A blue and black logo

Description automatically generated

|  |
| --- |
| Complementos de Bases de Dados 2023/2024  Licenciatura em Engª. Informática |
| 1ª e 2º Fase Relatório Técnico |

Turma: 2

Horário de Laboratório: 10:30 às 12:30

Docente: João Portelinha

Grupo

Nº202100728, André Rolo  
Nº202100744, Tomás Gonçalves

**Índice**

[1. Introdução 5](#_Toc155612796)

[2. Especificação de Requisitos 6](#_Toc155612797)

[3. Modelo Relacional (*Modelo de dados*) 6](#_Toc155612798)

[3.1 Diagrama do Modelo Entidade Relação 6](#_Toc155612799)

[3.2 Diagrama do Modelo Relacional 7](#_Toc155612800)

[4. Definição do Layout 8](#_Toc155612801)

[4.1 Identificação do espaço ocupado por tabela 8](#_Toc155612802)

[4.2 Especificação dos FIlegroups 9](#_Toc155612803)

[4.3 Schemas 9](#_Toc155612804)

[5. Verificação da migração de dados 11](#_Toc155612805)

[5.1 Consultas sobre a base de dados original 11](#_Toc155612806)

[5.2 Consultas sobre a nova base de dados 17](#_Toc155612807)

[6. Programação 36](#_Toc155612808)

[6.1 Views 36](#_Toc155612809)

[6.2 Functions 36](#_Toc155612810)

[6.3 Stored procedures 37](#_Toc155612811)

[6.4 Triggers 40](#_Toc155612812)

[7. Catálogo/Metadados 42](#_Toc155612813)

[7.1 Monitorização 42](#_Toc155612814)

[1. Índices 43](#_Toc155612815)

[1.1 Views 43](#_Toc155612816)

[1.2 Índices 44](#_Toc155612817)

[1.3 Otimização e Execução de Consultas 45](#_Toc155612818)

[2. Backup e Recuperação 50](#_Toc155612819)

[3. Segurança e Controlo de Acessos 54](#_Toc155612820)

[3.1 Níveis de acesso à informação 54](#_Toc155612821)

[3.2 Encriptação 59](#_Toc155612822)

[4. Controlo de Concorrência 60](#_Toc155612823)

[5. MongoDB 62](#_Toc155612824)

[8. Descrição da Demonstração 65](#_Toc155612825)

[8.1 Script de demonstração 65](#_Toc155612826)

[9. Conclusões 100](#_Toc155612827)

**Índice de Imagens**

[Figure 1 - Diagrama de Modelo Entidade Relação 7](#_Toc155612828)

[Figure 2 - Modelo Relacional 8](#_Toc155612829)

[Figure 3 - Tabela Products da OldData 12](#_Toc155612830)

[Figure 4 - Tabela Customer da OldData 13](#_Toc155612831)

[Figure 5 - Tabela Currency da OldData 14](#_Toc155612832)

[Figure 6 - Tabela ProductSubCategory da OldData 15](#_Toc155612833)

[Figure 7 - Tabela Sales2 da OldData 16](#_Toc155612834)

[Figure 8 - Tabela Salesterritory da OldData 17](#_Toc155612835)

[Figure 9 - Perocedimento MigrateModelData 18](#_Toc155612836)

[Figure 10 - Tabela Model 19](#_Toc155612837)

[Figure 11 - Percedimento MigrateColorData 20](#_Toc155612838)

[Figure 12 - Tabela Color 21](#_Toc155612839)

[Figure 13 - Percedimento MigrateProductCategoryData 22](#_Toc155612840)

[Figure 14 - Tabela ProductCategory 23](#_Toc155612841)

[Figure 15 - Percedimento InsertSalesTerritoryData 24](#_Toc155612842)

[Figure 16 - Tabela SalesTerritory 25](#_Toc155612843)

[Figure 17 - Percedimento MigrateCurrencyData 26](#_Toc155612844)

[Figure 18 - Tabela Currency 27](#_Toc155612845)

[Figure 19 - Percedimento MigrateProductSubCategoryData 28](#_Toc155612846)

[Figure 20 - Tabela ProductSubCategory 29](#_Toc155612847)

[Figure 21 - Percedimento MigrateCategoryListData 30](#_Toc155612848)

[Figure 22 - Tabela CategoryList 31](#_Toc155612849)

[Figure 23 - criação de nova Tabela Products 32](#_Toc155612850)

[Figure 24 - Percedimento MigrateProductsData 32](#_Toc155612851)

[Figure 25 - Tabela Products 33](#_Toc155612852)

[Figure 26 - criação de uma nova tabela Customer 34](#_Toc155612853)

[Figure 27 - Percedimento MigrateCustomerData 35](#_Toc155612854)

[Figure 28 - Tabela Customer 36](#_Toc155612855)

[Figure 29 Estatisticas Vendas por cidade (sem indices) 46](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612856)

[Figure 30 planos de execução vendas por cidade (sem indices) 47](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612857)

[Figure 31 Estatísticas vendas por ano (sem indices) 47](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612858)

[Figure 32 Plano de execução vendas por ano (sem indices) 47](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612859)

[Figure 33 Estatísticas Calcular Número de Produtos (sem indices) 48](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612860)

[Figure 34 Plano de execução Calcular Número de produtos (sem indices) 48](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612861)

[Figure 35 Estatisticas Vendas por cidade (com indices) 49](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612862)

[Figure 36 Plano de execução Vendas por cidade (com indices) 49](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612863)

[Figure 37 Estatisticas Vendas por ano (com indices) 50](#_Toc155612864)

[Figure 38 Plano de execução Vendas por ano (com indices) 50](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612865)

[Figure 39 Estatisticas Calcular número de produtos (com indices) 50](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612866)

[Figure 40 Plano de execução Calcular número de produtos (com indices) 51](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612867)

[Figure 41 Modelo de recuperação 51](#_Toc155612868)

[Figure 42 os três tipos de backups (integral, diferencial, log) 52](#_Toc155612869)

[Figure 43 exclusão acidental dos dados integral 52](#_Toc155612870)

[Figure 44 exclusão acidental dos dados diferencial 52](#_Toc155612871)

[Figure 45 exclusão acidental dos dados log 53](#_Toc155612872)

[Figure 46 corrupção de dados 53](#_Toc155612873)

[Figure 47 corrupção de dados integral 53](#_Toc155612874)

[Figure 48 corrupção de dados diferencial e log 54](#_Toc155612875)

[Figure 49 Administrador (login) 55](file:///E:\Escola\Faculdade\Licenciatura\1º%20Ano\Complementos%20de%20Base%20de%20Dados\202100744_202100728\FASE%202\Relatorio_tecnico-Fase1e2_202100744-202100728.docx#_Toc155612876)

[Figure 50 Administrador (criação de user) 55](#_Toc155612877)

[Figure 51 Administrador (conceder permissões para user) 56](#_Toc155612878)

[Figure 52 Administrador (conceder permissões para base de dados) 56](#_Toc155612879)

[Figure 53 Administrador (conceder permissões nível de servidor) 56](#_Toc155612880)

[Figure 54 SalesPersonLogin 57](#_Toc155612881)

[Figure 55 SalesPersonUser 57](#_Toc155612882)

[Figure 56 SalesPersonUser consulta 58](#_Toc155612883)

[Figure 57 PacificSales 58](#_Toc155612884)

[Figure 58 Encriptação 60](#_Toc155612885)

[Figure 59 MongoDB 63](#_Toc155612886)

[Figure 60 MongoDB importar dados 64](#_Toc155612887)

[Figure 61 MongoDB importado JSON 65](#_Toc155612888)

# Introdução

No presente relatório tem como objetivo explicar o que o grupo com dois elementos (André Rolo e Tomás Gonçalves) fizeram no projeto de fase 1 e 2 da Unidade Curricular de Complementos de Bases de Dados. Neste Projeto elaboramos com lógica a criação de novas tabelas a uma base de dados (usámos a migração de dados para facilitar a criação das novas tabelas e as suas relações), onde foi fornecida tabelas já criadas em Excel. Tivemos de aplicar os conhecimentos em aula. Uma das matérias que aplicámos foi os metadados, índices, backups, transações e outros conhecimentos adquiridos em outras disciplinas relacionadas a criação de base de dados (Sistemas de Gestão de Bases de Dados).

Nós esperamos que este projeto ajude-nos a melhorar os nossos conhecimentos de criação de tabelas eficientes e com um sistema de segurança, para isso declaramos funções e procedimentos, também usamos e declaramos as views, with e trigger, onde iremos explicar com mais detalhes no decorrer do relatório. Além disso, nosso objetivo é melhorar a capacidade de trabalhar em equipe e aprendermos a resolver problemas que possam aparecer no mundo do trabalho.

# Especificação de Requisitos

Nós ao vermos o enunciado dado pelo professor, foi no proposto fazer uma base de dados em que os produtos podiam ter categorias e subcategorias, onde fomos ter um discussão em grupo de decidirmos declarar uma tabela para categorias e outra para sub categorias. Como pode ver no modelo relacional e no modelo Entidade Relação

|  |  |  |
| --- | --- | --- |
| ID | Descrição | Implementado  (S/N) |
| *R01* | *Este R01 é o mesmo que o MER mas com a diferença de ter mais especificado as chaves primarias e estrangeiras mostrando as suas colunas.* | *S* |
| *MER01* | *Criamos um sistema que irá permitir adicionar clientes , produtos dar categorias aos produtos e nas categorias estão a usar subcategorias, assim sendo mais fácil procurar por um produto existente no banco de dados. Também foi adicionado o sendEmail para enviar um email ao cliente e assim mostrando possíveis alterações na sua conta (Ex.: Password alterada)* | *S* |

# Modelo Relacional (*Modelo de dados*)

## Diagrama do Modelo Entidade Relação

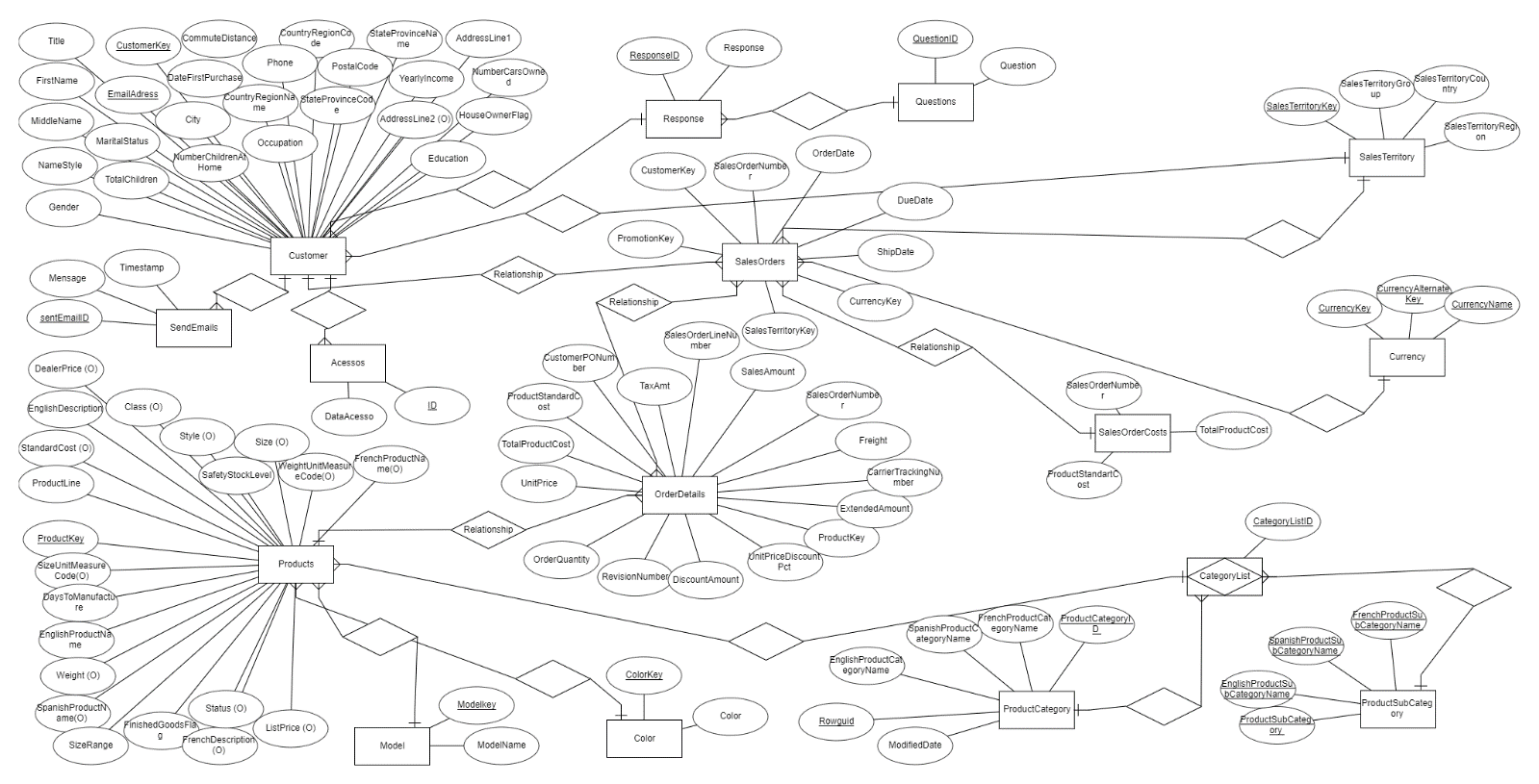
**

Figure 1 - Diagrama de Modelo Entidade Relação

## Diagrama do Modelo Relacional

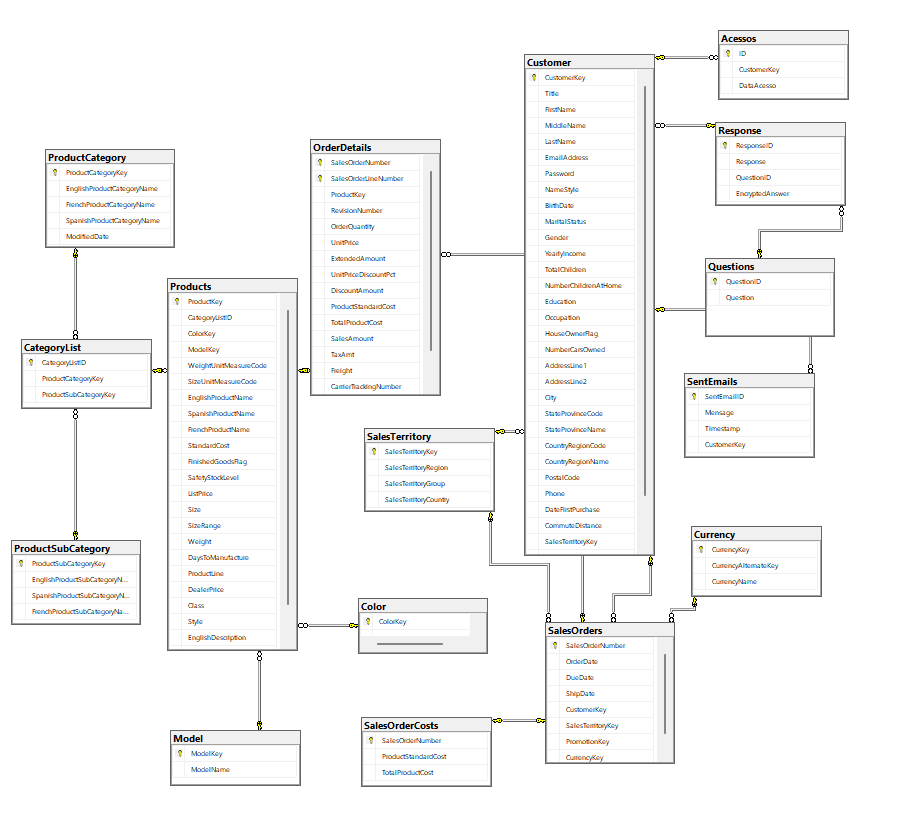


Figure - Modelo Relacional

# Definição do Layout

No nosso Layout começamos por criar a possibilidade de ver os espaços de cada tabela dando a informação do número de linhas e espaço que foi reservado para a tabela e seu espaço usado, para além de mostrar os espaços que está a ser ocupado pelos índices da tabela (são partes para melhorar o desempenho), e mostro ainda o espaço que ainda pode ser usado. Ainda fazemos um select em que mostra os nomes de todas as tabelas e exibe o tamanho máximo, número de registos e espaço total. Com base no select de cima geramos uma tabela temporária onde iremos mostrar a taxa de crescimento da tabela e ainda colocamos um gatilho em cada tabela para não crashar para quando se faz insert, update ou delete na tabela.

## Identificação do espaço ocupado por tabela

|  |  |  |
| --- | --- | --- |
| Nome Tabela | Dimensão do Registo | Nº de Registos (inicial/final) |
| *Acessos* | *72KB* | *0* |
| *CategoryList* | *37KB* | *37* |
| *Currency* | *288KB* | *105* |
| *Customer* | *7376KB* | *18484* |
| *ErrorLog* | *72KB* | *9* |
| *EstatisticasTabelas* | *72KB* | *18* |
| *MetadadosTabelas* | *72KB* | *139* |
| *Model* | *72KB* | *119* |
| *ProductCategory* | *72KB* | *4* |
| *Products* | *400KB* | *397* |
| *ProductSubCategory* | *288KB* | *37* |
| *Questions* | *72KB* | *2* |
| *Response* | *72KB* | *1* |
| *SalesOrders* | *856KB* | *6074* |
| *OrderDetails* | *1048KB* | *6074* |
| *SalesOrdersCosts* | *328KB* | *6074* |
| *SalesTerritory* | *144KB* | *11* |
| *SentEmails* | *72KB* | *65* |
| *sysdiagrams* | *280KB* | *1* |

## Especificação dos FIlegroups

|  |  |  |
| --- | --- | --- |
| Nome Filegroup | Tabelas associadas | Parâmetros |
| *PRIMARY* | *SalesOrders, OrderDetails, SalesOrdersCosts, Customer, ErrorLog, EstatisticasTabelas, MetadadosTabelas, Model, ProductCategory, Products, ProductSubCategory, Questions, Response, SalesTerritory, SentEmails* | *Nome, Localização de Ficheiros* |

## Schemas

|  |  |
| --- | --- |
| Nome | Descrição |
| *sp\_spaceused* | *Usado para obter rapidamente informações resumidas sobre o espaço ocupado por uma tabela. Ele fornece dados como o número de linhas, espaço total alocado, espaço usado para dados e índices, e a diferença entre o espaço reservado e o espaço realmente utilizado* |
| *sys.tables* | *Fornece uma lista de todas as tabelas no banco de dados, incluindo detalhes como o nome da tabela, a data de criação, a última data de modificação, e outras propriedades relacionadas às tabelas* |
| *sys.indexes* | *oferece uma lista de todos os índices no banco de dados, incluindo detalhes como o nome do índice, o tipo de índice, a tabela à qual o índice pertence, e informações sobre a fragmentação do índice* |
| *sys.partitions* | *fornece informações sobre as partições de tabelas e índices, incluindo detalhes como o nome do objeto, a função da partição, o tipo de índice, o número de linhas na partição e informações sobre o armazenamento de dados* |
| *sys.allocation\_units* | *oferece informações resumidas sobre como o espaço físico é alocado para objetos, como tabelas e índices, no banco de dados. Ela inclui detalhes sobre o tipo de alocação, objeto associado, quantidade de espaço alocado e localização física* |
| *sys.data\_spaces* | *oferece informações resumidas sobre a organização de espaços de dados no banco. Fornece dados como o nome do espaço de dados, tipo (filegroup ou partition scheme), e local de armazenamento* |
| *sys.filegroups* | *fornece informações resumidas sobre os filegroups no banco de dados, incluindo detalhes como nome, tipo e localização física* |
| *sys.database\_files* | *fornece informações resumidas sobre os arquivos físicos de um banco de dados, incluindo detalhes como nome, tipo, tamanho e localização.* |
| *sys.columns* | *fornece informações resumidas sobre as colunas em tabelas e exibições, incluindo nome, tipo de dados e outras propriedades.* |
| *sys.default\_constraints* | *fornece informações sobre restrições padrão em colunas de tabelas, incluindo nome da restrição, tabela, coluna e o valor padrão.* |
| *sys.types* | *fornece informações resumidas sobre os tipos de dados disponíveis no banco de dados, incluindo tipos padrão e definidos pelo utilizador.* |

# Verificação da migração de dados

## Consultas sobre a base de dados original

Na tabela antiga Products teremos a seguinte consulta:

