

|  |
| --- |
| Complementos de Bases de Dados 2022/2023  Licenciatura em Engª. Informática |
| Relatório Técnico |

Turma: 2ªL\_EI-SW-08

Horário de Laboratório: 14:30h – 16:30h

Docente: Gabriel Pestana

Grupo

Nº202000753, Nuno Reis

# Introdução

Este relatório tem como objetivo documentar e apresentar algumas justificações para as principais decisões tomadas na primeira fase de desenvolvimento do projeto de CBD, projeto esse que visa a familiarização com a administração de bases de dados relacionais. Ao realizar este projeto será consolidada e posta em prática toda a matéria teórica.

A Wide World Importers (WWI), é uma empresa importadora e distribuidora de produtos que opera no mercado de vendas a retalho. O seu sistema de informação está baseado numa base de dados não normalizada e está desatualizado. A administração tomou a decisão de reformular o sistema, num novo ERP que lhe permita gerir, de forma integrada, todo o processo de vendas.

O projeto irá incidir na modelação e integração de uma nova base de dados, que dará apoio a esse novo sistema de informação. Foram exportados e disponibilizados fragmentos de informação do sistema existente, estes apresentam-se fracamente relacionados e carecem de uma otimização segundo as boas praticas de modelação e regras da normalização.

# Especificação de Requisitos

|  |  |  |
| --- | --- | --- |
| ID | Descrição | Implementado  (S/N) |
| RF01 | O sistema não deverá permitir importar o mesmo registo duas vezes. | S |
| RF02 | O sistema deverá guardar o registo de tokens gerados (tokens gerados ao tentar recuperar password). | S |
| RF03 | O sistema deverá verificar se um token está ativo (tokens têm validade de 24 horas) | S |
| RF04 | O sistema não deverá permitir que numa venda existam produtos que necessitem de refrigeração e produtos que não necessitem. | S |
| RF05 | O sistema não deverá permitir a atribuição de uma promoção que não esteja ativa. | S |
| RF06 | O sistema deverá eliminar uma venda se forem removidos todos os produtos. | S |
| RM01 | O sistema deverá guardar a password do utilizador usando codificação uma em SHA1. | S |
| RM02 | O sistema deverá alterar a validade de um token quando um utilizador criar um segundo token. De forma a existir apenas um token valido para cada utilizador. | S |
| RM03 | O sistema deverá calcular a data prevista de entrega e os valores associados a uma venda, assim que esta é finalizada. | S |
| RM04 | O sistema deverá alterar o stock de um produto quando este é adicionado ou removido de uma venda. | S |
| RM05 | O sistema não deverá permitir que existam dois utilizadores com o mesmo email. | S |
| RM06 | O sistema não deverá permitir adicionar um produto a uma venda acabada. | S |

# Alterações/Melhorias ao Relatório da 1ª Fase

Nesta segunda fase passaram a ser utilizadas as convenções de escrita adequadas para o Microsoft SQL Server.

## Modelo Relacional

A tabela RH.ErrorLog deixou de estar ligada á tabela RH.SysUser.

## Layout

Os filegroups e schemas foram alterados:

* Filegroups para tabelas em que é mais frequente a escrita ou a leitura de registos (Read para as de leitura e Write para as de escrita).
* Schemas para as tabelas referentes a dados de utilizadores (RH), do armazém (Storage) e das vendas (Sales).

## Programação

* Funções e Stored Procedures:

Os erros, em todos os stored procedures e funções, passaram a ser lançados utilizando o RaiseError.

* Geradores:

Foram nesta fase adicionados todos os stored procedures geradores.

# Relacional (*Modelo de dados*)

## Diagrama do Modelo Relacional

Uma imagem com texto, interior, captura de ecrã

Descrição gerada automaticamente

# Definição do Layout

## Identificação do espaço ocupado por tabela

|  |  |  |
| --- | --- | --- |
| Nome Tabela | Dimensão do Registo | Nº de Registos (inicial/final) |
| RH.Country | 44 | 0 – 1 |
| RH.City | 108 | 116294 - 23272 |
| RH.StateProvince | 59 | 57 - 60 |
| RH.Category | 24 | 5 - 5 |
| RH.BuyingGroup | 24 | 0 - 2 |
| RH.SysUser | 124 | 0 - 421 |
| RH.Region\_Category | 24 | 0 - 401 |
| RH.Customer | 56 | 402 - 402 |
| RH.Employee | 15 | 212 - 20 |
| Storage.Package | 29 | 0 - 5 |
| Storage.Brand | 29 | 0 - 2 |
| Storage.ProductType | 29 | 0 - 2 |
| Storage.TaxRate | 12 | 0 - 6 |
| Storage.Product | 250 | 671 - 227 |
| Storage.Promotion | 110 | 0 - 1 |
| Storage.Product\_Promotion | 20 | 0 - 227 |
| Sales.Sale | 155 | 228265 - 70510 |
| Sales.ProductPromotion\_Sale | 12 | 0 - 228265 |
| RH.ErrorLog | 312 | 0 - |
| RH.Token | 28 | 0 - |

