Escola Superior de Tecnologia e Gestão de Beja

Curso de Engenharia de Informática
(regime nocturno)

Base de Dados II

Relatório do 2º Trabalho

Alunos:
Miguel Bilro Murta Soares nº2863
José Afonso Esteves Janeiro nº2467

Docente:
Artur Lança

2008

Índice

Introdução

Neste segundo trabalho vamos realizar a configuração e manutenção do servidor de bases de dados (SQL Server 2005), sendo implementada a base de dados desenvolvida no trabalho anterior. Para isso, vamos definir as políticas de segurança, políticas de recuperação, políticas de manutenção e automatização. Iremos ainda criar funções, Stored procedures, e triggers, para que seja possível inserir, apagar, actualizar e consultar dados.

O SQL Server 2005 fornece um forte modelo de segurança, que nos irá ajudar a evitar o acesso não autorizado aos dados. Este modelo é baseado nas permissões que damos aos *principals*, ou seja, aos indivíduos, grupos e processos. O modo de autenticação e os logins são o primeiro nível de segurança do SQL Server 2005. Assim sendo, nas políticas de segurança iremos definir um dos dois modos de autenticação, que são o Windows authentication e o Mixed Mode authentication, bem como criar logins para os utilizadores. Os logins são os principals do servidor que permitem o acesso por parte dos utilizadores. Para que os logins tenham acesso à base de dados é necessário criar um utilizador para cada login. A estes utilizadores iremos dar permissões através dos schemas e dos roles. No que diz respeito a segurança iremos ainda configurar a encriptação dos dados de algumas tabelas, cujos dados resultam das transacções dos utilizadores. A configuração da encriptação será feita através de certificates, *symmetric* e *asymmetric keys*.

No que diz respeito às políticas de recuperação iremos definir o modelo de recuperação de entre os três modelos que existem, que são *Full recovery model*, *Simple recovery model* e *Bulk-Logged recovery model*. Estes modelos determinam como o SQL trabalha com o transaction log, quando efectua o truncate do log (processo de remoção das transacções que foram *commited*, libertando espaço para novas transacções) e que opções de restauro estão disponíveis, como por exemplo, Full Backups, Differential Backups, Transaction Log Backups, Filegroup Backups, Mirrored Backups, etc.

Nas políticas de manutenção iremos tratar da fragmentação de índices, estatísticas, shrink da base de dados e de ficheiros, e integridade da base de dados. Os índices são um componente essencial para assegurar uma boa performance na execução de query, no entanto, os índices podem perder a sua performance se não tiverem uma correcta manutenção devido à fragmentação das páginas que os suportam. Os processos que envolvem operações de *delete* provocam a libertação de espaço nas páginas de índices, ficando estas apenas com uma fracção de linhas. Quando as

páginas não estão cheias como deveriam estar é chamado fragmentação interna. Os processos que envolvem operações de insert e update criam páginas adicionais de índices se as páginas onde necessitam de colocar os índices ou os dados já estão completamente cheias. Quando novas páginas são criadas ocorre uma divisão dos dados ou da informação de índices entre a página original e a nova página. Esta divisão de páginas provoca uma desordenação física das mesmas, ou seja a página original não vai ficar adjacente à nova página. Quando as páginas não estão ordenadas fisicamente ocorre a fragmentação externa. Outro aspecto importante na performance de querys é a informação estatística criada pelo SQL Server sobre a distribuição de valores numa coluna. As estatísticas são usadas na execução de querys para estimar o custo de utilizar um índice. A criação de estatísticas e a sua actualização pode ser feita automaticamente pelo SQL ou manualmente. O shrink da base de dados ou de ficheiros é usado para remover páginas que já não são usadas e recuperar espaço em disco. Estas páginas não usadas são o resultado de operações como eliminar grandes quantidades de dados. O shrink da base de dados ou de ficheiros pode ser feito manualmente ou automaticamente. O problema do shrink automático é que ao ser feito continuamente, leva à fragmentação ao nível de ficheiros, que pode ser bastante difícil de consertar, especialmente em base de dados com grande quantidade de dados. Além disso os DBAs apenas devem efectuar o shrink da base de dados ou de ficheiros se tiverem a certeza de que espaço não utilizado que vai ser libertado não voltará a ser preciso. No que diz respeito à integridade da base de dados o comando *DBCC CHECKDB* efectua uma variedade de verificações relativamente à alocação, integridade estrutural e lógica de todos os objectos na base de dados. Um aspecto a ter em conta na restauração da base de dados é que quando são reportados erros após a execução do comando *DBCC CHECKDB*, devemos restaurar a base de dados a partir de um backup recente. No entanto se não for possível restaurar a base de dados devido ao seu tamanho, devemos considerar executar os comandos *REPAIR_ALLOW_DATA_LOSS*, *REPAIR_FAST* ou *REPAIR_REBUILD*, que devem ser o ultimo recurso, pois estas operações não consideram os constraints que possam existir nas tabelas ou entre as tabelas, resultando numa perda de dados.

Relativamente à automatização, iremos usar o SQL Server Agent, para automatizar as operações de backups e de manutenção. Para isso iremos cria jobs, que irão ter steps, schedules, alerts e notifications. As notifications e os alerts necessitam de alguém para onde serão enviadas (operators). No nosso caso o operator será o administrador, ou seja nós próprios.

Stored procedures

Na nossa base de dados os utilizadores estão divididos por departamentos. Os departamentos são: compras, encomendas, reprodução e saúde. Para cada departamento criámos vários stored procedures. Decidimos criar stored procedures para cada departamento porque a tabela *dbo.animal* por exemplo, pertence a todos os departamentos. Assim o utilizador do departamento compras, apenas pode realizar operações de inserção, remoção e alteração sobre registos de animais que ele próprio tenha introduzido na tabela *dbo.animal*. Para que isso aconteça, foi-lhe dado permissão apenas de *select* para a tabela *dbo.animal* (`GRANT SELECT ON dbo.animal TO Compras_user)`, não tendo permissões para inserir, remover e alterar. Estas permissões são lhe dadas através do *role* que permite a execução de stored procedures do departamento compras. Assim sendo apenas pode inserir, remover e alterar através dos stored procedures sob certas condições que garantem que foi um utilizador do departamento compras que inseriu esse registo na tabela *dbo.animal*. Por exemplo, apenas pode eliminar um registo da tabela *dbo.animal* em que o *codigo_compra* não seja null, porque apenas os utilizadores do departamento compras podem inserir o *codigo_compra* nos registos de animais, também através de stored procedures, que não permitem que insira por exemplo o *codigo_encomenda*, que é inserido apenas pelos utilizadores do departamento encomendas, usando o mesmo processo. Para efectuar os updates à tabela *dbo.animal*, são também utilizados stored procedures que apenas permitem alterar os campos que dizem respeito ao departamento, no caso do departamento compras são todos os campos excepto o codigo_encomenda, *numero_sia_progenitor_macho* e *numero_sia_progenitor_femea,* pois o primeiro campo apenas pode ser inserido e actualizado pelo departamento encomendas e os dois últimos campos apenas podem ser inseridos e actualizados pelo departamento reprodução.

Para os vários departamentos foram ainda criados vários stored procedures para efectuar consultas sobre as tabelas que apenas dizem respeito aos diferentes departamentos. Por exemplo, para o departamento encomendas foi criado o stored procedure *dbo.clientesPagamentoEmAtraso* que devolve o codigo_cliente, o nome e o telefone dos clientes com pagamentos em atraso.

Os stored procedures utilizados pelos diferentes departamentos são:

Compras:

dbo.Compras_eliminarAnimal,

dbo.Compras_inserirAnimal,

dbo.Compras_updateAnimal_codigoCompra,

dbo.Compras_updateAnimal_codigoEspecie,

dbo.Compras_updateAnimal_dataNascimento,

dbo.Compras_updateAnimal_numeroParicular,

dbo.Compras_updateAnimal_sexo,

dbo.Compras_updateCompra_valorCompra

Encomendas:

dbo.clientesPagamentoEmAtraso,

dbo.Encomendas_updateAnimal_codigoEncomenda,

dbo.Encomendas_updateValorFactura,

dbo.verificaAnimaisDisponiveisParaVenda

Reprodução:

dbo.dataUltimoPartoDaFemea,

dbo.femeasSemPartoHaMaisDeUmAno,

dbo.Reproducao_eliminarAnimal,

dbo.Reproducao_inserirAnimal,

dbo.Reproducao_updateAnimal_codigoEspecie,

dbo.Reproducao_updateAnimal_dataNascimento,

dbo.Reproducao_updateAnimal_numeroParticular,

dbo.Reproducao_updateAnimal_numeroSiaProgenitorFemea,

dbo.Reproducao_updateAnimal_numeroSiaProgenitorMacho,

dbo.Reproducao_updateAnimal_sexo

Saúde:

dbo.cuidadosSaudeEfectuadosPorCadaVeterinario,

dbo.verificaAnimaisSemCuidadosSaudeUltimos2Meses

Foram criados dois stored procedures que são utilizados apenas pelo administrador, que somos nós. Estes stored procedures são *dbo.verCompraAudit* e *dbo.verFacturaAudit*. Servem para desencriptar e consultar os dados de duas tabelas que são *dbo.compra_audit* e *dbo.factura_audit*. Estas duas tabelas foram criadas já neste segundo trabalho e guardam os dados encriptados por certificates e symmetric key, dados esse que resultam de dois triggers que são executados quando um utilizador do departamento encomendas altera o valor das facturas ou quando um utilizador do departamento compras altera o valor das compras. A tabela dbo.*factura_audit* vai guardar o *codigo_factura*, o *valor_da_factura_antes* que é o valor da factura antes de ser alterado, *valor_da_factura_depois* que é o valor da factura depois de ser alterado, *data_audit* que é a data em que foi feita a alteração e o *utilizador_alterou* que é o utilizador que fez a alteração. A tabela *dbo.compra_audit* vai guardar o *codigo_compra*, *valor_da_compra_antes* que é o valor da compra antes de ser alterado, *valor_da_compra_depois* que é o valor da compra depois de ser alterado, *data_audit* que é a data em que foi feita a alteração e o *utilizador_alterou* que é o utilizador que fez a alteração.

