduzidas tendo em conta a época do ser logo aplicada uma taxa, não ser feito nas reservas em si. A outra era trabalhar um pouco com a disponibilidade das propriedades, onde a ideia era haver um *trigger* que quando fosse feita uma reserva a sua disponibilidade passasse de *true* para *false*. Por fim um *trigger* em que não permitisse inserir reservas caso a sua disponibilidade tivesse com o seu valor a *false*. Para mim estes *triggers* futuramente poderiam ser implementados.

Outras futuras implementações seria colocar a funcionar o que foi pensado no primeiro modelo entregue aos docentes, que seria dentro de uma reserva poder haver varias propriedades e datas diferentes. Acho que seria uma grande melhoria para o meu modelo de base de dados como para todas as base de dados do género.



Figura 7.1: *Futuras Implementações*

7. FUTURAS IMPLEMENTAÇÕES POSSÍVEIS

Tópicos de futuras implementações:

Resumo das Implementações possíveis , mas que testadas não funcionaram a 100%

- ★ *Trigger* - Propriedades ao serem introduzidas tendo em conta a época do ser logo aplicada uma taxa;
- ★ *Trigger* - Ao fazer uma reserva a sua disponibilidade passa - se de *true* para *false*;
- ★ *Trigger* - Não permitir inserir reservas caso a sua disponibilidade tivesse com o seu valor a *false*;
- ★ Implementar a possibilidade de haver várias propriedades dentro da mesma reserva;

No anexo I encontra se os *scripts* todos usados, nomeadamente da criação da estrutura da base de dados, assim como os *scripts* que não funcionaram perfeitamente portanto não os deixei a funcionar na base de dados. Todas as outras *querys* como referido anteriormente serão entregues juntas do relatório em pastas separadas.

Capítulo 8

Manutenção e automatização do servidor

Uma Base de dados numa forma geral, é bastante importante por gera e trabalha com muitos dados, sendo que alguns até podem ser críticos, onde a privacidade dos mesmos tem de sempre prevalecer. É imprescindível garantir que a mesma esteja sempre a funcionar corretamente, sendo que a melhor forma de fazer isso é realizando manutenção da mesma. É importante perceber que base de dados no mundo de trabalho serão utilizados constantemente e tem de responder corretamente e sem falhas, seja para a realização de consultas, armazenamento de informações e também alterações frequentes das mesmas, já que pode estar em funcionamento 24 horas por dia, 7 dias da semana – o chamado ambiente 24/7.

Portanto e sendo que “a melhor defesa é o ataque”, é preciso monitorizar a base de dados antes que um problema ocorra e cause sua indisponibilidade. Daí a importância de realizar manutenção do Servidor.



Figura 8.1: Manutenção Base de Dados

8. MANUTENÇÃO E AUTOMATIZAÇÃO DO SERVIDOR

Tópicos de Manutenção e Automatização da Base de Dados:

- ★ A realização de *backups* pode afetar o desempenho da base de dados. Mesmo que novas tecnologias reduzam esse impacto, o volume de dados e os recursos de *hardware* ainda são os fatores que definirão o tempo de execução de um *schedule* de *backups*. Portanto, é recomendado encontrar um horário ideal para que essa manutenção tenha pouco impacto sobre os utilizadores da mesma. Nesta base de Dados os *schedules* definidos tiveram em conta isto mesmo , como pode ser lido no 5.1.2.
- ★ A performance da base de dados é um fator crítico, já que reflete diretamente na usabilidade em qualquer tipo de equipamento ou plataforma.
Portanto, estar atento às mudanças dos dados no decorrer da vida útil daquele ambiente é essencial. Isso pode resultar em benefícios na velocidade dos procedimentos existentes na *database*. Acompanhar a efetividade dos índices criados é bastante importante já certos índices poderão deixar de fazer mais sentido atualmente, embora fosse útil antes;
- ★ Verificações da existência de atualizações no sistemas de base de dados, e mesmo no sistema em que a base de dados está a correr por cima é bastante importante, já que assim melhorar a performance e acima de tudo a segurança numa forma geral.

Capítulo 9

Conclusão

A criação e planeamento de uma base de dados nem sempre é fácil e corre como o planeado. Foi um trabalho longo, onde acho que desempenhei bem e tenho aqui um trabalho muito bem elaborado, e que acredito que me sirva para o futuro a aprendizagem. Os temas pretendidos foram todos abordados, no quais tentei desempenhar da melhor forma e consequentemente a sua explicação estar explicita e bastante objetiva.

Por fim este trabalho permitiu consolidar a maior parte dos conteúdos lecionados.

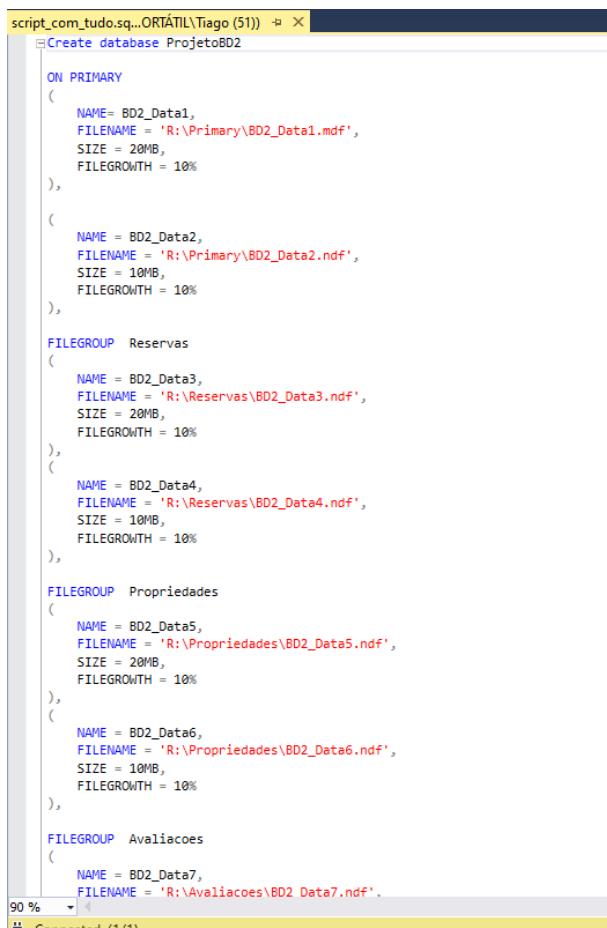
Bibliografia

- [SQL23] SQLHack. *Multiple methods for scheduling a SQL Server backup automatically.* <https://www.sqlshack.com/multiple-methods-for-scheduling-a-sql-server-backup-automatically/>. Consultado em 2023/01/2. 2023 (citado na página 26).

