

Complementos de Bases de Dados 2023/2024

Licenciatura em Eng^a. Informática

1^a 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.....	4
2. Especificação de Requisitos	5
3. Modelo Relacional (<i>Modelo de dados</i>)	5
3.1 Diagrama do Modelo Entidade Relação.....	5
3.2 Diagrama do Modelo Relacional.....	6
4. Definição do Layout	6
4.1 Identificação do espaço ocupado por tabela.....	7
4.2 Especificação dos Filegroups.....	8
4.3 Schemas	8
5. Verificação da migração de dados	10
5.1 Consultas sobre a base de dados original.....	10
5.2 Consultas sobre a nova base de dados	16
6. Programação	38
6.1 Views.....	38
6.2 Functions.....	38
6.3 Stored procedures.....	39
6.4 Triggers.....	42
7. Catálogo/Metadados	44
7.1 Monitorização	44
8. Descrição da Demonstração	46
8.1 Script de demonstração	46
9. Conclusões	58

Índice de Imagens

Figure 1 - Diagrama de Modelo Entidade Relação	5
Figure 2 - Modelo Relacional	6
Figure 3 - Tabela Products da OldData	10
Figure 4 - Tabela Customer da OldData	11
Figure 5 - Tabela Currency da OldData	12
Figure 6 - Tabela ProductSubCategory da OldData.....	13
Figure 7 - Tabela Sales2 da OldData.....	14
Figure 8 - Tabela Salesterritory da OldData	15
Figure 9 - Perocedimento MigrateModelData	16
Figure 10 - Tabela Model	17
Figure 11 - Percedimento MigrateColorData	18
Figure 12 - Tabela Color	19
Figure 13 - Percedimento MigrateProductCategoryData	20
Figure 14 - Tabela ProductCategory.....	21
Figure 15 - Percedimento InsertSalesTerritoryData	22
Figure 16 - Tabela SalesTerritory	23
Figure 17 - Percedimento MigrateCurrencyData	24
Figure 18 - Tabela Currency	25
Figure 19 - Percedimento MigrateProductSubCategoryData	26
Figure 20 - Tabela ProductSubCategory	27
Figure 21 - Percedimento MigrateCategoryListData	28
Figure 22 - Tabela CategoryList.....	29
Figure 23 - criação de nova Tabela Products	30
Figure 24 - Percedimento MigrateProductsData	30
Figure 25 - Tabela Products	31
Figure 26 - criação de uma nova tabela Customer	32
Figure 27 - Percedimento MigrateCustomerData.....	33
Figure 28 - Tabela Customer	34
Figure 29 - Criação de um novo Sales2	35
Figure 30 - Percedimento MigrateSalesData	36
Figure 31 - Tabela Sales2	37
Figure 32 - Perocedimento sp_SendPasswordResetEmail.....	47
Figure 33 - Procedimentos de Acesso	48
Figure 34 - Espaço ocupado por registo de cada tabela	49
Figure 35 - Espaço ocupado por cada tabela com o número atual de registo.....	50
Figure 36 - Taxa de crescimento por tabela.....	51
Figure 37 - Gatilho dinamico para erros	52
Figure 38 - Colocação de gatilhos para cada tabela.....	53
Figure 39 - histórico do esquema.....	55
Figure 40 - Visão instantânea dos dados mais recentes	56
Figure 41 - Histórico das estatísticas.....	57

1. 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 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 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 as declaramos de 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.

2. Especificação de Requisitos

Nós ao vermos o enunciado dado pelo professor, foi nos 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 na base 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