A explicação dos stored procedures encontra-se nos próprios scripts e estes no anexo.

Functions

À semelhança dos stored procedures foram criadas várias scalar-valued functions e uma tabled-valued function, para os diferentes departamentos.

As funções utilizadas pelos diferentes departamentos são:

Compras:
dbo.numeroBensAgricolasMuitoMauEstado,
dbo.numeroAnimaisComprados

Encomendas:
dbo.animaisDisponiveisParaVenda,

dbo.avaliacaoCliente

Reprodução:

dbo.numeroFemeasMaisDeUmAnoDeIdadeSemParto

Saúde:

dbo.mediaCuidadosSaudeVeterinarioPorMes

A explicação das funções encontra-se nos próprios scripts e estes no anexo.

Triggers

No que diz respeito a triggers e para o departemento compras foi criado um trigger já falado anteriormente (*dbo.compraAudit*), que é executado quando um utilizador do departamento compras altera o valor da compra. Este trigger vai guardar os campos já referidos anteriormente encriptados na tabela *dbo.compra_audit* que apenas pode ser consultada pelo administrador.

Para o departamento encomendas foram criados dois trigger, um deles (*dbo.facturaAudit*) semelhante ao referido atrás, mas para a tabela *dbo.factura_audit*. O outro trigger (*dbo.verificaDescontoParaCliente*) é executado quando é feito um insert na tabela *dbo.encomenda*, e devolve uma mensagem a indicar que o cliente tem direito a um desconto no caso deste ter efectuado mais de 3 encomenda nos últimos dois meses.

Para o departamento reprodução foi criado um trigger (*dbo.femeaSemCuidadoSaudeComParto*), que é executado quando é feito um insert na tabela *dbo.parto*, e devolve uma mensagem a indicar que a femea do parto inserido não tem cuidados de saúde há mais de seis meses, no caso desse facto se verificar e a data do último cuidado de saúde dessa fémea.

Politicas de segurança

No que diz respeito a políticas de segurança, foi criado um login para cada utilizador e um utilizador por departamento. Os logins criados foram Compras_Pedro, Encomendas_Rui, Reproducao_Vitor e Saude_Luis. Para a autenticação dos logins no SQL Server 2005, escolhemos o modo de autenticação Mixed Mode Authentication. O

modo de autenticação recomendado para sistemas operativos Windows é o Windows Autentication Mode, pois podemos beneficiar das vantagens das políticas de segurança centralizada do domínio do Active Directory. No entanto, visto não termos nenhum domínio e para evitar estar a criar utilizadores no computador, escolhemos o modo de autenticação Mixed Mode Authentication, que deve ser utilizado quando queremos dar acesso a utilizadores de sistemas operativos diferentes do Windows. Este modo de autenticação permite ambos os logins do Windows e do SQL Server. Todos os logins criados têm a password igual a 123456.

```
USE master
GO

CREATE LOGIN Compras_Pedro WITH PASSWORD='123456', DEFAULT_DATABASE=AgroTejo,
CHECK_EXPIRATION=OFF, CHECK_POLICY=ON
GO
```

Os utilizadores foram criados com o de DEFAULT_SCHEMA como sendo o *dbo.*

```
USE AgroTejo
GO
CREATE USER Pedro FOR LOGIN Compras_Pedro WITH DEFAULT_SCHEMA=dbo
GO
```

Para dar permissões aos utilizadores criámos os nossos próprios database roles para agrupar os utilizadores que têm as mesmas necessidades de acesso, atribuindo as permissões nos database roles, assim todos os membros desse role têm essa permissões.

```
USE AgroTejo
GO

CREATE ROLE Compras_user AUTHORIZATION db_owner

EXEC sp_addrolemember 'Compras_user', 'Pedro'

GRANT DELETE ON dbo.compra TO Compras_user
GRANT INSERT ON dbo.compra TO Compras_user
GRANT SELECT ON dbo.compra TO Compras_user
GRANT UPDATE ON dbo.compra TO Compras_user

GRANT SELECT ON dbo.especie TO Compras_user

GRANT DELETE ON dbo.bem_agricola TO Compras_user
GRANT INSERT ON dbo.bem_agricola TO Compras_user
GRANT SELECT ON dbo.bem_agricola TO Compras_user
GRANT UPDATE ON dbo.bem_agricola TO Compras_user
```

```
GRANT DELETE ON dbo.fornecedor TO Compras_user
GRANT INSERT ON dbo.fornecedor TO Compras_user
GRANT SELECT ON dbo.fornecedor TO Compras_user
GRANT UPDATE ON dbo.fornecedor TO Compras_user

GRANT SELECT ON dbo.animal TO Compras_user

GRANT EXECUTE ON dbo.Compras_eliminarAnimal TO Compras_user
GRANT EXECUTE ON dbo.Compras_inserirAnimal TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateAnimal_codigoCompra TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateAnimal_codigoEspecie TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateAnimal_dataNascimento TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateAnimal_numeroParicular TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateAnimal_sexo TO Compras_user
GRANT EXECUTE ON dbo.numeroAnimaisComprados TO Compras_user
GRANT EXECUTE ON dbo.numeroBensAgricolasMuitoMauEstado TO Compras_user
GRANT EXECUTE ON dbo.Compras_updateCompra_valorCompra TO Compras_user
GRANT VIEW DEFINITION ON SYMMETRIC KEY::ComprasKey TO Compras_user
GRANT CONTROL ON CERTIFICATE::CompraCert TO Compras_user
GO
```

Como explicado nos stored procedures, visto a tabela *dbo.animal* pertencer a todos os departamentos, nos roles apenas foram dadas permissões de *select* para a tabela *dbo.animal*. Para efectuar operações de *insert*, *delete* e *update*, foram dadas permissões de execução para os stored procedures que permitem efectuar essa operações sob certas condições impedindo os utilizadores mexer nos registos e colunas que não dizem respeito ao seu departamento.

O SQL Server 2005 fornece uma hierarquia de encriptação baseada no service master key, que é uma chave simétrica gerada automaticamente na instalação do Servidor. O service master key é utilizado para encriptar passwords do Servidor, Strings de ligação, credenciais de contas e todas as master keys da base de dados. Fizemos um backup do service master key que foi guardado no seguinte path C:\DATABASES\AgroTejoBackups\Master key\exportedmasterkey

```
USE AgroTejo
GO

/*Efectua um backup da service master key para o path
C:\DATABASES\AgroTejoBackups\Master key\exportedmasterkey, usando a password
dy&jfh39ji#ww$h7
para encriptar o backup*/
BACKUP SERVICE MASTER KEY TO FILE='C:\DATABASES\AgroTejoBackups\Master
key\exportedmasterkey'
ENCRYPTION BY PASSWORD = 'dy&jfh39ji#ww$h7'
```

O database master key é o próximo nível na hierarquia de encriptação, que é uma chave simétrica opcional que criámos ao nível da base de dados para encriptar certificados e chaves.

```
USE AgroTejo
GO

/*Cria uma master key encriptada com a password m2#2i1$0g75#. A master key é
uma chave simétrica opcional criada ao nível da base de dados para encriptar
certificados e chaves na base de dados*/
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'm2#2i1$0g75#'
GO
```

Como já foi explicado anteriormente, temos duas tabelas que guardam os dados encriptados (*dbo.compra_audit* e *dbo.factura_audit*). Estas tabelas são preenchidas através de um trigger que é disparado quando um utilizador altera o valor da compra ou o valor da factura. Para encriptar os dados, utilizámos para cada departamento (compras e encomendas) uma chave simétrica protegida com um certificado, pois esta opção fornece um bom equilíbrio entre segurança e performance.

Para o departamento compras:

```
USE AgroTejo
GO

/*Cria o certificado que usaremos para encriptar a chave simétrica*/
CREATE CERTIFICATE CompraCert
WITH SUBJECT = 'Compra audit',
START_DATE = '02/07/2008'
GO
```

```
USE AgroTejo
GO

--Cria a chave simétrica encriptada pelo certificado CompraCert
CREATE SYMMETRIC KEY ComprasKey
WITH ALGORITHM = DES --A encriptação usando AES não é suportada no Windows XP
ENCRYPTION BY CERTIFICATE CompraCert
GO
```

Para o departamento encomendas:

```
USE AgroTejo
GO

/*Cria o certificado que usaremos para encriptar a chave simétrica*/
CREATE CERTIFICATE EncomendaCert
WITH SUBJECT = 'Factura audit',
START_DATE = '02/07/2008'
GO
```

```
USE AgroTejo
GO

--Cria a chave simétrica encriptada pelo certificado EncomendaCert
CREATE SYMMETRIC KEY EncomendasKey
WITH ALGORITHM = DES --A encriptação usando AES não é suportada no Windows XP
ENCRYPTION BY CERTIFICATE EncomendaCert
GO
```

Assim sendo, para encriptar os dados para a tabela *dbo.compra_audit*, utilizamos o trigger *dbo.compraAudit* e para a tabela *dbo.factura_audit*, utilizamos o trigger *dbo.facturaAudit*. Para desencriptar e mostrar os dados utilizamos os stored procedures *dbo.verCompraAudit* e *dbo.verFacturaAudit* respectivamente.