## Especificação dos Filegroups

|  |  |  |
| --- | --- | --- |
| Nome Filegroup | Tabelas associadas | Parâmetros |
| Read | RH.Country  RH.City  RH.StateProvince  RH.Category  RH.BuyingGroup  RH.Region\_Category  Storage.Package  Storage.Brand  Storage.ProductType  Storage.TaxRate | Dimensão inicial: 2MB  Dimensão final: 4MB  Taxa de crescimento: 100% |
| Write | RH.SysUser  RH.Customer  RH.Employee  Storage.Product  Storage.Promotion  Storage.Product\_Promotion  Sales.Sale  Sales.ProductPromotion\_Sale  RH.ErrorLog  RH.Token | Dimensão inicial: 15MB  Dimensão final: 30MB  Taxa de crescimento: 100% |

## Schemas

|  |  |
| --- | --- |
| Nome | Descrição |
| RH | Este schema tem como objetivo o agrupamento das tabelas que contêm dados relacionados com os utilizadores. |
| Storage | Este schema tem como objetivo o agrupamento das tabelas que contêm dados relacionados com os produtos e promoções. |
| Sales | Este schema tem como objetivo o agrupamento das tabelas que contêm dados relacionados com as vendas. |

# Verificação da migração de dados

## Consultas sobre a base de dados original

No final do ficheiro *WWWI\_DS Query.sql* estão presentes as seguintes consultas:

* Número de clientes
* Número de clientes por categoria
* Total de vendas por funcionário
* Total monetário de vendas por produto
* Total monetário de vendas por produto por ano
* Total monetário de vendas por cidade por ano

## Consultas sobre a nova base de dados

No final do ficheiro *WWWIGlobal Query.sql* estão presentes as seguintes consultas:

* Número de clientes
* Número de clientes por categoria
* Total de vendas por funcionário
* Total monetário de vendas por produto
* Total monetário de vendas por produto por ano
* Total monetário de vendas por cidade por ano

# Programação

## Views

|  |  |
| --- | --- |
| Nome | Descrição |
| RH.viewNorthAmericaCountry | Esta view permite obter a lista de países na américa do norte. |
| RH.viewCitySalesTerritory | Esta view permite obter a lista de território de vendas. |
| RH.viewRegion\_Category | Esta view permite obter a lista de relações entre cidade, estado, continente e categoria, com os campos das chaves externas. |
| RH.viewCustomer | Esta view permite obter a lista de customers, com os campos das chaves externas. |
| RH.viewEmployee | Esta view permite obter a lista funcionários. |
| Storage.viewProduct | Esta view permite obter a lista de produtos, com os campos das chaves externas. |
| Storage.viewProductPromotion | Esta view permite obter a lista de produtos associados a promoções, com os campos das chaves externas. |
| Sales.viewProductPromotionSale | Esta view permite obter a lista de produtos e promoções associados a vendas, com os campos das chaves externas. |