3. Modelo Relacional (Modelo de dados)

3.1 Diagrama do Modelo Entidade Relação

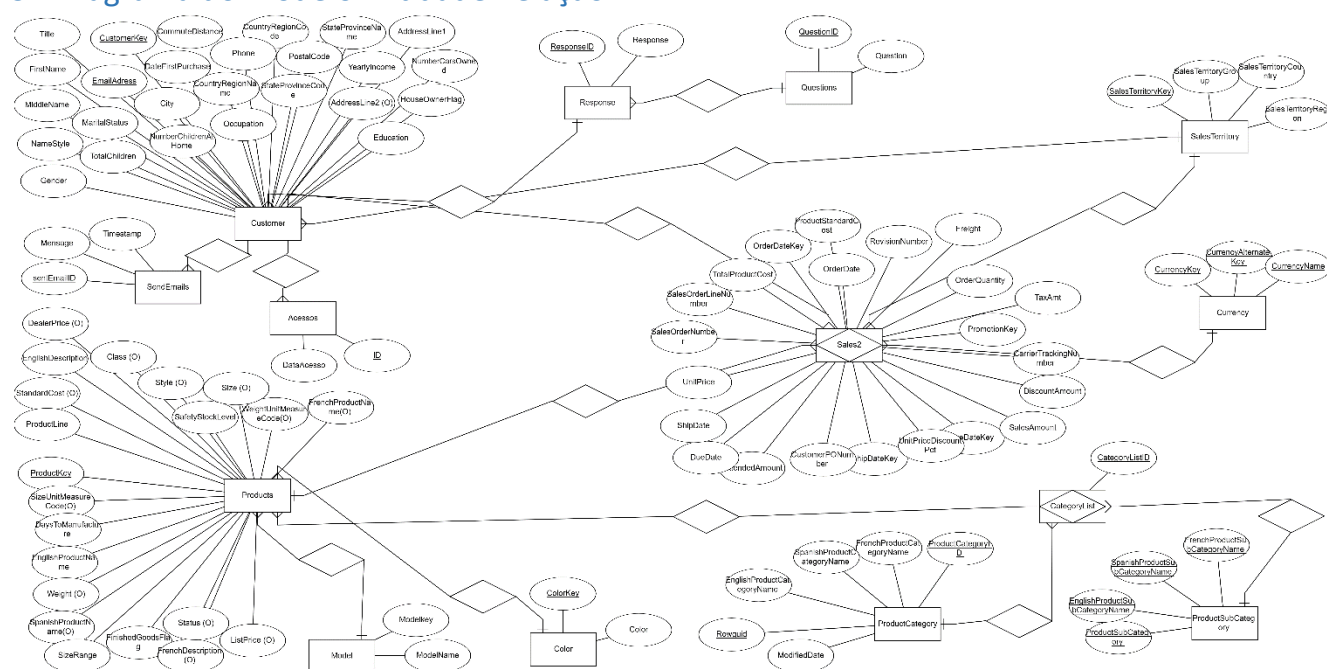


Figure 1 - Diagrama de Modelo Entidade Relação

3.2 Diagrama do Modelo Relacional

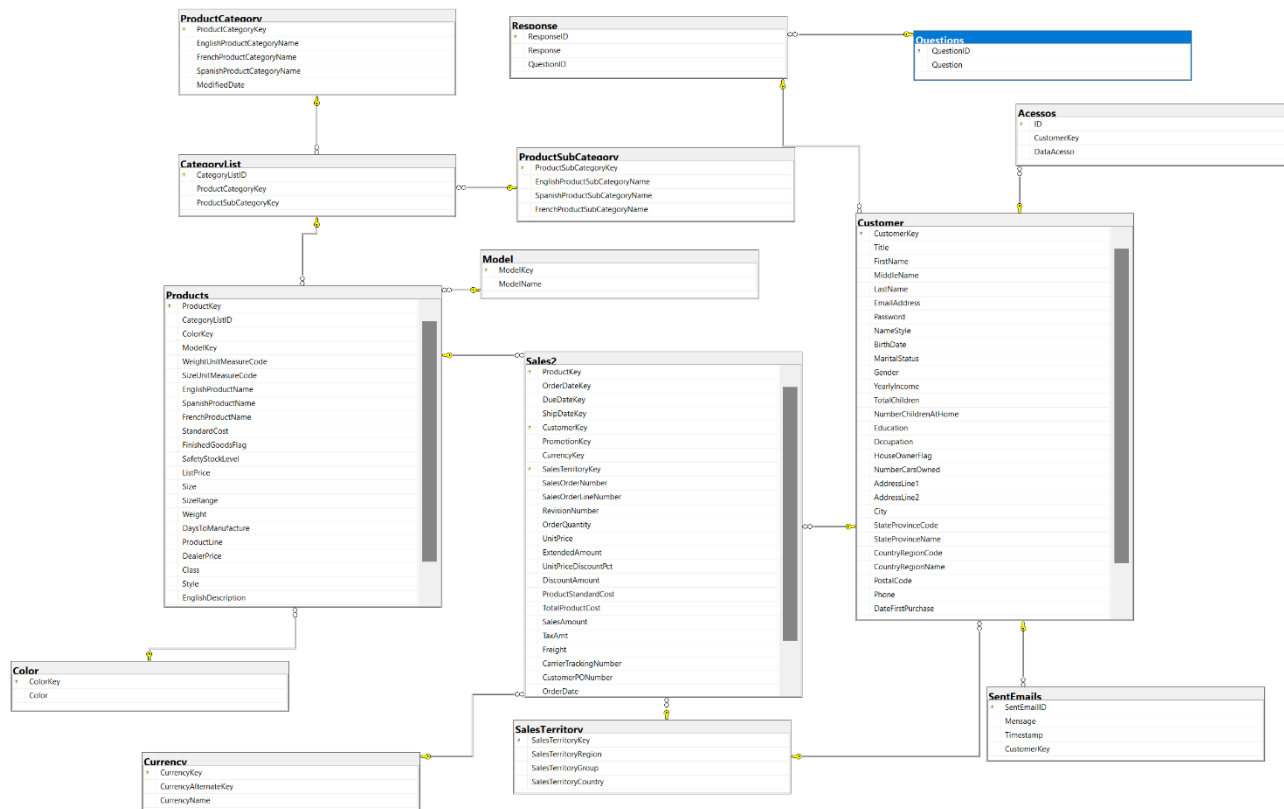


Figure 2 - Modelo Relacional

4. 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.