Politicas de recuperação

O modelo de recuperação que escolhemos foi o Full recovery model, pois neste modelo de recuperação o database engine efectua o log de todas as operações para o transaction log. Além disso nunca é efectuado o truncate do log, sendo possível restaurar a base de dados para um ponto em que falhou, ou para um ponto específico no tempo.

```
USE master
GO

ALTER DATABASE AgroTejo
SET RECOVERY FULL
```

Para os backups da base de dados decidimos usa a estratégia de efectuar um full backup vários differential backups e vários transaction log backups. O full backup irá guardar todos os dados existentes na base de dados, os differential backup irão guardar todas as alterações efectuadas na base de dados desde o último full backup, não necessitando dos differential backups anteriores, o que acontece com os backups incrementais. O principal objectivo do differential backup é reduzir o número de transaction log backups necessários para restaurar a base de dados. O transaction log backup contém apenas um subconjunto de dados e para o efectuar é necessário que tenhamos efectuado pelo menos um full backup à semelhança do differential backup.

Uma outra estratégia bastante aceitável que poderíamos ter escolhido seria efectuar filegroup backups, differential backups e transaction log backups, pois os filegroup backups são uma alternativa ao full backup.

```
USE master
GO

/*Cria um full backup da base de dados AgroTejo para o disco e para o seguinte
path
C:\DATABASES\AgroTejoBackups\AgroTejoFull.BAK, onde AgroTejoFull.BAK é o nome
do ficheiro*/
BACKUP DATABASE AgroTejo
TO DISK ='C:\DATABASES\AgroTejoBackups\AgroTejoFull.BAK'
WITH INIT


USE master
GO

/*Cria um differential backup da base de dados AgroTejo para o disco e para o
seguinte path
C:\DATABASES\AgroTejoBackups\AgroTejoDiff.BAK, onde AgroTejoDiff.BAK é o nome
do ficheiro*/
BACKUP DATABASE AgroTejo
TO DISK ='C:\DATABASES\AgroTejoBackups\AgroTejoDiff.BAK'
WITH DIFFERENTIAL


USE master
GO

/*Cria um backup do transactional log da base de dados AgroTejo para o disco e
para o seguinte path
C:\DATABASES\AgroTejoBackups\AgroTejoLog.BAK, onde AgroTejoLog.BAK é o nome do
ficheiro*/
BACKUP LOG AgroTejo
TO DISK ='C:\DATABASES\AgroTejoBackups\AgroTejoLog.TRN'
WITH INIT
```

Para recuperar a base de dados teremos de efectuar um restore de um full backup e só depois o restore do differential backup e do transaction log backup.

```
USE master
GO

/*O restore de um transaction log pode ser efectuado para recuperar a base de
dados para um ponto especifico. O restore de um transaction log pode ser também
aplicado a um full backup ou após um differential backup ter sido restaurado*/

/*Executa um restore de um full backup da base de dados AgroTejo a partir do
disco e do seguinte path C:\DATABASES\AgroTejoBackups\AgroTejoFull.BAK, onde
AgroTejoFull.BAK é o nome do ficheiro. A opcão WITH NORECOVERY deixa o estado
da base de dados disponivel RESTORING*/
RESTORE DATABASE AgroTejo FROM DISK =
'C:\DATABASES\AgroTejoBackups\AgroTejoFull.BAK'
WITH NORECOVERY

--Efectua a validação do backup
RESTORE VERIFYONLY FROM DISK = 'C:\DATABASES\AgroTejoBackups\AgroTejoFull.BAK'
```

```
/*Executa um restore de um differential backup da base de dados AgroTejo a
partir do disco e do seguinte path
C:\DATABASES\AgroTejoBackups\AgroTejoDiff.BAK, onde AgroTejoDiff.BAK é o nome
do ficheiro. A opcão WITH NORECOVERY deixa o estado da base de dados disponivel
RESTORING*/
RESTORE DATABASE AgroTejo FROM DISK =
'C:\DATABASES\AgroTejoBackups\AgroTejoDiff.BAK'
WITH NORECOVERY

--Efectua a validação do backup
RESTORE VERIFYONLY FROM DISK = 'C:\DATABASES\AgroTejoBackups\AgroTejoDiff.BAK'
```

```
/*Executa um restore de um transaction log da base de dados AgroTejo a partir
do disco e do seguinte path C:\DATABASES\AgroTejoBackups\AgroTejoLog.TRN, onde
AgroTejoLog.TRN é o nome do ficheiro. A opcão RECOVERY força a que não sejam
permitidas mais operações de restore*/
RESTORE LOG AgroTejo FROM DISK = 'C:\DATABASES\AgroTejoBackups\AgroTejoLog.TRN'
WITH RECOVERY

--Efectua a validação do backup
RESTORE VERIFYONLY FROM DISK = 'C:\DATABASES\AgroTejoBackups\AgroTejoLog.TRN'
```

Politicas de manutenção

Na política de manutenção tivemos em conta a verificação do nível de fragmentação dos índices, a sua reorganização caso necessário, a criação e actualização de estatísticas, o shrink de ficheiros ou da base de dados e a verificação e recuperação da integridade da base de dados.

A verificação do nível de fragmentação dos índices pode ser feita para todas as tabelas da base de dados ou para cada tabela individualmente.

Para todas as tabelas:

```
USE AgroTejo
GO

--Determina a fragmentação de indices para todas as tabelas da base de dados
SELECT OBJECT_NAME(dt.object_id), si.name, dt.avg_fragmentation_in_percent,
dt.avg_page_space_used_in_percent
FROM
(SELECT object_id, index_id, avg_fragmentation_in_percent,
avg_page_space_used_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID('AgroTejo'), NULL, NULL, NULL,
'DETAILED')
WHERE index_id <> 0) as dt
INNER JOIN sys.indexes si
ON si.object_id = dt.object_id
AND si.index_id = dt.index_id
```

Para as tabelas individualmente:

```
USE AgroTejo
GO

--Ver os niveis actuais de fragmentação da tabela dbo.animal
SELECT index_id, avg_fragmentation_in_percent, avg_page_space_used_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID('AgroTejo'),
OBJECT_ID('dbo.animal'), NULL, NULL, 'DETAILED')
WHERE index_id <> 0;
```

Para reorganizar ou reconstruir os índices temos de ter em conta o seguinte:

O ALTER INDEX ... REORGANIZE deve ser usado para desfragmentar indices em que o resultado de avg_page_space_used_in_percent é < 75 e > 60 ou quando o resusultado de avg__fragmentation_in_percent é > 10 e < 15, enquanto que o ALTER INDEX ... REBUILD deve ser usado para desfragmentar indices em que o resultado de avg_page_space_used_in_percent é < 60 ou quando o resultado de avg__fragmentation_in_percent é > 15.

Por este motivo a verificação, reorganização ou reconstrução dos indices não está incluida nos jobs utilizados no SQL Server Agent para a automatização da manutenção, sendo preferivel efectuar manualmente.

```
USE AgroTejo
GO

/*ALTER INDEX ... REORGANIZE deve ser usado para desfragmentar indices em que o
resultado de avg_page_space_used_in_percent é < 75 e > 60 ou quando o resultado
de avg__fragmentation_in_percent é > 10 e < 15*/


--Reorganizar o indice da tabela dbo.animal
ALTER INDEX ALL ON dbo.animal REORGANIZE;



USE AgroTejo
GO

/*ALTER INDEX ... REBUILD deve ser usado para desfragmentar indices em que o
resultado de avg_page_space_used_in_percent é < 60 ou quando o resusultado de
avg__fragmentation_in_percent é > 15*/


/*Reconstroi todos os indeces da tabela dbo.animal. Cria os indices com o fill
factor de 90, e permite que as operações de indices sejam efectuadas ONLINE*/
ALTER INDEX ALL ON dbo.animal REBUILD WITH (FILLFACTOR = 90, ONLINE = ON);
```

No que diz respeito a estatísticas, quando um DBA cria índices, o query optimizer guarda informações estatísticas sobre os índices. No entanto se o Servidor estiver configurado para criar estatisticas automaticamente, o database engine cria estatisticas nas colunas que não são índices mas que são usadas nos querys e o query optimizer actualiza automaticamente a informação estatística periodicamente à medida que os dados mudam nas tabelas.

Para criar estatisticas automaticamente:

```
USE master
GO


--Configurar a base de dados para criar estatisticas automáticamente
ALTER DATABASE AgroTejo
SET AUTO_CREATE_STATISTICS ON;
```

Para actualizar estatisticas automaticamente:

```
USE master
GO


--Configurar a base de dados para actualizar estatisticas automáticamente
ALTER DATABASE AgroTejo
SET AUTO_UPDATE_STATISTICS ON;
```

Também é importante criar estatisticas manualmente, porque podemos criar estatisticas que contêm densidades de valores para uma combinação de colunas e assim o database engine pode efectuar uma estimativa melhor para a execução de querys.

Para criar estatisticas para todas as tabelas:

```
USE AgroTejo
GO

/*Stored procedure do sistema que cria estatisticas para todas as colunas
necessitadas de todas as tabelas da base de dados AgroTejo.*/
EXEC sp_createstats;
```

Pelos motivos apresentados atrás não alterámos a predefinição para criar estatisticas automaticamente bem como para as actualizar. Além disso criámos ainda manualmente estatisticas para todas as colunas necessitadas de todas as tabelas da base de dados.

Como a criação e actualização de estatisticas estão configuradas para serem efectuadas automáticamente, não estão incluidas nos jobs utilizados no SQL Server Agent para a automatização da manutenção.