## Functions

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Atributos | Requisito | Descrição |
| RH.udf\_countryExists | @countryName  varchar (20),  @continentName varchar (20) | RF01 | Permite obter o id de um país com o nome do país e nome do continente passados, ou 0 caso não exista. |
| RH.udf\_countryExistsById | @countryId int | RF01 | Permite obter o id de um país com o id passado, ou 0 caso não exista. |
| RH.udf\_stateProvinceExists | @stateProvinceName  varchar (50) | RF01 | Permite obter o id de um estado com o nome passado, ou 0 caso não exista. |
| RH.udf\_stateProvinceExistsByCode | @stateProvinceCode varchar(50) | RF01 | Permite obter o id de um estado com o código passado, ou 0 caso não exista. |
| RH.udf\_stateProvinceExistsById | @stateProvinceId int | RF01 | Permite obter o id de um estado com o id passado, ou 0 caso não exista. |
| RH.udf\_cityExists | @cityName  varchar (50) | RF01 | Permite obter o id de uma cidade com o nome passado, ou 0 caso não exista. |
| RH.udf\_cityExistsById | @cityId int | RF01 | Permite obter o id de uma cidade com o id passado, ou 0 caso não exista. |
| RH.udf\_categoryExists | @categoryName  varchar (50) | RF01 | Permite obter o id de uma categoria com o nome passado, ou 0 caso não exista. |
| RH.udf\_categoryExistsById | @categoryId int | RF01 | Permite obter o id de uma categoria com o id passado, ou 0 caso não exista. |
| RH.udf\_regionCategoryExists | @countryId int,  @stateProvinceId int,  @cityId int,  @categoryId int | RF01 | Permite obter o id de uma relação entre pais, estado, cidade e categoria com os ids associados, passados, ou 0 caso não exista. |
| RH.udf\_regionCategoryExistsById | @regionCategoryId int | RF01 | Permite obter o id de uma relação entre pais, estado, cidade e categoria com o id passado, ou 0 caso não exista. |
| RH.udf\_buiyngGroupExists | @buiyngGroupName varchar (50) | RF01 | Permite obter o id de um grupo com o nome passado, ou 0 caso não exista. |
| RH.udf\_buiyngGroupExistsById | @buiyngGroupId int | RF01 | Permite obter o id de um grupo com o id passado, ou 0 caso não exista. |
| RH.udf\_sysUserExists | @sysUserEmail  varchar (50) | RF01 | Permite obter o id de um utilizador com o email passado, ou 0 caso não exista. |
| RH.udf\_sysUserExistsByName | @sysUserName varchar(50) | RF01 | Permite obter o id de um utilizador com o nome passado, ou 0 caso não exista. |
| RH.udf\_sysUserExistsById | @sysUserId int | RF01 | Permite obter o id de um utilizador com o id passado, ou 0 caso não exista. |
| RH.udf\_customerExists | @customerName varchar (50) | RF01 | Permite obter o id de um cliente com o nome passado, ou 0 caso não exista. |
| RH.udf\_customerExistsById | @customerId int | RF01 | Permite obter o id de um cliente com o id passado, ou 0 caso não exista. |
| RH.udf\_employeeExists | @employeeName varchar (50) | RF01 | Permite obter o id de um funcionário com o nome passado, ou 0 caso não exista. |
| RH.udf\_employeeExistsById | @employeeId int | RF01 | Permite obter o id de um funcionário com o id passado, ou 0 caso não exista. |
| RH.udf\_tokenExists | @token int | RF01 | Permite obter o id de um token com o token passado, ou 0 caso não exista. |
| RH.udf\_tokenExistsByUser | @token int, @userId int | RF01 | Permite obter o id de um token com o token e id de utilizador passados caso seja válido, ou 0 caso não exista. |
| RH.udf\_tokenExistsById | @tokenId int | RF01 | Permite obter o id de um token com o id passado, ou 0 caso não exista. |
| RH.udf\_fnHashPassword | @password varchar (20) | RF01 | Permite encriptar uma palavra-passe usando codificação SH1. |
| Storage.udf\_taxRateExists | @taxRate float | RF01 | Permite obter o id de uma taxa com o valor passado, ou 0 caso não exista. |
| Storage.udf\_taxRateExistsById | @taxRateId int | RF01 | Permite obter o id de uma taxa com o id passado, ou 0 caso não exista. |
| Storage.udf\_productTypeExists | @productType  varchar (25) | RF01 | Permite obter o id de um tipo de produto com o nome passado, ou 0 caso não exista. |
| Storage.udf\_productTypeExistsById | @productTypeId int | RF01 | Permite obter o id de um tipo de produto com o id passado, ou 0 caso não exista. |
| Storage.udf\_packageExists | @package varchar (25) | RF01 | Permite obter o id de um pacote com o nome passado, ou 0 caso não exista. |
| Storage.udf\_packageExistsById | @packageId int | RF01 | Permite obter o id de um pacote com o id passado, ou 0 caso não exista. |
| Storage.udf\_brandExists | @brand varchar (25) | RF01 | Permite obter o id de uma marca com o nome passado, ou 0 caso não exista. |
| Storage.udf\_brandExistsById | @brandId int | RF01 | Permite obter o id de uma marca com o id passado, ou 0 caso não exista. |
| Storage.udf\_productExists | @productName  varchar (100) | RF01 | Permite obter o id de um produto com o nome passado, ou 0 caso não exista. |
| Storage.udf\_productExistsById | @productId int | RF01 | Permite obter o id de um produto com o id passado, ou 0 caso não exista. |
| Storage.udf\_promotionExists | @promotionDescription varchar (100) | RF01 | Permite obter o id de uma promoção com a descrição passada, ou 0 caso não exista. |
| Storage.udf\_promotionExistsById | @promotionId int | RF01 | Permite obter o id de uma promoção com o id passado, ou 0 caso não exista. |
| Storage.udf\_productPromotionExists | @productId int,  @promotionId int | RF01 | Permite obter o id de uma relação entre produto e promoção com os ids associados, passados, ou 0 caso não exista. |
| Storage.udf\_productPromotionExistsById | @productPromotionId int | RF01 | Permite obter o id de uma relação entre produto e promoção com o id passado, ou 0 caso não exista. |
| Storage.udf\_productPromotionExistsByName | @name varchar(100),  @promotion varchar(100) | RF01 | Permite obter o id de uma relação entre produto e promoção com os nomes passados, ou 0 caso não exista. |
| Storage.udf\_productStock | @product varchar(100) | RF01 | Permite obter o stock de um entre produto com o nome passado. |
| Sales.udf\_saleExistsById | @saleId int | RF01 | Permite obter o id de uma venda com o id passado, ou 0 caso não exista. |
| Sales.udf\_productPromotionSaleExists | @productPromotionId int,  @saleId int | RF01 | Permite obter o id de uma relação entre produto-promoção e venda com os ids associados, passado, ou 0 caso não exista. |
| Sales.udf\_saleExistsByDescription | @description  varchar(100) | RF01 | Permite obter o id de uma venda com a descrição passada, ou 0 caso não exista. |
| Sales.udf\_saleType | @sale varchar(100) | RF01 | Permite obter o tipo de uma venda com a descrição passada. |