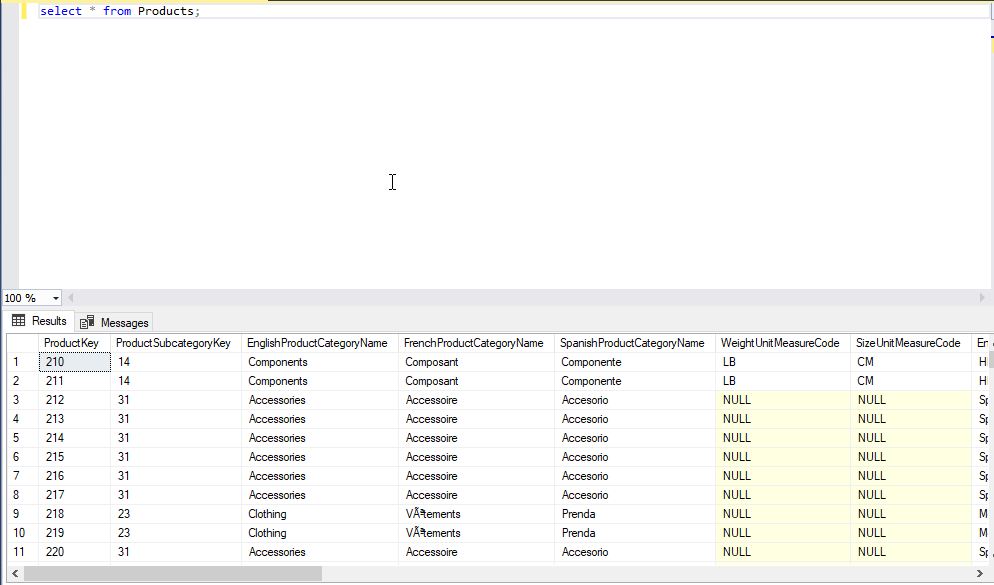


Figure 3 - Tabela Products da OldData

Na tabela antiga Customer teremos a seguinte consulta:

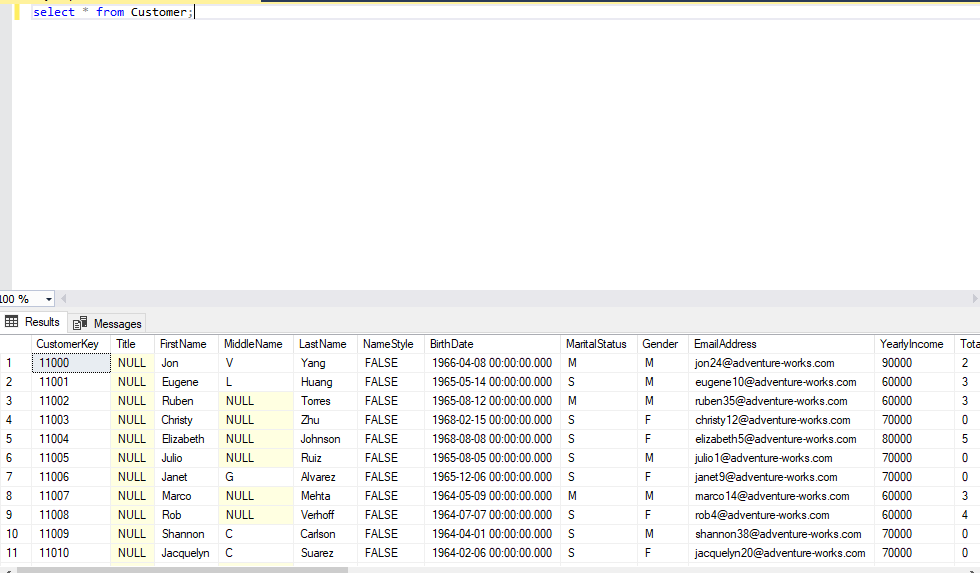


Figure 4 - Tabela Customer da OldData

Na tabela antiga Currency teremos a seguinte consulta:

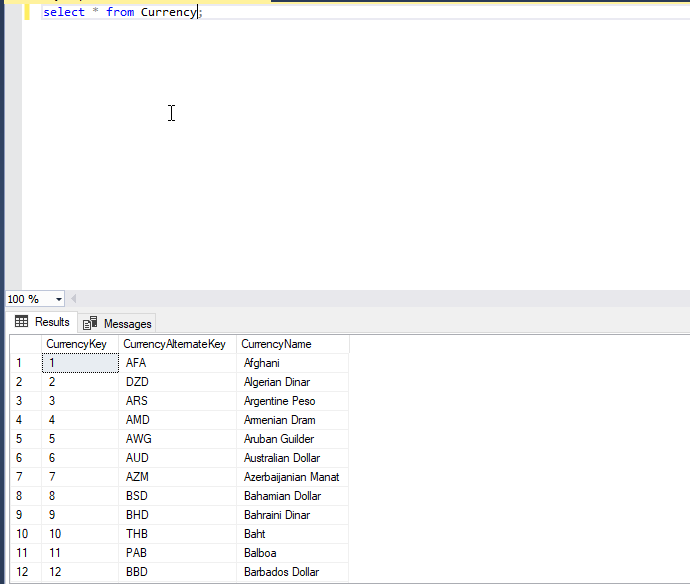


Figure 5 - Tabela Currency da OldData

Na tabela antiga ProductSubCategory teremos a seguinte consulta:

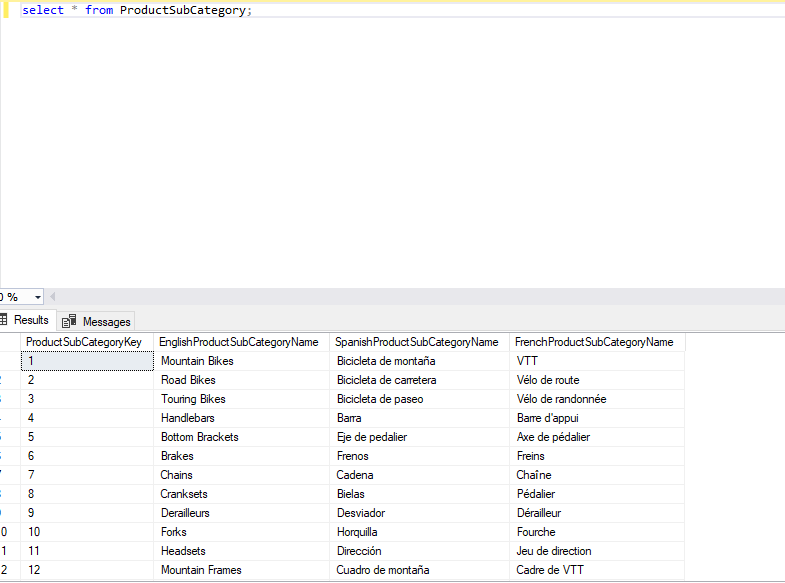


Figure 6 - Tabela ProductSubCategory da OldData

Na tabela antiga Sales2 teremos a seguinte consulta:

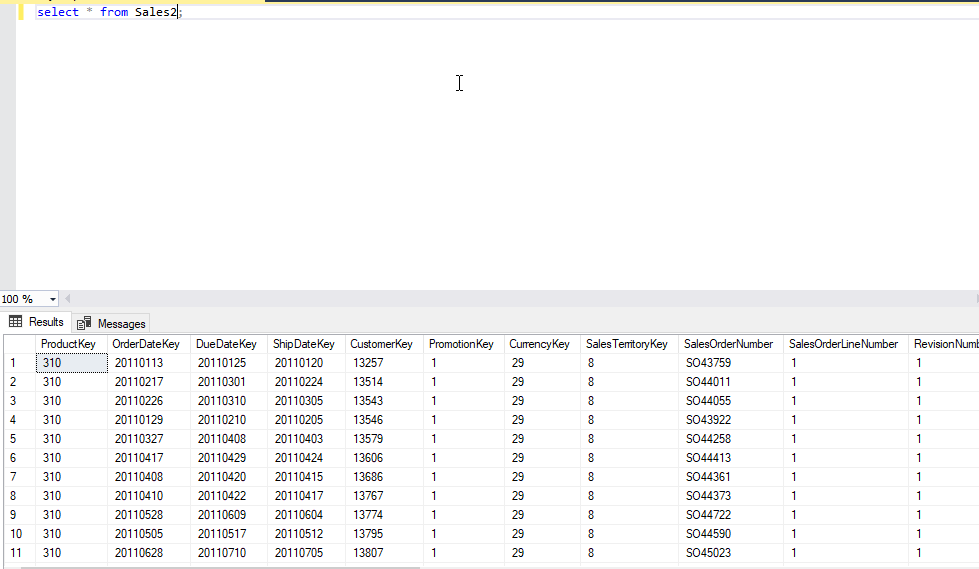


Figure 7 - Tabela Sales2 da OldData

Na tabela antiga SalesTerritory teremos a seguinte consulta:

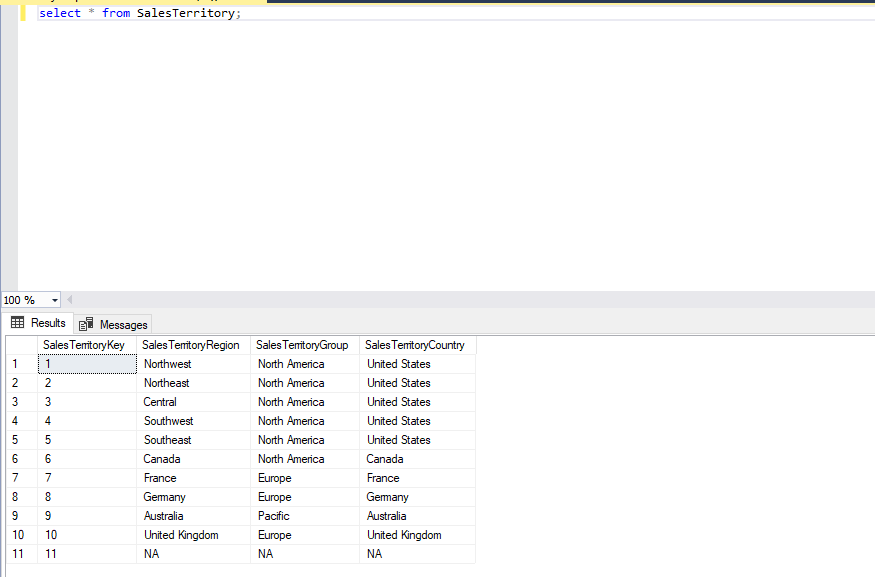


Figure 8 - Tabela Salesterritory da OldData

## Consultas sobre a nova base de dados

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Model”, para fazer a separação dos dados da tabela Products.

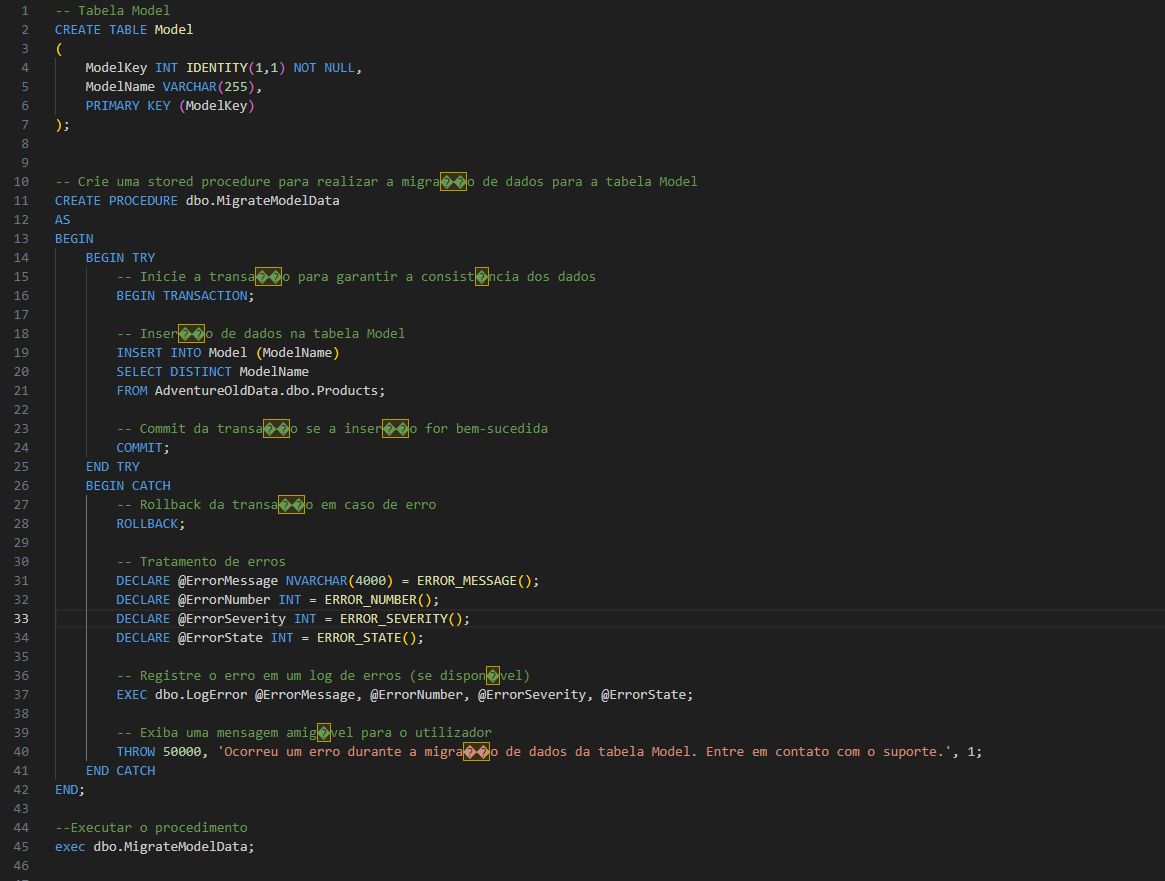


Figure - Perocedimento MigrateModelData

Na seguinte imagem mostra a consulta da tabela “Model”.

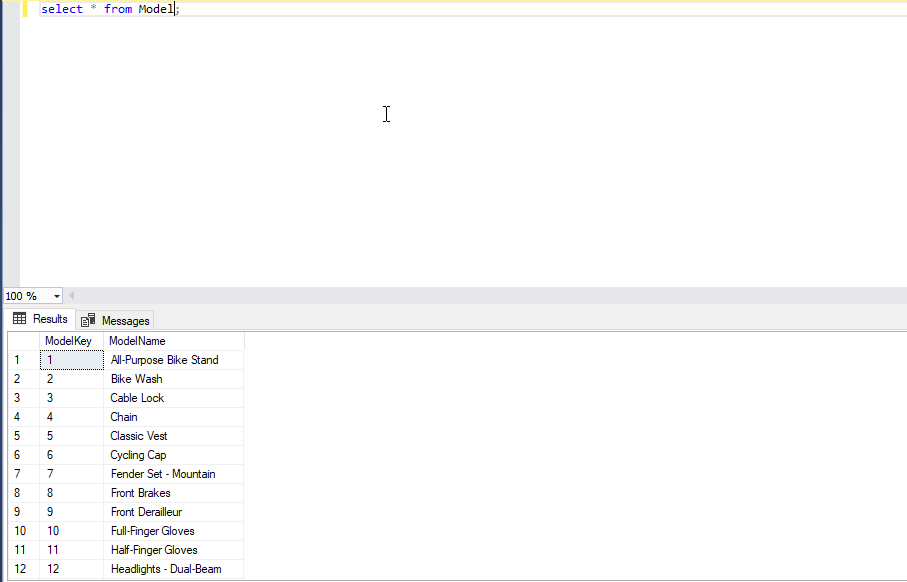


Figure - Tabela Model

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Color”, para fazer a separação dos dados da tabela Products.

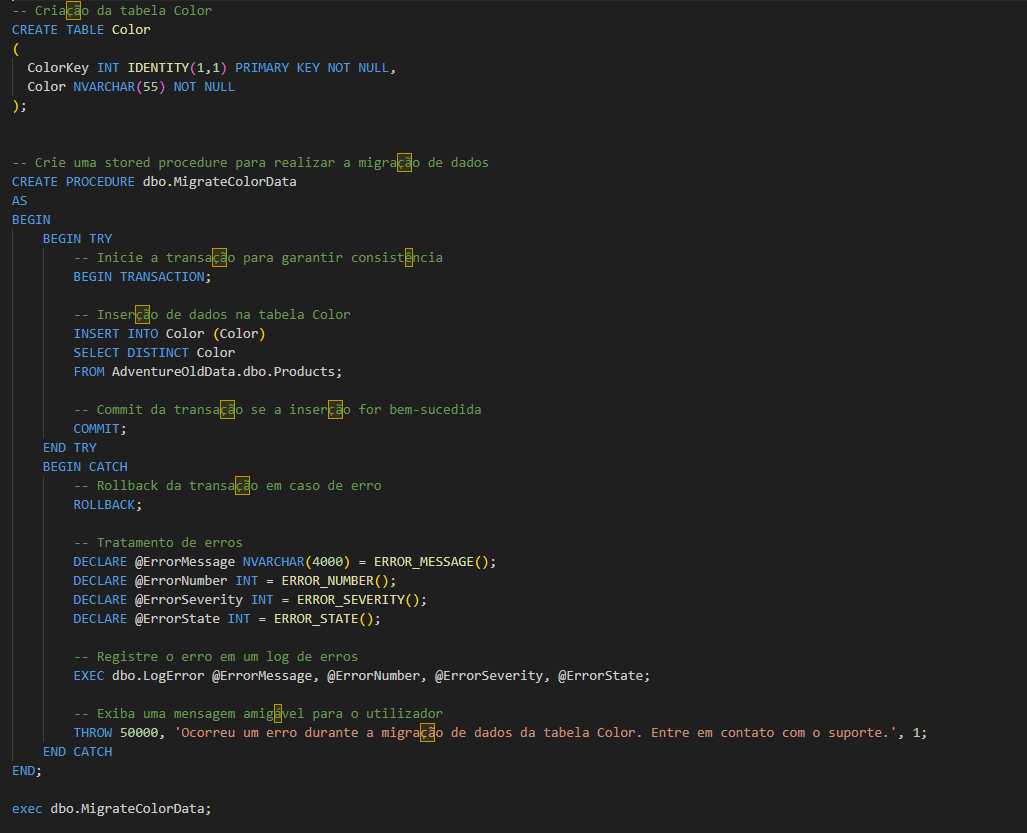


Figure - Percedimento MigrateColorData

Na seguinte imagem mostra a consulta da tabela “Color”.

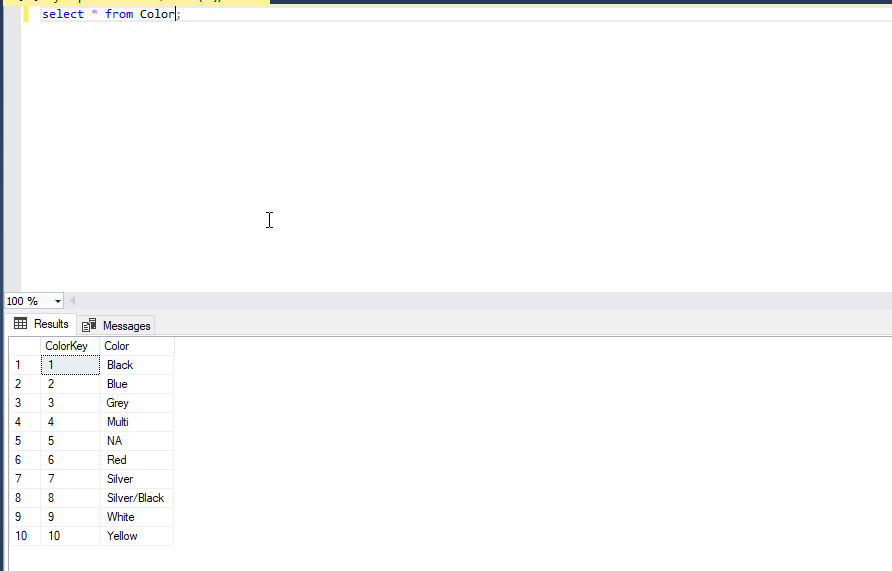


Figure - Tabela Color

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “ProductCategory”, para fazer a separação dos dados da tabela Products.



Figure - Percedimento MigrateProductCategoryData

Na seguinte imagem mostra a consulta da tabela “ProductCategory”.

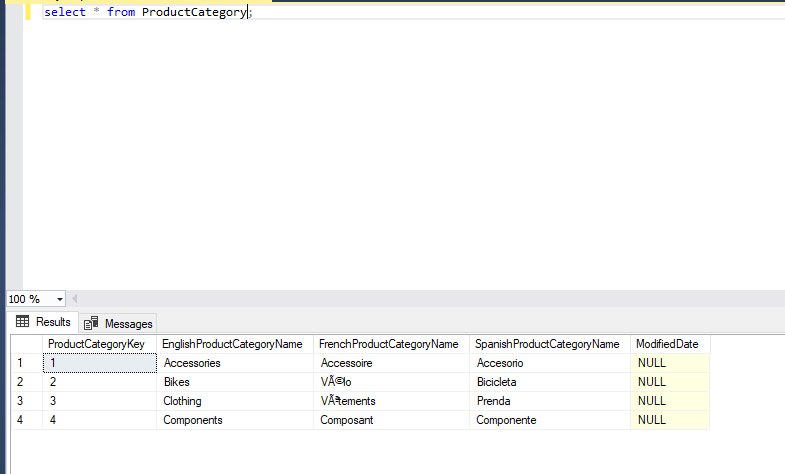


Figure - Tabela ProductCategory

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “SalesTerritory” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.