Anexos

Anexo I

Script total



```
script_com_tudo.sql..ORTÁTIL\Tiago (51))  x
Create database ProjetoBD2
ON PRIMARY
(
    NAME = BD2_Data1,
    FILENAME = 'R:\Primary\BD2_Data1.mdf',
    SIZE = 20MB,
    FILEGROWTH = 10%
),
(
    NAME = BD2_Data2,
    FILENAME = 'R:\Primary\BD2_Data2.ndf',
    SIZE = 10MB,
    FILEGROWTH = 10%
),
FILEGROUP Reservas
(
    NAME = BD2_Data3,
    FILENAME = 'R:\Reservas\BD2_Data3.ndf',
    SIZE = 20MB,
    FILEGROWTH = 10%
),
(
    NAME = BD2_Data4,
    FILENAME = 'R:\Reservas\BD2_Data4.ndf',
    SIZE = 10MB,
    FILEGROWTH = 10%
),
FILEGROUP Propriedades
(
    NAME = BD2_Data5,
    FILENAME = 'R:\Propriedades\BD2_Data5.ndf',
    SIZE = 20MB,
    FILEGROWTH = 10%
),
(
    NAME = BD2_Data6,
    FILENAME = 'R:\Propriedades\BD2_Data6.ndf',
    SIZE = 10MB,
    FILEGROWTH = 10%
),
FILEGROUP Avaliacoes
(
    NAME = BD2_Data7,
    FILENAME = 'R:\Avaliacoes\BD2_Data7.ndf'.

```

Figura I.1: Anexo 1 - 1

I. SCRIPT TOTAL

```
script_com_tudo.sql...ORTATIL\Tiago (51)  ↵ X
|
| FILEGROUP Avaliacoes
| (
|   NAME = BD2_Data7,
|   FILENAME = 'R:\Avaliacoes\BD2_Data7.ndf',
|   SIZE = 10MB,
|   FILEGROWTH = 10%
| ),
| (
|   NAME = BD2_Data8,
|   FILENAME = 'R:\Avaliacoes\BD2_Data8.ndf',
|   SIZE = 10MB,
|   FILEGROWTH = 10%
| ),
|
| FILEGROUP Clientes
| (
|   NAME = BD2_Data9,
|   FILENAME = 'R:\Clientes\BD2_Data9.ndf',
|   SIZE = 20MB,
|   FILEGROWTH = 10%
| ),
| (
|   NAME = BD2_Data10,
|   FILENAME = 'R:\Clientes\BD2_Data10.ndf',
|   SIZE = 10MB,
|   FILEGROWTH = 10%
| ),
|
| FILEGROUP Indices
| (
|   NAME = BD2_Data11,
|   FILENAME = 'R:\Indices\BD2_Data11.ndf',
|   SIZE = 25MB,
|   FILEGROWTH = 15%
| )
|
| LOG ON
| (
|   NAME = BD2_Log,
|   FILENAME = 'R:\Logs\BD2_Log.ldf',
|   SIZE = 10MB,
|   FILEGROWTH = 10%
| ),
| (
|   NAME = BD2_Log2,
|   FILENAME = 'R:\Logs\BD2_Log2.ldf',
|   SIZE = 10MB,
|   FILEGROWTH = 10%
| );
|
| GO
```

Figura I.2: Anexo 1 - 2

```
    NAME = BD2_Log2,
    FILENAME = 'R:\Logs\BD2_Log2.ldf',
    SIZE = 10MB,
    FILEGROWTH = 10%
);
GO

use ProjetoBD2
GO

CREATE SCHEMA Propriedades
CREATE SCHEMA Reservas
CREATE SCHEMA Clientes
CREATE SCHEMA Avaliacoes

use ProjetoBD2
GO

CREATE TABLE Reservas.Taxas (
    ID_Taxa smallint primary key,
    Taxa_APLICADA Decimal
)ON Reservas;

CREATE TABLE Reservas.Metodos_Pagamento (
    ID_Metodo_Pagamento smallint primary key,
    metodo_pagamento varchar(50)
)ON Reservas;

CREATE TABLE Propriedades.Continente (
    ID_continente smallint primary key,
    continente varchar(60)
)ON Propriedades;

CREATE TABLE Propriedades.Country (
    ID_country smallint primary key,
    country varchar(60),
    ID_continente SMALLINT NOT NULL,
    FOREIGN KEY (ID_continente)
    REFERENCES Propriedades.Continente (ID_continente)
    on update cascade
    on delete cascade
)ON Propriedades;

CREATE TABLE Propriedades.Localizacao (
    ID_localizacao smallint primary key,
) %
```

Figura I.3: Anexo 1 - 3

I. SCRIPT TOTAL

```
script_com_tudo.sql...ORTÁTIL\Tiago (51) + X
    on update cascade
    on delete cascade
)ON Propriedades;

CREATE TABLE Propriedades.Localizacao (
    ID_localizacao smallint primary key,
    nome_cidade varchar(60),
    cod_postal varchar(60),
    morada varchar(60),
    ID_country SMALLINT NOT NULL,
    FOREIGN KEY (ID_country)
    REFERENCES Propriedades.Country (ID_country)
    on update cascade
    on delete cascade,
)ON Propriedades;

CREATE TABLE Propriedades.Tipos_Propriedades (
    ID_Tipos_Propriedades smallint primary key,
    tipo_propriedade varchar(60),
    descricao varchar(60)
)ON Propriedades;

CREATE TABLE Propriedades.Dono_Propriedade (
    ID_Dono_Propriedade smallint primary key,
    nome varchar(60),
    idade int
)ON Propriedades;

CREATE TABLE Propriedades.media (
    ID_media smallint primary key,
    descricao varchar(60),
    fotos varchar(60),
    videos varchar(60)
)ON Propriedades;

Create table Propriedades.Propriedades (
    ID_propriedades smallint primary key,
    nome_propriedade varchar(60),
    descrição varchar(60),
    preco money,
    Disponibilidade bit,
    wc int,
    quartos int,
    camas int,
    cama_tipo varchar(60),
    tipo_casa varchar(60),
    
```