Relativamente ao shrink de ficheiros e da base de dados, podemos efectuar o shrink automáticamente para a base de dados ou manualmente para a base de dados ou ficheiros. No entanto como já referido na introdução o problema do shrink automático é que ao ser feito continuamente, leva à fragmentação ao nível de ficheiros, que pode ser bastante difícil de consertar, especialmente em base de dados com grande quantidade de dados. Além disso os DBAs apenas devem efectuar o shrink da base de dados ou de ficheiros se tiverem a certeza de que espaço não utilizado que vai ser libertado não voltará a ser preciso. Por este motivo optámos por efectuar o shrink manualmente e por esta razão não está incluido nos jobs utilizados no SQL Server Agent para a automatização da manutenção.

Para ver o tamanho actual da base de dados:

```
USE AgroTejo
GO

--Ver o tamanho actual da base de dados
SELECT file_id, name, physical_name, size, max_size FROM sys.database_files;
```

Para efectuar o shrink de toda a base de dados:

```
USE AgroTejo
GO

--Efectua o shrink da base de dados , sem deixar espaço livre
DBCC SHRINKDATABASE(AgroTejo, 0)
```

Para efectuar o shrink de ficheiros da base de dados:

```
USE AgroTejo
GO

--Efectua o shrink do transaction log  Agro_Log1, para o seu tamanho inicial
que são 3MB
DBCC SHRINKFILE('Agro_Log1', 3)
```

No que diz respeito à integridade da base de dados, incluimos a sua verificação nos jobs utilizados no SQL Server Agent para a automatização da manutenção.

Para verificar a integridade da base de dados:

```
USE master
GO

--Verifica a integridade da base de dados e mostra todas as mensagens de erro
DBCC CHECKDB('AgroTejo') WITH ALL_ERRORMSGS
```

Se a integridade da base de dados tiver de ser restaurada temos de ter em conta que a melhor opção é restaurar a base de dados a partir de um backup recente. No entanto se não for possível restaurar a base de dados devido ao seu tamanho, devemos considerar executar os comandos *REPAIR_ALLOW_DATA_LOSS*, *REPAIR_FAST* ou *REPAIR_REBUILD*, que devem ser o ultimo recurso, devido ao facto de estas operações não considerarem os constraints que possam existir nas tabelas ou entre as tabelas, resultando numa perda de dados. Por este motivo o restauro da integridade da base de dados não está incluido nos jobs utilizados no SQL Server Agent para a automatização da manutenção.

Para restaurar a integridade da base de dados:

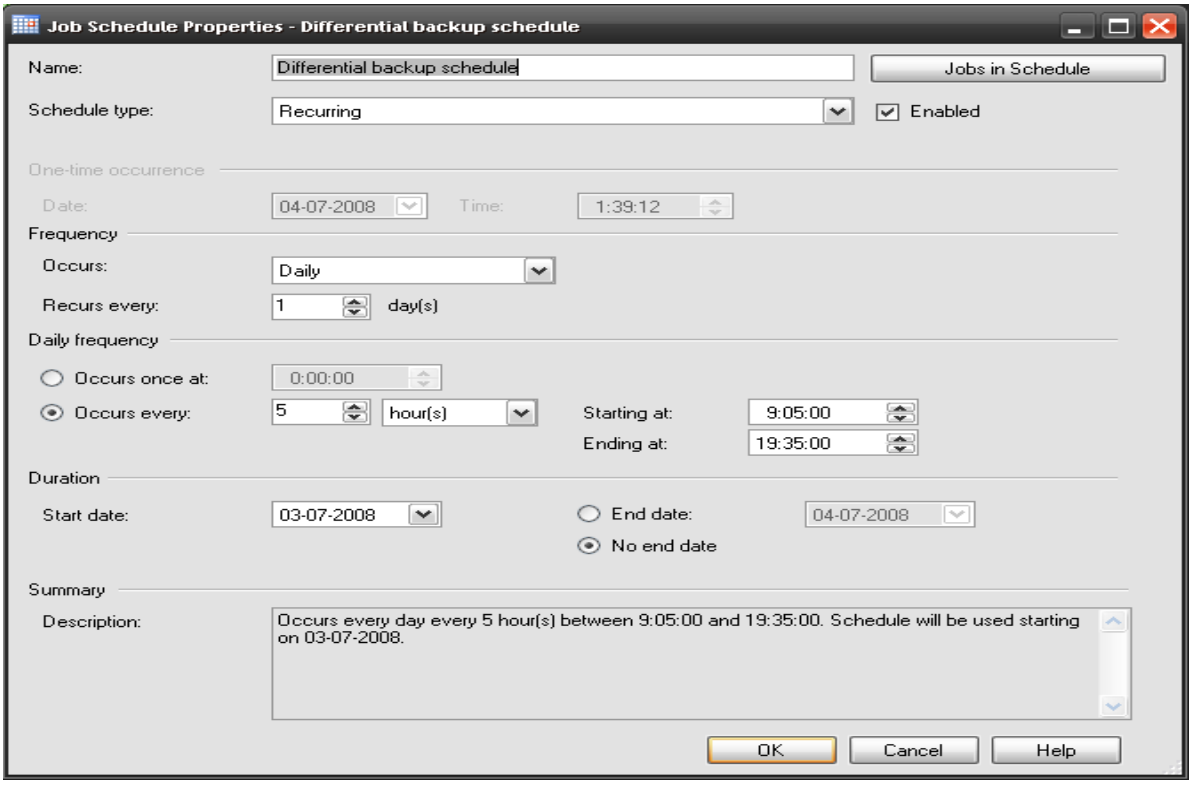
```
USE master
GO


/*Altera a base de dados para single-user, pois para usar uma das três opções
de reparação de DBCC CHECKDB, a base de dados tem que estar em modo single-
user.*/
ALTER DATABASE AgroTejo
SET SINGLE_USER;


/* REPAIR_ALLOW_DATA_LOSS, REPAIR_FAST, ou REPAIR_REBUILD, apenas devem ser
usados se não for possível reparar a base de dados por causa do seu tamanho,
porque estas operações de reparação não consideram os constraints que possam
existir nas tabelas.*/

--Permite reparar a base de dados com possível perda de dados.
DBCC CHECKDB('AgroTejo', 'REPAIR_ALLOW_DATA_LOSS') WITH ALL_ERRORMSGS;


ALTER DATABASE AgroTejo
SET MULTI_USER;
```

Politicas de automatização

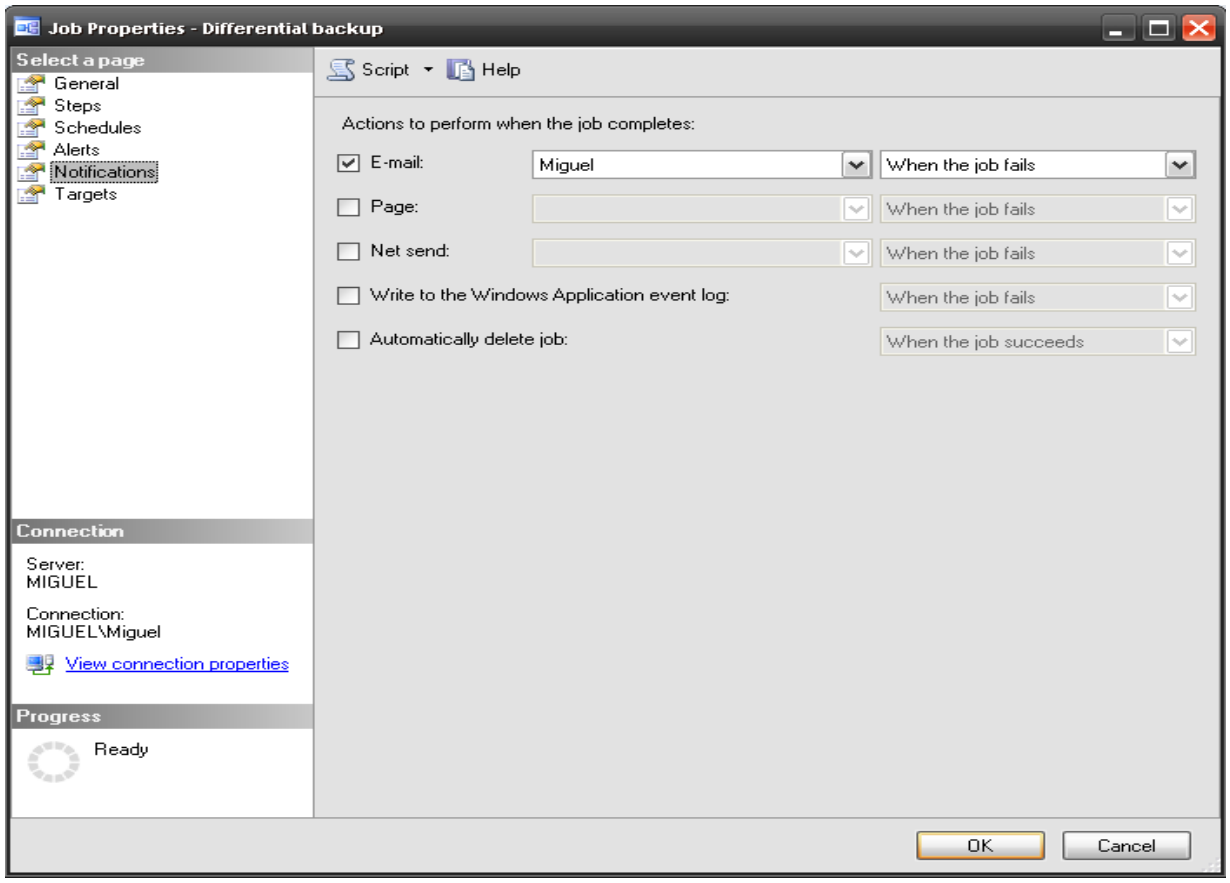
Na automatização da manutenção da base de dados, optámos por utilizar o SQL Server Agent, no qual criámos jobs, steps, schedules, notifications e alerts para full backups, differential backups, transaction log backups e verificação da integridade da base de dados. O full backup foi configurado para ser efectuado apenas uma vez e antes dos differential backups e transaction log backups que foram configurados para serem efectuados várias vezes ao dia. A verificação da integridade da base de dados foi configurada para ser efectuada apenas uma vez por dia. No caso destas operações falharem será enviado um e-mail para o operator dando essa informação. No caso da verificação da integridade e caso haja erros, será enviado um alert para o operator. Para enviar por e-mail as notificações e o alert criado para a verificação da integridade da base de dados, criámos um operator.