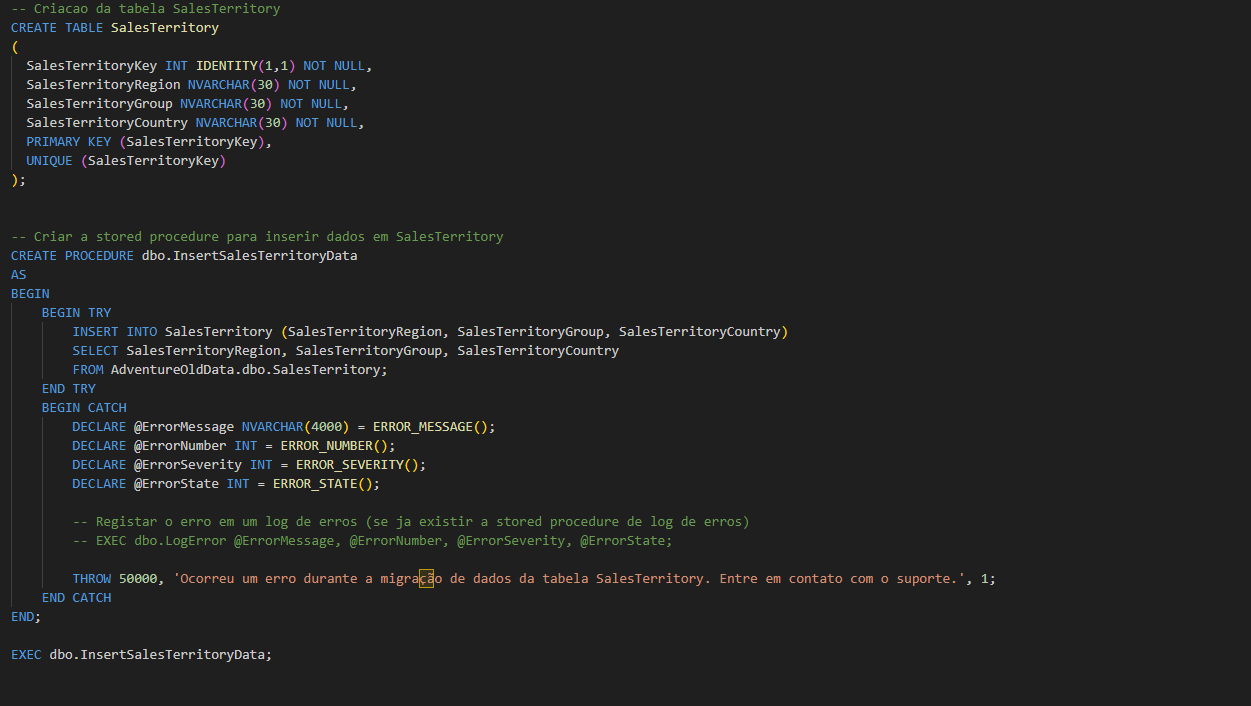


Figure - Percedimento InsertSalesTerritoryData

Na seguinte imagem mostra a consulta da tabela “SalesTerritory”.

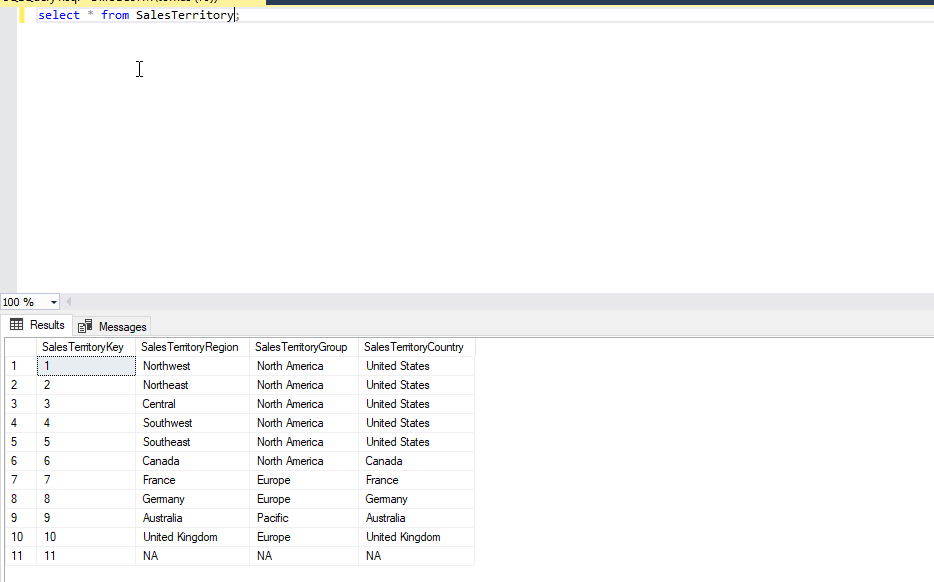


Figure - Tabela SalesTerritory

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Currency” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.



Figure - Percedimento MigrateCurrencyData

Na seguinte imagem mostra a consulta da tabela “Currency”.

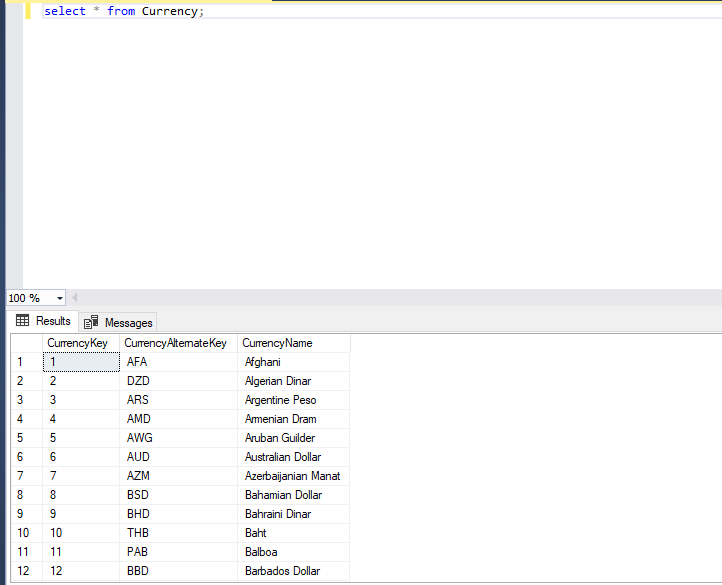


Figure - Tabela Currency

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “ProductSubCategory” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.

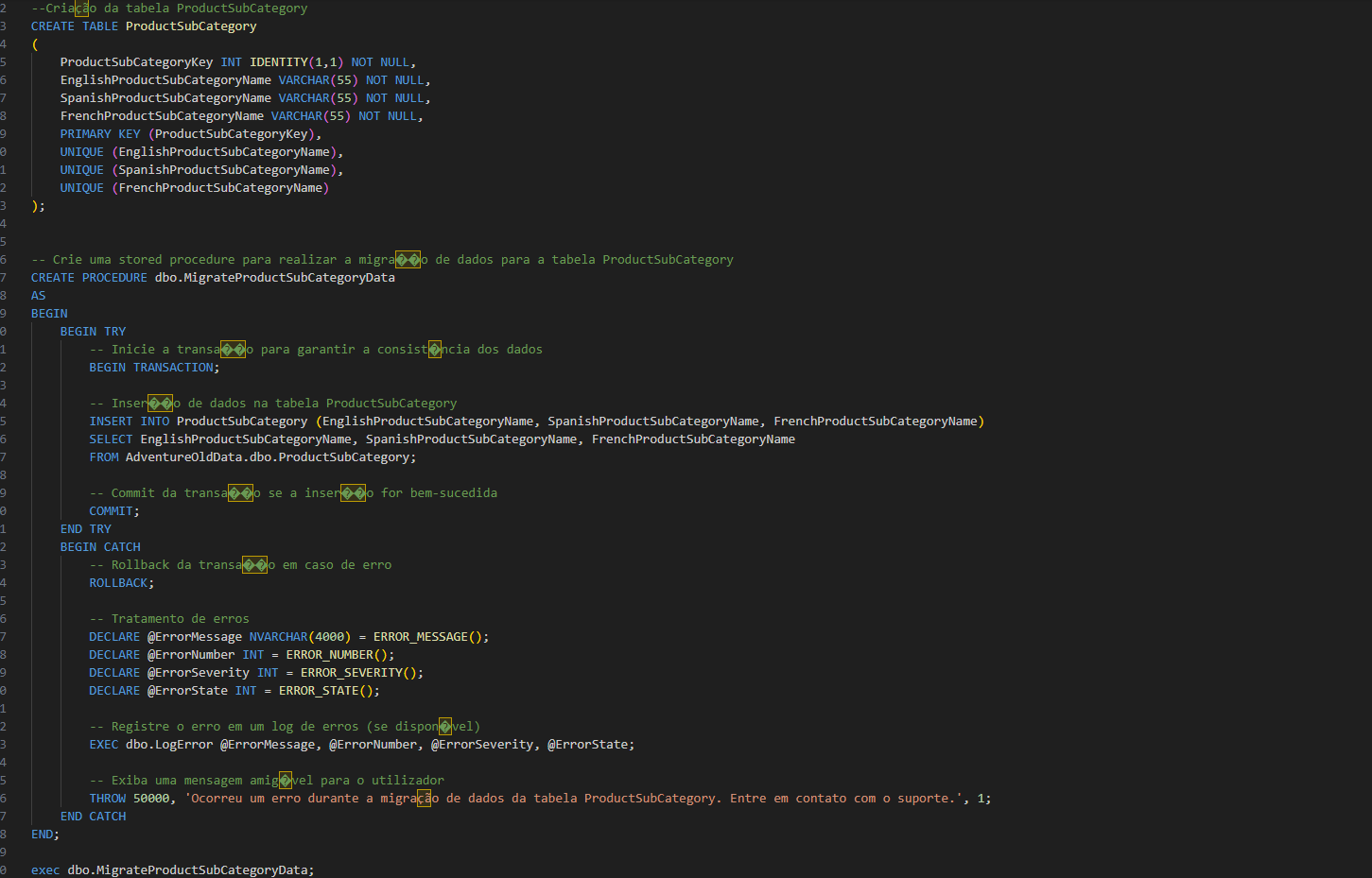


Figure - Percedimento MigrateProductSubCategoryData

Na seguinte imagem mostra a consulta da tabela “ProductsSubCategory”.

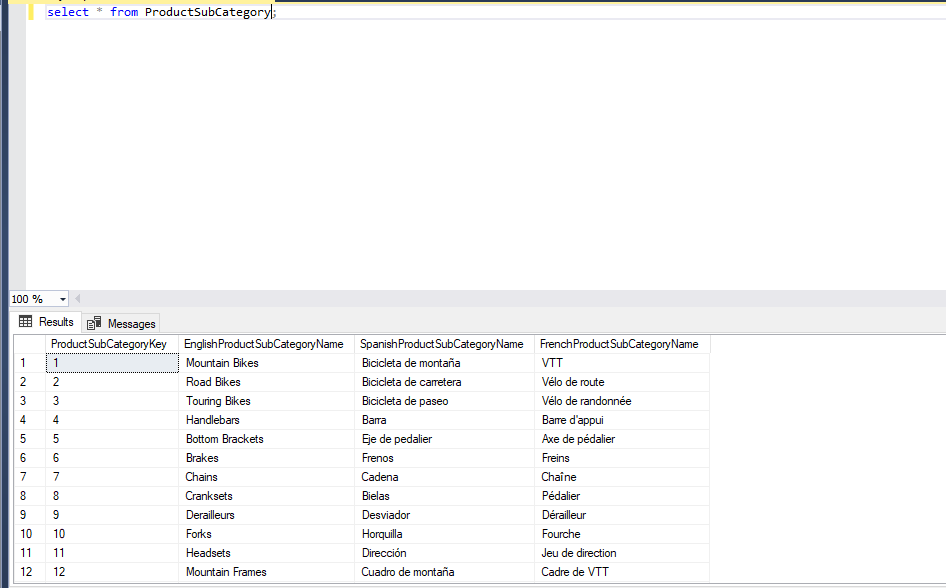


Figure - Tabela ProductSubCategory

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “CategoryList” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente, com isso teremos de utilizar a tabela “ProductCategory” e “ProductSubCategory”.

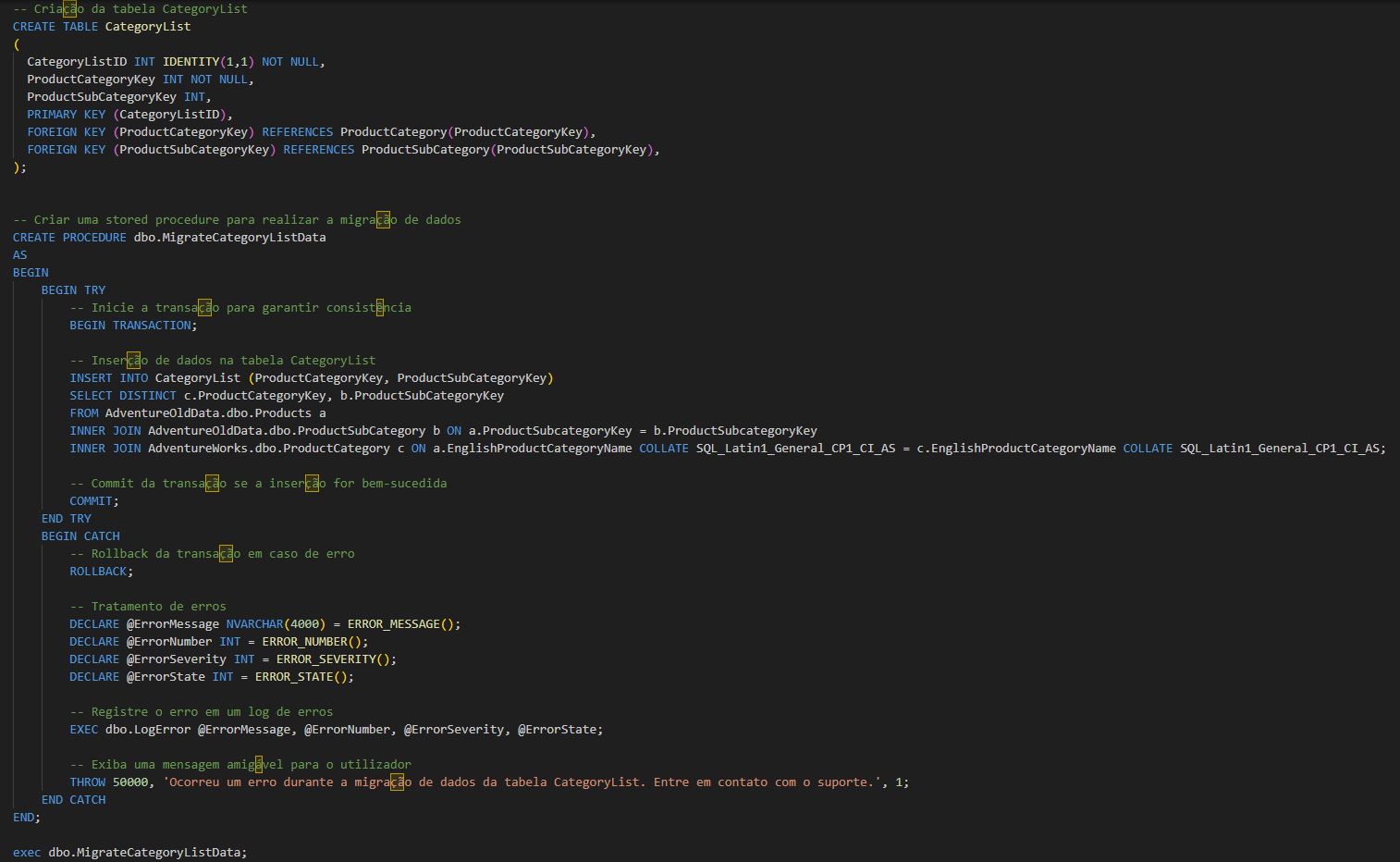


Figure - Percedimento MigrateCategoryListData

Na seguinte imagem mostra a consulta da tabela “CategoryList”.

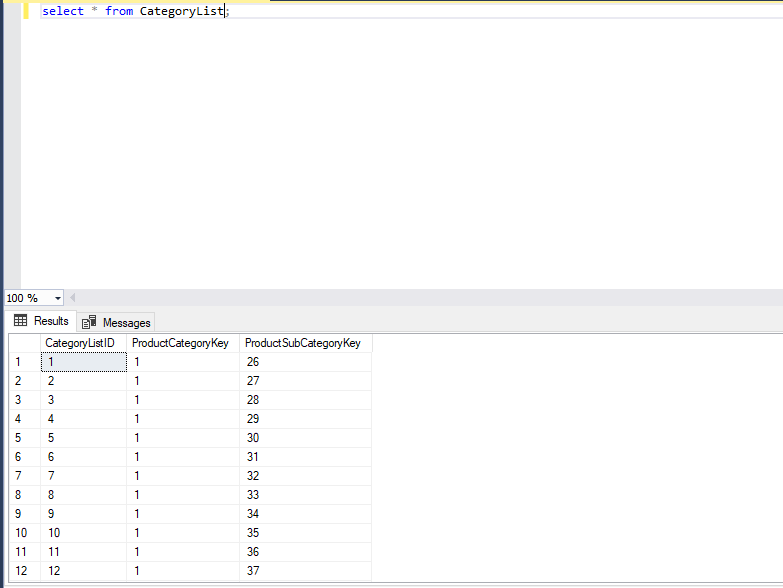


Figure 22 - Tabela CategoryList

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Products” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.

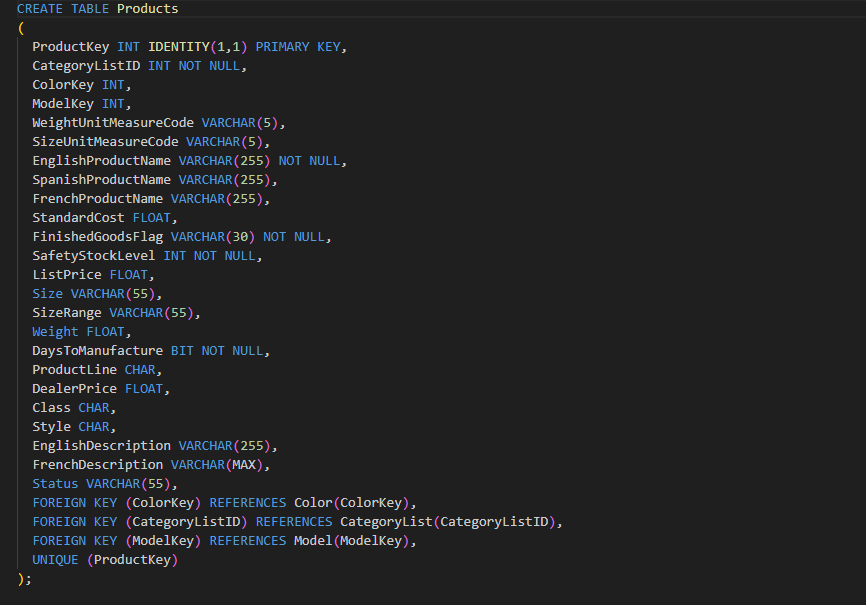


Figure 23 - criação de nova Tabela Products

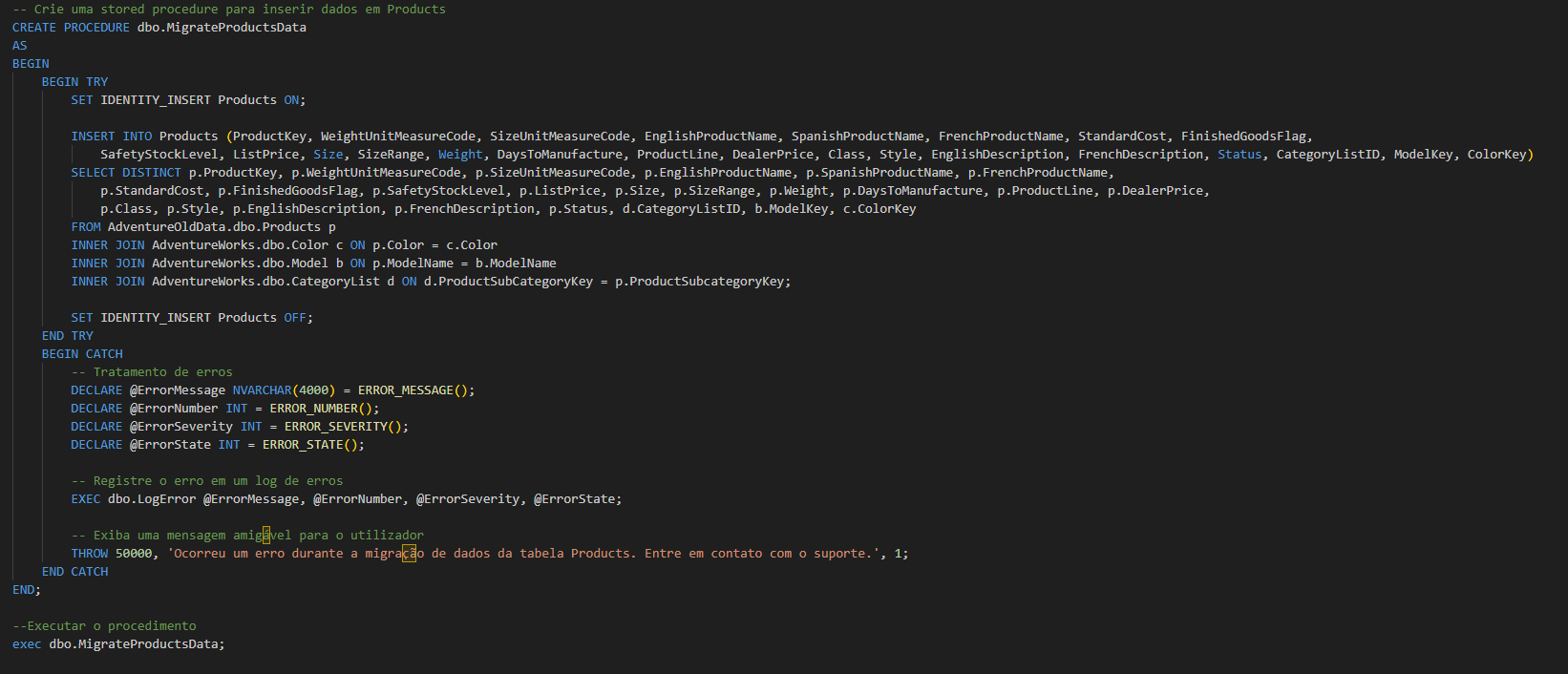


Figure - Percedimento MigrateProductsData

Na seguinte imagem mostra a consulta da tabela “Products”.