## Stored procedures

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Atributos | Requisito | Descrição |
| dbo.Migrate\_OldData\_CityTable |  | RF01 | Permite migrar os registos da tabela OldData.City para as tabelas UsersInfo.Country, UsersInfo.StateProvince e UsersInfo.City |
| dbo.Migrate\_OldData\_StatesTable |  | RF01 | Permite migrar os registos da tabela OldData.States para a tabela UsersInfo.StateProvince |
| dbo.Migrate\_OldData\_lookupTable |  | RF01 | Permite migrar os registos da tabela OldData.lookup para a tabela UsersInfo.Category |
| dbo.Migrate\_OldData\_CustomerTable |  | RF01 | Permite migrar os registos da tabela OldData.Customer para as tabelas UsersInfo.SysUser, UsersInfo.Customer, UsersInfo.Region\_Category e UsersInfo.BuyingGroup |
| dbo.Migrate\_OldData\_EmployeeTable |  | RF01 | Permite migrar os registos da tabela OldData.Employee para as tabelas UsersInfo.Employee e UsersInfo.Region\_Category |
| dbo.Migrate\_OldData\_ItemStockTable |  | RF01 | Permite migrar os registos da tabela OldData.[Stock Item] para as tabelas ProductsInfo.Package, ProductsInfo. Brand, ProductsInfo.TaxRate, ProductsInfoProductType, ProductsInfo.Product e ProductsInfo.Product\_Promotion |
| dbo.Migrate\_OldData\_SaleTable |  | RF01 | Permite migrar os registos da tabela OldData.Sale para as tabelas SalesInfo.Sale e SalesInfo.ProductPromotion\_Sale |
| dbo.MigrateAll |  | RF01 | Permite executar todos os stored procedures de migração |
| RH.sp\_recuperarPassword | @userEmail VARCHAR (50),  @token int,  @newUserPassword VARCHAR (20) |  | Permite que se altere a palavra-passe de um utilizador. |
| Sales.sp\_finishSale | @id int |  | Permite finalizar uma venda. |

## Triggers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome | Tipo | Tabela | Requisito | Descrição |
| RH.tr\_validade\_token | INSERT | UsersInfo.Token | RM02 | Altera a data de validade de um token repetido. |
| Sales.tr\_eliminateSale | DELETE | SalesInfo.ProductPromotion\_Sale | RF06 | Elimina uma venda se forem removidos todos os produtos. |
| Sales.tr\_calculateSaleInfo | UPDATE | SalesInfo.Sale | RM03 | Calcula a data prevista de entrega e os valores associados a uma venda finalizada. |