Pelo facto do servidor estar a correr como um serviço local, este não pode aceder a servidores SMTP remotos para enviar os e-mails com as notificações ou o alert. Assim sendo instalámos um servidor de e-mail local que é o Ability Mail Server 2 e testámos com êxito o envio destas notificações quando, por exemplo, falha um differential backup.

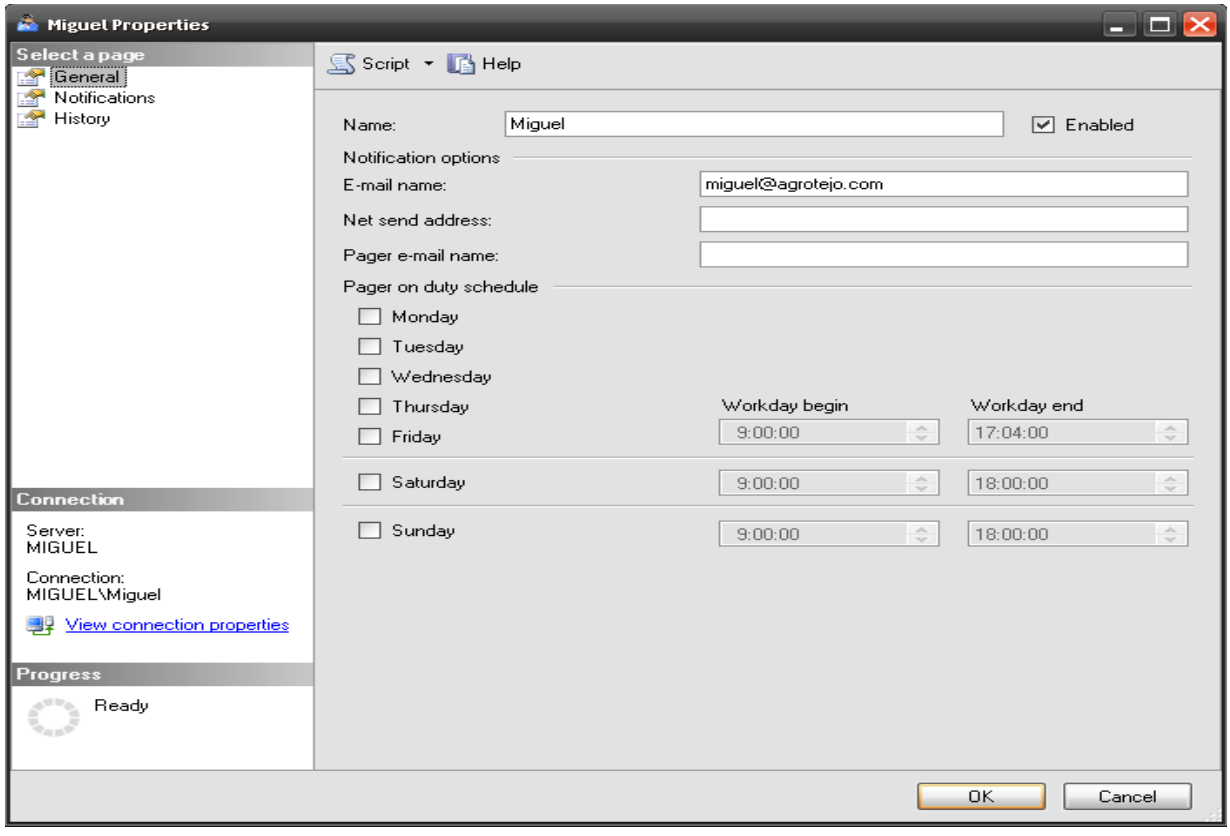
Schedule para o differential backup:



Notificação no caso do differential backup falhar:



Operator para onde são enviados os e-mails:



Conclusão

Pensamos ter atingido todos os objectivos deste trabalho, bem como ter adquirido os conhecimentos básicos na configuração do SQL Server 2005.

Nem todos os scripts utilizados na base de dados foram apresentados neste relatório.

Anexo

Stored Procedures

Compras:

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_eliminarAnimal', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_eliminarAnimal;
GO

/*Procedimento armazenado para eliminar o registo do animal que tenha o
numero_sia igual a @numeroSia e o codigo_compra igual a @codigoCompra,
respeitando a integridade referêncial*/
CREATE PROCEDURE dbo.Compras_eliminarAnimal
      @numeroSia smallint,
      @codigoCompra smallint
AS

IF (SELECT DISTINCT numero_sia FROM dbo.cuidado_saude_animal WHERE numero_sia =
@numeroSia)
      IS NULL AND (SELECT DISTINCT numero_sia FROM dbo.parto WHERE numero_sia =
@numeroSia)
      IS NULL
BEGIN
      DELETE FROM dbo.animal WHERE numero_sia = @numeroSia AND codigo_compra =
@codigoCompra
END
ELSE
PRINT 'Não é possível eliminar o registo por questões de integridade
referêncial!!!'
GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_eliminarAnimal @numeroSia = 1006, @codigoCompra = 96;
GO


            ********************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_inserirAnimal', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_inserirAnimal;
GO

/*Procedimento armazenado para inserir um registo de animal na condição de que
o
codigo_compra a introduzir no animal tem de existir na tabela dbo.compra*/
CREATE PROCEDURE dbo.Compras_inserirAnimal
      @numeroParticular int,
```

```
      @sexo varchar(5),
      @dataNascimento datetime,
      @codigoCompra smallint,
      @codigo_especie smallint

AS

IF (SELECT codigo_compra FROM dbo.compra WHERE codigo_compra = @codigoCompra)
IS NOT NULL

BEGIN
      INSERT INTO dbo.animal(numero_particular, sexo, data_nascimento,
                  codigo_compra, codigo_especie) VALUES(@numeroParticular,
@sexo, @dataNascimento, @codigoCompra, @codigo_especie)
END
ELSE
PRINT 'O codigo_compra que tentou introduzir no animal não existe na tabela
dbo.compra!!!'
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_inserirAnimal @numeroParticular = 276453653, @sexo = 'Macho',
                        @dataNascimento = '2002/04/27', @codigoCompra = 345,
@codigo_especie = 3;

GO


            ******************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateAnimal_codigoCompra', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateAnimal_codigoCompra;
GO

/*Procedimento armazenado para actualizar o campo codigo_compra do animal que
tenha o numero_sia igual a
@numeroSia e o codigo_compra igual a @codigoCompraActual*/
CREATE PROCEDURE dbo.Compras_updateAnimal_codigoCompra
      @numeroSia smallint,
      @codigoCompraActual smallint,
      @codigoCompra smallint

AS

UPDATE dbo.animal SET codigo_compra = @codigoCompra
      WHERE numero_sia = @numeroSia AND codigo_compra = @codigoCompraActual
GO


USE AgroTejo
GO
```

```
/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateAnimal_codigoCompra @numeroSia = 1000,
@codigoCompraActual = 292, @codigoCompra = 222;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateAnimal_codigoEspecie', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateAnimal_codigoEspecie;
GO

/*Procedimento armazenado para actualizar o campo codigo_especie do animal que
tenha o numero_sia igual a @numeroSia e o codigo_compra igual a @codigoCompra*/
CREATE PROCEDURE dbo.Compras_updateAnimal_codigoEspecie
      @numeroSia smallint,
      @codigoCompra smallint,
      @codigo_especie smallint

AS

UPDATE dbo.animal SET codigo_especie = @codigo_especie
      WHERE numero_sia = @numeroSia AND codigo_compra = @codigoCompra
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateAnimal_codigoEspecie @numeroSia = 1000, @codigoCompra =
292, @codigo_especie = 3;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateAnimal_dataNascimento', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateAnimal_dataNascimento;
GO

/*Procedimento armazenado para actualizar o campo data_nascimento do animal que
tenha o numero_sia igual a @numeroSia e o codigo_compra igual a @codigoCompra*/
CREATE PROCEDURE dbo.Compras_updateAnimal_dataNascimento
      @numeroSia smallint,
      @codigoCompra smallint,
      @dataNascimento datetime

AS

UPDATE dbo.animal SET data_nascimento = @dataNascimento
      WHERE numero_sia = @numeroSia AND codigo_compra = @codigoCompra
GO
```

```
USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateAnimal_dataNascimento @numeroSia = 1000, @codigoCompra =
292, @dataNascimento = '2009/03/25';
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateAnimal_numeroParicular', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateAnimal_numeroParicular;
GO

/*Procedimento armazenado para actualizar o campo numero_particular do animal
que tenha o numero_sia igual a @numeroSia e o codigo_compra igual a
@codigoCompra*/
CREATE PROCEDURE dbo.Compras_updateAnimal_numeroParicular
      @numeroSia smallint,
      @codigoCompra smallint,
      @numeroParticular int