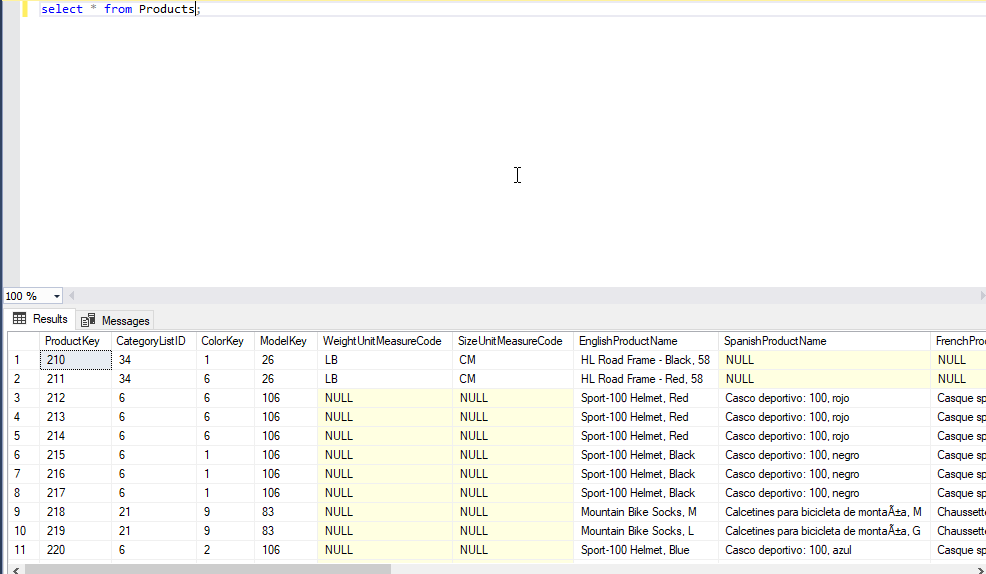


Figure - Tabela Products

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Customer” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.

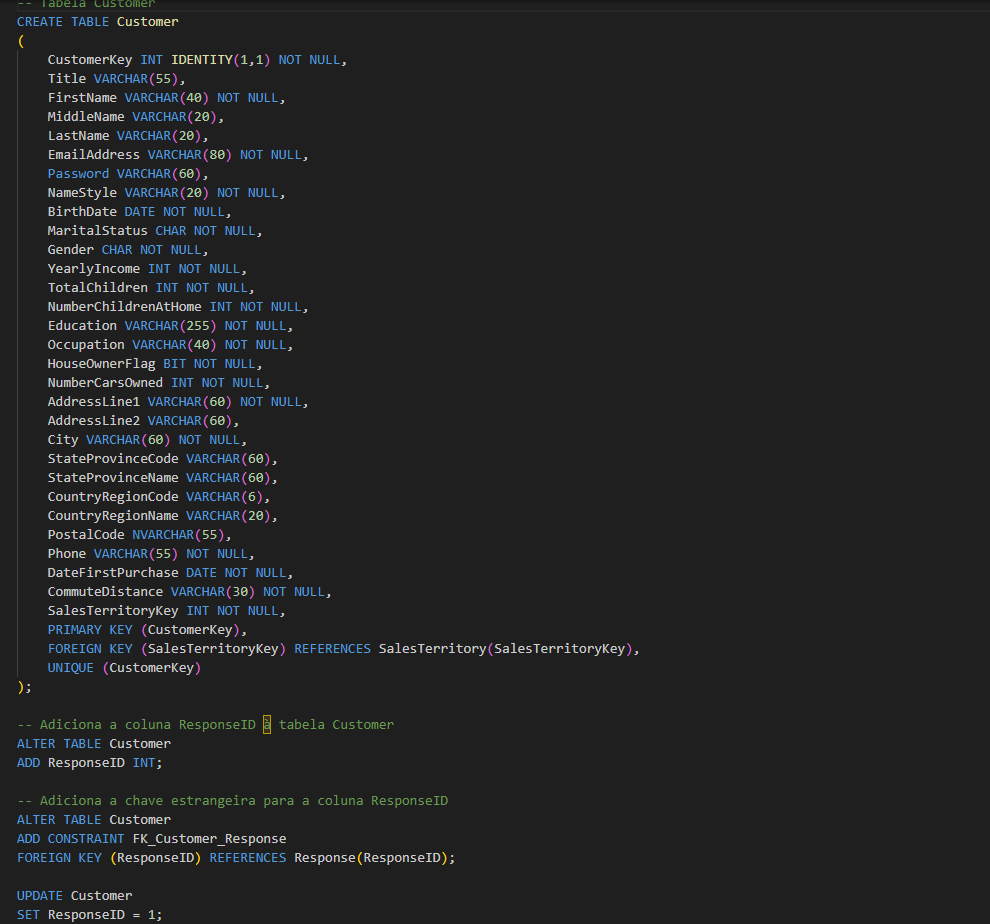


Figure 26 - criação de uma nova tabela Customer

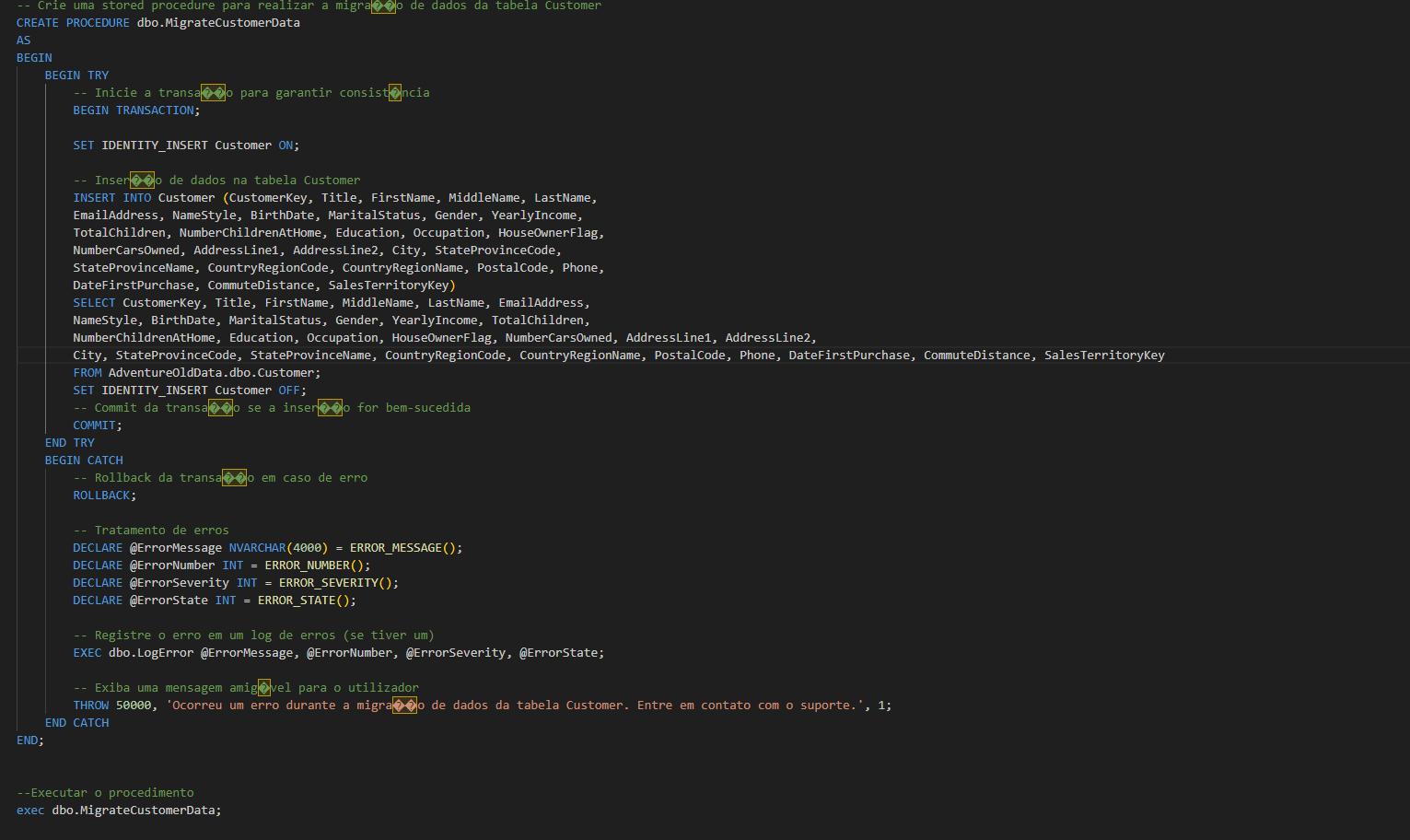


Figure - Percedimento MigrateCustomerData

Na seguinte imagem mostra a consulta da tabela “Customer”.

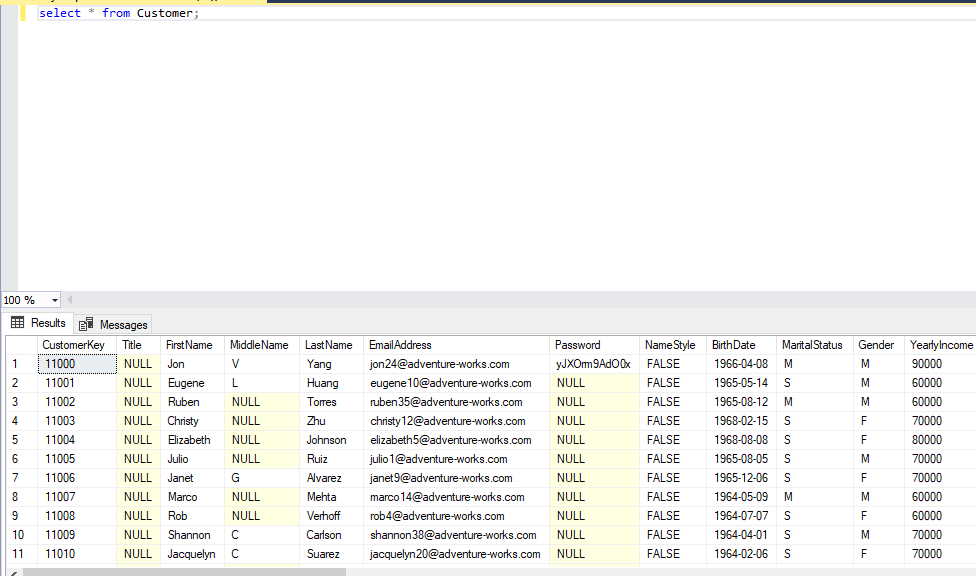


Figure - Tabela Customer

# Programação

## Views

|  |  |
| --- | --- |
| Nome | Descrição |
| *CustomerDetailsView* | *Esta view permite obter a lista de clientes* |
| *CustomerPurchasesView* | *Esta View permite obter a lista de os produtos que os clientes compraram* |
| *HistoricoComprasCliente* | *Esta View permite obter a lista de todos os clientes que fizeram compras* |
| *VendasPorTerritorioView* | *Esta Views permite obter a lista de locais onde foram mais compradas* |

## Functions

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Atributos | Requisito | Descrição |
| *dbo.udf\_getUtilizador* | *@id\_user INT* | *R0#* | *Permite obter informação detalhada sobre um utilizador* |
|  |  |  |  |

## Stored procedures

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Atributos | Requisito | Descrição |
| *dbo.MigrateCategoryListData* | *@nome VARCHAR(50)* | *R0#* | *Este procedimento dbo.MigrateCategoryListData move dados da tabela CategoryList de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateColorData* |  |  | *Este procedimento dbo.MigrateColorData move dados da tabela Color de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateCurrencyData* |  |  | *Este procedimento dbo.MigrateCurrencyData move dados da tabela Currency de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateCustomerData* |  |  | *Este procedimento dbo.MigrateCustomerData move dados da tabela Customer de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateModelData* |  |  | *Este procedimento dbo.MigrateModelData move dados da tabela Model de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateProductCategoryData* |  |  | *Este procedimento dbo.MigrateProductCategoryData move dados da tabela ProductCategory de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateProductsData* |  |  | *Este procedimento dbo.MigrateProductsData move dados da tabela Products de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateProductSubCategoryData* |  |  | *Este procedimento dbo.MigrateProductsSubCategoryData move dados da tabela ProductSubCategory de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateSalesOrder* |  |  | *Este procedimento dbo.MigrateSalesOrder move dados da tabela Sales2 de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateSalesOrderCosts* |  |  | *Este procedimento dbo.MigrateSalesOrderCosts move dados da tabela Sales2 de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.MigrateOrderDetails* |  |  | *Este procedimento dbo.MigrateSalesOrder move dados da tabela Sales2 de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.InsertSalesTerritoryData* |  |  | *Este procedimento dbo.InsertSalesTerritory move dados da tabela SalesTerritory de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.* |
| *dbo.CreateErrorTrigger* | *@TableName NVARCHAR(255)* |  | *O procedimento dbo.CreateErrorTrigger cria gatilhos de erro dinâmicos para tabelas específicas. Recebe o nome da tabela como parâmetro e gera um gatilho que é acionado após operações de inserção, atualização ou exclusão nessa tabela. Esse gatilho simula um erro (que pode ser substituído por lógica real de verificação de erro) e, se condições específicas forem atendidas, regista o erro em um log por meio de um procedimento chamado dbo.ErrorLog. Essa abordagem oferece flexibilidade na criação de gatilhos personalizados para diferentes tabelas da base de dados.* |
| *GetSalesByYear* |  |  | *Este procedimento armazenado chamado GetSalesByYear busca obter informações de vendas por ano. Ele seleciona o ano da data do pedido e a soma total dos valores de vendas da tabela Sales2, agrupando os resultados por ano. Se ocorrer um erro durante a execução da consulta, o bloco CATCH captura o erro e utiliza RAISERROR para sinalizar a ocorrência do erro, exibindo a mensagem de erro* |
| *GetSalesByProductCategory* |  |  | *Este procedimento GetSalesByProductCategory busca obter informações de vendas agregadas por categoria de produto. Ele realiza uma consulta que junta tabelas como Sales2, Products, CategoryList e ProductCategory para calcular a soma total dos valores de vendas (SalesAmount) para cada categoria de produto (EnglishProductCategoryName). Se ocorrer algum erro durante a execução da consulta, o bloco CATCH captura e sinaliza o erro, exibindo a mensagem de erro, sua severidade e estado.* |
| *GetSalesByProductSubCategory* |  |  | *Este procedimento GetSalesByProductSubCategory busca informações de vendas agregadas por subcategoria de produto. Ele realiza uma consulta que junta tabelas como Sales2, Products, CategoryList e ProductSubCategory para calcular a soma total dos valores de vendas (SalesAmount) para cada subcategoria de produto (EnglishProductSubCategoryName). Se algo der errado durante a execução da consulta, o bloco CATCH captura o erro e sinaliza, exibindo detalhes como mensagem de erro, severidade e estado.* |
| *GetProductsSoldByCategoryAndYear* |  |  | *Esse procedimento GetProductsSoldByCategoryAndYear busca informações sobre os produtos vendidos, agregando por categoria de produto e ano de venda. Realiza uma consulta complexa unindo tabelas como Sales2, Products, CategoryList e ProductCategory para contar o número de produtos vendidos por categoria e ano. Em caso de erro durante a execução da consulta, o bloco CATCH captura e sinaliza o erro, exibindo detalhes como a mensagem de erro, a severidade e o estado.* |

## Triggers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome | Tipo | Tabela | Requisito | Descrição |
| *dbo.trg\_Customer\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo.Customer* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Customer, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_CategoryList\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. CategoryList* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela CategoryList, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_Color\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Color* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Color, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_Model\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Model* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Model, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_ProductCategory\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. ProductCategory* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela ProductCategory, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_Products\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Products* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Products, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_ProductSubCategory \_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. ProductSubCategory* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela ProductSubCategory, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.trg\_ Questions \_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Questions* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Questions, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.tr\_Response\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Response* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela response, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.tr\_Sales2\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. Sales2* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Sales2, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.tr\_SalesTerritory\_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. SalesTerritory* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Customer, mas atualmente está configurado apenas para simular um erro.* |
| *dbo.tr\_SalesTerritory \_Error* | *AFTER UPDATE, INSERT, DELETE* | *dbo. SalesTerritory* | *R0#* | *Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela SalesTerritory, mas atualmente está configurado apenas para simular um erro.* |

# Catálogo/Metadados

## Monitorização

|  |  |  |
| --- | --- | --- |
| Nome | Atributos | Descrição |
| *dbo.MetadadosTabelas* | *TabelaID INT IDENTITY(1,1) PRIMARY KEY,*  *NomeTabela NVARCHAR(128),*  *NomeColuna NVARCHAR(128),*  *TipoDados NVARCHAR(128),*  *Tamanho INT,*  *Restricoes NVARCHAR(512),*  *DataAlteracao DATETIME* | *Cria uma tabela chamada MetadadosTabelas para armazenar informações sobre tabelas e colunas numa base de dados. Ele extrai dados das tabelas do sistema para preencher essa tabela com detalhes como nome da tabela, nome da coluna, tipo de dados, tamanho, restrições e data de alteração. É uma forma de registar informações sobre a estrutura das tabelas e colunas presentes no banco de dados.* |
| *dbo.MetadadosTabelas\_View* | *NomeTabela,*  *NomeColuna,*  *TipoDados,*  *Tamanho,*  *Restricoes,*  *DataAlteracao* | *Cria uma view chamada MetadadosTabelas\_View que seleciona informações específicas, como nome da tabela, nome da coluna, tipo de dados, tamanho, restrições e data da última alteração, da tabela MetadadosTabelas. A view retorna apenas os metadados mais recentes das tabelas, usando a informação da data de alteração mais atual disponível na tabela.* |
| *dbo.EstatisticasTabelas* | *TabelaID INT IDENTITY(1,1) PRIMARY KEY,*  *NomeTabela NVARCHAR(128),*  *NomeColuna NVARCHAR(128),*  NumRegistros INT,  EspacoOcupadoKB DECIMAL(10, 2),  DataRegistro DATETIME | *Cria uma tabela chamada EstatisticasTabelas para armazenar estatísticas sobre outras tabelas da base de dados. Em seguida, ele coleta informações sobre o número de registos e o espaço ocupado por cada tabela existente no banco de dados e insere esses dados na tabela EstatisticasTabelas* |

# Índices

## Views

|  |  |
| --- | --- |
| Nome | Descrição |
| *dbo.VendasPorCidade* | *Esta view permite o número de vendas distintas e o total de vendas por cidade e estado.* |
| *dbo.VendasPorAno1* | *Esta view permite as vendas anuais por categoria de produto, mostrando o total de vendas, as vendas do ano anterior, e a taxa de crescimento em relação ao ano anterior para cada categoria de produto.* |
| *Dbo.CalcularNumeroProdutos* | *Esta view lista o número de produtos distintos por cor, utilizando dados das tabelas de detalhes de pedidos, produtos e cores.* |