1ª Fase Relatório Técnico – Complementos de Bases de Dados

4.1 Identificação do espaço ocupado por tabela

Nome Tabela	Dimensão do Registo	Nº de Registos (inicial/final)
<i>Acessos</i>	72KB	0
<i>CategoryList</i>	37KB	37
<i>Currency</i>	288KB	105
<i>Customer</i>	7376KB	18484
<i>ErrorLog</i>	72KB	9
<i>EstatisticasTabelas</i>	72KB	18
<i>MetadadosTabelas</i>	72KB	139
<i>Model</i>	72KB	119
<i>ProductCategory</i>	72KB	4
<i>Products</i>	400KB	397
<i>ProductSubCategory</i>	288KB	37
<i>Questions</i>	72KB	2
<i>Response</i>	72KB	1
<i>Sales2</i>	840KB	6072
<i>SalesTerritory</i>	144KB	11

1ª Fase Relatório Técnico – Complementos de Bases de Dados

<i>SentEmails</i>	<i>72KB</i>	<i>65</i>
<i>sysdiagrams</i>	<i>280KB</i>	<i>1</i>

4.2 Especificação dos Filegroups

Nome Filegroup	Tabelas associadas	Parâmetros
<i>PRIMARY</i>	<i>Sales2, Customer, ErrorLog, EstatisticasTabelas, MetadadosTabelas, Model, ProductCategory, Products, ProductSubCategory, Questions, Response, SalesTerritory, SentEmails</i>	<i>Nome, Localização de Ficheiros</i>

4.3 Schemas

Nome	Descrição
<i>sp_spaceused</i>	<i>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</i>
<i>sys.tables</i>	<i>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</i>
<i>sys.indexes</i>	<i>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</i>
<i>sys.partitions</i>	<i>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</i>

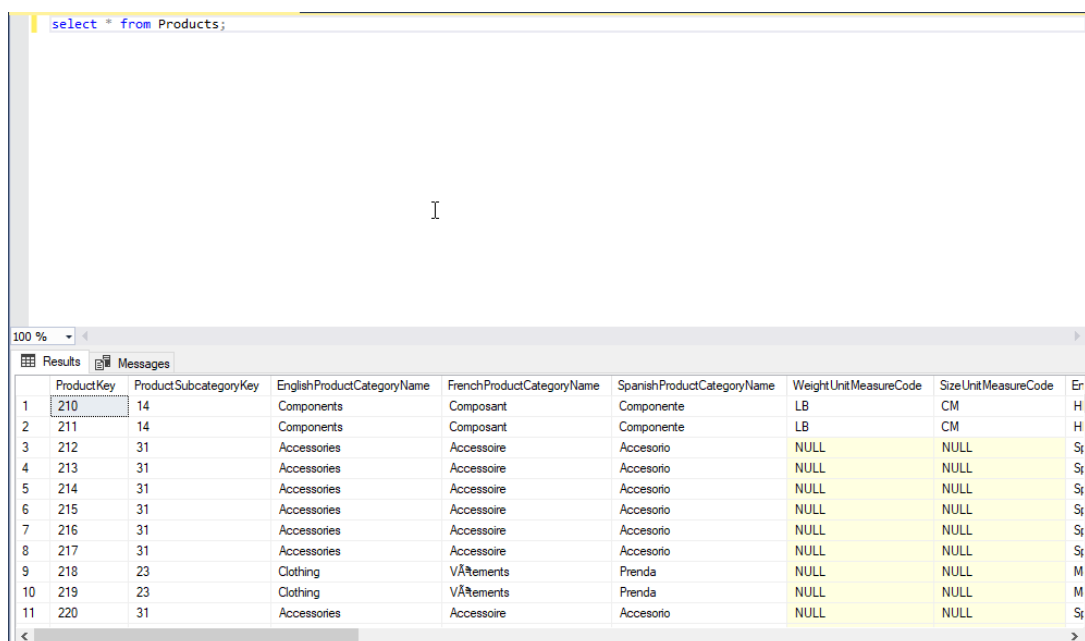
1ª Fase Relatório Técnico – Complementos de Bases de Dados

<i>sys.allocation_units</i>	<i>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</i>
<i>sys.data_spaces</i>	<i>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</i>
<i>sys.filegroups</i>	<i>fornece informações resumidas sobre os filegroups no banco de dados, incluindo detalhes como nome, tipo e localização física</i>
<i>sys.database_files</i>	<i>fornece informações resumidas sobre os arquivos físicos de um banco de dados, incluindo detalhes como nome, tipo, tamanho e localização.</i>
<i>sys.columns</i>	<i>fornece informações resumidas sobre as colunas em tabelas e exibições, incluindo nome, tipo de dados e outras propriedades.</i>
<i>sys.default_constraints</i>	<i>fornece informações sobre restrições padrão em colunas de tabelas, incluindo nome da restrição, tabela, coluna e o valor padrão.</i>
<i>sys.types</i>	<i>fornece informações resumidas sobre os tipos de dados disponíveis no banco de dados, incluindo tipos padrão e definidos pelo utilizador.</i>

5. Verificação da migração de dados

5.1 Consultas sobre a base de dados original

Na tabela antiga Products teremos a seguinte consulta:



The screenshot shows a SQL query window with the following content:

```
select * from Products;
```

Below the query window, the results are displayed in a table with the following columns: ProductKey, ProductSubcategoryKey, EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName, WeightUnitMeasureCode, SizeUnitMeasureCode, and ProductName. The table contains 11 rows of data.