AS

UPDATE dbo.animal SET numero_particular = @numeroParticular
      WHERE numero_sia = @numeroSia AND codigo_compra = @codigoCompra
GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateAnimal_numeroParicular @numeroSia = 1000,
@codigoCompra = 292, @numeroParticular = 9999999;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateAnimal_sexo', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateAnimal_sexo;
GO

/*Procedimento armazenado para actualizar o campo sexo do animal que tenha o
numero_sia igual a @numeroSia e o codigo_compra igual a @codigoCompra*/
CREATE PROCEDURE dbo.Compras_updateAnimal_sexo
      @numeroSia smallint,
      @codigoCompra smallint,
      @sexo varchar(5)


AS
```

```
UPDATE dbo.animal SET sexo = @sexo  WHERE numero_sia = @numeroSia AND
codigo_compra = @codigoCompra
GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateAnimal_sexo @numeroSia = 1000, @codigoCompra = 292,
@sexo = Macho;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Compras_updateCompra_valorCompra', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Compras_updateCompra_valorCompra;
GO

/*Procedimento armazenado para actualizar o campo valor_da_compra da compra que
tenha o codigo_compra igual a @codigoCompra*/
CREATE PROCEDURE dbo.Compras_updateCompra_valorCompra
      @codigoCompra smallint,
      @valorCompra money

AS

UPDATE dbo.compra SET valor_da_compra = @valorCompra WHERE codigo_compra =
@codigoCompra
GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Compras_updateCompra_valorCompra @codigoCompra = 6,
@valorCompra = 222;
GO
```

****

Encomendas:

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.clientesPagamentoEmAtraso', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.clientesPagamentoEmAtraso;
GO
```

```
/*Procedimento armazenado para devolver os clientes que efectuaram uma
encomenda há mais de um mês e ainda não pagaram a factura*/
CREATE PROCEDURE dbo.clientesPagamentoEmAtraso

AS

SELECT C.codigo_cliente, C.nome, C.telefone
FROM cliente C
WHERE C.codigo_cliente IN (SELECT E.codigo_cliente
                                    FROM encomenda E
                                    WHERE E.codigo_encomenda NOT IN
                               (SELECT F.codigo_encomenda  FROM factura F )AND
DATEDIFF( month , E.data_encomenda, GETDATE() ) > 1)

GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.clientesPagamentoEmAtraso;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Encomendas_updateAnimal_codigoEncomenda', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Encomendas_updateAnimal_codigoEncomenda;
GO

/*Procedimento armazenado para actualizar o campo codigo_encomenda do animal
que tenha o numero_sia igual a @numeroSia e o codigo_encomenda igual a
@codigoEncomendaActual*/
CREATE PROCEDURE dbo.Encomendas_updateAnimal_codigoEncomenda
      @numeroSia smallint,
      @codigoEncomendaActual smallint,
      @codigoEncomenda smallint

AS

UPDATE dbo.animal SET codigo_encomenda = @codigoEncomenda
      WHERE numero_sia = @numeroSia AND codigo_encomenda =
@codigoEncomendaActual
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Encomendas_updateAnimal_codigoEncomenda @numeroSia = 5,
@codigoEncomendaActual = 391, @codigoEncomenda = 444;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Encomendas_updateValorFactura', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Encomendas_updateValorFactura;
GO

/*Procedimento armazenado para actualizar o campo valor_da_factura da factura
que tenha o codigo_encomenda igual a  @codigoEncomenda*/
CREATE PROCEDURE dbo.Encomendas_updateValorFactura
      @codigoEncomenda smallint,
      @valorFactura smallint

AS

UPDATE dbo.factura SET valor_da_factura = @valorFactura
      WHERE codigo_encomenda = @codigoEncomenda
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Encomendas_updateValorFactura @codigoEncomenda = 15,
@valorFactura = 156;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.verificaAnimaisDisponiveisParaVenda', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.verificaAnimaisDisponiveisParaVenda;
GO

/*Procedimento armazenado para actualizar o campo codigo_compra do animal que
tenha o numero_sia igual a @numeroSia e o codigo_compra igual a
@codigoCompraActual*/
CREATE PROCEDURE dbo.verificaAnimaisDisponiveisParaVenda
      @nomeEspecie varchar(15)

AS

SELECT A.numero_sia, A.sexo, A.data_nascimento,
DATEDIFF( month , A.data_nascimento, GETDATE() )  as NumeroDeMeses
FROM animal A WHERE DATEDIFF( month , A.data_nascimento, GETDATE() ) > 6
AND A.codigo_especie IN (SELECT E.codigo_especie FROM especie E
                                    WHERE E.nome_especie LIKE @nomeEspecie)
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.verificaAnimaisDisponiveisParaVenda @nomeEspecie = 'Cavalo';
GO
```

****

Reprodução:

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.dataUltimoPartoDaFemea', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.dataUltimoPartoDaFemea;
GO

/*Procedimento armazenado para devolver a data do último parto da femea*/
CREATE PROCEDURE dbo.dataUltimoPartoDaFemea
            @numeroSia smallint
AS

SELECT P.numero_sia, MAX(P.data_parto) AS data_ultimo_parto
FROM parto P
WHERE P.numero_sia = @numeroSia
GROUP BY P.numero_sia

GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.dataUltimoPartoDaFemea @numeroSia = 2;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.femeasSemPartoHaMaisDeUmAno', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.femeasSemPartoHaMaisDeUmAno;
GO

/*Procedimento armazenado para devolver os numero_sia das femeas que não têm um
parto há mais de um ano*/
CREATE PROCEDURE dbo.femeasSemPartoHaMaisDeUmAno

AS

SELECT A.numero_sia FROM animal A
WHERE A.numero_sia IN (SELECT P.numero_sia FROM parto P
                       WHERE DATEDIFF( year , P.data_parto, GETDATE() ) > 1)

GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.femeasSemPartoHaMaisDeUmAno;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_eliminarAnimal', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Reproducao_eliminarAnimal;
GO

/*Procedimento armazenado para eliminar o registo do animal que tenha o
numero_sia igual a @numeroSia, o numero_sia_progenitor_femea diferente de null,
respeitando a integridade referêncial*/
CREATE PROCEDURE dbo.Reproducao_eliminarAnimal
      @numeroSia smallint

AS

IF (SELECT DISTINCT numero_sia FROM dbo.cuidado_saude_animal WHERE numero_sia =
@numeroSia)
      IS NOT NULL OR (SELECT DISTINCT numero_sia FROM dbo.parto
WHERE numero_sia = @numeroSia) IS NOT NULL

      BEGIN
            PRINT 'Não é possível eliminar o registo por questões de
integridade referêncial!!!'
      END
ELSE
      IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NULL

            BEGIN
                  PRINT 'Não tem permissão para actualizar o registo, pois não
foi introduzido pelo departamento Reproducao'
            END
      ELSE
            DELETE FROM dbo.animal WHERE numero_sia = @numeroSia
GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_eliminarAnimal @numeroSia = 798;
GO


            ******************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_inserirAnimal', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Reproducao_inserirAnimal;
GO

/*Procedimento armazenado para inserir um registo de animal na condição de que
o codigo_compra a introduzir no animal tem de existir na tabela dbo.compra*/
CREATE PROCEDURE dbo.Reproducao_inserirAnimal
      @numeroParticular int,
      @sexo varchar(5),
      @dataNascimento datetime,
      @numeroSiaProgenitorMacho smallint,
      @numeroSiaProgenitorFemea smallint,
```

```
      @codigoEspecie smallint

AS

IF (SELECT numero_sia FROM dbo.animal WHERE numero_sia =
@numeroSiaProgenitorMacho AND sexo = 'Macho' AND codigo_especie =
@codigoEspecie) IS NULL

      BEGIN
            PRINT 'O numero_sia_progenitor_macho que tentou inserir no animal,
não corresponde a nenhum numero_sia de um animal macho da mesma especie na
tabela dbo.animal!!!'

      END
ELSE
      IF (SELECT numero_sia FROM dbo.animal WHERE numero_sia =
@numeroSiaProgenitorFemea AND sexo = 'Femea') IS NULL

            BEGIN
                  PRINT 'O numero_sia_progenitor_femea que tentou inserir no
animal, não corresponde a nenhum numero_sia de um animal femea da mesma especie
na tabela dbo.animal!!!'