# Catálogo/Metadados

## Geradores

|  |  |  |
| --- | --- | --- |
| Nome | Atributos | Descrição |
| RH.errorLog\_insert | @error varchar (300) | Implementa o procedimento para inserir registos na tabela RH.ErrorLog. |
| RH.country\_insert | @countryName varchar (20),  @continentName varchar (20) | Implementa o procedimento para inserir registos na tabela RH.Country. |
| RH.country\_update | @countryId int,  @countryName varchar (20),  @continentName varchar (20) | Implementa o procedimento para atualizar registos na tabela RH.Country. |
| RH.country\_delete | @countryId int | Implementa o procedimento para apagar registos na tabela RH.Country. |
| RH.stateProvince\_insert | @stateProvinceName varchar (50),  @stateProvinceCode varchar (5) | Implementa o procedimento para inserir registos na tabela RH.StateProvince. |
| RH.stateProvince\_update | @stateProvinceId int,  @stateProvinceName varchar (50),  @stateProvinceCode varchar (5) | Implementa o procedimento para atualizar registos na tabela RH.StateProvince. |
| RH.stateProvince\_delete | @stateProvinceId int | Implementa o procedimento para apagar registos na tabela RH.StateProvince. |
| RH.city\_insert | @cityName varchar (50),  @citySalesTerritory varchar (50),  @cityLastPopulationRecord int | Implementa o procedimento para inserir registos na tabela RH.City. |
| RH.city\_update | @cityId int,  @cityName varchar (50),  @citySalesTerritory varchar (50),  @cityLastPopulationRecord int | Implementa o procedimento para atualizar registos na tabela RH.City. |
| RH.city\_delete | @cityId int | Implementa o procedimento para apagar registos na tabela RH.City. |
| RH.category\_insert | @categoryName varchar (50) | Implementa o procedimento para inserir registos na tabela RH.Category. |
| RH.category\_update | @categoryId int,  @categoryName varchar (50) | Implementa o procedimento para atualizar registos na tabela RH.Category. |
| RH.category\_delete | @categoryId int | Implementa o procedimento para apagar registos na tabela RH.Category. |
| RH.regionCategory\_insert | @countryId int,  @stateProvinceId int,  @cityId int,  @categoryId int,  @postalCode int | Implementa o procedimento para inserir registos na tabela RH.Region\_Category. |
| RH.regionCategory\_update | @regionCategoryId int,  @countryId int,  @stateProvinceId int,  @cityId int,  @categoryId int,  @postalCode int | Implementa o procedimento para atualizar registos na tabela RH.Region\_Category. |
| RH.regionCategory\_delete | @regionCategoryId int | Implementa o procedimento para apagar registos na tabela RH.Region\_Category. |
| RH.buiyngGroup\_insert | @buiyngGroupName varchar (50) | Implementa o procedimento para inserir registos na tabela RH.BuyingGroup. |
| RH.buiyngGroup\_update | @buiyngGroupId int,  @buiyngGroupName varchar (50) | Implementa o procedimento para atualizar registos na tabela RH.BuyingGroup. |
| RH.buiyngGroup\_delete | @buiyngGroupId int | Implementa o procedimento para apagar registos na tabela RH.BuyingGroup. |
| RH.sysUser\_insert | @sysUserName varchar (50),  @sysUserEmail varchar (50),  @sysUserPassword varchar (50) | Implementa o procedimento para inserir registos na tabela RH.SysUser. |
| RH.sysUser\_update | @sysUserId int,  @sysUserName varchar (50),  @sysUserEmail varchar (50),  @sysUserPassword varchar (50) | Implementa o procedimento para atualizar registos na tabela RH.SysUser. |
| RH.sysUser\_delete | @sysUserId int | Implementa o procedimento para apagar registos na tabela RH.SysUser. |
| RH.customer\_insert | @userId int,  @headquartersId int, @regionCategoryId int, @buyingGroupId int, @primaryContact varchar (40) | Implementa o procedimento para inserir registos na tabela RH.Customer. |
| RH.customer\_update | @userId int,  @headquartersId int, @regionCategoryId int, @buyingGroupId int, @primaryContact varchar (40) | Implementa o procedimento para atualizar registos na tabela RH.Customer. |
| RH.customer\_delete | @userId int | Implementa o procedimento para apagar registos na tabela RH.Customer. |
| RH.employee\_insert | @userId int,  @employeePreferedName  varchar (10),  @employeeIsSalesPerson BIT | Implementa o procedimento para inserir registos na tabela RH.Employee. |
| RH.employee\_update | @userId int,  @employeePreferedName  varchar (10),  @employeeIsSalesPerson BIT | Implementa o procedimento para atualizar registos na tabela RH.Employee. |
| RH.employee\_delete | @userId int | Implementa o procedimento para apagar registos na tabela RH.Employee. |
| RH.token\_insert | @tokenUserId INT | Implementa o procedimento para inserir registos na tabela RH.Token. |
| RH.token\_update | @tokenId int | Implementa o procedimento para atualizar registos na tabela RH.Token. |
| Storage.taxRate\_insert | @taxRate float | Implementa o procedimento para inserir registos na tabela Storage.TaxRate. |
| Storage.taxRate\_update | @taxRateId int,  @taxRate float | Implementa o procedimento para atualizar registos na tabela Storage.TaxRate. |
| Storage.taxRate\_delete | @taxRateId int | Implementa o procedimento para apagar registos na tabela Storage.TaxRate. |
| Storage.productType\_insert | @productType varchar (25) | Implementa o procedimento para inserir registos na tabela Storage.ProductType. |
| Storage.productType\_update | @productTypeId int,  @productType varchar (25) | Implementa o procedimento para atualizar registos na tabela Storage.ProductType. |
| Storage.productType\_delete | @productTypeId int | Implementa o procedimento para apagar registos na tabela Storage.ProductType. |
| Storage.package\_insert | @package varchar (25) | Implementa o procedimento para inserir registos na tabela Storage.Package. |
| Storage.package\_update | @packageId int,  @package varchar (25) | Implementa o procedimento para atualizar registos na tabela Storage.Package. |
| Storage.package\_delete | @packageId int | Implementa o procedimento para apagar registos na tabela Storage.Package. |
| Storage.brand\_insert | @brand varchar (25) | Implementa o procedimento para inserir registos na tabela Storage.Brand. |
| Storage.brand\_update | @brandId int,  @brand varchar (25) | Implementa o procedimento para atualizar registos na tabela Storage.Brand. |
| Storage.brand\_delete | @brandId int | Implementa o procedimento para apagar registos na tabela Storage.Brand. |
| Storage.product\_insert | @brandId int,  @taxRateId int,  @productTypeId int,  @buyingPackageId int,  @sellingPackageId int,  @productName varchar (100),  @productColor varchar (50),  @productSize varchar (20),  @productLeadTimeDays int,  @productQuantityPerOuter int,  @productStock int,  @productBarCode varchar (20),  @productUnitPrice float,  @productRecommendedRetailPrice float,  @productTypicalWeightPerUnit  float | Implementa o procedimento para inserir registos na tabela Storage.Product. |
| Storage.product\_update | @productId int,  @brandId int,  @taxRateId int,  @productTypeId int,  @buyingPackageId int,  @sellingPackageId int,  @productName varchar (100),  @productColor varchar (50),  @productSize varchar (20),  @productLeadTimeDays int,  @productQuantityPerOuter int,  @productStock int,  @productBarCode varchar (20),  @productUnitPrice float,  @productRecommendedRetailPrice float,  @productTypicalWeightPerUnit float | Implementa o procedimento para atualizar registos na tabela Storage.Product. |
| Storage.product\_delete | @productId int | Implementa o procedimento para apagar registos na tabela Storage.Product. |
| Storage.promotion\_insert | @promotionDescription  varchar (100),  @promotionStartDate varchar (20),  @promotionEndDate varchar (20) | Implementa o procedimento para inserir registos na tabela Storage. Promotion. |
| Storage.promotion\_update | @promotionId int,  @promotionDescription  varchar (100),  @promotionStartDate varchar (20),  @promotionEndDate varchar (20) | Implementa o procedimento para atualizar registos na tabela Storage.Promotion. |
| Storage.promotion\_delete | @promotionId int | Implementa o procedimento para apagar registos na tabela Storage.Promotion. |
| Storage.productPromotion\_insert | @productId int,  @promotionId int | Implementa o procedimento para inserir registos na tabela Storage.Product\_Promotion. |
| Storage.productPromotion\_update | @productPromotionId int,  @productId int,  @promotionId int | Implementa o procedimento para atualizar registos na tabela Storage.Product\_Promotion. |
| Storage.productPromotion\_delete | @productPromotionId int | Implementa o procedimento para apagar registos na tabela Storage.Product\_Promotion. |
| Sales.sale\_insert | @saleID int,  @customerId int,  @employeeId int,  @saleDescription varchar (100) | Implementa o procedimento para inserir registos na tabela Sales. Sale. |
| Sales.sale\_update | @saleID int,  @customerId int,  @employeeId int,  @saleDescription varchar (100) | Implementa o procedimento para atualizar registos na tabela Sales. Sale. |
| Sales.sale\_delete | @saleID int | Implementa o procedimento para apagar registos na tabela Sales. Sale. |
| Sales.productPromotionSale\_insert | @productPromotionId int,  @saleID int,  @quantity int | Implementa o procedimento para inserir registos na tabela Sales.ProductPromotion\_Sale. |
| Sales.productPromotionSale\_update | @productPromotionId int,  @saleID int,  @quantity int | Implementa o procedimento para atualizar registos na tabela Sales.ProductPromotion\_Sale. |
| Sales.productPromotionSale\_delete | @productPromotionId int,  @saleID int | Implementa o procedimento para apagar registos na tabela Sales.ProductPromotion\_Sale. |