Figura I.4: Anexo 1 - 4

```
script_com_tudo.sql...ORTÁTIL\Tiago (51)  ✎ ×
```

```
| ID_media smallint primary key,  
| descricao varchar(60),  
| fotos varchar(60),  
| videos varchar(60)  
)ON Propriedades;
```

```
|  
| Create table Propriedades.Propriedades (
```

```
| ID_propriedades smallint primary key,  
| nome_propriedade varchar(60),  
| descrição varchar(60),  
| preco money,  
| Disponibilidade bit,  
| wc int,  
| quartos int,  
| camas int,  
| cama_tipo varchar(60),  
| tipo_casa varchar(60),  
| arcondicionado int,  
| internet bit ,  
| tv int,  
| publicacao_criada dateTime,  
| publicacao_atualizada dateTime,
```

```
|  
| ID_Tipos_Propriedades SMALLINT NOT NULL,  
| FOREIGN KEY (ID_Tipos_Propriedades)  
| REFERENCES Propriedades.Tipos_Propriedades (ID_Tipos_Propriedades)  
| on update cascade  
| on delete cascade,
```

```
| ID_localizacao smallint not null,  
| foreign key (ID_localizacao)  
| references Propriedades.Localizacao (ID_localizacao)  
| on update cascade  
| on delete cascade,
```

```
| ID_media smallint not null,  
| foreign key (ID_media)  
| references Propriedades.media (ID_media)  
| on update cascade  
| on delete cascade,
```

```
| ID_Dono_Propriedade smallint not null,  
| foreign key (ID_Dono_Propriedade)  
| references Propriedades.Dono_Propriedade (ID_Dono_Propriedade)  
| on update cascade  
| on delete cascade
```

```
90 %
```

Figura I.5: Anexo 1 - 5

I. SCRIPT TOTAL

```
script_com_tudo.sql...ORTÁTIL\Tiago (51) ➔ X
    on delete cascade,
    ID_Dono_Propriedade smallint not null,
    foreign key (ID_Dono_Propriedade)
    references Propriedades.Dono_Propriedade (ID_Dono_Propriedade)
    on update cascade
    on delete cascade

)ON Propriedades;

Create table Reservas.Estado_Reserva (
    ID_Estado_Reserva smallint primary key,
    descricao varchar(60),
    estado_reserva varchar(60)
)ON Reservas;

Create table Reservas.Reservas (
    ID_Reserva smallint primary key,
    inicio_data dateTime,
    fim_data dateTime,
    preco_total money,
    criado_quando dateTime ,
    atualizada_quando dateTime,
    ID_Estado_Reserva smallint not null,
    foreign key (ID_Estado_Reserva)
    references Reservas.Estado_Reserva (ID_Estado_Reserva)
    on update cascade
    on delete cascade,
    ID_Propriedades smallint not null,
    foreign key (ID_Propriedades)
    references Propriedades.Propriedades (ID_Propriedades)
    on update cascade
    on delete cascade,
    ID_Taxa smallint not null,
    foreign key (ID_Taxa)
    references Reservas.Taxas (ID_Taxa)
    on update cascade
    on delete cascade

)ON Reservas;

Create table Clientes.cliente (
    ID_cliente smallint primary key,
```

Figura I.6: Anexo 1 - 6

```
script_com_tudo.sql...ORTÁTIL\Tiago (51)  ↗ X

    )ON Reservas;

Create table Clientes.cliente (
    ID_cliente smallint primary key,
    primeiro_nome varchar(60),
    ultimo_nome varchar(60),
    nome_completo varchar(60)
)ON Clientes;

Create table Clientes.Utilizador (
    ID_user smallint primary key,
    user_nick varchar(60),
    email varchar(60),
    pa$$word varchar(60),
    criado_quando dateTime,
    contacto_telefonico varchar(60),
    descricao varchar(60),
    imagem_perfil varchar(60),

    ID_cliente smallint not null,
    foreign key (ID_cliente)
        references Clientes.cliente (ID_cliente)
        on update cascade
        on delete cascade,
    )ON Clientes;

Create table Reservas.Eventos_Reserva (
    ID_Eventos_Reserva smallint primary key,
    pagamento_inicial money,
    pagamento_total_executado bit,
    pagamento_cancelado bit,
    data_pagamento_cobrança dateTime,
    data_pagamento_total datetime,

    ID_Reserva smallint not null,
    foreign key (ID_Reserva)
        references Reservas.Reservas (ID_Reserva)
        on update cascade
        on delete cascade,

    ID_user smallint not null,
    foreign key (ID_user)
        references Clientes.Utilizador (ID_user)
        on update cascade
        on delete cascade,
    90 %
```

Figura I.7: Anexo 1 - 7

I. SCRIPT TOTAL

The screenshot shows a database script editor window with the following content:

```
script_com_tudo.sql...ORTÁTIL\Tiago (51) X
on delete cascade,
ID_Metodo_Pagamento smallint not null,
foreign key (ID_Metodo_Pagamento)
references Reservas.Metodos_Pagamento (ID_Metodo_Pagamento)
on update cascade
on delete cascade

)ON Reservas;

Create table Avaliacoes.Avaliacao_Propriedade (
ID_avaliacao smallint primary key,
descricao varchar(60),
pontuacao int ,

ID_user smallint not null,
foreign key (ID_user)
references Clientes.Utilizador (ID_user)
on update cascade
on delete cascade,
ID_propriedades smallint not null,
foreign key (ID_propriedades)
references Propriedades.Propriedades (ID_propriedades)
on update cascade
on delete cascade
)ON Avaliacoes;

Create table Avaliacoes.Avaliacao_Dono (
ID_avaliacao_dono smallint primary key,
descricao varchar(60),
pontuacao int ,

ID_Dono_Propriedade smallint not null,
foreign key (ID_Dono_Propriedade)
references Propriedades.Dono_Propriedade (ID_Dono_Propriedade)
on update cascade
on delete cascade,
ID_user smallint not null,
foreign key (ID_user)
references Clientes.Utilizador (ID_user)
on update cascade
on delete cascade
90 % < 
Connected. (1/1)
```

Figura I.8: Anexo 1 - 8

```

script_com_tudo.sql...ORTÁTIL\Tiago (51) ➔ ×
foreign key (ID_Dono_Propriedade)
references Propriedades.Dono_Propriedade (ID_Dono_Propriedade)
on update cascade
on delete cascade,
ID_user smallint not null,
foreign key (ID_user)
references Clientes.Utilizador (ID_user)
on update cascade
on delete cascade
)ON Avaliacoes;

create nonclustered INDEX ID_Cliente_Index ON Clientes.Utilizador (ID_cliente) on Indices
create nonclustered INDEX Nomes_Index ON Clientes.cliente (primeiro_nome,ultimo_nome) on Indices
create nonclustered INDEX Propriedades ON Reservas.Reservas (ID_Propriedades) on Indices
create nonclustered INDEX preco_nome ON Propriedades.Propriedades (nome_propriedade,preco) on Indices

INSERT INTO Reservas.Eventos_Reserva
VALUES (3,NULL,0,0,2002-01-15,2002-01-16, 2, 2);

select * from Reservas.Reservas
select * from Propriedades.Propriedades
select * from Reservas.Eventos_Reserva

BACKUP DATABASE [ProjetoBD2] TO DISK = N'E:\Backups\ProjetoBD2_full_20296.bak'
WITH NOFORMAT, NOINIT, NAME = N'ProjetoBD2- Full Database Backup non encrypted',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

-- backup Diferencial
BACKUP DATABASE [ProjetoBD2] TO DISK = N'E:\Backups\ProjetoBD2_diffteste.bak'
WITH DIFFERENTIAL , NOFORMAT, NOINIT, NAME = N'ProjetoBD2-diff Database Backup', SKIP, NOREWIND, NOUNLOAD, STATS = 10, CHECKSUM
GO

-- Backup Log
BACKUP LOG [ProjetoBD2] TO DISK = N'E:\Backups\ProjetoBD2_logteste.bak'
WITH NOFORMAT, NOINIT, NAME = N'Backup_Restore-Trasaction Log Backup',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

--encrypted backup
use master;
GO

CREATE MASTER KEY
90 %
CREATE CERTIFICATE [Anexo 1 - 9]
90 %
Connected. (1/1)