            END
      ELSE

      INSERT INTO dbo.animal(numero_particular, sexo, data_nascimento,
                        numero_sia_progenitor_macho,
numero_sia_progenitor_femea, codigo_especie)
                        VALUES(@numeroParticular, @sexo, @dataNascimento,
                              @numeroSiaProgenitorMacho,
@numeroSiaProgenitorFemea, @codigoEspecie)
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_inserirAnimal @numeroParticular = 111111111, @sexo =
'Macho', @dataNascimento = '2002/04/27', @numeroSiaProgenitorMacho = 9,
@numeroSiaProgenitorFemea = 21, @codigoEspecie = 5;

GO

            ********************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_codigoEspecie', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_codigoEspecie;
GO

/*Procedimento armazenado para actualizar o campo codigo_especie do animal que
tenha o numero_sia igual a @numeroSia e o numero_sia_progenitor_femea diferente
de null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_codigoEspecie
      @numeroSia smallint,
      @codigo_especie smallint

AS
```

```
IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL

BEGIN
      UPDATE dbo.animal SET codigo_especie = @codigo_especie WHERE numero_sia =
@numeroSia

END

ELSE
      PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'

GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_codigoEspecie @numeroSia = 1000,
@codigo_especie = 3;
GO

            ******************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_dataNascimento', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_dataNascimento;
GO

/*Procedimento armazenado para actualizar o campo data_nascimento do animal que
tenha o numero_sia igual a @numeroSia e o numero_sia_progenitor_femea diferente
de null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_dataNascimento
      @numeroSia smallint,
      @dataNascimento datetime

AS

IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL
BEGIN
      UPDATE dbo.animal SET data_nascimento = @dataNascimento
WHERE numero_sia = @numeroSia
END
ELSE
      PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_dataNascimento @numeroSia = 1000,
      @dataNascimento = '2008/08/27';
GO

            ******************************************************
```

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_numeroParticular', 'P' ) IS NOT
NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_numeroParticular;
GO

/*Procedimento armazenado para actualizar o campo numero_particular do animal
que tenha o numero_sia igual a @numeroSia e o numero_sia_progenitor_femea
diferente de null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_numeroParticular
      @numeroSia smallint,
      @numeroParticular int

AS

IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL
BEGIN
      UPDATE dbo.animal SET numero_particular = @numeroParticular WHERE
numero_sia = @numeroSia
END
ELSE
      PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_numeroParticular @numeroSia = 1000,
      @numeroParticular = 378234463;
GO


            **********************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_numeroSiaProgenitorFemea', 'P' ) IS
NOT NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_numeroSiaProgenitorFemea;
GO

/*Procedimento armazenado para actualizar o campo numero_sia_progenitor_femea
do animal que tenha o numero_sia igual a @numeroSia e o
numero_sia_progenitor_femea diferente de null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_numeroSiaProgenitorFemea
      @numeroSia smallint,
      @numeroSiaProgenitorFemea smallint

AS

IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL
BEGIN
      UPDATE dbo.animal SET numero_sia_progenitor_femea =
@numeroSiaProgenitorFemea  WHERE numero_sia = @numeroSia
```

```
END
ELSE
	PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'

GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_numeroSiaProgenitorFemea @numeroSia = 1000,
	@numeroSiaProgenitorFemea = 4489;
GO


		**************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_numeroSiaProgenitorMacho', 'P' ) IS
NOT NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_numeroSiaProgenitorMacho;
GO

/*Procedimento armazenado para actualizar o campo numero_sia_progenitor_macho
do animal que tenha o numero_sia igual a @numeroSia e o
numero_sia_progenitor_femea diferente de null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_numeroSiaProgenitorMacho
	@numeroSia smallint,
	@numeroSiaProgenitorMacho smallint

AS

IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL
BEGIN
	UPDATE dbo.animal SET numero_sia_progenitor_macho =
@numeroSiaProgenitorMacho WHERE numero_sia = @numeroSia
END
ELSE
	PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'

GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_numeroSiaProgenitorMacho @numeroSia = 1000,
		@numeroSiaProgenitorMacho = 4567;
GO


		******************************************************
```

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.Reproducao_updateAnimal_sexo', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.Reproducao_updateAnimal_sexo;
GO

/*Procedimento armazenado para actualizar o campo sexo do animal que tenha o
numero_sia igual a @numeroSia e o numero_sia_progenitor_femea diferente de
null*/
CREATE PROCEDURE dbo.Reproducao_updateAnimal_sexo
      @numeroSia smallint,
      @sexo varchar(5)

AS

IF (SELECT numero_sia_progenitor_femea FROM dbo.animal WHERE numero_sia =
@numeroSia) IS NOT NULL
BEGIN
      UPDATE dbo.animal SET sexo = @sexo WHERE numero_sia = @numeroSia
END
ELSE
      PRINT 'Não tem permissão para actualizar o registo, pois não foi
introduzido pelo departamento Reproducao'

GO

USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.Reproducao_updateAnimal_sexo @numeroSia = 3, @sexo = 'Macho';
GO
```

****

Saude:

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.cuidadosSaudeEfectuadosPorCadaVeterinario', 'P' ) IS NOT
NULL
    DROP PROCEDURE dbo.cuidadosSaudeEfectuadosPorCadaVeterinario;
GO

/*Procedimento armazenado para devolver o numero de cuidados de saude
efectuados por cada veterinario*/
CREATE PROCEDURE dbo.cuidadosSaudeEfectuadosPorCadaVeterinario

AS

SELECT V.codigo_veterinario, V.nome, COUNT(*) AS numeroDeCuidadosSaude
FROM veterinario V, cuidado_saude C
WHERE V.codigo_veterinario = C.codigo_veterinario
GROUP BY V.codigo_veterinario, V.nome
```

```
GO
USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.cuidadosSaudeEfectuadosPorCadaVeterinario;
GO


            ******************************************************


USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.verificaAnimaisSemCuidadosSaudeUltimos2Meses', 'P' ) IS NOT
NULL
    DROP PROCEDURE dbo.verificaAnimaisSemCuidadosSaudeUltimos2Meses;
GO

/*Procedimento armazenado para devolver os numeros_sia dos animais que não
recebem um cuidado de saude há mais de dois meses*/
CREATE PROCEDURE dbo.verificaAnimaisSemCuidadosSaudeUltimos2Meses

AS

SELECT A.numero_sia
FROM animal A
WHERE A.numero_sia IN (SELECT CS.numero_sia
                                FROM cuidado_saude_animal CS
                                WHERE CS.cod_cuidado_saude IN
                (SELECT cod_cuidado_saude FROM cuidado_saude C
            WHERE DATEDIFF( month , C.data_cuidado_saude, GETDATE()) > 2 ))
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.verificaAnimaisSemCuidadosSaudeUltimos2Meses;
GO


            ******************************************************
```

Administrador:

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.verCompraAudit', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.verCompraAudit;
GO

/*Procedimento armazenado para devolver os campos desencriptados da tabela
compras_audit*/
CREATE PROCEDURE dbo.verCompraAudit

AS
```

```
OPEN SYMMETRIC KEY ComprasKey
DECRYPTION BY CERTIFICATE CompraCert;


SELECT codigo_audit, CONVERT(smallint, DecryptByKey(codigo_compra)) AS
codigo_compra,
CONVERT(money, DecryptByKey(valor_da_compra_antes)) AS valor_da_compra_antes,
CONVERT(money, DecryptByKey(valor_da_compra_depois)) AS valor_da_compra_depois,
CONVERT(datetime, DecryptByKey(data_audit)) AS data_audit,
CONVERT(sysname, DecryptByKey(utilizador_alterou)) AS utilizador_alterou
FROM dbo.compra_audit
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.verCompraAudit;
GO
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ( 'dbo.verFacturaAudit', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.verFacturaAudit;
GO

/*Procedimento armazenado para devolver os campos desencriptados da tabela
factura_audit*/
CREATE PROCEDURE dbo.verFacturaAudit

AS

OPEN SYMMETRIC KEY EncomendasKey
DECRYPTION BY CERTIFICATE EncomendaCert;


SELECT codigo_audit, CONVERT(smallint, DecryptByKey(codigo_encomenda)) AS
codigo_encomenda,
CONVERT(money, DecryptByKey(valor_da_factura_antes)) AS valor_da_factura_antes,
CONVERT(money, DecryptByKey(valor_da_factura_depois)) AS
valor_da_factura_depois,
CONVERT(datetime, DecryptByKey(data_audit)) AS data_audit,
CONVERT(sysname, DecryptByKey(utilizador_alterou)) AS utilizador_alterou
FROM dbo.factura_audit
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
EXEC dbo.verFacturaAudit;
GO
```

****

Triggers

Compras:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.compraAudit', 'TR') IS NOT NULL
   DROP TRIGGER dbo.compraAudit
GO


/*Trigger que é executado após um update à tabela dbo.compra e que insere na
tabela dbo.compra_audit o codigo_compra, o valor_da_compra antes de ser
alterado, o valor_da_compra depois de ser alterado e o nome do login do
utilizador que efectuou o update. Todos estes valores são introduzidos
encriptados*/
CREATE TRIGGER dbo.compraAudit
ON dbo.compra
AFTER UPDATE
AS

OPEN SYMMETRIC KEY ComprasKey
DECRYPTION BY CERTIFICATE CompraCert;


INSERT INTO dbo.compra_audit
(codigo_compra, valor_da_compra_antes, valor_da_compra_depois, data_audit,
utilizador_alterou)
SELECT EncryptByKey(Key_GUID('ComprasKey'), CONVERT(varbinary,
INSERTED.codigo_compra)),
       EncryptByKey(Key_GUID('ComprasKey'), CONVERT(varbinary,
DELETED.valor_da_compra)),
       EncryptByKey(Key_GUID('ComprasKey'), CONVERT(varbinary,
INSERTED.valor_da_compra)),
       EncryptByKey(Key_GUID('ComprasKey'), CONVERT(varbinary, getdate())),
       EncryptByKey(Key_GUID('ComprasKey'), suser_sname())
FROM INSERTED INNER JOIN DELETED ON INSERTED.codigo_compra =
DELETED.codigo_compra;


          ******************************************************
```

Encomendas:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.facturaAudit', 'TR') IS NOT NULL
   DROP TRIGGER dbo.facturaAudit
GO

/*Trigger que é executado após um update à tabela dbo.factura e que insere na
tabela dbo.factura_audit o codigo_encomenda, o valor_factura antes de ser
```

```
alterado, o valor_factura depois de ser alterado e o nome do login do
utilizador que efectuou o update*/
CREATE TRIGGER dbo.facturaAudit
ON dbo.factura
AFTER UPDATE
AS

OPEN SYMMETRIC KEY EncomendasKey
DECRYPTION BY CERTIFICATE EncomendaCert;