## Monitorização

|  |  |  |
| --- | --- | --- |
| Nome | Atributos | Descrição |
| dbo.viewLastMonitorizacaoColunas |  | Monitora os espaços ocupados por cada resisto e por cada tabela. |

# Índices

## Views

|  |  |
| --- | --- |
| Nome | Descrição |
| dbo.viewSalesPerCity\_OldData | Esta view permite a pesquisa de vendas por cidade sobre a base de dados original. |
| dbo.viewSalesPerCity | Esta view permite a pesquisa de vendas por cidade sobre a base de dados otimizada. |
| dbo.viewYearGrowthPerSale\_OldData | Esta view permite o cálculo para as vendas da taxa de crescimento de cada ano, face ao ano anterior, por categoria de cliente sobre a base de dados original. |
| dbo.viewYearGrowthPerSale | Esta view permite o cálculo para as vendas da taxa de crescimento de cada ano, face ao ano anterior, por categoria de cliente sobre a base de dados otimizada. |
| dbo.viewNProductsPerColor\_OldData | Esta view permite o nº de produtos nas vendas por cor sobre a base de dados original. |
| dbo.viewNProductsPerColor | Esta view permite obter o nº de produtos nas vendas por cor sobre a base de dados otimizada. |

## Índices

|  |  |  |
| --- | --- | --- |
| Designação | Tabela | Justificação/Consultas |
| \_dta\_index\_City\_6\_645577338\_\_K1\_K2 | RH.City | Consultas: dbo.viewSalesPerCity.  Esta consulta foi otimizada com este índice, pois esta acede às colunas CitId e CitName da tabela RH.City. |
| \_dta\_index\_Sale\_6\_1365579903\_\_K2\_K3\_8 | Sales.Sale | Consultas: dbo.viewSalesPerCity, dbo.viewYearGrowthPerSale.  Estas consultas foram otimizadas com este índice, pois ambas acedem às colunas SalCustomerId e SalEmployeeId da tabela Sales.Sale. |
| \_dta\_index\_ProductPromotion\_Sale\_6\_1429580131\_\_K1 | Sales.ProductPromotion\_Sale | Consultas: dbo.viewNProductsPerColor  Esta consulta foi otimizada com este índice, pois esta acede à coluna ProdProm\_SalProductPromotionId da tabela Sales.ProductPromotion\_Sale. |

## Otimização e Execução de Consultas

Os resultados da execução das consultas foram os seguintes:

* dbo.viewSalesPerCity\_OldData sobre a base de dados original (não normalizada);
* SQL Server Profiler

CPU – 3321

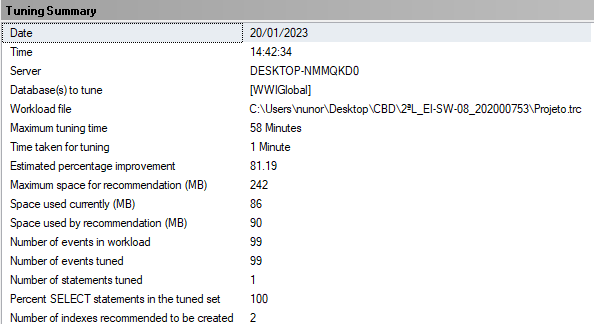
Reads – 1499966

Writes – 3216

Duration – 2442

* Database Engine TurningAdvisor

Tab Reports



* dbo.viewSalesPerCity sobre a base de dados otimizada (normalizada) sem índices;
* SQL Server Profiler

CPU – 125

Reads – 3752

Writes – 0

Duration – 248

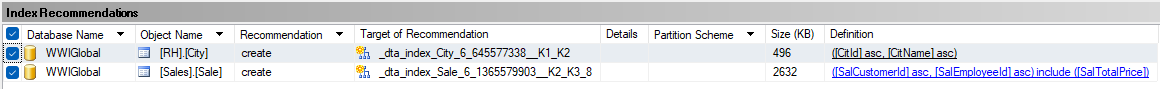
* Database Engine TurningAdvisor

Tab Reports

Uma imagem com texto

Descrição gerada automaticamente

Tab Recommendations



* dbo.viewSalesPerCity sobre a base de dados otimizada (normalizada) com índices.
* SQL Server Profiler