	ProductKey	ProductSubcategoryKey	EnglishProductCategoryName	FrenchProductCategoryName	SpanishProductCategoryName	WeightUnitMeasureCode	SizeUnitMeasureCode	ProductName
1	210	14	Components	Composant	Componente	LB	CM	Hu
2	211	14	Components	Composant	Componente	LB	CM	Hu
3	212	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
4	213	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
5	214	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
6	215	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
7	216	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
8	217	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq
9	218	23	Clothing	Vêtements	Prenda	NULL	NULL	M
10	219	23	Clothing	Vêtements	Prenda	NULL	NULL	M
11	220	31	Accessories	Accessoire	Accesorio	NULL	NULL	Sq

Figure 3 - Tabela Products da OldData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na tabela antiga Customer teremos a seguinte consulta:

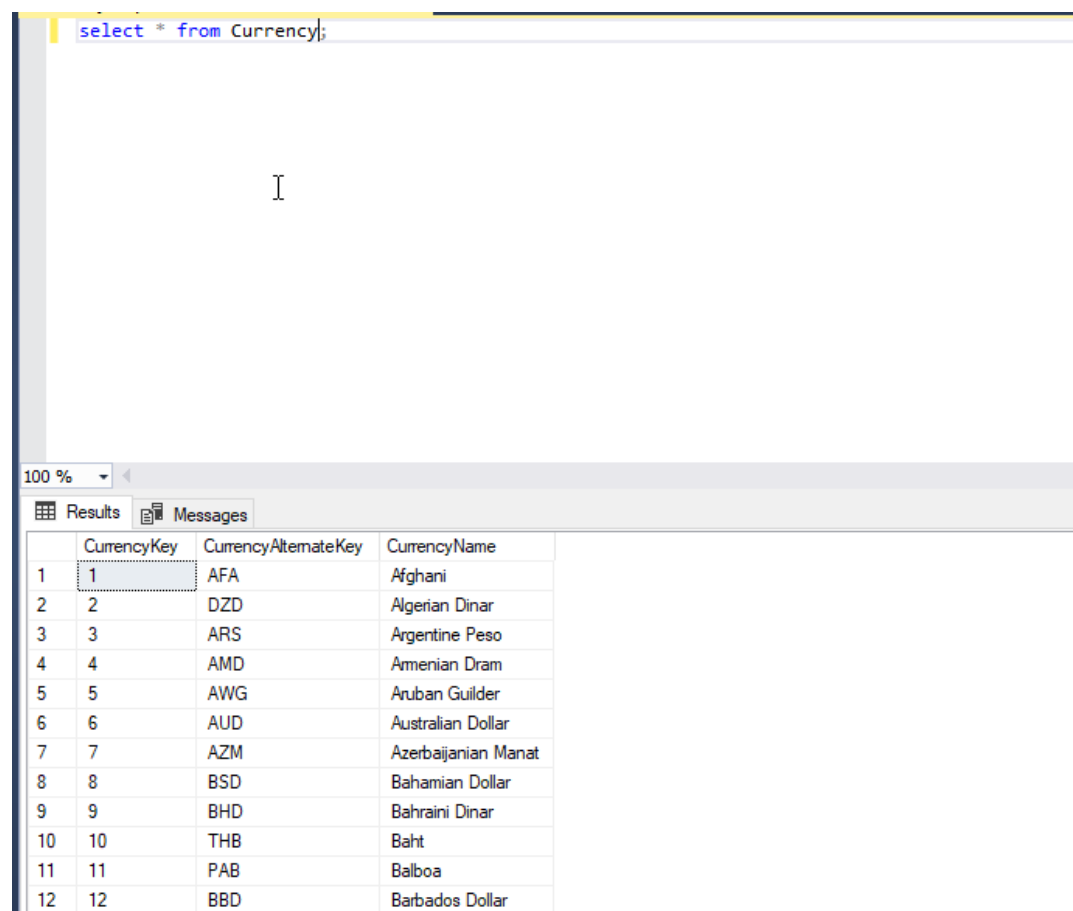
```
select * from Customer;
```

	CustomerKey	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Gender	EmailAddress	YearlyIncome	Total
1	11000	NULL	Jon	V	Yang	FALSE	1966-04-08 00:00:00.000	M	M	jon24@adventure-works.com	90000	2
2	11001	NULL	Eugene	L	Huang	FALSE	1965-05-14 00:00:00.000	S	M	eugene10@adventure-works.com	60000	3
3	11002	NULL	Ruben	NULL	Tones	FALSE	1965-08-12 00:00:00.000	M	M	ruben35@adventure-works.com	60000	3
4	11003	NULL	Christy	NULL	Zhu	FALSE	1968-02-15 00:00:00.000	S	F	christy12@adventure-works.com	70000	0
5	11004	NULL	Elizabeth	NULL	Johnson	FALSE	1968-08-08 00:00:00.000	S	F	elizabeth5@adventure-works.com	80000	5
6	11005	NULL	Julio	NULL	Ruiz	FALSE	1965-08-05 00:00:00.000	S	M	julio1@adventure-works.com	70000	0
7	11006	NULL	Janet	G	Alvarez	FALSE	1965-12-06 00:00:00.000	S	F	janet9@adventure-works.com	70000	0
8	11007	NULL	Marco	NULL	Mehta	FALSE	1964-05-09 00:00:00.000	M	M	marco14@adventure-works.com	60000	3
9	11008	NULL	Rob	NULL	Verhoff	FALSE	1964-07-07 00:00:00.000	S	F	rob4@adventure-works.com	60000	4
10	11009	NULL	Shannon	C	Carlson	FALSE	1964-04-01 00:00:00.000	S	M	shannon38@adventure-works.com	70000	0
11	11010	NULL	Jacquelyn	C	Suarez	FALSE	1964-02-06 00:00:00.000	S	F	jacquelyn20@adventure-works.com	70000	0

Figure 4 - Tabela Customer da OldData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na tabela antiga Currency teremos a seguinte consulta:



```
select * from Currency;
```