INSERT INTO dbo.factura_audit
(codigo_encomenda, valor_da_factura_antes, valor_da_factura_depois, data_audit,
utilizador_alterou)
SELECT EncryptByKey(Key_GUID('EncomendasKey'),
CONVERT(varbinary,INSERTED.codigo_encomenda)),
       EncryptByKey(Key_GUID('EncomendasKey'), CONVERT(varbinary,
DELETED.valor_da_factura)),
         EncryptByKey(Key_GUID('EncomendasKey'), CONVERT(varbinary,
INSERTED.valor_da_factura)),
         EncryptByKey(Key_GUID('EncomendasKey'), CONVERT(varbinary,
getdate())),
         EncryptByKey(Key_GUID('EncomendasKey'), suser_sname())
FROM INSERTED INNER JOIN DELETED ON INSERTED.codigo_encomenda =
DELETED.codigo_encomenda;
```

****

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.verificaDescontoParaCliente', 'TR') IS NOT NULL
   DROP TRIGGER dbo.verificaDescontoParaCliente
GO

/*Trigger que é executado quando é feito um insert na tabela dbo.encomenda, e
que devolve uma mensagem a indicar que o cliente tem direito a um desconto no
caso deste ter efectuado mais de 3 encomenda nos últimos dois meses*/
CREATE TRIGGER dbo.verificaDescontoParaCliente
ON dbo.encomenda
FOR INSERT
AS

DECLARE @numeroDeEncomendasUltimosDoisMeses smallint;

SET @numeroDeEncomendasUltimosDoisMeses = (SELECT COUNT(E.codigo_cliente) FROM
encomenda E, INSERTED WHERE E.codigo_cliente = INSERTED.codigo_cliente
                    AND DATEDIFF( month , E.data_encomenda, GETDATE() ) < 2)


IF(@numeroDeEncomendasUltimosDoisMeses >3)

BEGIN
PRINT 'O cliente para o qual inserio uma encomenda tem direito a um
desconto!!!'
END
```

****

Reprodução:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.femeaSemCuidadoSaudeComParto', 'TR') IS NOT NULL
   DROP TRIGGER dbo.femeaSemCuidadoSaudeComParto
GO

/*Trigger que é executado quando é feito um insert na tabela dbo.parto, e a
femea desse parto não tem cuidados de saude há mais de seis meses*/
CREATE TRIGGER dbo.femeaSemCuidadoSaudeComParto
ON dbo.parto
FOR INSERT
AS

DECLARE @dataMaisRecente DATETIME;
DECLARE @data_maisRecente varchar(20);

SET @dataMaisRecente = (SELECT MAX(CS.data_cuidado_saude)
                  FROM cuidado_saude CS LEFT JOIN cuidado_saude_animal CSA
                  ON CS.cod_cuidado_saude = CSA.cod_cuidado_saude, INSERTED
                              WHERE CSA.numero_sia = INSERTED.numero_sia)

IF(DATEDIFF( month , @dataMaisRecente, GETDATE() )) > 6

BEGIN

SET @data_maisRecente = convert(varchar(20), @dataMaisRecente);

PRINT 'A femea do parto inserido não tem cuidados de saúde há mais de seis
meses!!!'

PRINT 'O último cuidado de saude foi efectuado em ' + @data_maisRecente

END
```

****

Functions

Compras:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.numeroBensAgricolasMuitoMauEstado', 'FN') IS NOT NULL
    DROP FUNCTION dbo.numeroBensAgricolasMuitoMauEstado;
GO


/*Criar função que devolve o numero de bens agricolas em muito mau estado*/
CREATE FUNCTION dbo.numeroBensAgricolasMuitoMauEstado()
RETURNS INT
AS
```

```
BEGIN

DECLARE @numeroBensAgricolasMuitoMauEstado INT;

SET @numeroBensAgricolasMuitoMauEstado = (SELECT count(B.codigo_bem_agricola)
                                          FROM bem_agricola B
                                WHERE B.estado_conservacao LIKE 'Muito mau')

RETURN @numeroBensAgricolasMuitoMauEstado;

END;

GO



/*Chamar função*/
SELECT dbo.numeroBensAgricolasMuitoMauEstado() AS
numeroBensAgricolasMuitoMauEstado;



            ********************************************************



USE AgroTejo
GO

IF OBJECT_ID ('dbo.numeroAnimaisComprados', 'FN') IS NOT NULL
    DROP FUNCTION dbo.numeroAnimaisComprados;
GO


/*Criar função que devolve o numero de animais que foram comprados a
fornecedores*/
CREATE FUNCTION dbo.numeroAnimaisComprados()
RETURNS INT
AS

BEGIN

DECLARE @numeroAnimaisComprados INT;

SET @numeroAnimaisComprados = (SELECT count(A.numero_sia)
                                          FROM animal A
                                          WHERE A.codigo_compra IS NOT NULL)

RETURN @numeroAnimaisComprados;

END;

GO

/*Chamar função*/
SELECT dbo.numeroAnimaisComprados() AS NumeroDeAnimaisComprados;



            ********************************************************
```

Encomendas:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.animaisDisponiveisParaVenda', 'IF') IS NOT NULL
    DROP FUNCTION dbo.animaisDisponiveisParaVenda;
GO


/*Criar função que devolve o numero_sia, o sexo, data_nascimento e a idade em
meses dos animais que têm mais de seis meses de idade e estão disponiveis para
venda*/
CREATE FUNCTION dbo.animaisDisponiveisParaVenda(@nomeEspecie varchar(15))
RETURNS TABLE
AS

RETURN(
SELECT A.numero_sia, A.sexo, A.data_nascimento, DATEDIFF( month ,
A.data_nascimento, GETDATE() ) AS NumeroDeMeses
FROM animal A WHERE DATEDIFF( month , A.data_nascimento, GETDATE() ) > 6
AND A.codigo_especie IN (SELECT E.codigo_especie
                  FROM especie E WHERE E.nome_especie LIKE @nomeEspecie))
GO

/*Chamar função*/
SELECT * FROM dbo.animaisDisponiveisParaVenda('Bovino');



          ******************************************************


USE AgroTejo
GO

IF OBJECT_ID ('dbo.avaliacaoCliente', 'FN') IS NOT NULL
    DROP FUNCTION dbo.avaliacaoCliente;
GO


/*Procedimento para avaliar os clientes no que diz respeito ao numero de
encomendas nos últimos quatro meses*/
CREATE FUNCTION dbo.avaliacaoCliente(@codigoCliente smallint)
RETURNS SMALLINT

AS

BEGIN

DECLARE @avaliacao smallint;
DECLARE @numeroDeEcomendas smallint;

SET @numeroDeEcomendas = (SELECT COUNT(*) FROM encomenda E
WHERE E.codigo_cliente = @codigoCliente AND
DATEDIFF( month , e.data_encomenda, GETDATE() ) < 4)

IF(@numeroDeEcomendas > 15)
      SET @avaliacao = 3;
ELSE IF(@numeroDeEcomendas > 7)
      SET @avaliacao = 2;
```

```
ELSE
SET @avaliacao = 1;

RETURN @avaliacao;

END;
GO


USE AgroTejo
GO

/*Executa o procedimento armazenado*/
SELECT dbo.avaliacaoCliente(24) AS avaliacao;
GO
```

****

Reprodução:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.numeroFemeasMaisDeUmAnoDeIdadeSemParto', 'FN') IS NOT NULL
    DROP FUNCTION dbo.numeroFemeasMaisDeUmAnoDeIdadeSemParto;
GO


/*Criar função que devolve o numero de femeas com mais de um ano e que ainda
não tiveram partos*/
CREATE FUNCTION dbo.numeroFemeasMaisDeUmAnoDeIdadeSemParto()
RETURNS INT
AS

BEGIN

DECLARE @numeroFemeasMaisDeUmAnoDeIdadeSemParto INT;

SET @numeroFemeasMaisDeUmAnoDeIdadeSemParto = (SELECT count(A.numero_sia)
                  FROM animal A WHERE A.sexo LIKE 'Femea'
                       AND DATEDIFF( YEAR , A.data_nascimento, GETDATE() ) > 1
AND A.numero_sia NOT IN (SELECT P.numero_sia FROM parto P))

RETURN @numeroFemeasMaisDeUmAnoDeIdadeSemParto;

END;

GO

/*Chamar função*/
SELECT dbo.numeroFemeasMaisDeUmAnoDeIdadeSemParto() AS
numeroFemeasMaisDeUmAnoDeIdadeSemParto;
```

****

Saude:

```
USE AgroTejo
GO

IF OBJECT_ID ('dbo.mediaCuidadosSaudeVeterinarioPorMes', 'FN') IS NOT NULL
    DROP FUNCTION dbo.mediaCuidadosSaudeVeterinarioPorMes;
GO


/*Criar função que devolve o numero médio de cuidados de saude efectuados por
mês por um determinado veterinario num determinado ano*/
CREATE FUNCTION dbo.mediaCuidadosSaudeVeterinarioPorMes(@codigoVeterinario
smallint, @ano smallint)
RETURNS INT
AS

BEGIN

DECLARE @numeroCuidadosSaudeVeterinarioNumAno INT;

DECLARE @mediaCuidadosSaudeVeterinarioPorMes INT;


SET @numeroCuidadosSaudeVeterinarioNumAno = (SELECT count(C.cod_cuidado_saude)
                  FROM cuidado_saude C WHERE YEAR(C.data_cuidado_saude) = @ano
                                AND C.codigo_veterinario = @codigoVeterinario)

SET @mediaCuidadosSaudeVeterinarioPorMes =
@numeroCuidadosSaudeVeterinarioNumAno / 12;

RETURN @mediaCuidadosSaudeVeterinarioPorMes;

END;

GO

/*Chamar função*/
SELECT dbo.mediaCuidadosSaudeVeterinarioPorMes(477, 2007) AS
mediaCuidadosSaudeVeterinarioPorMes;
```

****