CPU – 93

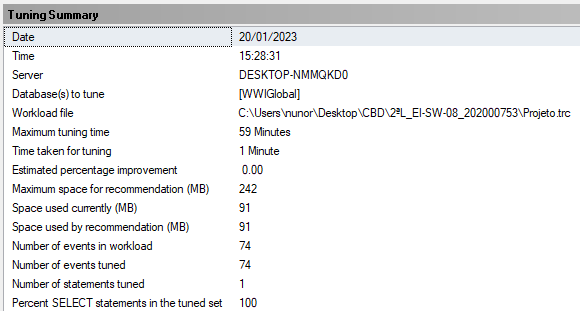
Reads – 1280

Writes – 0

Duration – 165

* Database Engine TurningAdvisor

Tab Reports



* dbo.viewYearGrowthPerSale \_OldData sobre a base de dados original (não normalizada);
* SQL Server Profiler

CPU – 30

Reads – 6204

Writes – 0

Duration – 22

* Database Engine TurningAdvisor

Tab Reports

Uma imagem com texto, mesa

Descrição gerada automaticamente

* dbo.viewYearGrowthPerSale sobre a base de dados otimizada (normalizada) sem índices;
* SQL Server Profiler

CPU – 109

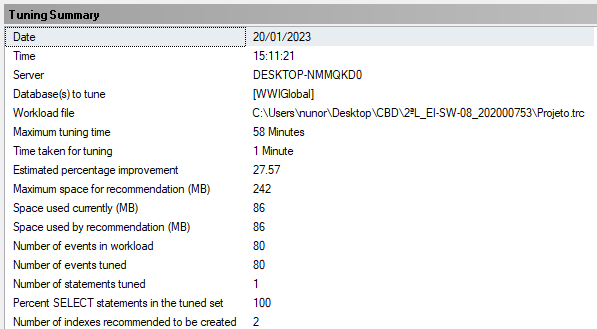
Reads – 1894

Writes – 0

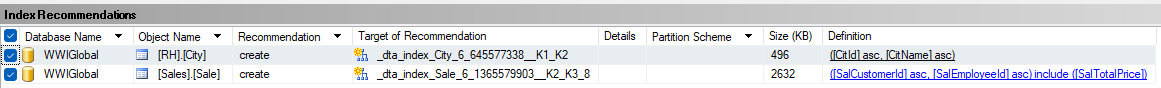
Duration – 172

* Database Engine TurningAdvisor

Tab Reports



Tab Recommendations



* dbo.viewYearGrowthPerSale sobre a base de dados otimizada (normalizada) com índices.
* SQL Server Profiler

CPU – 31

Reads – 1102

Writes – 0

Duration – 57

* Database Engine TurningAdvisor

Tab Reports

Uma imagem com mesa

Descrição gerada automaticamente

* dbo.viewNProductsPerColor \_OldData sobre a base de dados original (não normalizada);
* SQL Server Profiler

CPU – 0

Reads – 6207

Writes – 0

Duration – 21

* Database Engine TurningAdvisor

Tab Reports

Uma imagem com mesa

Descrição gerada automaticamente

* dbo.viewNProductsPerColor sobre a base de dados otimizada (normalizada) sem índices;
* SQL Server Profiler

CPU – 15

Reads – 978

Writes – 0

Duration – 53

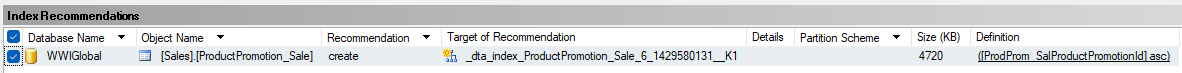
* Database Engine TurningAdvisor

Tab Reports

Uma imagem com mesa

Descrição gerada automaticamente

Tab Recommendations



* dbo.viewNProductsPerColor sobre a base de dados otimizada (normalizada) com índices.
* SQL Server Profiler

CPU – 15

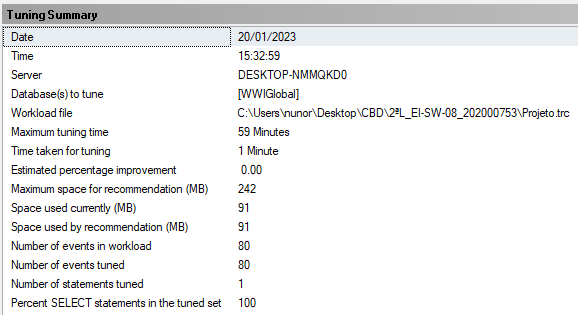
Reads – 473

Writes – 0

Duration – 20

* Database Engine TurningAdvisor

Tab Reports



# Backup e Recuperação

Optei por utilizar o modelo de recuperação Full, apesar de mais dispendioso em espaço necessário e menor desempenho tem uma proteção mais elevada contra a perda de informação.

O tipo de backups que decidi usar é Backups completos de sete em sete dias e Backups diferenciais a cada vinte e quatro horas.

Pode ser necessário recuperar apenas os dados inseridos ou alterados depois do fim da semana, neste caso a recuperação seria feita através do backup diferencial. Caso fossem perdidos todos os dados a recuperação seria feita usando o ficheiro do backup completo e o ficheiro do backup diferencial.

# Segurança e Controlo de Acessos

## Níveis de acesso à informação

Roles:

* administrador - tem acesso a toda a informação.
* employeeSalesPerson - tem acesso total às tabelas de suporte às vendas, e apenas acesso em modo de consulta às restantes tabelas.
* salesTerritory - Apenas pode consultar a informação relativa ao seu território (Rocky Mountain).