```

Figura I.9: Anexo 1 - 9

I. SCRIPT TOTAL

```
script_com_tudo.sql...ORTÁTIL\Tiago (51) -> X
    SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

-- backup Diferencial
BACKUP DATABASE [ProjetoBD2] TO  DISK = N'E:\Backups\ProjetoBD2_diffteste.bak'
WITH DIFFERENTIAL , NOFORMAT, NOINIT, NAME = N'ProjetoBD2-diff Database Backup', SKIP, NOREWIND, NOUNLOAD,  STATS = 10, CHI
GO

-- Backup Log
BACKUP LOG [ProjetoBD2] TO DISK = N'E:\Backups\ProjetoBD2_logteste.bak'
WITH NOFORMAT, NOINIT, NAME = N'Backup_Restore-Trasaction Log Backup',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

--encrypted backup
use master;
GO

CREATE MASTER KEY
    ENCRYPTION BY PASSWORD = 'Pa$$w0rd'
GO

CREATE CERTIFICATE ProjetoBD2Cert
    WITH SUBJECT = 'SQL-BD2 self-signed backup'
GO

BACKUP DATABASE ProjetoBD2
    TO DISK = N'E:\Backups\ProjetoBD2_EncryptedV5.bak'
    with
        Encryption (
            Algorithm = AES_256,
            SERVER CERTIFICATE = ProjetoBD2Cert
        );
GO

BACKUP CERTIFICATE ProjetoBD2Cert
    TO FILE = N'R:\Backups\ProjetoBD2Cert.bak'
GO

RESTORE DATABASE ProjetoBD2
    FROM DISK = N'E:\Backups\ProjetoBD2_Encrypted.bak'
GO

create procedure All_Clientes
as (select * from Clientes.cliente)
create procedure All_Reservas
GO
```

Figura I.10: Anexo 1 - 10

```
script_com_tudo.sql...ORTÁIL\Tiago (51)  # X
GO

BACKUP CERTIFICATE ProjetoBD2Cert
    TO FILE = N'R:\Backups\ProjetoBD2Cert.bak'
GO

RESTORE DATABASE ProjetoBD2
    FROM DISK = N'E:\Backups\ProjetoBD2_Encrypted.bak'
GO

create procedure All_Clientes
as (select * from Clientes.cliente)

create procedure All_Reservas
as (select * from Reservas.Reservas)

create procedure All_Propriedades
as ( Select * from Propriedades.Propriedades)

create procedure All_Eventos_Cobranca
as ( Select * from Reservas.Eventos_Pagamento)

use ProjetoBD2
GO

create procedure All_Reservas_Verao
as (select * from Reservas.Reservas where inicio_data between '2023/06/01' and '2023/10/01')

create procedure All_Reservas_Inverno
as (select * from Reservas.Reservas where inicio_data between '2022/06/01' and '2023/05/31')

exec All_Clientes
exec All_Propriedades
exec All_Reservas
exec All_Eventos_Cobranca
exec All_Reservas_Verao
exec All_Reservas_Inverno
```

Figura I.11: Anexo 1 - 11

I. SCRIPT TOTAL

```

script_com_tudo.sql...ORTÁTIL\Tiago (51)
exec All_Reservas_Verão
exec All_Reservas_Inverno

-- id = 1 - taxa hj
-- id = 2 - taxa verão
-- id = 3 - taxa inverno
-- id = 4 - taxa poshj

CREATE_TRIGGER Reservas_Incorreta -- RESERVA N PODE SER FEITA
ON Reservas.Reservas
AFTER INSERT AS
declare @data_inicial date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint

set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @data_inicial = (SELECT inicio_data FROM Reservas WHERE ID_Reserva=@ultimoID_valor_reservas)
set @converte_data_sistema = (select convert(date, GETDATE()))

IF @data_inicial < @converte_data_sistema
BEGIN
    Print 'A data introduzida esta incorreta'
    rollback transaction
END;

SELECT * FROM Propriedades_Propriedades
select * from Reservas.Reservas
SELECT *FROM Reservas.Taxas
INSERT INTO Reservas.Reservas
VALUES (1002,'2023-06-28','2023-06-30',NULL,'2002-01-15','2002-01-16', 1,1,3);

-- id = 1 - taxa hj
-- id = 2 - taxa verão
-- id = 3 - taxa inverno
-- id = 4 - taxa poshj

CREATE_TRIGGER reservas_full -- RESERVA DIA DE HJ
ON Reservas.Reservas
AFTER INSERT AS
declare @data_inicial date

```

Figura I.12: Anexo 1 - 12

```

script_com_tudo.sql...ORTÁTIL\Tiago (51)
CREATE_TRIGGER reservas_full -- RESERVA DIA DE HJ
ON Reservas.Reservas
AFTER INSERT AS
declare @data_inicial date
declare @converte_data_sistema date
declare @ultimoID_valor_reservas smallint
declare @return_countDays_entre2datas int
declare @taxa1 decimal(19,4)
declare @taxa2 decimal(19,4)
declare @taxa3 decimal(19,4)
declare @taxa4 decimal(19,4)

set @taxa1 = (Select Taxa_Aplicada From Taxas where ID_Taxa=2)
set @taxa2 = (Select Taxa_Aplicada From Taxas where ID_Taxa=1)
set @taxa3 = (Select Taxa_Aplicada From Taxas where ID_Taxa=3)
set @taxa4 = (Select Taxa_Aplicada From Taxas where ID_Taxa=4)
set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
set @data_inicial = (SELECT inicio_data FROM Reservas WHERE ID_Reserva=@ultimoID_valor_reservas)
set @converte_data_sistema = (select convert(date, GETDATE()))
set @return_countDays_entre2datas = (select DATEDIFF(dw, inicio_data, fim_data) FROM Reservas WHERE ID_Reserva=@ultimoID_valor_reservas)

IF @data_inicial > @converte_data_sistema AND (@data_inicial < '2023-06-01' OR @data_inicial > '2023-10-01')
BEGIN
    UPDATE Reservas
    SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa4) * @taxa3) * @return_countDays_entre2datas
    WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
END;

ELSE IF @data_inicial > @converte_data_sistema AND (@data_inicial > '2023-06-01' OR @data_inicial < '2023-10-01')
BEGIN
    UPDATE Reservas
    SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa4) * @taxa2) * @return_countDays_entre2datas
    WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
END;

ELSE IF @data_inicial = @converte_data_sistema AND (@data_inicial < '2023-06-01' OR @data_inicial > '2023-10-01')
BEGIN
    UPDATE Reservas
    SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa1) * @taxa3) * @return_countDays_entre2datas
    WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
END;

ELSE
BEGIN
    UPDATE Reservas
    SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) * @taxa1) * @taxa2) * @return_countDays_entre2datas
    WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
END;