## Índices

|  |  |  |
| --- | --- | --- |
| Designação | Tabela | Justificação/Consultas |
| *IX\_Customer\_City* | *Customer* | *Está a ser criado na tabela Customer do esquema [dbo]. Esse índice será baseado na coluna City. Isso significa que quando consultas são feitas usando a coluna City na tabela Customer, o índice ajudará a acelerar a busca, tornando as consultas mais rápidas.* |
| *IX\_SalesOrders\_CustomerKey* | *SalesOrders* | *Está a ser criado na tabela SalesOrders do esquema dbo. Este índice será baseado na coluna CustomerKey. Semelhante ao primeiro índice, isso otimiza a busca e recuperação de dados quando consultas são feitas usando a coluna CustomerKey na tabela SalesOrders* |
| *IX\_SalesOrder\_OrderDate* | *SalesOrders* | *Melhora a velocidade de consultas que envolvem a coluna OrderDate nesta tabela.* |
| *IX\_OrderDetails\_SalesOrderNumber* | *OrderDetails* | *Melhora a velocidade de consultas que envolvem a coluna SalesOrderNumber nesta tabela* |
| *IX\_OrderDetails\_ProductKey* | *OrderDetails* | *Melhorar a velocidade de consultas que envolvem a coluna ProductKey nesta tabela.* |
| *IX\_Products\_ProductKey\_CategoryListID* | *Products* | *Este tipo de índice composto éútil para consultas que envolvem ambas as colunas.* |
| *IX\_CategoryList\_CategoryListID* | *CategoryList* | *Melhorar a velocidade de consultas que envolvem a coluna CategoryListID nesta tabela.* |
| *IX\_Color\_ColorKey* | *Color* | *É útil para melhorar a velocidade de consultas que envolvem a coluna ColorKey na tabela Color.* |
| *IX\_Products\_ColorKey\_ProductKey* | *Products* | *É útil para consultas que envolvem ambas as colunas ColorKey e ProductKey na tabela Products* |

## Otimização e Execução de Consultas

*Apresentar os resultados da execução das consultas, com informação sobre as estatísticas e planos de execução, nos seguintes cenários:*

**- Sobre a base de dados otimizada (normalizada) sem índices;**

*Uma imagem com texto, captura de ecrã, menu, número

Descrição gerada automaticamente*

Figure 29 Estatisticas Vendas por cidade (sem indices)

*Uma imagem com texto, file, diagrama, Gráfico

Descrição gerada automaticamente*

Figure 30 planos de execução vendas por cidade (sem indices)

*Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente*

Figure 31 Estatísticas vendas por ano (sem indices)

*Uma imagem com diagrama, file, Gráfico, texto

Descrição gerada automaticamente*

Figure 32 Plano de execução vendas por ano (sem indices)

*Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente*

Figure 33 Estatísticas Calcular Número de Produtos (sem indices)

*Uma imagem com texto, file, diagrama, Gráfico

Descrição gerada automaticamente*

Figure 34 Plano de execução Calcular Número de produtos (sem indices)

**- Sobre a base de dados otimizada (normalizada) com índices.**

*Uma imagem com texto, captura de ecrã, menu, número

Descrição gerada automaticamente*

Figure 35 Estatisticas Vendas por cidade (com indices)

*Uma imagem com texto, file, Tipo de letra, Gráfico

Descrição gerada automaticamente*

Figure 36 Plano de execução Vendas por cidade (com indices)

*Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente*

Figure 37 Estatisticas Vendas por ano (com indices)

*Uma imagem com file, Gráfico, diagrama, captura de ecrã

Descrição gerada automaticamente*

Figure 38 Plano de execução Vendas por ano (com indices)

*Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente*

Figure 39 Estatisticas Calcular número de produtos (com indices)

*Uma imagem com texto, captura de ecrã, file, diagrama

Descrição gerada automaticamente*

Figure 40 Plano de execução Calcular número de produtos (com indices)

# Backup e Recuperação

**Modelo de Recuperação:** Utilização do modelo de recuperação "Full" para a base de dados AdventureWorks, permitindo backups diferenciais e de log.

Como mostra a próxima figura:  
Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figure 41 Modelo de recuperação

**Tipos de Backup:**

**Backup Integral (Full):** Realizado periodicamente para armazenar todos os dados da base de dados. Armazenamento em diferentes pastas, incluindo a pasta do user e a pasta padrão do SQL Server.

**Backup Diferencial:** Realizado para armazenar apenas as alterações desde o último backup integral. Também armazenado em pastas distintas.

**Backup de Log:** Registando transações, útil para a recuperação até um ponto específico no tempo.

Na próxima figura mostro o exemplo dos três backups;

Uma imagem com captura de ecrã, texto

Descrição gerada automaticamente

Figure 42 os três tipos de backups (integral, diferencial, log)

**Cenários de ocorrência de “crash”**

**Exclusão Acidental dos dados:**

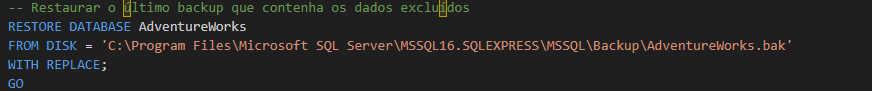
****

Figure 43 exclusão acidental dos dados integral

Este comando mencionado acima restaura a base de dados AdventureWorks a partir do último backup completo disponível.

A opção WITH REPLACE substitui qualquer base de dados existente com o mesmo nome, se houver.



Figure 44 exclusão acidental dos dados diferencial

Este comando mencionado acima aplica um backup diferencial à base de dados restaurado.

A opção WITH NORECOVERY deixa a base de dados num estado que permite a aplicação de logs adicionais, sem trazê-lo completamente online.



Figure 45 exclusão acidental dos dados log

Esse comando mencionado acima aplica um backup de log à base de dados.

Da mesma forma que o comando anterior, utiliza a opção WITH NORECOVERY para deixar a base de dados num estado que permite a aplicação de mais logs de transações.

**Corrupção de dados:**



Figure 46 corrupção de dados

Este comando executa uma verificação de consistência na base de dados AdventureWorks (DBCC CHECKDB). Ele examina a integridade física e lógica dos objetos da base de dados.

A opção WITH NO\_INFOMSGS é usada para suprimir mensagens informativas, tornando o resultado mais conciso.

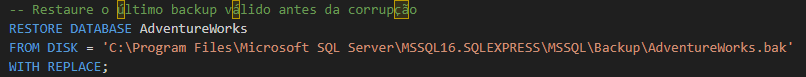


Figure 47 corrupção de dados integral

Esse comando tenta restaurar a base de dados AdventureWorks a partir do último backup válido antes da detecção da corrupção.

A opção WITH REPLACE substitui qualquer base de dados existente com o mesmo nome, se houver.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figure 48 corrupção de dados diferencial e log

Esses comandos visam aplicar backups diferenciais e de log base de dados restaurado a partir do último backup válido.

Ambos os comandos usam a opção WITH NORECOVERY para manter a base de dados num estado "pendente", permitindo a aplicação adicional de logs de transações.

**Supondo que as tabelas de base (exemplo: City, Sales Territory, State Province) não são sujeitas frequentemente a alterações:**

**Teremos de aplicar o Backup Parcial ou Diferenciado:**

Ao invés de fazer backups completos da base de dados inteiro, podemos realizar backups parciais ou diferenciados das tabelas estáticas:

**Backup Parcial**: Fazemos backups apenas das tabelas estáticas que raramente mudam, em vez de fazer um backup completo de toda a base de dados. Isso reduzirá significativamente o tempo e o espaço necessário para os backups.

**Filegroup Backup:** Se essas tabelas estiverem em filegroups separados, podemos fazer backups desses filegroups específicos, permitindo restaurações seletivas.

**Backup Offline:** Podemos realizar backups enquanto essas tabelas estão offline ou em momentos de baixa atividade no Sistema.

# Segurança e Controlo de Acessos

**Utilizadores:** Indivíduos ou entidades com permissão de acesso a um sistema, identificados por nomes exclusivos e autenticados por senhas, tokens ou biometria.

**Roles (Funções):** Conjuntos de permissões atribuídos a utilizadores ou grupos para controlar o acesso a recursos. Simplificam a gestão de acesso ao atribuir conjuntos predefinidos de permissões.

**Schemas:** Estruturas lógicas em base de dados que definem a organização e estrutura dos dados, incluindo tabelas, colunas, restrições e índices. Garantem a integridade e a separação lógica dos dados.

**Encriptação:** Processo de transformar dados em formato ilegível usando algoritmos criptográficos e chaves, protegendo informações confidenciais contra acesso não autorizado. Pode ser aplicada em vários níveis para garantir a segurança dos dados, incluindo armazenamento e transmissão de informações.

## Níveis de acesso à informação

**Administrador:**

Figure 49 Administrador (login)

Cria um login chamado "Tomas" com a senha '12345678'.



Figure 50 Administrador (criação de user)

Cria um user na base de dados associado ao login "Tomas".

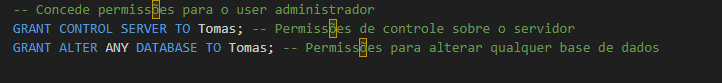


Figure Administrador (conceder permissões para user)

Concede permissão de CONTROLE DO SERVIDOR ao user "Tomas", que concede controle total sobre o servidor SQL.

Concede permissão para ALTERAR QUALQUER BASE DE DADOS ao user "Tomas", permitindo alterações em qualquer base de dados no servidor.



Figure Administrador (conceder permissões para base de dados)

Concede permissões de SELECT, INSERT, UPDATE e DELETE no esquema "dbo" ao user "Tomas", permitindo manipulação de dados nesse esquema.

Concede permissão para EXECUTAR PROCEDURE no esquema "dbo" ao user "Tomas", permitindo a execução de procedimentos armazenados nesse esquema.

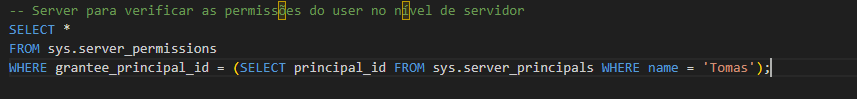


Figure Administrador (conceder permissões nível de servidor)

O código faz uma consulta na tabela sys.server\_permissions para verificar as permissões concedidas ao user "Tomas" no nível do servidor, filtrando pelos privilégios atribuídos a esse user.

**SalesPerson:**

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

Figure 54 SalesPersonLogin

Cria um login chamado "SalesPersonLogin" com a senha '12345678'.

Cria um user na base de dados associado ao login "SalesPersonLogin".

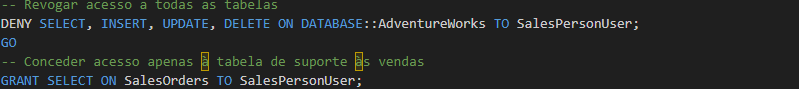


Figure 55 SalesPersonUser

Revoga explicitamente todas as permissões de SELECT, INSERT, UPDATE e DELETE para o user "SalesPersonUser" na base de dados "AdventureWorks". Isso nega o acesso a todas as tabelas a esse user.

Concede permissão de SELECT apenas na tabela "SalesOrders" ao usuário "SalesPersonUser". Essa é a única tabela à qual esse user terá acesso.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figure 56 SalesPersonUser consulta

A consulta é feita nas tabelas do sistema (sys.database\_permissions, sys.database\_principals, sys.objects) para mostrar o tipo de permissões (SELECT, INSERT, etc.) que foram concedidas ao user "SalesPersonUser" na base de dados. A consulta retorna informações sobre as permissões concedidas para esse usuário, mostrando o tipo de objeto (tabela) e o estado das permissões.

**SalesTerritory:**

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figure 57 PacificSales

**CREATE VIEW PacificSales AS**

Cria uma view chamada "PacificSales" que contém os dados da tabela "SalesTerritory" onde o país do território de vendas é 'Australia'.

**CREATE LOGIN SalesTerritoryLogin WITH PASSWORD = '12345678';**

Cria um login chamado "SalesTerritoryLogin" com a senha '12345678'.

**CREATE USER SalesTerritoryUser FOR LOGIN SalesTerritoryLogin;**

Cria um user na base de dados associado ao login "SalesTerritoryLogin".

**GRANT SELECT ON PacificSales TO SalesTerritoryUser;**

Concede permissão de SELECT para a view "PacificSales" ao user "SalesTerritoryUser".

**SELECT \* FROM PacificSales;**

Seleciona os dados da view recém-criada "PacificSales".

**EXEC sp\_helplogins 'SalesTerritoryLogin';**

Executa um procedimento armazenado chamado sp\_helplogins para obter informações sobre o login "SalesTerritoryLogin". Ele retorna detalhes sobre esse login, como suas propriedades e permissões.

**EXEC sp\_helpuser 'SalesTerritoryUser';**

Executa um procedimento armazenado chamado sp\_helpuser para obter informações sobre o user "SalesTerritoryUser". Isso retorna detalhes sobre esse usuário na base de dados, como seu mapeamento para o login correspondente e outras propriedades.

## Encriptação

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figure 58 Encriptação

Escolhemos o algoritmo SHA-512 porque é baseada na sua robustez e na capacidade de produzir um valor de hash que é muito difícil de reverter para a senha original. Além disso, o uso do NEWID() para concatenar com os dados antes de gerar o hash adiciona uma camada de aleatoriedade, tornando os hashes ainda mais únicos e menos suscetíveis a ataques de força bruta.

# Controlo de Concorrência

**Read Uncommitted (Leitura não confirmada):** Nesse nível, uma transação pode ler dados modificados por outras transações que ainda não foram confirmadas. Isso pode levar a problemas de inconsistência de dados.

**Read Committed (Leitura confirmada):** Aqui, uma transação só pode ler dados que foram confirmados por outras transações. Isso ajuda a evitar a leitura de dados sujeitos a alterações não confirmadas, mas ainda pode permitir que problemas de 'leitura suja' ocorram.

**Repeatable Read (Leitura repetível):** Nesse nível, uma transação garante que, uma vez que leu um conjunto de dados, esses dados permanecerão consistentes para leitura subsequente durante a mesma transação, mesmo se outras transações modifiquem os dados.

**Serializable (Serializável):** É o nível mais restritivo. Garante que as transações ocorram em série, como se estivessem sendo executadas uma após a outra, evitando qualquer tipo de anomalia devido a transações concorrentes.

Também existem vários cenários possíveis que podem ocorrer no acesso concorrente ao site suportado pela base de dados como:

**Concorrência na Atualização de Inventário:**

**Cenário:** Vários utilizadores tentam simultaneamente atualizar o inventário de um produto, seja para adicionar ou subtrair unidades do estoque.

**Solução**: Utilizar bloqueios de transação para garantir a consistência do estoque. Isso pode ser feito usando mecanismos como "Locks" ou "Locking hints" no banco de dados para garantir que apenas uma transação possa modificar o estoque de um produto por vez, evitando assim que duas transações atualizem os mesmos dados ao mesmo tempo. Além disso, é possível implementar uma lógica que minimize a duração do bloqueio, permitindo que outros usuários acessem o estoque para leitura, mas bloqueiem a atualização até que a transação de modificação seja concluída.

**Concorrência em Pedidos ou Reservas de Produtos:**

**Cenário**: Vários utilizadores tentam fazer pedidos ou reservas de um mesmo produto com estoque limitado.

**Solução**: Utilizar transações e atomicidade para garantir que a reserva ou pedido de um produto seja uma operação atômica. Isso significa que a operação de verificação do estoque disponível, reserva do item e atualização do estoque devem ser tratadas como uma única unidade. Isso evita que duas transações concorrentes reservem o mesmo item simultaneamente. Além disso, é possível implementar uma lógica de timeout para reservas, liberando automaticamente o item reservado se a transação não for concluída dentro de um tempo determinado, evitando que itens fiquem presos em reservas não finalizadas.

# MongoDB



Figure 59 MongoDB

**SalesByModelProductYear:**

**Coleção:** SalesByModelProductYear

**Estrutura de Dados:**

**model**: String representando o modelo do produto.

**product**: String representando o nome do produto.

**year**: Ano da venda (numérico).

**sales**: Valor das vendas para o modelo e produto específicos naquele ano.

**Justificação**: Esta coleção organiza as vendas por modelo de produto e ano, permitindo uma pesquisa rápida e eficiente por vendas específicas, filtrando pelo modelo e ano desejado. Utiliza uma estrutura simples com campos-chave para identificar vendas específicas e facilitar consultas.

**MonthlyAnnualSalesBalance:**

**Coleção**: MonthlyAnnualSalesBalance

**Estrutura de Dados:**

**year**: Ano das vendas (numérico).

**monthlySales**: Array de objetos, cada um representando as vendas por mês.

**month**: Número do mês.

**sales**: Valor das vendas para aquele mês específico.

**annualBalance**: Valor total das vendas no ano.

**Justificação**: Esta coleção armazena as vendas mensais e o balanço anual. Usa um array de objetos para manter as vendas mensais, permitindo uma análise detalhada mês a mês. O campo annualBalance contém o total de vendas no ano, o que pode ser útil para resumir rapidamente as vendas de um ano específico.

**Importar os dados do sql server para mongodb**

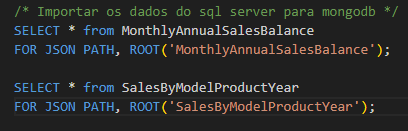


Figure 60 MongoDB importar dados

Esta consulta SQL retorna um conjunto de dados que mostra a quantidade total de pedidos de produtos agrupados por modelo de produto, chave do produto e ano do pedido, formatados em JSON para ser utilizado no mongoDB.

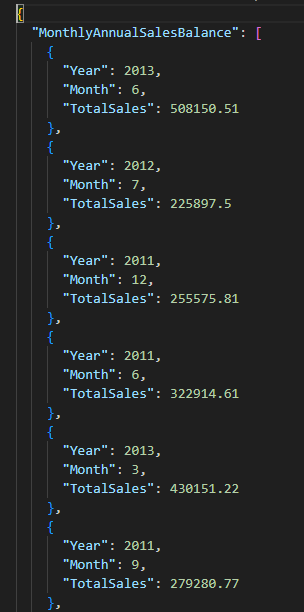


Figure 61 MongoDB importado JSON

Os dados fornecidos representam um conjunto de informações sobre vendas de produtos que foram extraídos da consulta SQL, referida na imagem anterior.

# Descrição da Demonstração

Neste trabalho nós tivemos de aplicar a migração de dados onde tínhamos uma base de dados delegada “AdventureOldData” onde tinha várias tabelas críticas (Currency, Customer, ProductSubCategory, Products, SalesTerritory e sales2), mas o objetivo principal fornecido pelo enunciado do projeto, que nos pedia para nós criarmos as melhorias na estrutura da base de dados “AdventureWorks”, esta seria a base de dados que a empresa ia começar a usar.

Para alcançar este Objetivo tivemos de implementar procedimentos (PROCEDURE) para cada tabela. Cada procedimento tem como objetivo transferir os dados de forma segura e eficiente, assim fazendo com que os dados não sejam corrompidos ou perdidos, mas caso desse erro, esse erro é registado no “LogError” e assim mantendo os dados seguros. Esta mudança permitiu não apenas a movimentação dos dados, mas também fazer melhorias significativas na estrutura da base de dados.

Essa abordagem não apenas cumpriu os requisitos do projeto, mas também possibilitou a realização de melhorias substanciais na qualidade e integridade dos dados na nova base de dados. O facto de usar os procedimentos não só facilitou, mas também fez com que fosse mais fácil fazer manutenção e reaproveitamento do código.