	CurrencyKey	CurrencyAlternateKey	CurrencyName
1	1	AFA	Afghani
2	2	DZD	Algerian Dinar
3	3	ARS	Argentine Peso
4	4	AMD	Amenian Dram
5	5	AWG	Aruban Guilder
6	6	AUD	Australian Dollar
7	7	AZM	Azerbaijani Manat
8	8	BSD	Bahamian Dollar
9	9	BHD	Bahraini Dinar
10	10	THB	Baht
11	11	PAB	Balboa
12	12	BBD	Barbados Dollar

Figure 5 - Tabela Currency da OldData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na tabela antiga ProductSubCategory teremos a seguinte consulta:

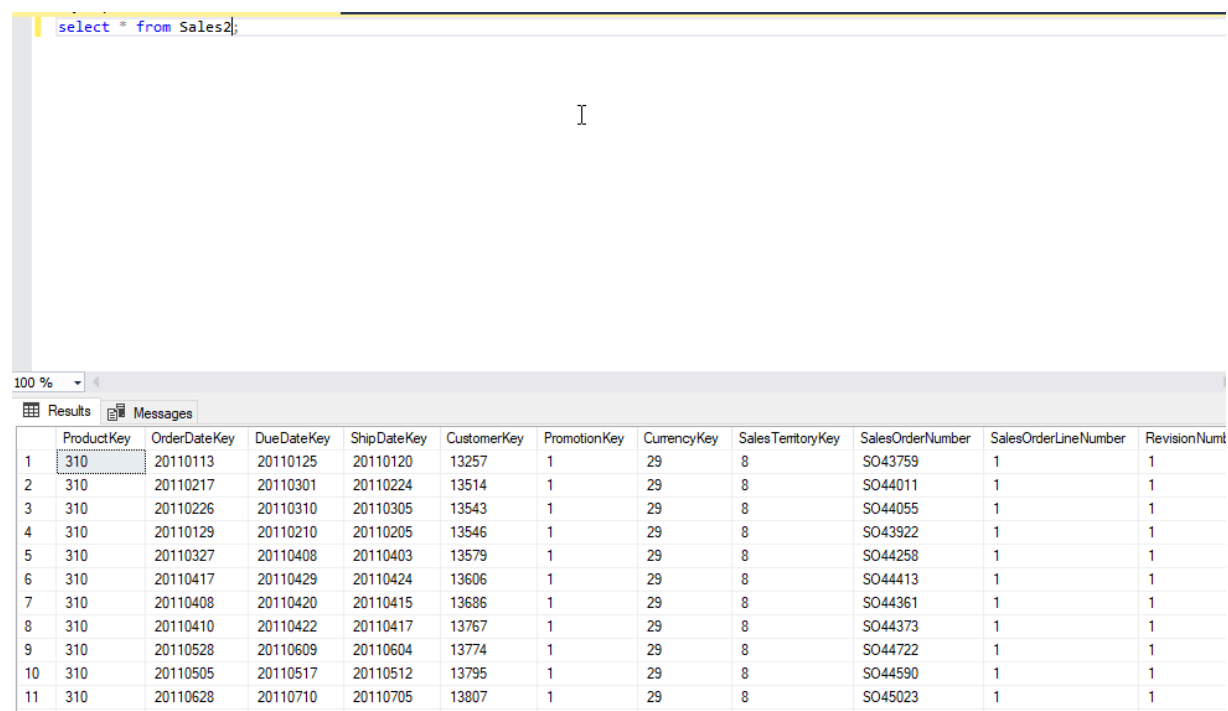
```
select * from ProductSubCategory;
```

ProductSubCategoryKey	EnglishProductSubCategoryName	SpanishProductSubCategoryName	FrenchProductSubCategoryName
1	Mountain Bikes	Bicicleta de montaña	VTT
2	Road Bikes	Bicicleta de carretera	Vélo de route
3	Touring Bikes	Bicicleta de paseo	Vélo de randonnée
4	Handlebars	Barra	Barre d'appui
5	Bottom Brackets	Eje de pedalier	Axe de pédalier
6	Brakes	Frenos	Freins
7	Chains	Cadena	Chaîne
8	Cranksets	Bielas	Pédalier
9	Derailleurs	Desviador	Dérailleur
10	Forks	Horquilla	Fourche
11	Headsets	Dirección	Jeu de direction
12	Mountain Frames	Cuadro de montaña	Cadre de VTT

Figure 6 - Tabela ProductSubCategory da OldData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na tabela antiga Sales2 teremos a seguinte consulta:



The screenshot shows a SQL query window with the following SQL statement:

```
select * from Sales2;
```