```

Figura I.13: Anexo 1 - 13

```

script_com_tudo.sql...ORTÁTIL\Tiago (51) - X
    BEGIN
        UPDATE Reservas
        SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) *@taxa1)*@taxa3) * @return_countDays_entre2datas
        WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
    END;

    ELSE
        BEGIN
            UPDATE Reservas
            SET preco_total = (((select preco from Propriedades.Propriedades WHERE ID_propriedades = Reservas.ID_propriedades) *@taxa1)*@taxa2) * @return_countDays_entre2datas
            WHERE Reservas.ID_Reserva = @ultimoID_valor_reservas
        END;
    */

    SELECT * FROM Propriedades.Propriedades
    select * from Reservas.Reservas
    SELECT *FROM Reservas.Taxas
    INSERT INTO Reservas.Reservas
    VALUES (1007,'2023-05-28','2023-05-30',NULL,'2002-01-15','2002-01-16', 1,1,1);

    --CREATE TRIGGER Reserva_Data_Incorretav2 -- RESERVA DIA DE HO
    ON Reservas.Reservas
    AFTER INSERT AS
    declare @fim_data date
    declare @converte_data_sistema date
    declare @ultimoID_valor_reservas smallint
    set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
    set @fim_data = (SELECT fim_data FROM Reservas where ID_Reserva=@ultimoID_valor_reservas)
    set @converte_data_sistema = (select convert(date, GETDATE()));

    IF @fim_data < @converte_data_sistema
    BEGIN
        Print 'A data introduzida esta incorreta'
        rollback transaction
    END;

    --CREATE TRIGGER Reserva_Data_Incorretav3 -- RESERVA DIA DE HO
    ON Reservas.Reservas
    AFTER INSERT AS
    declare @fim_data date
    declare @converte_data_sistema date
    declare @ultimoID_valor_reservas smallint
    declare @data_inicial date
    90 %

```

Figura I.14: Anexo 1 - 14

```

script_com_tudo.sql...ORTÁTIL\Tiago (51) - X
    declare @fim_data date
    declare @converte_data_sistema date
    declare @ultimoID_valor_reservas smallint
    declare @data_inicial date
    set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
    set @fim_data = (SELECT fim_data FROM Reservas where ID_Reserva=@ultimoID_valor_reservas)
    set @converte_data_sistema = (select convert(date, GETDATE()))
    set @data_inicial = (SELECT inicio_data FROM Reservas where ID_Reserva=@ultimoID_valor_reservas)

    IF @fim_data < @data_inicial
    BEGIN
        Print 'A data introduzida esta incorreta'
        rollback transaction
    END;

    --CREATE TRIGGER Eventos_Reserva_cobranca
    ON Reservas.Eventos_Reserva
    AFTER INSERT AS
    declare @looo int
    set @looo = (SELECT MAX(ID_Reserva) FROM Reservas)

    UPDATE Reservas.Eventos_Reserva
    SET pagamento_inicial = (select preco_total from Reservas.Reservas WHERE ID_Reserva = Eventos_Reserva.ID_Reserva) * 0.10
    WHERE Eventos_Reserva.ID_Eventos_Reserva = ID_Reserva
    print 'O pagamento inicial da reserva foi atualizado';

    -- testes nao funcionais

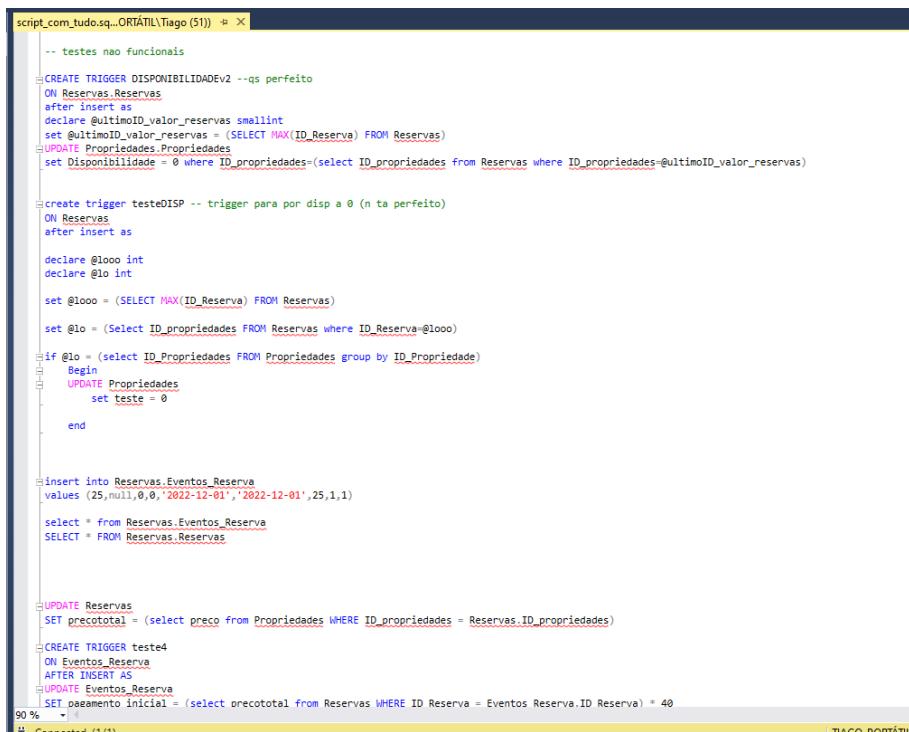
    --CREATE TRIGGER DISPONIBILIDADE2 --qs perfeito
    ON Reservas.Reservas
    after Insert as
    declare @ultimoID_valor_reservas smallint
    set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
    UPDATE Propriedades.Propriedades
    set disponibilidade = 0 where ID_propriedades=(select ID_propriedades from Reservas where ID_propriedades=@ultimoID_valor_reservas)

    --create trigger testeDISP -- trigger para por disp a 0 (na te perfeito)
    ON Reservas
    after Insert as
    declare @looo int
    declare @lo int
    set @looo = (SELECT MAX(ID_Reserva) FROM Reservas)
    set @lo = (Select ID_propriedades FROM Reservas where ID_Reserva=@looo)
    90 %