## Script de demonstração

CREATE TABLE ErrorLog (

ErrorLogID INT IDENTITY(1,1) PRIMARY KEY,

ErrorMessage NVARCHAR(4000),

ErrorNumber INT,

ErrorSeverity INT,

ErrorState INT,

Timestamp DATETIME

);

GO

-- Criar uma stored procedure para inserir logs de erro

CREATE PROCEDURE dbo.LogError

@ErrorMessage NVARCHAR(4000),

@ErrorNumber INT,

@ErrorSeverity INT,

@ErrorState INT

AS

BEGIN

-- Inser��o na tabela ErrorLog

INSERT INTO ErrorLog (ErrorMessage, ErrorNumber, ErrorSeverity, ErrorState, Timestamp)

VALUES (@ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState, GETDATE());

END;

GO

CREATE TABLE Model

(

ModelKey INT IDENTITY(1,1) NOT NULL,

ModelName VARCHAR(255),

PRIMARY KEY (ModelKey)

);

GO

-- Crie uma stored procedure para realizar a migra��o de dados para a tabela Model

CREATE PROCEDURE dbo.MigrateModelData

AS

BEGIN

BEGIN TRY

-- Inicie a transa��o para garantir a consist�ncia dos dados

BEGIN TRANSACTION;

-- Inser��o de dados na tabela Model

INSERT INTO Model (ModelName)

SELECT DISTINCT ModelName

FROM AdventureOldData.dbo.Products;

-- Commit da transa��o se a inser��o for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transa��o em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo do erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exibir uma mensagem amig�vel para o utilizador

THROW 50000, 'Ocorreu um erro durante a migra��o de dados da tabela Model. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar o procedimento

exec dbo.MigrateModelData;

GO

CREATE TABLE Color

(

ColorKey INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

Color NVARCHAR(55) NOT NULL

);

GO

-- Crie uma stored procedure para realizar a migração de dados

CREATE PROCEDURE dbo.MigrateColorData

AS

BEGIN

BEGIN TRY

-- Inicie a transação para garantir consistência

BEGIN TRANSACTION;

-- Inserção de dados na tabela Color

INSERT INTO Color (Color)

SELECT DISTINCT Color

FROM AdventureOldData.dbo.Products;

-- Commit da transação se a inserção for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transação em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo do erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exibir uma mensagem amigável para o utilizador

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Color. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

exec dbo.MigrateColorData;

GO

CREATE TABLE ProductCategory

(

ProductCategoryKey INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

EnglishProductCategoryName NVARCHAR(55),

FrenchProductCategoryName NVARCHAR(55) NOT NULL,

SpanishProductCategoryName NVARCHAR(55),

ModifiedDate DATE

);

GO

-- Crie uma stored procedure para realizar a migracao de dados para a tabela ProductCategory

CREATE PROCEDURE dbo.MigrateProductCategoryData

AS

BEGIN

BEGIN TRY

-- Inicia a transacao para garantir a consistencia dos dados

BEGIN TRANSACTION;

-- Inserir de dados na tabela ProductCategory

INSERT INTO ProductCategory (EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName)

SELECT DISTINCT EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName

FROM AdventureOldData.dbo.Products;

-- Commit da transacao se a insercao for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transacao em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registar o erro em um log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exiba uma mensagem amigavel para o utilizador

THROW 50000, 'Ocorreu um erro durante a migracao de dados da tabela ProductCategory. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar o procedimento

exec dbo.MigrateProductCategoryData;

GO

CREATE TABLE Questions

(

QuestionID INT IDENTITY(1,1) NOT NULL,

Question VARCHAR(255) NOT NULL,

PRIMARY KEY (QuestionID),

);

GO

-- insercao de um registo na tabela Questions

INSERT INTO Questions (Question)

VALUES ('Qual é o nome do teu cão?');

GO

CREATE TABLE Response

(

ResponseID INT IDENTITY(1,1) NOT NULL,

Response VARCHAR(255) NOT NULL,

QuestionID INT

PRIMARY KEY (ResponseID),

FOREIGN KEY (QuestionID) REFERENCES Questions(QuestionID)

);

GO

-- inserção de um registo na tabela Response

INSERT INTO Response (Response, QuestionID)

VALUES ('Billy', 1);

GO

CREATE TABLE SalesTerritory

(

SalesTerritoryKey INT IDENTITY(1,1) NOT NULL,

SalesTerritoryRegion NVARCHAR(30) NOT NULL,

SalesTerritoryGroup NVARCHAR(30) NOT NULL,

SalesTerritoryCountry NVARCHAR(30) NOT NULL,

PRIMARY KEY (SalesTerritoryKey),

UNIQUE (SalesTerritoryKey)

);

GO

-- Criar a stored procedure para inserir dados na SalesTerritory

CREATE PROCEDURE dbo.InsertSalesTerritoryData

AS

BEGIN

BEGIN TRY

INSERT INTO SalesTerritory (SalesTerritoryRegion, SalesTerritoryGroup, SalesTerritoryCountry)

SELECT SalesTerritoryRegion, SalesTerritoryGroup, SalesTerritoryCountry

FROM AdventureOldData.dbo.SalesTerritory;

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela SalesTerritory. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

EXEC dbo.InsertSalesTerritoryData;

GO

CREATE TABLE Currency

(

CurrencyKey INT IDENTITY(1,1) PRIMARY KEY,

CurrencyAlternateKey NVARCHAR(5) NOT NULL,

CurrencyName NVARCHAR(55) NOT NULL,

UNIQUE (CurrencyKey),

UNIQUE (CurrencyAlternateKey),

UNIQUE (CurrencyName)

);

GO

-- Crie uma stored procedure para realizar a migra��o de dados da tabela Currency

CREATE PROCEDURE dbo.MigrateCurrencyData

AS

BEGIN

BEGIN TRY

-- Inicie a transa��o para garantir consist�ncia

BEGIN TRANSACTION;

-- Inser��o de dados na tabela Currency

INSERT INTO Currency (CurrencyAlternateKey, CurrencyName)

SELECT CurrencyAlternateKey, CurrencyName

FROM AdventureOldData.dbo.Currency;

-- Commit da transa��o se a inser��o for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transa��o em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo o erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exibir uma mensagem amig�vel para o utilizador

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Currency. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

exec dbo.MigrateCurrencyData;

GO

CREATE TABLE ProductSubCategory

(

ProductSubCategoryKey INT IDENTITY(1,1) NOT NULL,

EnglishProductSubCategoryName VARCHAR(55) NOT NULL,

SpanishProductSubCategoryName VARCHAR(55) NOT NULL,

FrenchProductSubCategoryName VARCHAR(55) NOT NULL,

PRIMARY KEY (ProductSubCategoryKey),

UNIQUE (EnglishProductSubCategoryName),

UNIQUE (SpanishProductSubCategoryName),

UNIQUE (FrenchProductSubCategoryName)

);

GO

-- Criar uma stored procedure para realizar a migracao de dados para a tabela ProductSubCategory

CREATE PROCEDURE dbo.MigrateProductSubCategoryData

AS

BEGIN

BEGIN TRY

-- Inicio da transacao para garantir a consistencia dos dados

BEGIN TRANSACTION;

-- Inser��o de dados na tabela ProductSubCategory

INSERT INTO ProductSubCategory (EnglishProductSubCategoryName, SpanishProductSubCategoryName, FrenchProductSubCategoryName)

SELECT EnglishProductSubCategoryName, SpanishProductSubCategoryName, FrenchProductSubCategoryName

FROM AdventureOldData.dbo.ProductSubCategory;

-- Commit da transacao se a insercao for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transacao em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo do erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exiba uma mensagem amigavel para o utilizador

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela ProductSubCategory. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

exec dbo.MigrateProductSubCategoryData;

GO

CREATE TABLE CategoryList

(

CategoryListID INT IDENTITY(1,1) NOT NULL,

ProductCategoryKey INT NOT NULL,

ProductSubCategoryKey INT,

PRIMARY KEY (CategoryListID),

FOREIGN KEY (ProductCategoryKey) REFERENCES ProductCategory(ProductCategoryKey),

FOREIGN KEY (ProductSubCategoryKey) REFERENCES ProductSubCategory(ProductSubCategoryKey),

);

GO

-- Criar uma stored procedure para realizar a migração de dados

CREATE PROCEDURE dbo.MigrateCategoryListData

AS

BEGIN

BEGIN TRY

-- Inicie a transação para garantir consistência

BEGIN TRANSACTION;

-- Inserção de dados na tabela CategoryList

INSERT INTO CategoryList (ProductCategoryKey, ProductSubCategoryKey)

SELECT DISTINCT c.ProductCategoryKey, b.ProductSubCategoryKey

FROM AdventureOldData.dbo.Products a

INNER JOIN AdventureOldData.dbo.ProductSubCategory b ON a.ProductSubcategoryKey = b.ProductSubcategoryKey

INNER JOIN AdventureWorks.dbo.ProductCategory c ON a.EnglishProductCategoryName COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS = c.EnglishProductCategoryName COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS;

-- Commit da transação se a inserção for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transação em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo do erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exibir uma mensagem amigável para o utilizador

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela CategoryList. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

exec dbo.MigrateCategoryListData;

GO

CREATE TABLE Products

(

ProductKey INT IDENTITY(1,1) PRIMARY KEY,

CategoryListID INT NOT NULL,

ColorKey INT,

ModelKey INT,

WeightUnitMeasureCode VARCHAR(5),

SizeUnitMeasureCode VARCHAR(5),

EnglishProductName VARCHAR(255) NOT NULL,

SpanishProductName VARCHAR(255),

FrenchProductName VARCHAR(255),

StandardCost FLOAT,

FinishedGoodsFlag VARCHAR(30) NOT NULL,

SafetyStockLevel INT NOT NULL,

ListPrice FLOAT,

Size VARCHAR(55),

SizeRange VARCHAR(55),

Weight FLOAT,

DaysToManufacture BIT NOT NULL,

ProductLine CHAR,

DealerPrice FLOAT,

Class CHAR,

Style CHAR,

EnglishDescription VARCHAR(255),

FrenchDescription VARCHAR(MAX),

Status VARCHAR(55),

FOREIGN KEY (ColorKey) REFERENCES Color(ColorKey),

FOREIGN KEY (CategoryListID) REFERENCES CategoryList(CategoryListID),

FOREIGN KEY (ModelKey) REFERENCES Model(ModelKey),

UNIQUE (ProductKey)

);

GO

-- Criar uma stored procedure para inserir dados em Products

CREATE PROCEDURE [dbo].[MigrateProductsData]

AS

BEGIN

BEGIN TRY

SET IDENTITY\_INSERT Products ON;

INSERT INTO Products (ProductKey, WeightUnitMeasureCode, SizeUnitMeasureCode, EnglishProductName, SpanishProductName, FrenchProductName, StandardCost, FinishedGoodsFlag,

SafetyStockLevel, ListPrice, Size, SizeRange, Weight, DaysToManufacture, ProductLine, DealerPrice, Class, Style, EnglishDescription, FrenchDescription, Status, CategoryListID, ModelKey, ColorKey)

SELECT DISTINCT p.ProductKey, p.WeightUnitMeasureCode, p.SizeUnitMeasureCode, p.EnglishProductName, p.SpanishProductName, p.FrenchProductName,

p.StandardCost, p.FinishedGoodsFlag, p.SafetyStockLevel, p.ListPrice, p.Size, p.SizeRange, p.Weight, p.DaysToManufacture, p.ProductLine, p.DealerPrice,

p.Class, p.Style, p.EnglishDescription, p.FrenchDescription, p.Status, d.CategoryListID, b.ModelKey, c.ColorKey

FROM AdventureOldData.dbo.Products p

INNER JOIN AdventureWorks.dbo.Color c ON p.Color COLLATE Latin1\_General\_CI\_AS = c.Color

INNER JOIN AdventureWorks.dbo.Model b ON p.ModelName COLLATE Latin1\_General\_CI\_AS = b.ModelName

INNER JOIN AdventureWorks.dbo.CategoryList d ON d.ProductSubCategoryKey = p.ProductSubcategoryKey;

SET IDENTITY\_INSERT Products OFF;

END TRY

BEGIN CATCH

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registre o erro em um log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exiba uma mensagem amigável para o usuário

THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Products. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar o procedimento

exec dbo.MigrateProductsData;

GO

CREATE TABLE Customer

(

CustomerKey INT IDENTITY(1,1) NOT NULL,

Title VARCHAR(55),

FirstName VARCHAR(40) NOT NULL,

MiddleName VARCHAR(20),

LastName VARCHAR(20),

EmailAddress VARCHAR(80) NOT NULL,

Password NVARCHAR(500),

NameStyle VARCHAR(20) NOT NULL,

BirthDate DATE NOT NULL,

MaritalStatus CHAR NOT NULL,

Gender CHAR NOT NULL,

YearlyIncome INT NOT NULL,

TotalChildren INT NOT NULL,

NumberChildrenAtHome INT NOT NULL,

Education VARCHAR(255) NOT NULL,

Occupation VARCHAR(40) NOT NULL,

HouseOwnerFlag BIT NOT NULL,

NumberCarsOwned INT NOT NULL,

AddressLine1 VARCHAR(60) NOT NULL,

AddressLine2 VARCHAR(60),

City VARCHAR(60) NOT NULL,

StateProvinceCode VARCHAR(60),

StateProvinceName VARCHAR(60),

CountryRegionCode VARCHAR(6),

CountryRegionName VARCHAR(20),

PostalCode NVARCHAR(55),

Phone VARCHAR(55) NOT NULL,

DateFirstPurchase DATE NOT NULL,

CommuteDistance VARCHAR(30) NOT NULL,

SalesTerritoryKey INT NOT NULL,

ResponseID INT,

PRIMARY KEY (CustomerKey),

FOREIGN KEY (ResponseID) REFERENCES Response(ResponseID),

FOREIGN KEY (SalesTerritoryKey) REFERENCES SalesTerritory(SalesTerritoryKey),

UNIQUE (CustomerKey)

);

GO

UPDATE Customer

SET ResponseID = 1;

GO

-- Crie uma stored procedure para realizar a migra��o de dados da tabela Customer

CREATE PROCEDURE dbo.MigrateCustomerData

AS

BEGIN

BEGIN TRY

-- Inicie a transa��o para garantir consist�ncia

BEGIN TRANSACTION;

SET IDENTITY\_INSERT Customer ON;

-- Inser��o de dados na tabela Customer

INSERT INTO Customer (CustomerKey, Title, FirstName, MiddleName, LastName,

EmailAddress, NameStyle, BirthDate, MaritalStatus, Gender, YearlyIncome,

TotalChildren, NumberChildrenAtHome, Education, Occupation, HouseOwnerFlag,

NumberCarsOwned, AddressLine1, AddressLine2, City, StateProvinceCode,

StateProvinceName, CountryRegionCode, CountryRegionName, PostalCode, Phone,

DateFirstPurchase, CommuteDistance, SalesTerritoryKey)

SELECT CustomerKey, Title, FirstName, MiddleName, LastName, EmailAddress,

NameStyle, BirthDate, MaritalStatus, Gender, YearlyIncome, TotalChildren,

NumberChildrenAtHome, Education, Occupation, HouseOwnerFlag, NumberCarsOwned, AddressLine1, AddressLine2,

City, StateProvinceCode, StateProvinceName, CountryRegionCode, CountryRegionName, PostalCode, Phone, DateFirstPurchase, CommuteDistance, SalesTerritoryKey

FROM AdventureOldData.dbo.Customer;

SET IDENTITY\_INSERT Customer OFF;

-- Commit da transa��o se a inser��o for bem-sucedida

COMMIT;

END TRY

BEGIN CATCH

-- Rollback da transa��o em caso de erro

ROLLBACK;

-- Tratamento de erros

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- Registo do erro num log de erros

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

-- Exibir uma mensagem amig�vel para o utilizador

THROW 50000, 'Ocorreu um erro durante a migra��o de dados da tabela Customer. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar o procedimento

exec dbo.MigrateCustomerData;

GO

CREATE TABLE SentEmails

(

SentEmailID INT IDENTITY(1,1) NOT NULL,

Mensage NVARCHAR(MAX),

Timestamp DATETIME,

CustomerKey INT NOT NULL,

PRIMARY KEY (SentEmailID),

FOREIGN KEY (CustomerKey) REFERENCES Customer(CustomerKey)

);

GO

CREATE TABLE Acessos (

ID INT IDENTITY(1,1) PRIMARY KEY,

CustomerKey INT,

DataAcesso DATETIME,

FOREIGN KEY (CustomerKey) REFERENCES Customer(CustomerKey)

);

GO

CREATE TABLE SalesOrders

(

SalesOrderNumber NVARCHAR(255) PRIMARY KEY,

OrderDate DATE NOT NULL,

DueDate DATE NOT NULL,

ShipDate DATE,

CustomerKey int NOT NULL,

SalesTerritoryKey int NOT NULL,

PromotionKey float NOT NULL,

CurrencyKey int NOT NULL,

FOREIGN KEY (CustomerKey) REFERENCES Customer(CustomerKey),

FOREIGN KEY (SalesTerritoryKey) REFERENCES SalesTerritory(SalesTerritoryKey),

FOREIGN KEY (CurrencyKey) REFERENCES Currency(CurrencyKey)

);

GO

-- Criacao a stored procedure para mover os dados

CREATE PROCEDURE dbo.MigrateSalesOrder

AS

BEGIN

BEGIN TRY

--Insercao da nova tabela SalesOrders

INSERT INTO SalesOrders

(

CustomerKey, PromotionKey, CurrencyKey, SalesTerritoryKey,

SalesOrderNumber, OrderDate, DueDate, ShipDate

)

--Consulta para ir buscar os dados a tabela SalesOrders da base de dados antiga

SELECT

CustomerKey, PromotionKey, CurrencyKey, SalesTerritoryKey,

SalesOrderNumber, OrderDate, DueDate, ShipDate

FROM AdventureOldData.dbo.sales2;

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

THROW 50000, 'Ocorreu um erro durante a migra��o de dados da tabela SalesOrders. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar procedimento

EXEC dbo.MigrateSalesOrder;

GO

CREATE TABLE SalesOrderCosts

(

SalesOrderNumber NVARCHAR(255) PRIMARY KEY,

ProductStandardCost float NOT NULL,

TotalProductCost float NOT NULL,

FOREIGN KEY (SalesOrderNumber) REFERENCES SalesOrders(SalesOrderNumber)

);

GO

CREATE PROCEDURE dbo.MigrateSalesOrderCosts

AS

BEGIN

BEGIN TRY

--Insercao da nova tabea SalesOrderCosts

INSERT INTO SalesOrderCosts

(

SalesOrderNumber, ProductStandardCost, TotalProductCost

)

--Consulta para ir buscar os dados a tabela Sales2 da base de dados antiga

SELECT

SalesOrderNumber, ProductStandardCost, TotalProductCost

FROM AdventureOldData.dbo.sales2;

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

THROW 50000, 'Ocorreu um erro durante a migra��o de dados da tabela SalesOrderCosts. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar o procedimento

EXEC dbo.MigrateSalesOrderCosts;

GO

CREATE TABLE OrderDetails

(

SalesOrderNumber NVARCHAR(255) NOT NULL,

SalesOrderLineNumber float NOT NULL,

ProductKey int NOT NULL,

RevisionNumber float NOT NULL,

OrderQuantity float NOT NULL,

UnitPrice float NOT NULL,

ExtendedAmount float NOT NULL,

UnitPriceDiscountPct float NOT NULL,

DiscountAmount float NOT NULL,

ProductStandardCost float NOT NULL,

TotalProductCost float NOT NULL,

SalesAmount float NOT NULL,

TaxAmt float NOT NULL,

Freight float NOT NULL,

CarrierTrackingNumber NVARCHAR(255),

CustomerPONumber NVARCHAR(255),

PRIMARY KEY (SalesOrderNumber, SalesOrderLineNumber),

FOREIGN KEY (SalesOrderNumber) REFERENCES SalesOrders(SalesOrderNumber),

FOREIGN KEY (ProductKey) REFERENCES Products(ProductKey)

);

GO

-- Criacao a stored procedure para mover os dados

CREATE PROCEDURE dbo.MigrateOrderDetails

AS

BEGIN

BEGIN TRY

--Insercao da nova tabela OrderDetails

INSERT INTO OrderDetails

(

ProductKey, SalesOrderNumber, SalesOrderLineNumber, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct,

DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight, CarrierTrackingNumber,

CustomerPONumber

)

--Consulta para ir buscar os dados a tabela OrderDetails da base de dados antiga

SELECT

ProductKey, SalesOrderNumber, SalesOrderLineNumber, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct,

DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight, CarrierTrackingNumber,

CustomerPONumber

FROM AdventureOldData.dbo.sales2;

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT = ERROR\_NUMBER();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

THROW 50000, 'Ocorreu um erro durante a migra��o de dados da tabela OrderDetails. Entre em contato com o suporte.', 1;

END CATCH

END;

GO

--Executar procedimento

EXEC dbo.MigrateOrderDetails;

GO

-- Layout da BD

-- Espaço ocupado por registo de cada tabela;

EXEC sp\_spaceused 'Acessos';

EXEC sp\_spaceused 'CategoryList';

EXEC sp\_spaceused 'Currency';

EXEC sp\_spaceused 'Customer';

EXEC sp\_spaceused 'ErrorLog';

EXEC sp\_spaceused 'EstatisticasTabelas';

EXEC sp\_spaceused 'MetadadosTabelas';

EXEC sp\_spaceused 'Model';

EXEC sp\_spaceused 'ProductCategory';

EXEC sp\_spaceused 'Products';

EXEC sp\_spaceused 'ProductSubCategory';

EXEC sp\_spaceused 'Questions';

EXEC sp\_spaceused 'Response';

EXEC sp\_spaceused 'OrderDetails';

EXEC sp\_spaceused 'SalesOrders';

EXEC sp\_spaceused 'SalesOrderCosts';

EXEC sp\_spaceused 'SalesTerritory';

EXEC sp\_spaceused 'SentEmails';

EXEC sp\_spaceused 'sysdiagrams';

-- Espaço ocupado por cada tabela com o número atual de registos;

SELECT

t.name AS 'Nome da Tabela',

ds.name AS 'Filegroup',

f.type\_desc AS 'Tipo de Filegroup',

df.growth AS 'Taxa de Crescimento (KB)',

df.max\_size \* 8 AS 'Tamanho Máximo (KB)',

SUM(p.rows) AS 'Número de Registros',

SUM(a.total\_pages) \* 8 AS 'Espaço Total (KB)'

FROM sys.tables t

INNER JOIN sys.indexes i ON t.object\_id = i.object\_id

INNER JOIN sys.partitions p ON i.object\_id = p.object\_id AND i.index\_id = p.index\_id

INNER JOIN sys.allocation\_units a ON p.partition\_id = a.container\_id

INNER JOIN sys.data\_spaces ds ON ds.data\_space\_id = i.data\_space\_id

INNER JOIN sys.filegroups f ON f.data\_space\_id = ds.data\_space\_id

INNER JOIN sys.database\_files df ON f.data\_space\_id = df.data\_space\_id

GROUP BY t.name, ds.name, f.type\_desc, df.growth, df.max\_size

ORDER BY SUM(a.total\_pages) DESC;

-- Propor uma taxa de crescimento por tabela

-- Criação de tabela temporaria 'HistoricalGrowth'

WITH HistoricalGrowth AS (

-- Consulta para obter informações sobre o espaço da tabela

SELECT

t.name AS TableName,

MAX(p.rows) AS MaxRows,

MAX(a.total\_pages) \* 8 AS MaxSpaceKB

FROM sys.tables t

INNER JOIN sys.indexes i ON t.object\_id = i.object\_id

INNER JOIN sys.partitions p ON i.object\_id = p.object\_id AND i.index\_id = p.index\_id

INNER JOIN sys.allocation\_units a ON p.partition\_id = a.container\_id

GROUP BY t.name

)

SELECT

TableName,

CAST(

COALESCE(

NULLIF(

ROUND(

AVG(CAST(MaxSpaceKB AS DECIMAL) / NULLIF(CAST(MaxRows AS DECIMAL), 0)),

2

),

0

),

0

) AS DECIMAL(18, 2)

) AS AverageGrowthRate

FROM HistoricalGrowth

GROUP BY TableName;

-- Dimensionar o nº e tipos de acessos.