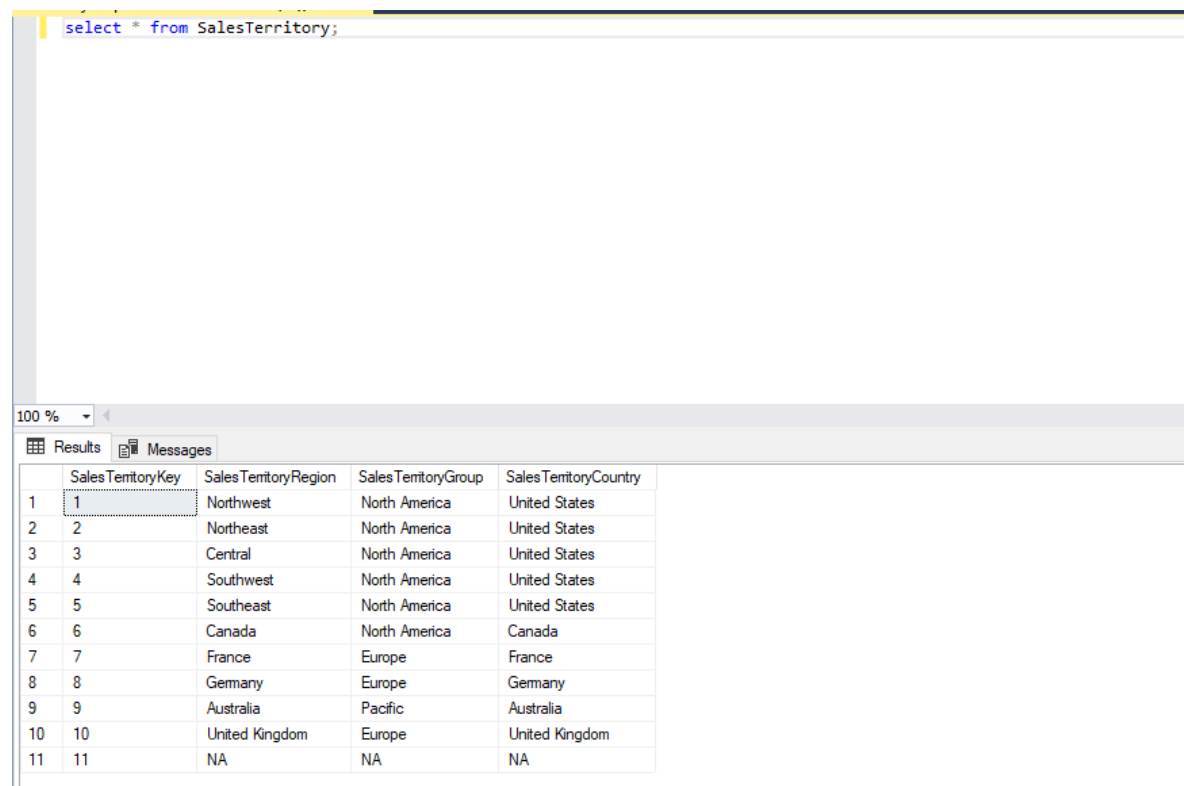
The results are displayed in a table grid with the following columns:

	ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	PromotionKey	CurrencyKey	SalesTerritoryKey	SalesOrderNumber	SalesOrderLineNumber	RevisionNumt
1	310	20110113	20110125	20110120	13257	1	29	8	SO43759	1	1
2	310	20110217	20110301	20110224	13514	1	29	8	SO44011	1	1
3	310	20110226	20110310	20110305	13543	1	29	8	SO44055	1	1
4	310	20110129	20110210	20110205	13546	1	29	8	SO43922	1	1
5	310	20110327	20110408	20110403	13579	1	29	8	SO44258	1	1
6	310	20110417	20110429	20110424	13606	1	29	8	SO44413	1	1
7	310	20110408	20110420	20110415	13686	1	29	8	SO44361	1	1
8	310	20110410	20110422	20110417	13767	1	29	8	SO44373	1	1
9	310	20110528	20110609	20110604	13774	1	29	8	SO44722	1	1
10	310	20110505	20110517	20110512	13795	1	29	8	SO44590	1	1
11	310	20110628	20110710	20110705	13807	1	29	8	SO45023	1	1

Figure 7 - Tabela Sales2 da OldData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na tabela antiga SalesTerritory teremos a seguinte consulta:



The screenshot displays a SQL query window with the command `select * from SalesTerritory;`. Below the query, the 'Results' pane shows a table with 11 rows and 4 columns: SalesTerritoryKey, SalesTerritoryRegion, SalesTerritoryGroup, and SalesTerritoryCountry. The data is as follows:

	SalesTerritoryKey	SalesTerritoryRegion	SalesTerritoryGroup	SalesTerritoryCountry
1	1	Northwest	North America	United States
2	2	Northeast	North America	United States
3	3	Central	North America	United States
4	4	Southwest	North America	United States
5	5	Southeast	North America	United States
6	6	Canada	North America	Canada
7	7	France	Europe	France
8	8	Germany	Europe	Germany
9	9	Australia	Pacific	Australia
10	10	United Kingdom	Europe	United Kingdom
11	11	NA	NA	NA