Utilizadores:

* Adminis - tem como credenciais de acesso login: “Admi”; password = “PASSWORD”; Tem o role “administrador”.
* EmpSalPerson - tem como credenciais de acesso login: “EmployeeSales”; password = “PASSWORD”; Tem o role “employeeSalesPerson”.
* SalTerri - tem como credenciais de acesso login: “SalesTer”; password = “PASSWORD”; Tem o role “salesTerritory”.

## Encriptação

A nível de encriptação, os campos de password da tabela RH.SysUser são guardadas usando a encriptação com chaves assimétricas (SHA-1), apesar de diminuírem a performance são mais complexas e seguras.

# MongoDB

Primeiro foi criada a base de dados WWWIWeb em MongoDB depois foram criadas as consultas necessárias para popular a base de dados anterior (estas consultas estão no ficheiro *MongoDB.sql*), através destas consultas foram exportados os dados para ficheiros .csv (presentes na pasta MongoDB) e por último foram importados os dados desses ficheiros para a base de dados WWWIWeb.

## Coleções

BuyingGroup:

* BuyGrouId (number)
* BuyGrouName (string)

Brand:

* BraId (number)
* BraName (string)

Customer:

* CusUserId (number)
* CusHeadquartersId (number)
* CusRegion\_CategoryId (number)
* CusBuyingGroupId (number)
* CusPrimaryContact (string)

Product:

* ProdId (number)
* ProdBrandId (number)
* ProdTaxRateId (number)
* ProdProductTypeId (number)
* ProdBuyingPakageId (number)
* ProdSellingPakageId (number)
* ProdName (string)
* ProdColor (string)
* ProdSize (string)
* ProdLeadTimesDays (number)
* ProdQuantityPerOuter (number)
* ProdStock (number)
* ProdBarCode (number)
* ProdUnitPrice (double)
* ProdRecommendedRetailPrice (double)
* ProdTypicalWeightPerUnit (double)

Product\_Promotion:

* Prod\_PromProductPromotionId (number)
* Prod\_PromProductId (number)
* Prod\_PromPromotionId (number)
* ProdNewPrice (string)

ProductPromotion\_Sale:

* ProdProm\_SalProductPromotionId (number)
* ProdProm\_SalSaleId (number)
* ProdProm\_SalQuantity (number)

Promotion:

* PromId (number)
* PromDescription (string)
* PromStartDate (date)
* PromEndDate (date)

Sale:

* SalID (number)
* SalCustomerId (number)
* SalEmployeeId (number)
* SalDate (date)
* SalDeliveryDate (date)
* SalDescription (string)
* SalProfit (double)
* SalTotalPrice (double)
* SalTotalExcludingTax (double)
* SalTaxAmount (double)
* SalIsFinished (boolean)

SysUser

* SysUseId (number)
* SysUseEmail (string)
* SysUsePassword (string)
* SysUseName (string)

# Descrição da Demonstração

## Script de demonstração sobre a base de dados relacional

O primeiro passo é criar a base de dados para tal temos de executar o ficheiro *CriacaoBD.sql* (cria a base de dados WWWIGlobal, schemas e tabelas), podemos também executar os ficheiros *Functions.sql*, *Geradores.sql*, *Triggers.sql*, *StoredProcedures.sql* e *Views.sql* para criar as funções, stored procedures geradores, triggers, stored procedures e views.

Para a importação dos dados da base de dados antiga para a nova teremos de realizar os seguintes passos:

* Executar todos os ficheiros da pasta Scripts de migração (cria e importa as tabelas e registos da base de dados antiga WWW\_DS), a ordem da execução dos ficheiros apenas importa nos ficheiros Sale…, a ordem tem de ser *Sale1.sql*, *Sale2.sql*, *Sale3.sql* e por fim *Sale4.sql*.
* Para verificar que as tabelas foram bem criadas e os registos foram bem importados podemos executar as queries no final do ficheiro *WWWGlobal Query.sql* (apenas as queries associadas ao schema OldData).
* Por fim basta executar o ficheiro *OldDataMigration.sql* para que todos os registos sejam tratados e inseridos nas novas tabelas.
* Podemos também conferir se os registos foram bem tratados e inseridos executando as queries dos ficheiros do *WWWGlobal Query.sql* (queries associadas aos schemas RH, Storage e Sales).

Depois disso basta executar as funcionalidades que pretendemos testar (a forma como se executa todas as funcionalidades está nos ficheiros, no ficheiro *SQLQuery2.sql* são apresentadas algumas das mais importantes).

Para gerar e guardar as informações relativas á monitorização e espaço ocupado pelas tabelas e colunas precisamos de executar o ficheiro *monitorização.sql*.

## Script de demonstração sobre a base de dados NoSQL

O primeiro passo é criar a base de dados WWWIWeb no MongoDB Compass, depois criar as coleções indicadas no ponto 13.1. Para importar os dados do MSSQL basta fazer a importação para as coleções através dos ficheiros presentes na pasta MongoDB.

Para verificar o correto funcionamento da base de dados NoSQL basta realizar as consultas do ponto 13.2.

# Conclusões

Com a realização deste projeto fiquei mais familiarizado com a administração de uma base de dados relacional, apesar de não ter conseguido implementar tudo o que era proposto e ambicionei. Consegui aplicar o conhecimento que adquiri em anos anteriores na disciplina Bases de Dados e no ano atual em Complementos de Base de Dados.

Fazer este projeto sozinho foi muito desafiador pois tive de desenvolver todo o projeto apenas com a ajuda do professor e a pesquisa na internet.