```

Figura I.15: Anexo 1 - 15

I. SCRIPT TOTAL



```
-- testes nao funcionais

CREATE TRIGGER DISPONIBILIDADE2 --qs perfeito
ON Reservas.Reservas
after insert as
declare @ultimoID_valor_reservas smallint
set @ultimoID_valor_reservas = (SELECT MAX(ID_Reserva) FROM Reservas)
UPDATE Propriedades.Propriedades
set Disponibilidade = 0 where ID_propriedades=(select ID_propriedades from Reservas where ID_propriedades=@ultimoID_valor_reservas)

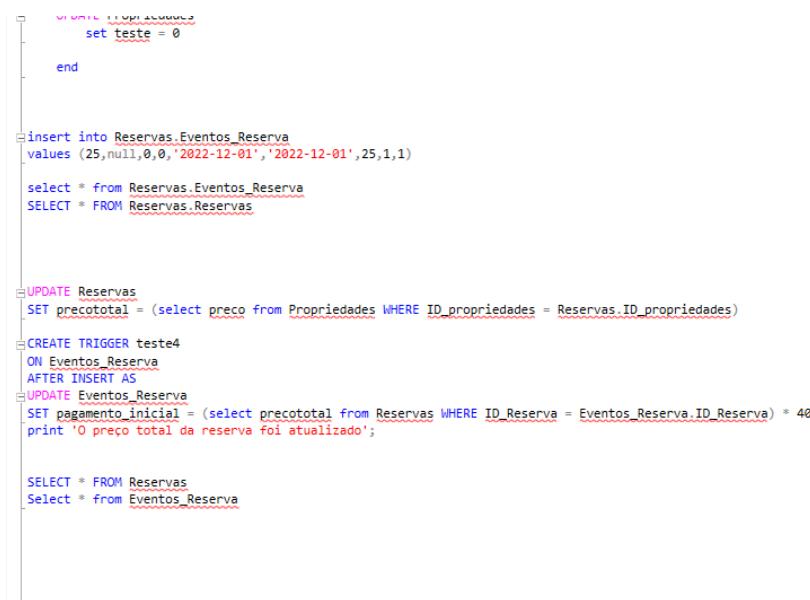
create trigger testeDISP -- trigger para por disp a 0 (n ta perfeito)
ON Reservas
after insert as
declare @looo int
declare @lo int
set @looo = (SELECT MAX(ID_Reserva) FROM Reservas)
set @lo = (Select ID_propriedades FROM Reservas where ID_Reserva=@looo)
if @lo = (select ID_Propriedades FROM Propriedades group by ID_Propriedade)
begin
    UPDATE Propriedades
    set teste = 0
end

insert into Reservas.Eventos_Reserva
values (25,null,0,0,'2022-12-01','2022-12-01',25,1,1)
select * from Reservas.Eventos_Reserva
SELECT * FROM Reservas.Reservas

UPDATE Reservas
SET precototal = (select preco from Propriedades WHERE ID_propriedades = Reservas.ID_propriedades)

CREATE TRIGGER teste4
ON Eventos_Reserva
AFTER INSERT AS
UPDATE Eventos_Reserva
SET pagamento_inicial = (select precototal from Reservas WHERE ID_Reserva = Eventos_Reserva.ID_Reserva) * 40
90%
Connected /1/1
```

Figura I.16: Anexo 1 - 16



```
/*
SET teste = 0
end

insert into Reservas.Eventos_Reserva
values (25,null,0,0,'2022-12-01','2022-12-01',25,1,1)
select * from Reservas.Eventos_Reserva
SELECT * FROM Reservas.Reservas

UPDATE Reservas
SET precototal = (select preco from Propriedades WHERE ID_propriedades = Reservas.ID_propriedades)

CREATE TRIGGER teste4
ON Eventos_Reserva
AFTER INSERT AS
UPDATE Eventos_Reserva
SET pagamento_inicial = (select precototal from Reservas WHERE ID_Reserva = Eventos_Reserva.ID_Reserva) * 40
print 'O preço total da reserva foi atualizado';

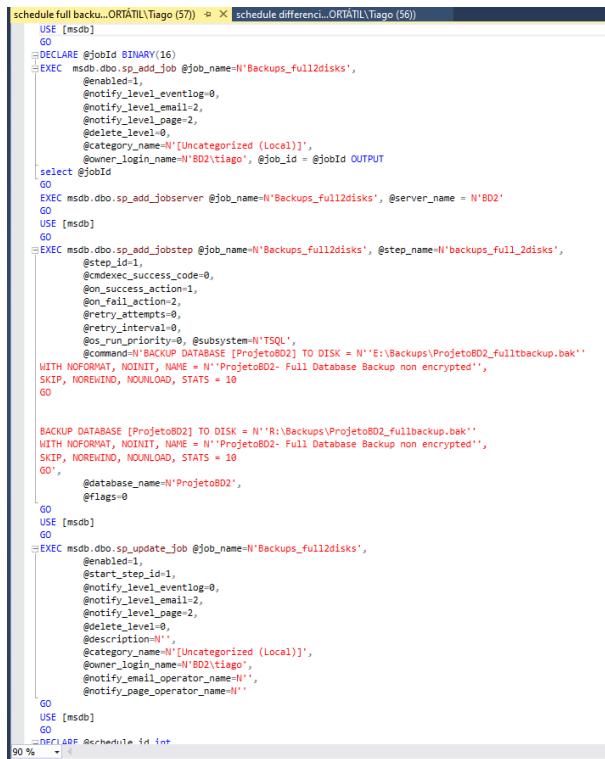
SELECT * FROM Reservas
Select * from Eventos_Reserva
```