Figure 8 - Tabela Salesterritory da OldData

5.2 Consultas sobre a nova base de dados

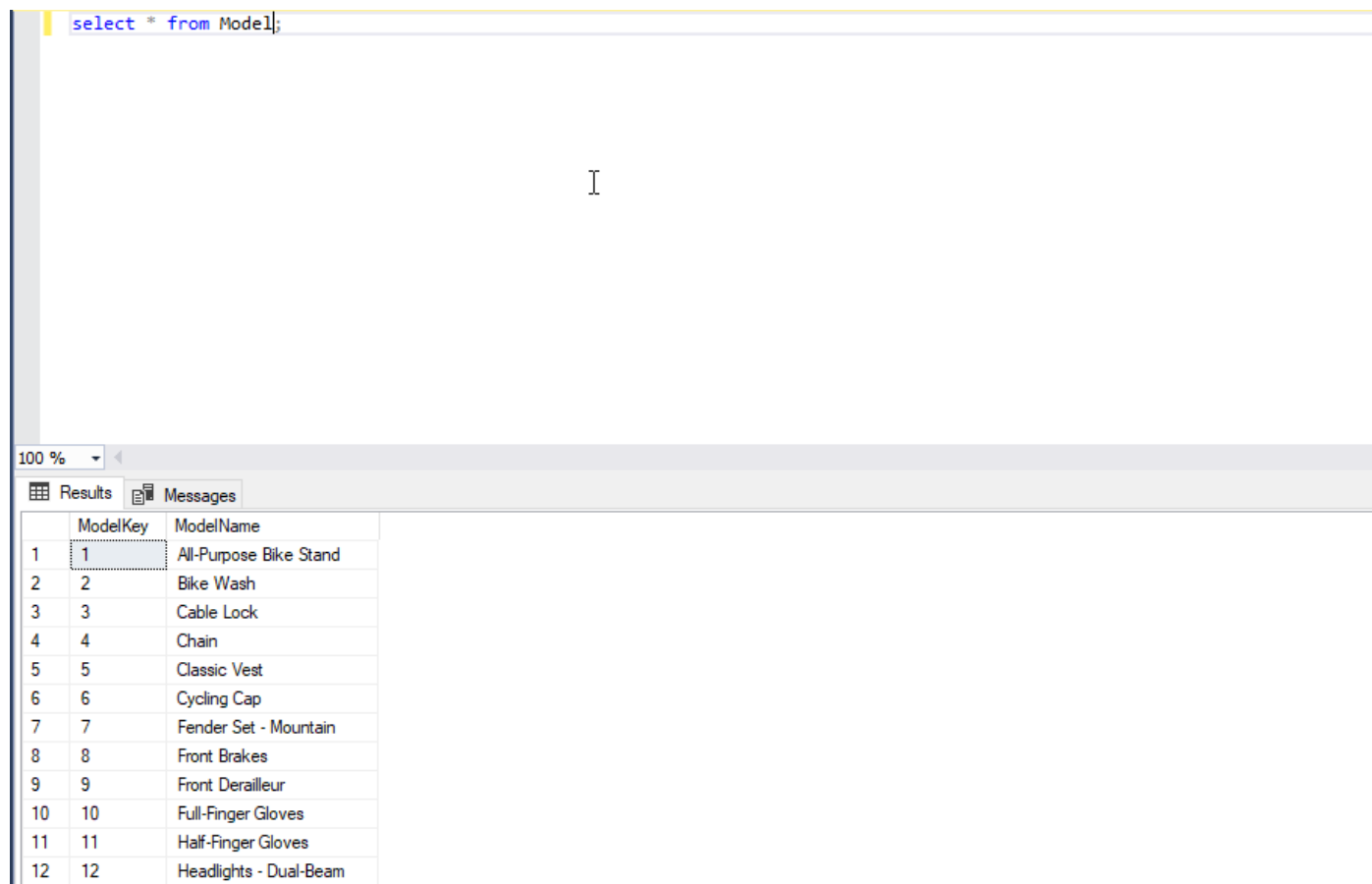
Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Model”, para fazer a separação dos dados da tabela Products.

```
1  -- Tabela Model
2  CREATE TABLE Model
3  (
4      ModelKey INT IDENTITY(1,1) NOT NULL,
5      ModelName VARCHAR(255),
6      PRIMARY KEY (ModelKey)
7  );
8
9
10 -- Crie uma stored procedure para realizar a migração de dados para a tabela Model
11 CREATE PROCEDURE dbo.MigrateModelData
12 AS
13 BEGIN
14     BEGIN TRY
15         -- Inicie a transação para garantir a consistência dos dados
16         BEGIN TRANSACTION;
17
18         -- Insere dados na tabela Model
19         INSERT INTO Model (ModelName)
20         SELECT DISTINCT ModelName
21         FROM AdventureOldData.dbo.Products;
22
23         -- Commit da transação se a inserção for bem-sucedida
24         COMMIT;
25     END TRY
26     BEGIN CATCH
27         -- Rollback da transação em caso de erro
28         ROLLBACK;
29
30         -- Tratamento de erros
31         DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
32         DECLARE @ErrorNumber INT = ERROR_NUMBER();
33         DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
34         DECLARE @ErrorState INT = ERROR_STATE();
35
36         -- Registre o erro em um log de erros (se disponível)
37         EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;
38
39         -- Exiba uma mensagem amigável para o utilizador
40         THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Model. Entre em contato com o suporte.', 1;
41     END CATCH
42 END;
43
44 --Executar o procedimento
45 exec dbo.MigrateModelData;
46
```

Figure 9 - Perocedimento MigrateModelData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a database query interface. At the top, a SQL query is entered in a text box: `select * from Model;`. Below the query box, a cursor is visible. The interface includes a toolbar with a zoom level set to 100% and tabs for 'Results' and 'Messages'. The 'Results' tab is active, showing a table with 12 rows of data. The table has two columns: 'ModelKey' and 'ModelName'. The data is as follows:

	ModelKey	ModelName
1	1	All-Purpose Bike Stand
2	2	Bike Wash
3	3	Cable Lock
4	4	Chain
5	5	Classic Vest
6	6	Cycling Cap
7	7	Fender Set - Mountain
8	8	Front Brakes
9	9	Front Derailleur
10	10	Full-Finger Gloves
11	11	Half-Finger Gloves
12	12	Headlights - Dual-Beam

Figure 10 - Tabela Model

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
-- Criação da tabela Color
CREATE TABLE Color
(
    ColorKey INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
    Color NVARCHAR(55) NOT NULL
);