ALTER DATABASE AdventureWorks

ADD FILEGROUP FG1;

-- Adicionar um ficheiro de dados ao filegroup 'FG\_Dados1' com dimensão inicial e taxa de crescimento específicas

ALTER DATABASE AdventureWorks

ADD FILE

(

NAME = 'DadosFile1',

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\DadosFile1.ndf', -- Caminho para o ficheiro de dados

SIZE = 100MB, -- Dimensão inicial

MAXSIZE = UNLIMITED, -- Dimensão máxima

FILEGROWTH = 50MB -- Taxa de crescimento

)

TO FILEGROUP FG1;

GO

-- Metadados -> Stored Procedure para gerar entradas na tabela de metadados:

CREATE PROCEDURE dbo.GerarEntradasTabelaMetadados

AS

BEGIN

-- Criação de uma tabela de metadados

IF NOT EXISTS (SELECT 1 FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'MetadadosTabelas')

BEGIN

CREATE TABLE dbo.MetadadosTabelas (

TabelaID INT IDENTITY(1,1) PRIMARY KEY,

NomeTabela NVARCHAR(128),

NomeColuna NVARCHAR(128),

TipoDados NVARCHAR(128),

Tamanho INT,

Restricoes NVARCHAR(512),

DataAlteracao DATETIME

);

END

-- Inserir as informações das tabelas e colunas no catalogo

INSERT INTO dbo.MetadadosTabelas (NomeTabela, NomeColuna, TipoDados, Tamanho, Restricoes, DataAlteracao)

SELECT

t.name AS NomeTabela,

c.name AS NomeColuna,

ty.name AS TipoDados,

c.max\_length AS Tamanho,

dc.definition AS Restricoes,

GETDATE() AS DataAlteracao

FROM sys.tables t

INNER JOIN sys.columns c ON t.object\_id = c.object\_id

LEFT JOIN sys.default\_constraints dc ON c.default\_object\_id = dc.object\_id

INNER JOIN sys.types ty ON c.user\_type\_id = ty.user\_type\_id;

END;

GO

-- criar a view MetadadosTabelas\_View

CREATE VIEW dbo.MetadadosTabelas\_View

AS

SELECT

NomeTabela,

NomeColuna,

TipoDados,

Tamanho,

Restricoes,

DataAlteracao

FROM dbo.MetadadosTabelas

WHERE DataAlteracao = (SELECT MAX(DataAlteracao) FROM dbo.MetadadosTabelas);

GO

-- Metadados: Stored Procedure para registrar o n�mero de registros e estimativa de espa�o ocupado:

CREATE PROCEDURE RegistrarEstatisticasTabelas

AS

BEGIN

-- Criar tabela de estatísticas se ainda não existir

IF NOT EXISTS (SELECT 1 FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'EstatisticasTabelas')

BEGIN

CREATE TABLE EstatisticasTabelas (

TabelaID INT IDENTITY(1,1) PRIMARY KEY,

NomeTabela NVARCHAR(128),

NumRegistros INT,

EspacoOcupadoKB DECIMAL(18, 2),

DataRegistro DATETIME

);

END

-- Tabela temporária para armazenar resultados atuais

CREATE TABLE #TempResults (

NomeTabela NVARCHAR(128),

NumRegistros INT,

EspacoOcupadoKB DECIMAL(18, 2)

);

-- Inserir dados na tabela temporária

INSERT INTO #TempResults (NomeTabela, NumRegistros, EspacoOcupadoKB)

SELECT

t.name AS NomeTabela,

SUM(p.rows) AS NumRegistros,

SUM(a.total\_pages) \* 8 / 1024 AS EspacoOcupadoKB

FROM sys.tables t

INNER JOIN sys.indexes i ON t.object\_id = i.object\_id

INNER JOIN sys.partitions p ON i.object\_id = p.object\_id AND i.index\_id = p.index\_id

INNER JOIN sys.allocation\_units a ON p.partition\_id = a.container\_id

GROUP BY t.name;

-- Inserir dados na tabela de estatísticas

INSERT INTO EstatisticasTabelas (NomeTabela, NumRegistros, EspacoOcupadoKB, DataRegistro)

SELECT

NomeTabela,

NumRegistros,

EspacoOcupadoKB,

GETDATE() AS DataRegistro

FROM #TempResults;

-- Apagar tabela temporária

DROP TABLE #TempResults;

END;

GO

--Executar Procedimento GerarEntradasTabelaMetadados

EXEC dbo.GerarEntradasTabelaMetadados;

-- Consultar a Tabela MetadadosTabelas

SELECT \* FROM dbo.MetadadosTabelas;

-- Consultar a VIEW MetadadosTabelas\_View

SELECT \* FROM dbo.MetadadosTabelas\_View;

--Executar Procedimento RegistrarEstatisticasTabelas

EXEC dbo.RegistrarEstatisticasTabelas;

-- Consultar a Tabela EstatisticasTabelas

SELECT \* FROM dbo.EstatisticasTabelas;

CREATE VIEW CustomerPurchasesView AS

SELECT

OD.SalesOrderNumber,

OD.SalesOrderLineNumber,

SO.OrderDate,

SO.DueDate,

SO.ShipDate,

P.EnglishProductName AS ProductName,

P.ListPrice AS ProductPrice,

SO.CustomerKey

FROM OrderDetails AS OD

INNER JOIN SalesOrders AS SO ON OD.SalesOrderNumber = SO.SalesOrderNumber

INNER JOIN Products AS P ON OD.ProductKey = P.ProductKey;

GO

-- Query view ComprasCliente

SELECT \*

FROM CustomerPurchasesView

WHERE CustomerKey = 11001;

GO

-- Criar uma view CustomerDetailsView

CREATE VIEW CustomerDetailsView AS

SELECT \*

FROM Customer;

GO

-- Query para a view CustomerDetailsView

SELECT \*

FROM CustomerDetailsView

WHERE CustomerKey = 11000;

GO

-- Create view VendasPorTerritorioView

CREATE VIEW VendasPorTerritorioView AS

SELECT

ST.SalesTerritoryRegion AS RegiaoDeVendas,

COUNT(SO.SalesOrderNumber) AS TotalDeVendas

FROM SalesOrders AS SO

INNER JOIN SalesTerritory AS ST ON SO.SalesTerritoryKey = ST.SalesTerritoryKey

INNER JOIN OrderDetails AS OD ON SO.SalesOrderNumber = OD.SalesOrderNumber

GROUP BY ST.SalesTerritoryRegion;

GO

-- Query da view VendasPorTerritorioView

SELECT \* FROM VendasPorTerritorioView;

GO

--Criacao da view ProdutosEmEstoqueView

CREATE VIEW ProdutosEmEstoqueView AS

SELECT

P.EnglishProductName AS NomeDoProduto,

P.ListPrice AS PrecoDoProduto,

P.StandardCost AS CustoDoProduto

FROM Products AS P

WHERE P.FinishedGoodsFlag = 'TRUE';

GO

--Query da view ProdutosEmEstoqueView

SELECT \* FROM ProdutosEmEstoqueView;  
GO

-- Criar view HistoricoComprasCliente

CREATE VIEW HistoricoComprasCliente AS

SELECT

OD.SalesOrderNumber,

SO.OrderDate,

P.EnglishProductName AS NomeDoProduto,

P.ListPrice AS PrecoDoProduto,

SO.CustomerKey

FROM OrderDetails AS OD

INNER JOIN SalesOrders AS SO ON OD.SalesOrderNumber = SO.SalesOrderNumber

INNER JOIN Products AS P ON OD.ProductKey = P.ProductKey;

GO

-- Query da view HistoricoComprasCliente

SELECT \*

FROM HistoricoComprasCliente

WHERE CustomerKey = 11000;

GO

--Create procedimento AdicionarAcesso

CREATE PROCEDURE AdicionarAcesso

@UserID INT,

@RecipientEmail NVARCHAR(80),

@EmailSubject NVARCHAR(255)

AS

BEGIN

INSERT INTO Acessos (CustomerKey, DataAcesso)

VALUES (@UserID, CURRENT\_TIMESTAMP);

-- Verificar se a inserção foi bem-sucedida

IF @@ROWCOUNT > 0

BEGIN

PRINT 'Acesso adicionado com sucesso.';

-- recuperação da senha

DECLARE @CharacterSet NVARCHAR(255) = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

DECLARE @NewPassword NVARCHAR(60) = '';

DECLARE @Counter INT = 1;

WHILE @Counter <= 12

BEGIN

SET @NewPassword = @NewPassword + SUBSTRING(@CharacterSet, CAST(RAND() \* LEN(@CharacterSet) + 1 AS INT), 1);

SET @Counter = @Counter + 1;

END

-- Simular o envio do e-mail

PRINT 'Simulating email sending to ' + @RecipientEmail;

PRINT 'Subject: ' + @EmailSubject;

PRINT 'Body: Your new password is ' + @NewPassword;

-- Insira os dados na tabela SentEmails

INSERT INTO SentEmails (Mensage, Timestamp, CustomerKey)

VALUES ('Your new password is ' + @NewPassword, GETDATE(), @UserID);

-- Atualiza a senha na tabela Customer

UPDATE Customer

SET Password = @NewPassword

WHERE CustomerKey = @UserID;

PRINT 'Senha atualizada com sucesso!';

END

ELSE

BEGIN

PRINT 'Erro ao adicionar o acesso.';

END

END

GO

EXEC AdicionarAcesso

@UserID = 11000,

@RecipientEmail = 'jon24@adventure-works.com',

@EmailSubject = 'Password Reset'

GO

-- Criar um procedimento RemoverAcesso

CREATE PROCEDURE RemoverAcesso

@UserID INT

AS

BEGIN

DELETE FROM Acessos

WHERE CustomerKey = @UserID;

-- Verificar se a exclus�o foi bem-sucedida

IF @@ROWCOUNT > 0

BEGIN

PRINT 'Acesso removido com sucesso.';

END

ELSE

BEGIN

PRINT 'Erro ao remover o acesso. Utilizador n�o encontrado.';

END

END

GO

--Executar o procedimento com um id especifico

exec RemoverAcesso 11001

GO

-- Criar procedimento para EditarAcessos

CREATE PROCEDURE EditarAcessos

@UserID INT

AS

BEGIN

UPDATE Acessos

SET DataAcesso = CURRENT\_TIMESTAMP

WHERE CustomerKey = @UserID;

-- Verificar se a atualiza��o foi bem-sucedida

IF @@ROWCOUNT > 0

BEGIN

PRINT 'Acesso atualizado com sucesso.';

END

ELSE

BEGIN

PRINT 'Utilizador nao encontrado ou acesso nao atualizado.';

END

END

GO

--Executar o procedimento com um id especifico

exec EditarAcessos 11000  
GO

-- Criação da procedure sp\_SendPasswordResetEmail

CREATE PROCEDURE sp\_SendPasswordResetEmail

@RecipientEmail NVARCHAR(80),

@EmailSubject NVARCHAR(255),

@CustomerKey INT,

@SecurityQuestion NVARCHAR(255),

@SecurityAnswer NVARCHAR(255)

AS

BEGIN

-- Verificar se o email e o CustomerKey correspondem

IF EXISTS (

SELECT 1

FROM Customer

WHERE EmailAddress = @RecipientEmail

AND CustomerKey = @CustomerKey

)

BEGIN

-- Gera uma senha aleatória

DECLARE @CharacterSet NVARCHAR(255) = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

DECLARE @NewPassword NVARCHAR(60) = '';

DECLARE @Counter INT = 1;

WHILE @Counter <= 12

BEGIN

SET @NewPassword = @NewPassword + SUBSTRING(@CharacterSet, CAST(RAND() \* LEN(@CharacterSet) + 1 AS INT), 1);

SET @Counter = @Counter + 1;

END

-- Simular o envio do e-mail

PRINT 'Simulating email sending to ' + @RecipientEmail;

PRINT 'Subject: ' + @EmailSubject;

PRINT 'Body: Your new password is ' + @NewPassword;

-- Insira os dados na tabela SentEmails

INSERT INTO SentEmails (Mensage, Timestamp, CustomerKey)

VALUES ('Your new password is ' + @NewPassword, GETDATE(), @CustomerKey);

-- Verifica a resposta à pergunta de segurança usando a tabela Response

IF EXISTS (

SELECT 1

FROM Response AS r

JOIN Questions AS q ON r.QuestionID = q.QuestionID

WHERE r.Response = @SecurityAnswer

AND q.Question = @SecurityQuestion

)

BEGIN

-- Simular uma atualização de senha na tabela Customer

UPDATE Customer

SET Password = @NewPassword

WHERE CustomerKey = @CustomerKey;

PRINT 'Senha atualizada com sucesso!';

END

ELSE

BEGIN

PRINT 'Resposta à pergunta de segurança incorreta. A senha não foi alterada.';

END

END

ELSE

BEGIN

PRINT 'O email fornecido não corresponde ao ID do cliente.';

-- Não prosseguir com a redefinição de senha

END

END;

GO

EXEC sp\_SendPasswordResetEmail

@RecipientEmail = 'jon24@adventure-works.com',

@EmailSubject = 'Password Reset',

@CustomerKey = 11000,

@SecurityQuestion = 'Qual é o nome do teu cão?',

@SecurityAnswer = 'Billy';

GO

-- Criação dos índices

CREATE INDEX IX\_Customer\_City ON [dbo].[Customer] ([City]);

CREATE INDEX IX\_SalesOrders\_CustomerKey ON [dbo].[SalesOrders] ([CustomerKey]);

GO

-- Consulta para Pesquisa de vendas por cidade com contagem de vendas corrigida

CREATE VIEW VendasPorCidade AS

SELECT

c.City,

c.StateProvinceName AS Estado,

COUNT(DISTINCT so.SalesOrderNumber) AS NumeroVendas,

SUM(od.SalesAmount) AS TotalVendas

FROM

[dbo].[SalesOrders] so

JOIN

[dbo].[Customer] c ON so.CustomerKey = c.CustomerKey

JOIN

[dbo].[OrderDetails] od ON so.SalesOrderNumber = od.SalesOrderNumber

GROUP BY

c.City, c.StateProvinceName;

GO

Select \* from VendasPorCidade;

------------------------------------------------------------

-- Para o total de vendas calcular a taxa de crescimento de cada ano, face ao ano anterior, por categoria de produto;

-- �ndices

CREATE INDEX IX\_SalesOrder\_OrderDate ON SalesOrders (OrderDate);

CREATE INDEX IX\_OrderDetails\_SalesOrderNumber ON OrderDetails (SalesOrderNumber);

CREATE INDEX IX\_OrderDetails\_ProductKey ON OrderDetails (ProductKey);

CREATE INDEX IX\_Products\_ProductKey\_CategoryListID ON Products (ProductKey, CategoryListID);

CREATE INDEX IX\_CategoryList\_CategoryListID ON CategoryList (CategoryListID);

GO

-- Consulta para calcular a taxa de crescimento de vendas por categoria de produto

CREATE VIEW VendasPorAno1 AS

WITH VendasPorAno AS (

SELECT

YEAR(SO.OrderDate) AS Ano,

CL.ProductCategoryKey,

SUM(OD.SalesAmount) AS TotalVendas

FROM

SalesOrders AS SO

JOIN

OrderDetails AS OD ON SO.SalesOrderNumber = OD.SalesOrderNumber

JOIN

Products AS P ON OD.ProductKey = P.ProductKey

JOIN

CategoryList AS CL ON P.CategoryListID = CL.CategoryListID

GROUP BY

YEAR(SO.OrderDate), CL.ProductCategoryKey

)

SELECT

Ano,

ProductCategoryKey,

TotalVendas,

LAG(TotalVendas) OVER (PARTITION BY ProductCategoryKey ORDER BY Ano) AS VendasAnoAnterior,

CASE

WHEN LAG(TotalVendas) OVER (PARTITION BY ProductCategoryKey ORDER BY Ano) = 0 THEN NULL

ELSE 100 \* (TotalVendas - LAG(TotalVendas) OVER (PARTITION BY ProductCategoryKey ORDER BY Ano)) / LAG(TotalVendas) OVER (PARTITION BY ProductCategoryKey ORDER BY Ano)

END AS TaxaCrescimento

FROM

VendasPorAno;

GO

Select \* from VendasPorAno1;

---------------------------------------------------------------------

-- N�mero de produtos nas vendas por cor.

-- Índices

CREATE INDEX IX\_Color\_ColorKey ON [dbo].[Color] ([ColorKey]);

CREATE INDEX IX\_Products\_ColorKey\_ProductKey ON [dbo].[Products] ([ColorKey], [ProductKey]);

GO

-- Consulta para calcular o número de produtos nas vendas por cor

Create VIEW CalcularNumeroProdutos AS

SELECT

c.Color,

COUNT(DISTINCT p.ProductKey) AS NumeroProdutos

FROM

[dbo].[OrderDetails] od

JOIN

[dbo].[Products] p ON od.ProductKey = p.ProductKey

JOIN

[dbo].[Color] c ON p.ColorKey = c.ColorKey

GROUP BY

c.Color;

GO

Select \* from CalcularNumeroProdutos;

GO

/\*

2.2 Backup e Recupera��o

-- modelo de recuperação como 'Full' para permitir backups de log \*/

ALTER DATABASE AdventureWorks SET RECOVERY FULL;

GO

/\*

Backup Integral (Full):

\*/

BACKUP DATABASE AdventureWorks TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorks.bak';

GO

/\*

Backup Diferencial:

\*/

BACKUP DATABASE AdventureWorks TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorksDiferencial.bak' WITH DIFFERENTIAL;

GO

/\*

Backup de Log:

\*/

BACKUP LOG AdventureWorks TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\BackLog.trn';

GO

-- Restaurar o último backup que contenha os dados excluídos

RESTORE DATABASE AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorks.bak'

WITH REPLACE;

GO

-- Aplicar backups diferenciais e de log

RESTORE DATABASE AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorksDiferencial.bak'

WITH NORECOVERY;

GO

RESTORE LOG AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\BackLog.trn'

WITH NORECOVERY;

GO

-- Identifique a extensão da corrupção e os objetos afetados

DBCC CHECKDB ('AdventureWorks') WITH NO\_INFOMSGS;

GO

-- Restaure o último backup válido antes da corrupção

RESTORE DATABASE AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorks.bak'

WITH REPLACE;

GO

-- Aplicar backups diferenciais e de log

RESTORE DATABASE AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\AdventureWorksDiferencial.bak'

WITH NORECOVERY;

GO

RESTORE LOG AdventureWorks

FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\BackLog.trn'

WITH NORECOVERY;

GO

-- Criação de um login para o administrador

CREATE LOGIN Tomas WITH PASSWORD = '12345678';

GO

-- Criação de um user na base de dados associado ao login do administrador

CREATE USER Tomas FOR LOGIN Tomas;

GO

-- Concede permissões para o user administrador

GRANT CONTROL SERVER TO Tomas; -- Permissões de controle sobre o servidor

GRANT ALTER ANY DATABASE TO Tomas; -- Permissões para alterar qualquer base de dados

GO

-- Concede permissões específicas na base de dados

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA :: dbo TO Tomas; -- Permissões de manipulação de dados no esquema 'dbo'

GRANT EXECUTE ON SCHEMA :: dbo TO Tomas; -- Permissões para executar procedimentos armazenados no esquema 'dbo'

GO

-- Server para verificar as permissões do user no nível de servidor

SELECT \*

FROM sys.server\_permissions

WHERE grantee\_principal\_id = (SELECT principal\_id FROM sys.server\_principals WHERE name = 'Tomas');

GO

CREATE LOGIN SalesPersonLogin WITH PASSWORD = '12345678';

CREATE USER SalesPersonUser FOR LOGIN SalesPersonLogin;