Figura I.17: Anexo 1 - 17

Anexo II

Schedule Backups

Backups *Full*



The screenshot shows a Microsoft SQL Server Management Studio (SSMS) window with a dark theme. It displays a T-SQL script for creating a scheduled backup job. The script uses the msdb database and the sp_add_job and sp_add_jobstep stored procedures. The job, named 'Backups_full2disks', is configured to run at level 1 (Differential) with various notification options (eventlog, email, page) and delete options. It then performs a full backup of the 'ProjetoBD2' database to disk E:\Backups\ProjetoBD2_fullbackup.bak with specific options like NOFORMAT, NOINIT, and NAME. Finally, it updates the job with the new step information.

```
USE [msdb]
GO
=DECLARE @jobId BINARY(16)
=EXEC msdb.dbo.sp_add_job @job_name=N'Backups_full2disks',
    @enabled=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\tiago', @job_id = @jobId OUTPUT
,select @jobId
GO
EXEC msdb.dbo.sp_add_jobserver @job_name=N'Backups_full2disks', @server_name = N'BD2'
GO
USE [msdb]
GO
=EXEC msdb.dbo.sp_add_jobstep @job_name=N'Backups_full2disks', @step_name=N'backups_full_2disks',
    @step_id=1,
    @cmdexec_success_code=0,
    @on_success_action=1,
    @on_fail_action=2,
    @retry_attempts=0,
    @retry_interval=0,
    @os_run_priority=0, @subsystem=N'TSQL',
    @command=N'BACKUP DATABASE [ProjetoBD2] TO DISK = N''E:\Backups\ProjetoBD2_fullbackup.bak'''
WITH NOFORMAT, NOINIT, NAME = N'ProjetoBD2- Full Database Backup non encrypted'',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO

BACKUP DATABASE [ProjetoBD2] TO DISK = N'R:\Backups\ProjetoBD2_fullbackup.bak''
WITH NOFORMAT, NOINIT, NAME = N'ProjetoBD2- Full Database Backup non encrypted'',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO',
    @database_name=N'ProjetoBD2',
    @flags=0
GO
USE [msdb]
GO
=EXEC msdb.dbo.sp_update_job @job_name=N'Backups_full2disks',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @description=N'',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\tiago',
    @notify_email_operator_name=N'',
    @notify_page_operator_name=N''
GO
USE [msdb]
GO
--NFCIAFP Gerado automaticamente
90 %
--
```

Figura II.1: Anexo 2 - 1

II. SCHEDULE BACKUPS

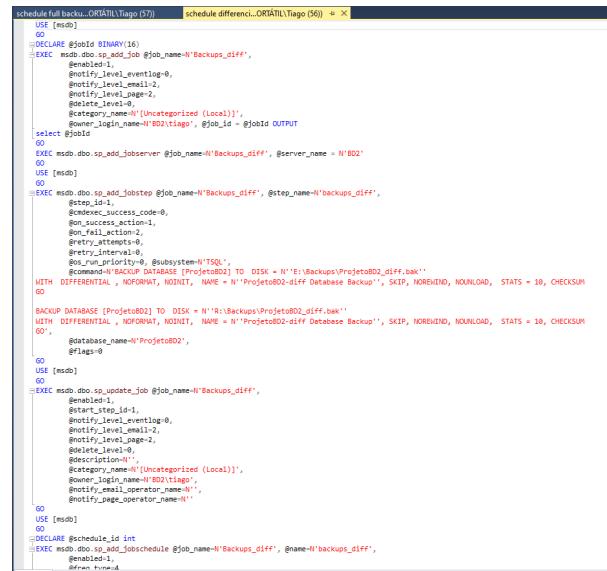
```

USE [msdb]
GO
EXEC msdb.dbo.sp_update_job @job_name=N'Backups_full2disks',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @description=N '',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\Tiago',
    @notify_email_operator_name=N '',
    @notify_page_operator_name=N ''
GO
USE [msdb]
GO
DECLARE @schedule_id int
EXEC msdb.dbo.sp_add_jobschedule @job_name=N'Backups_full2disks', @name=N'backups_full',
    @enabled=1,
    @freq_type=4,
    @freq_interval=1,
    @freq_subday_type=1,
    @freq_subday_interval=0,
    @freq_relative_interval=0,
    @freq_recurrence_factor=1,
    @active_start_date=20230115,
    @active_end_date=99991231,
    @active_start_time=30000,
    @active_end_time=235959, @schedule_id = @schedule_id OUTPUT
select @schedule_id
GO

```

Figura II.2: Anexo 2 - 2

Backups Diferencial



```

USE [msdb]
GO
DECLARE @jobId BINARY(16)
EXEC msdb.dbo.sp_add_job @job_name=N'Backups_diff',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\Tiago', @job_id = @jobId OUTPUT
select @jobId
GO
EXEC msdb.dbo.sp_add_jobstep @job_name=N'Backups_diff', @step_name=N'backups_diff',
    @cmd='
        EXEC msdb.dbo.sp_difffiles @job_name=N'Backups_diff', @server_name = N'BD2'
    ',
    @retry_attempts=0,
    @retry_interval=0,
    @on_success_action=0,
    @on_fail_action=2,
    @error_handling=0,
    @script=0
GO
EXEC msdb.dbo.sp_update_job @job_name=N'Backups_diff', @step_name=N'backups_diff',
    @enabled=1,
    @on_success_action=0,
    @on_fail_action=2,
    @retry_attempts=0,
    @retry_interval=0,
    @description=N '',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\Tiago',
    @notify_email_operator_name=N '',
    @notify_page_operator_name=N ''
GO
USE [msdb]
GO
DECLARE @schedule_id int
EXEC msdb.dbo.sp_add_jobschedule @job_name=N'Backups_diff', @name=N'backups_diff',
    @enabled=1,
    @freq_type=4

```

Figura II.3: Anexo 2 - 3

```

USE [msdb]
GO
:EXEC msdb.dbo.sp_update_job @job_name=N'Backups_diff',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @description=N'',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\tiago',
    @notify_email_operator_name=N '',
    @notify_page_operator_name=N ''
GO
USE [msdb]
GO
:DECLARE @schedule_id int
:EXEC msdb.dbo.sp_add_jobschedule @job_name=N'Backups_diff', @name=N'backups_diff',
    @enabled=1,
    @freq_type=4,
    @freq_interval=1,
    @freq_subday_type=8,
    @freq_subday_interval=4,
    @freq_relative_interval=0,
    @freq_recurrence_factor=1,
    @active_start_date=20230115,
    @active_end_date=99991231,
    @active_start_time=0,
    @active_end_time=235959, @schedule_id = @schedule_id OUTPUT
select @schedule_id
GO

```

Figura II.4: Anexo 2 - 4

Backups Logs

```

schedule.logs.sql -> ORTÁTIL\Tiago (58)  => schedule full backu...ORTÁTIL\Tiago (57)  schedule differenci...ORTÁTIL\Tiago (56)
USE [msdb]
GO
:DECLARE @jobId BINARY(16)
:EXEC msdb.dbo.sp_add_job @job_name=N'backups_log',
    @enabled=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\tiago', @job_id = @jobId OUTPUT
select @jobId
GO
EXEC msdb.dbo.sp_add_jobserver @job_name=N'backups_log', @server_name = N'BD2'
GO
USE [msdb]
GO
:EXEC msdb.dbo.sp_add_jobstep @job_name=N'backups_log', @step_name=N'backups log',
    @step_id=1,
    @onexes_success_code=0,
    @on_success_action=1,
    @on_fail_action=2,
    @retry_attempts=0,
    @retry_interval=0,
    @os_run_priority=0, @subsystem=N'TSQL',
    @command=N'BACKUP LOG [ProjetoBD2] TO DISK = N''E:\Backups\ProjetoBD2_log.bsk'' WITH NOFORMAT, NOINIT, NAME = N'Backup_Restore-Transaction Log Backup',
    SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
BACKUP LOG [ProjetoBD2] TO DISK = N'R:\Backups\ProjetoBD2_log.bak' WITH NOFORMAT, NOINIT, NAME = N'Backup_Restore-Transaction Log Backup',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO,
    @database_name=N'ProjetoBD2',
    @flags=0
GO
USE [msdb]
GO
:EXEC msdb.dbo.sp_update_job @job_name=N'backups_log',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @description=N'',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'BD2\tiago',
    @notify_email_operator_name=N '',
    @notify_page_operator_name=N ''
GO
USE [msdb]
GO
:DECLARE @schedule_id int
:EXEC msdb.dbo.sp_add_schedule @job_name=N'backups_log', @name=N'backups_log'