GO

-- Revogar acesso a todas as tabelas

DENY SELECT, INSERT, UPDATE, DELETE ON DATABASE::AdventureWorks TO SalesPersonUser;

GO

-- Conceder acesso apenas à tabela de suporte às vendas

GRANT SELECT ON SalesOrders TO SalesPersonUser;

GO

SELECT

princ.type\_desc AS [Type],

perm.permission\_name,

perm.state\_desc AS [Permission State],

obj.type\_desc AS [Object Type],

obj.name AS [Object Name]

FROM sys.database\_permissions perm

LEFT JOIN sys.database\_principals princ ON perm.grantee\_principal\_id = princ.principal\_id

LEFT JOIN sys.objects obj ON perm.major\_id = obj.object\_id

WHERE princ.name = 'SalesPersonUser';

GO

-- Criar a view para o território Australia

CREATE VIEW PacificSales AS

SELECT \* FROM SalesTerritory WHERE SalesTerritoryCountry = 'Australia';

GO

-- Criar conta SalesTerritory

CREATE LOGIN SalesTerritoryLogin WITH PASSWORD = '12345678';

CREATE USER SalesTerritoryUser FOR LOGIN SalesTerritoryLogin;

GO

-- Conceder acesso à view específica

GRANT SELECT ON PacificSales TO SalesTerritoryUser;

GO

-- Selecionar os dados da view 'PacificSales'

SELECT \* FROM PacificSales;

GO

-- Para um login específico

EXEC sp\_helplogins 'SalesTerritoryLogin';

GO

-- Para um user específico

EXEC sp\_helpuser 'SalesTerritoryUser';

GO

UPDATE Customer

SET Password = HASHBYTES('SHA2\_512', CONCAT(Password, NEWID()));

GO

UPDATE Questions

SET Question = HASHBYTES('SHA2\_512', CONCAT(Question, NEWID()));

GO

-- Criar a chave mestra se ainda n�o existir

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = '##MS\_DatabaseMasterKey##')

BEGIN

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123';

END;

GO

-- Abrir a chave mestra na sess�o atual

OPEN MASTER KEY DECRYPTION BY PASSWORD = '123';

GO

-- Criar a chave sim�trica se n�o existir

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = 'MySymmetricKey')

BEGIN

CREATE SYMMETRIC KEY MySymmetricKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY PASSWORD = '123';

END

ELSE

BEGIN

-- Se j� existir, abra a chave sim�trica

OPEN SYMMETRIC KEY MySymmetricKey

DECRYPTION BY PASSWORD = '123';

END;

GO

-- Criar o certificado se n�o existir

IF NOT EXISTS (SELECT \* FROM sys.certificates WHERE name = 'MyCertificate')

BEGIN

CREATE CERTIFICATE MyCertificate

WITH SUBJECT = 'Certificado';

END;

GO

-- Inserindo dados criptografados na tabela Response

UPDATE Response

SET Response = HASHBYTES('SHA2\_512', CONCAT(Response, NEWID()));

GO

-- Fechar a chave sim�trica

CLOSE SYMMETRIC KEY MySymmetricKey;

GO

-- Fechar a chave mestra

CLOSE MASTER KEY;

select \* from Customer;

select \* from Questions;

select \* from Response;

GO

-- User 1 - Transa��o A

BEGIN TRANSACTION;

-- Information -> SalesOrders

DECLARE @SalesOrderNumber1 VARCHAR(255) = 'SO50000';

DECLARE @OrderDate1 DATE = GETDATE();

DECLARE @DueDate1 DATE;

DECLARE @ShipDate1 DATE;

DECLARE @CustomerKey1 INT = 11003;

DECLARE @SalesTerritoryKey1 INT = 9;

DECLARE @PromotionKey1 INT = 1;

DECLARE @CurrencyKey1 INT = 6;

-- Information -> OrderDetails

DECLARE @ProductKey1 INT = 210;

DECLARE @OrderQuantity1 INT = 2; -- Definir um valor para OrderQuantity

DECLARE @UnitPrice1 FLOAT;

DECLARE @UnitPriceDiscountPct1 FLOAT = 5.0;

DECLARE @SalesOrderLineNumber1 INT = 1;

DECLARE @RevisionNumber1 INT = 1;

DECLARE @ExtendedAmount1 FLOAT;

DECLARE @DiscountAmount1 FLOAT;

DECLARE @ProductStandardCost1 FLOAT; -- SalesOrderCosts

DECLARE @TotalProductCost1 FLOAT; -- SalesOrderCosts

DECLARE @SalesAmount1 FLOAT;

DECLARE @TaxAmt1 FLOAT;

DECLARE @Freight1 FLOAT;

-- Obter informa��es do produto

SELECT

@UnitPrice1 = ISNULL(ListPrice, 0)

FROM

Products

WHERE

ProductKey = @ProductKey1;

BEGIN TRY

-- Iniciar transa��o A

BEGIN TRANSACTION;

-- Cont -> SalesOrders

IF NOT EXISTS (SELECT 1 FROM dbo.SalesOrders WHERE SalesOrderNumber = @SalesOrderNumber1)

BEGIN

-- SalesOrder n�o existe, prosseguir com a inser��o

SET @DueDate1 = DATEADD(YEAR, 1, @OrderDate1);

SET @ShipDate1 = DATEADD(DAY, -1, @DueDate1);

-- Cont -> OrderDetails

SET @ExtendedAmount1 = @UnitPrice1 \* @OrderQuantity1;

SET @DiscountAmount1 = 0.0;

SET @ProductStandardCost1 = 5.0;

SET @TotalProductCost1 = @ExtendedAmount1 + @DiscountAmount1 + @ProductStandardCost1;

SET @SalesAmount1 = @ExtendedAmount1 - @DiscountAmount1;

SET @TaxAmt1 = 2.0;

SET @Freight1 = 3.0;

-- Utilizar bloco TRY-CATCH para controle de erros

BEGIN TRY

-- Inserir na tabela SalesOrders

INSERT INTO dbo.SalesOrders (SalesOrderNumber, OrderDate, DueDate, ShipDate, CustomerKey, SalesTerritoryKey, PromotionKey, CurrencyKey)

VALUES (@SalesOrderNumber1, @OrderDate1, @DueDate1, @ShipDate1,@CustomerKey1,@SalesTerritoryKey1, @PromotionKey1, @CurrencyKey1);

-- Verificar se a inser��o foi bem-sucedida

DECLARE @RowCount1 INT = @@ROWCOUNT;

IF @RowCount1 > 0

BEGIN

-- PRINT 'Inser��o na SalesOrders bem-sucedida.';

-- Inserir na tabela OrderDetails

INSERT INTO dbo.OrderDetails(SalesOrderNumber, SalesOrderLineNumber, ProductKey, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct, DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight)

VALUES (@SalesOrderNumber1, @SalesOrderLineNumber1, @ProductKey1, @RevisionNumber1, @OrderQuantity1, @UnitPrice1, @ExtendedAmount1, @UnitPriceDiscountPct1, @DiscountAmount1, @ProductStandardCost1, @TotalProductCost1, @SalesAmount1, @TaxAmt1, @Freight1);

-- Verificar se a inser��o na OrderDetails foi bem-sucedida

DECLARE @RowCountOrderDetails1 INT = @@ROWCOUNT;

IF @RowCountOrderDetails1 > 0

BEGIN

-- PRINT 'Inser��o na OrderDetails bem-sucedida.';

-- add SalesOrderConts

INSERT INTO dbo.SalesOrderCosts(SalesOrderNumber, ProductStandardCost, TotalProductCost)

VALUES(@SalesOrderNumber1, @ProductStandardCost1, @TotalProductCost1);

END

ELSE

BEGIN

PRINT 'Falha na inser��o na OrderDetails.';

END

END

ELSE

BEGIN

PRINT 'Falha na inser��o na SalesOrders.';

END

-- Efetivar a transa��o A

COMMIT;

END TRY

BEGIN CATCH

-- Em caso de erro, fazer rollback da transa��o A

ROLLBACK;

PRINT ERROR\_MESSAGE();

END CATCH;

END

ELSE

BEGIN

-- SalesOrder j� existe, imprimir mensagem de erro

PRINT 'SalesOrderNumber1 j� existe na tabela SalesOrders.';

END

END TRY

BEGIN CATCH

-- Lidar com erros

PRINT ERROR\_MESSAGE();

END CATCH;

GO

-- User 2 - Transa��o B

DECLARE @SalesOrderNumber2 VARCHAR(255) = 'SO80000';

DECLARE @OrderDate2 DATE = GETDATE();

DECLARE @DueDate2 DATE;

DECLARE @ShipDate2 DATE;

DECLARE @CustomerKey2 INT = 11003;

DECLARE @SalesTerritoryKey2 INT = 9;

DECLARE @PromotionKey2 INT = 1;

DECLARE @CurrencyKey2 INT = 6;

-- Information -> OrderDetails

DECLARE @ProductKey2 INT = 210;

DECLARE @OrderQuantity2 INT = 2; -- Definir um valor para OrderQuantity

DECLARE @UnitPrice2 FLOAT;

DECLARE @UnitPriceDiscountPct2 FLOAT = 5.0;

DECLARE @SalesOrderLineNumber2 INT = 1;

DECLARE @RevisionNumber2 INT = 1;

DECLARE @ExtendedAmount2 FLOAT;

DECLARE @DiscountAmount2 FLOAT;

DECLARE @ProductStandardCost2 FLOAT; -- SalesOrderCosts

DECLARE @TotalProductCost2 FLOAT; -- SalesOrderCosts

DECLARE @SalesAmount2 FLOAT;

DECLARE @TaxAmt2 FLOAT;

DECLARE @Freight2 FLOAT;

-- Obter informa��es do produto

SELECT

@UnitPrice2 = ISNULL(ListPrice, 0)

FROM

Products

WHERE

ProductKey = @ProductKey2;

BEGIN TRY

-- Iniciar transa��o B

BEGIN TRANSACTION;

-- Cont -> SalesOrders

IF NOT EXISTS (SELECT 1 FROM dbo.SalesOrders WHERE SalesOrderNumber = @SalesOrderNumber2)

BEGIN

-- SalesOrder n�o existe, prosseguir com a inser��o

SET @DueDate2 = DATEADD(YEAR, 1, @OrderDate2);

SET @ShipDate2 = DATEADD(DAY, -1, @DueDate2);

-- Cont -> OrderDetails

SET @ExtendedAmount2 = @UnitPrice2 \* @OrderQuantity2;

SET @DiscountAmount2 = 0.0;

SET @ProductStandardCost2 = 6.0;

SET @TotalProductCost2 = @ExtendedAmount2 + @DiscountAmount2 + @ProductStandardCost2;

SET @SalesAmount2 = @ExtendedAmount2 - @DiscountAmount2;

SET @TaxAmt2 = 3.0;

SET @Freight2 = 4.0;

-- Utilizar bloco TRY-CATCH para controle de erros

BEGIN TRY

-- Inserir na tabela SalesOrders

INSERT INTO dbo.SalesOrders (SalesOrderNumber, OrderDate, DueDate, ShipDate, CustomerKey, SalesTerritoryKey, PromotionKey, CurrencyKey)

VALUES (@SalesOrderNumber2, @OrderDate2, @DueDate2, @ShipDate2,@CustomerKey2,@SalesTerritoryKey2, @PromotionKey2, @CurrencyKey2);

DECLARE @RowCount1 INT = @@ROWCOUNT;

IF @RowCount1 > 0

BEGIN

-- Inserir na tabela OrderDetails

INSERT INTO dbo.OrderDetails(SalesOrderNumber, SalesOrderLineNumber, ProductKey, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct, DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight)

VALUES (@SalesOrderNumber2, @SalesOrderLineNumber2, @ProductKey2, @RevisionNumber2, @OrderQuantity2, @UnitPrice2, @ExtendedAmount2, @UnitPriceDiscountPct2, @DiscountAmount2, @ProductStandardCost2, @TotalProductCost2, @SalesAmount2, @TaxAmt2, @Freight2);

DECLARE @RowCountOrderDetails1 INT = @@ROWCOUNT;

IF @RowCountOrderDetails1 > 0

BEGIN

PRINT 'Inser��o na OrderDetails bem-sucedida.';

-- add SalesOrderConts

INSERT INTO dbo.SalesOrderCosts(SalesOrderNumber, ProductStandardCost, TotalProductCost)

VALUES(@SalesOrderNumber2, @ProductStandardCost2, @TotalProductCost2);

END

ELSE

BEGIN

PRINT 'Falha na inser��o na OrderDetails.';

END

END

ELSE

BEGIN

PRINT 'Falha na inser��o na SalesOrders.';

END

-- Efetivar a transa��o B

COMMIT;

END TRY

BEGIN CATCH

-- Em caso de erro, fazer rollback da transa��o B

ROLLBACK;

PRINT '1.';

PRINT ERROR\_MESSAGE();

END CATCH;

END

ELSE

BEGIN

-- SalesOrder j� existe, imprimir mensagem de erro

PRINT 'SalesOrderNumber2 j� existe na tabela SalesOrders.';

END

END TRY

BEGIN CATCH

-- Lidar com erros

PRINT ERROR\_MESSAGE();

END CATCH;

COMMIT;

-- Selecionar resultados

SELECT \* FROM dbo.OrderDetails where SalesOrderNumber = 'SO50000' or SalesOrderNumber = 'SO80000';

GO

-- User 1 TRANSACTION A

BEGIN TRANSACTION;

-- variaveis

DECLARE @ProductKey1 INT = 210;

DECLARE @NewUnitPrice1 FLOAT = 10.0;

-- Verificar se a condição ainda é válida

IF EXISTS (SELECT 1 FROM Products WHERE ProductKey = @ProductKey1)

BEGIN

-- Alterar o preço do Produto 1

UPDATE Products

SET ListPrice = @NewUnitPrice1

WHERE ProductKey = @ProductKey1;

-- Verificar se a alteração está correta

SELECT \* FROM Products WHERE ProductKey = @ProductKey1;

-- Confirmar a transação do User 1

COMMIT;

END

ELSE

BEGIN

-- Se a condição não for mais válida, reverter a transação do User 1

ROLLBACK;

PRINT 'Condição inválida para User 1. Transação revertida.';

END

-- User 2 TRANSACTION B

BEGIN TRANSACTION;

-- variaveis

DECLARE @ProductKey2 INT = 210;

DECLARE @NewUnitPrice2 FLOAT = 15.0;

-- Verificar se a condição ainda é válida

IF EXISTS (SELECT 1 FROM Products WHERE ProductKey = @ProductKey2)

BEGIN

-- Alterar o preço do Produto 2

UPDATE Products

SET ListPrice = @NewUnitPrice2

WHERE ProductKey = @ProductKey2;

-- Verificar se a alteração está correta

SELECT \* FROM Products WHERE ProductKey = @ProductKey2;

-- Confirmar a transação do User 2

COMMIT;

END

ELSE

BEGIN

-- Se a condição não for mais válida, reverter a transação do User 2

ROLLBACK;

PRINT 'Condição inválida para User 2. Transação revertida.';

END

GO

-- Parâmetro de data para o início da transação

DECLARE @inicio\_transacao DATETIME = GETDATE();

-- Declarar variáveis para armazenar os totais de vendas

DECLARE @total\_vendas\_user1 MONEY;

DECLARE @total\_vendas\_user2 MONEY;

-- Iniciar a transação para User 1

BEGIN TRANSACTION;

-- Calcular o total das vendas até o início da transação para User 1

SELECT @total\_vendas\_user1 = SUM(p.ListPrice)

FROM [AdventureWorks].[dbo].[SalesOrders] so

JOIN [AdventureWorks].[dbo].[OrderDetails] od ON so.SalesOrderNumber = od.SalesOrderNumber

JOIN [AdventureWorks].[dbo].[Products] p ON od.ProductKey = p.ProductKey

WHERE so.OrderDate <= @inicio\_transacao

AND so.CustomerKey = 11000;

-- Confirmar a transação para User 1

COMMIT;

-- Iniciar a transação para User 2

BEGIN TRANSACTION;

-- Calcular o total das vendas até o início da transação para User 2

SELECT @total\_vendas\_user2 = SUM(p.ListPrice)

FROM [AdventureWorks].[dbo].[SalesOrders] so

JOIN [AdventureWorks].[dbo].[OrderDetails] od ON so.SalesOrderNumber = od.SalesOrderNumber

JOIN [AdventureWorks].[dbo].[Products] p ON od.ProductKey = p.ProductKey

WHERE so.OrderDate <= @inicio\_transacao

AND so.CustomerKey = 11001; -- Supondo que CustomerKey 2 seja associado ao User 2

-- Confirmar a transação para User 2

COMMIT;

-- Exibir os resultados formatados

SELECT 'User 1 Total Vendas' AS user\_label, CONVERT(VARCHAR, @total\_vendas\_user1, 1) AS total\_vendas\_user11000;

SELECT 'User 2 Total Vendas' AS user\_label, CONVERT(VARCHAR, @total\_vendas\_user2, 1) AS total\_vendas\_user11001;

GO

-- 1 Criacao da base de dados

use AdventurWorksWeb

--3

-- Criar uma tabela para armazenar os resultados

CREATE TABLE SalesByModelProductYear (

ModelKey INT,

ProductKey INT,

[Year] INT,

TotalQuantity INT

)

GO

-- Popular a tabela temporária com as vendas por modelo, produto e ano

INSERT INTO SalesByModelProductYear (ModelKey, ProductKey, [Year], TotalQuantity)

SELECT P.ModelKey, P.ProductKey, YEAR(SO.OrderDate) AS [Year], SUM(OD.OrderQuantity) AS TotalQuantity

FROM AdventureWorks.dbo.SalesOrders SO

JOIN AdventureWorks.dbo.OrderDetails OD ON SO.SalesOrderNumber = OD.SalesOrderNumber

JOIN AdventureWorks.dbo.Products P ON OD.ProductKey = P.ProductKey

GROUP BY P.ModelKey, P.ProductKey, YEAR(SO.OrderDate)

GO

-- Exibir o conteúdo da tabela

SELECT \* FROM SalesByModelProductYear

-- Criar uma tabela para armazenar os resultados

CREATE TABLE MonthlyAnnualSalesBalance (

[Year] INT,

[Month] INT,

TotalSales DECIMAL(18, 2)

);

GO

-- Popular a tabela com o balanço mensal e anual do total de vendas

INSERT INTO MonthlyAnnualSalesBalance ([Year], [Month], TotalSales)

SELECT YEAR(OrderDate) AS [Year], MONTH(OrderDate) AS [Month], SUM(od.SalesAmount) AS TotalSales

FROM AdventureWorks.dbo.SalesOrders so

INNER JOIN AdventureWorks.dbo.OrderDetails od ON so.SalesOrderNumber = od.SalesOrderNumber

GROUP BY YEAR(OrderDate), MONTH(OrderDate);

GO

-- Exibir o conteúdo da tabela MonthlyAnnualSalesBalance

SELECT \* FROM MonthlyAnnualSalesBalance;

GO

/\* Importar os dados do sql server para mongodb \*/

SELECT P.ModelKey, P.ProductKey, YEAR(SO.OrderDate) AS [Year], SUM(OD.OrderQuantity) AS TotalQuantity

FROM AdventureWorks.dbo.SalesOrders SO

JOIN AdventureWorks.dbo.OrderDetails OD ON SO.SalesOrderNumber = OD.SalesOrderNumber

JOIN AdventureWorks.dbo.Products P ON OD.ProductKey = P.ProductKey

GROUP BY P.ModelKey, P.ProductKey, YEAR(SO.OrderDate)

FOR JSON PATH, ROOT('Orders');

# Conclusões

Com base nas etapas concluídas no projeto de complemento da base de dados, podemos afirmar que houve um trabalho abrangente e detalhado em todos os aspectos relacionados à migração, estruturação, gestão dos dados, índices, backup/recuperação, encriptação, controlo de transações e o mongoDB. A migração bem-sucedida dos dados demonstra uma transição eficiente e cuidadosa do sistema anterior para o novo, garantindo a integridade e consistência das informações, os índices serve para organização, categorização e acesso a informações a conjuntos de dados extensos, o backup e recuperação serve para proteger dados e sistemas contra perdas, os níveis de acesso à informação serve para controlar melhor o acesso dos utilizadores, a encriptação serve para melhor segurança dos dados dos utilizadores da base de dados.

Os metadados e modelos (relacional e entidade-relacionamento) desenvolvidos refletem uma compreensão profunda dos dados e de sua organização, proporcionando uma base sólida para consultas e manipulações futuras. A deteção e resolução de erros durante o processo de programação mostram um compromisso com a qualidade e a precisão dos dados.

Além disso, a gestão de acessos e a criação de views específicas para análise de compras demonstram uma preocupação com a segurança dos dados e a facilidade de utilização para finalidades específicas, o que é fundamental para garantir a privacidade e a eficiência na análise de informações sensíveis.

O projeto alcançou diversos marcos importantes, desde a migração eficaz até a implementação de medidas de segurança e acesso, estabelecendo uma base sólida para o uso futuro da base de dados e a geração de insights relevantes para a tomada de decisões.