```

Figura II.5: Anexo 2 - 5

II. SCHEDULE BACKUPS

```
USE [msdb]
GO
EXEC msdb.dbo.sp_update_job @job_name=N'backups_log',
    @enabled=1,
    @start_step_id=1,
    @notify_level_eventlog=0,
    @notify_level_email=2,
    @notify_level_page=2,
    @delete_level=0,
    @description='',
    @category_name=N'[Uncategorized (Local)]',
    @owner_login_name=N'B02\Tiago',
    @notify_email_operator_name='',
    @notify_page_operator_name=''
GO
USE [msdb]
GO
DECLARE @schedule_id int
EXEC msdb.dbo.sp_add_jobschedule @job_name=N'backups_log', @name=N'backups_log',
    @enabled=1,
    @freq_type=4,
    @freq_interval=1,
    @freq_subday_type=4,
    @freq_subday_interval=15,
    @freq_relative_interval=0,
    @freq_recurrence_factor=1,
    @active_start_date=20230115,
    @active_end_date=99991231,
    @active_start_time=0,
    @active_end_time=235959, @schedule_id = @schedule_id OUTPUT
select @schedule_id
GO
```

Figura II.6: Anexo 2 - 6

Anexo III

Utilizadores e *Logins*

Utilizadores *Windows + SQL*



The screenshot shows a SQL Server Management Studio (SSMS) window with the following details:

- Title Bar:** users base de dado...ORTÁTIL\Tiago (62) - logins base de dad...ORTÁTIL\Tiago (61)) schedule logs.sql ~..ORTÁTIL\Tiago (58))
- Text Area:** The code lists the creation of several logins, including:
 - User [Admin_SQL] FOR LOGIN [Admin] WITH DEFAULT_SCHEMA=[Admin_SQL_Schema]
 - User [Administrator] FOR LOGIN [BD2\ADMIN] WITH DEFAULT_SCHEMA=[AdminSchema]
 - User [Cliente_BD] FOR LOGIN [Cliente] WITH DEFAULT_SCHEMA=[Cliente_Schema]
 - User [Clientes_Manager] FOR LOGIN [BD2\User3] WITH DEFAULT_SCHEMA=[Clientes_Manager_Schema]
 - User [Insert_ALL] FOR LOGIN [BD2\Admin_InsertData] WITH DEFAULT_SCHEMA=[Insert_ALL_Schema]
 - User [insert_ALL_SQL] FOR LOGIN [Insert_Data_User] WITH DEFAULT_SCHEMA=[insert_ALL_SQL_Schema]
 - User [Propiedades_Manager] FOR LOGIN [BD2\User2] WITH DEFAULT_SCHEMA=[Propiedades_Manager_Schema]
 - User [Reservas_Manager] FOR LOGIN [BD2\User1] WITH DEFAULT_SCHEMA=[Reservas_Manager_Schema]
 - User [Select_ALL] FOR LOGIN [BD2\Admin_Only_Select] WITH DEFAULT_SCHEMA=[Select_ALL_Schema]
 - User [Select_ALL_SQL] FOR LOGIN [Select_Data_User] WITH DEFAULT_SCHEMA=[Select_ALL_SQL_Schema]
- Bottom:** The code concludes with several ALTER ROLE statements, such as:
 - ALTER ROLE [db_ddladmin] ADD MEMBER [Admin_SQL]
 - ALTER ROLE [db_datareader] ADD MEMBER [Admin_SQL]
 - ALTER ROLE [db_datawriter] ADD MEMBER [Admin_SQL]
 - ALTER ROLE [db_ddladmin] ADD MEMBER [Administrator]
 - ALTER ROLE [db_datareader] ADD MEMBER [Administrator]
 - ALTER ROLE [db_datawriter] ADD MEMBER [Administrator]
 - ALTER ROLE [db_datawriter] ADD MEMBER [Insert_ALL]
 - ALTER ROLE [db_datawriter] ADD MEMBER [insert_ALL_SQL]
 - ALTER ROLE [db_datareader] ADD MEMBER [Select_ALL]
 - ALTER ROLE [db_datareader] ADD MEMBER [Select_ALL_SQL]

Figura III.1: Anexo 3 - 1

III. UTILIZADORES E *Logins*

Logins Windows + SQL

```
users base de dado...ORTÁTIL\Tiago (62)          logins base de dad...ORTÁTIL\Tiago (61)  -> X schedule logs.sql -...ORTÁTIL\Tiago (58)      schedule full back
-- logins sql
USE [master]
GO
CREATE LOGIN [Admin] WITH PASSWORD=N'Pa$$w0rd', DEFAULT_DATABASE=[ProjetoBD2], CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO

USE [master]
GO
CREATE LOGIN [Cliente] WITH PASSWORD=N'Pa$$w0rd', DEFAULT_DATABASE=[ProjetoBD2], CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO

USE [master]
GO
CREATE LOGIN [Insert_Data_User] WITH PASSWORD=N'Pa$$w0rd', DEFAULT_DATABASE=[ProjetoBD2], CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO

USE [master]
GO
CREATE LOGIN [Select_Data_Userr] WITH PASSWORD=N'Pa$$w0rd', DEFAULT_DATABASE=[ProjetoBD2], CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO

-- logins windows
USE [master]
GO
CREATE LOGIN [BD2\Admin] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO

USE [master]
GO
CREATE LOGIN [BD2\Admin_InsertData] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO

USE [master]
GO
CREATE LOGIN [BD2\Admin_Only_Select] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO

USE [master]
GO
CREATE LOGIN [BD2\User1] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO

USE [master]
GO
CREATE LOGIN [BD2\User2] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO

USE [master]
GO
CREATE LOGIN [BD2\User3] FROM WINDOWS WITH DEFAULT_DATABASE=[ProjetoBD2]
GO
```

Figura III.2: Anexo 3 - 2