-- 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();

        -- Registre o erro em um log de erros
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

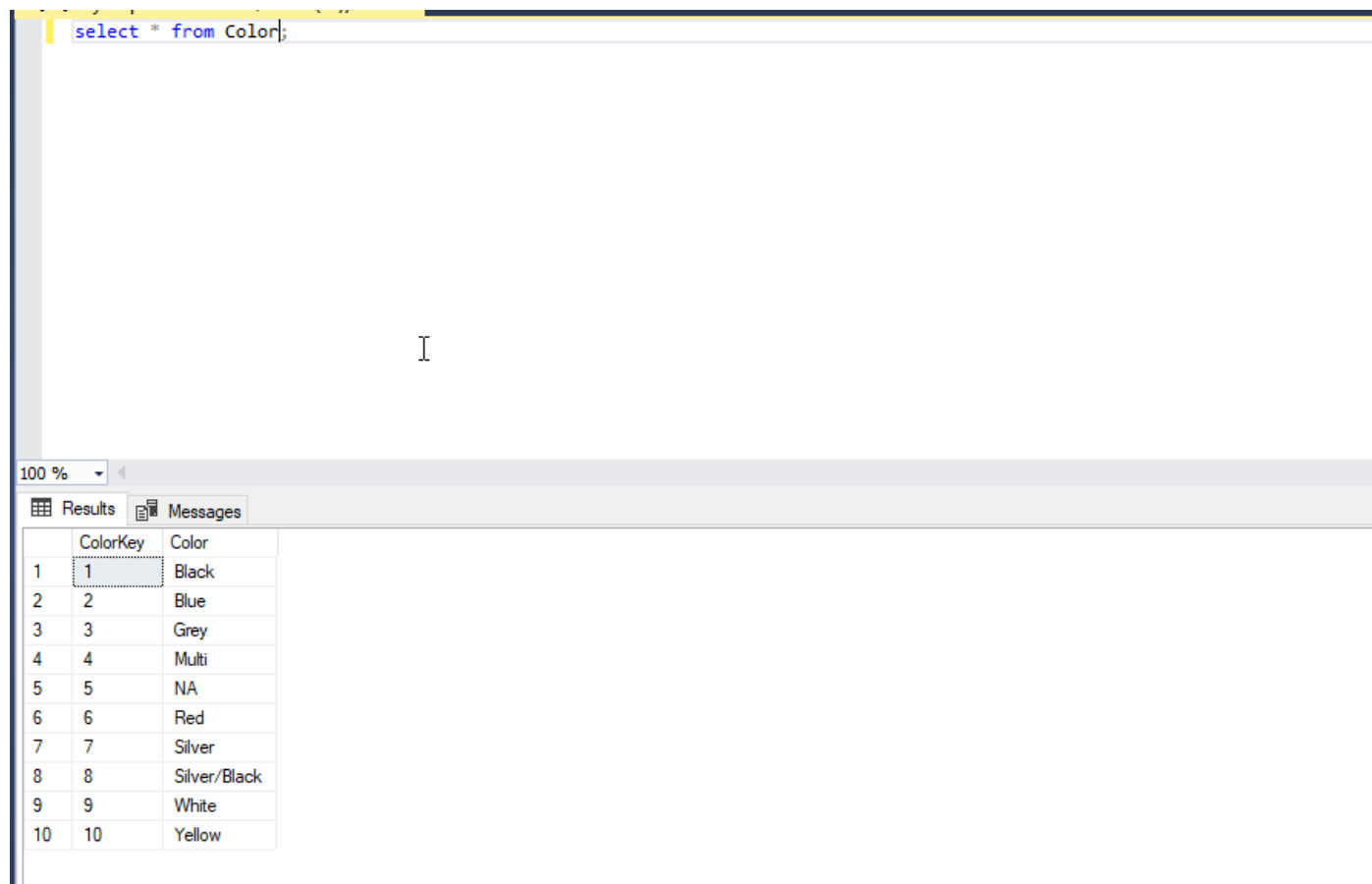
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Color. Entre em contato com o suporte.', 1;
    END CATCH
END;

exec dbo.MigrateColorData;
```

Figure 11 - Percedimento MigrateColorData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot shows a SQL query execution interface. At the top, a query editor contains the text `select * from Color;`. Below the editor, a results pane displays the data. The results pane has a tab labeled 'Results' and a zoom level of '100 %'. The data is presented in a table with two columns: 'ColorKey' and 'Color'. The table contains 10 rows of data, with the first row (ColorKey 1, Color Black) highlighted. The interface also includes a 'Messages' tab.

	ColorKey	Color
1	1	Black
2	2	Blue
3	3	Grey
4	4	Multi
5	5	NA
6	6	Red
7	7	Silver
8	8	Silver/Black
9	9	White
10	10	Yellow

Figure 12 - Tabela Color

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
-- Tabela ProductCategory
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
);

-- Crie uma stored procedure para realizar a migração de dados para a tabela ProductCategory
CREATE PROCEDURE dbo.MigrateProductCategoryData
AS
BEGIN
    BEGIN TRY
        -- Inicie a transação para garantir a consistência dos dados
        BEGIN TRANSACTION;

        -- Insere dados na tabela ProductCategory
        INSERT INTO ProductCategory (EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName)
        SELECT DISTINCT EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName
        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();

        -- Registre o erro em um log de erros (se disponível)
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

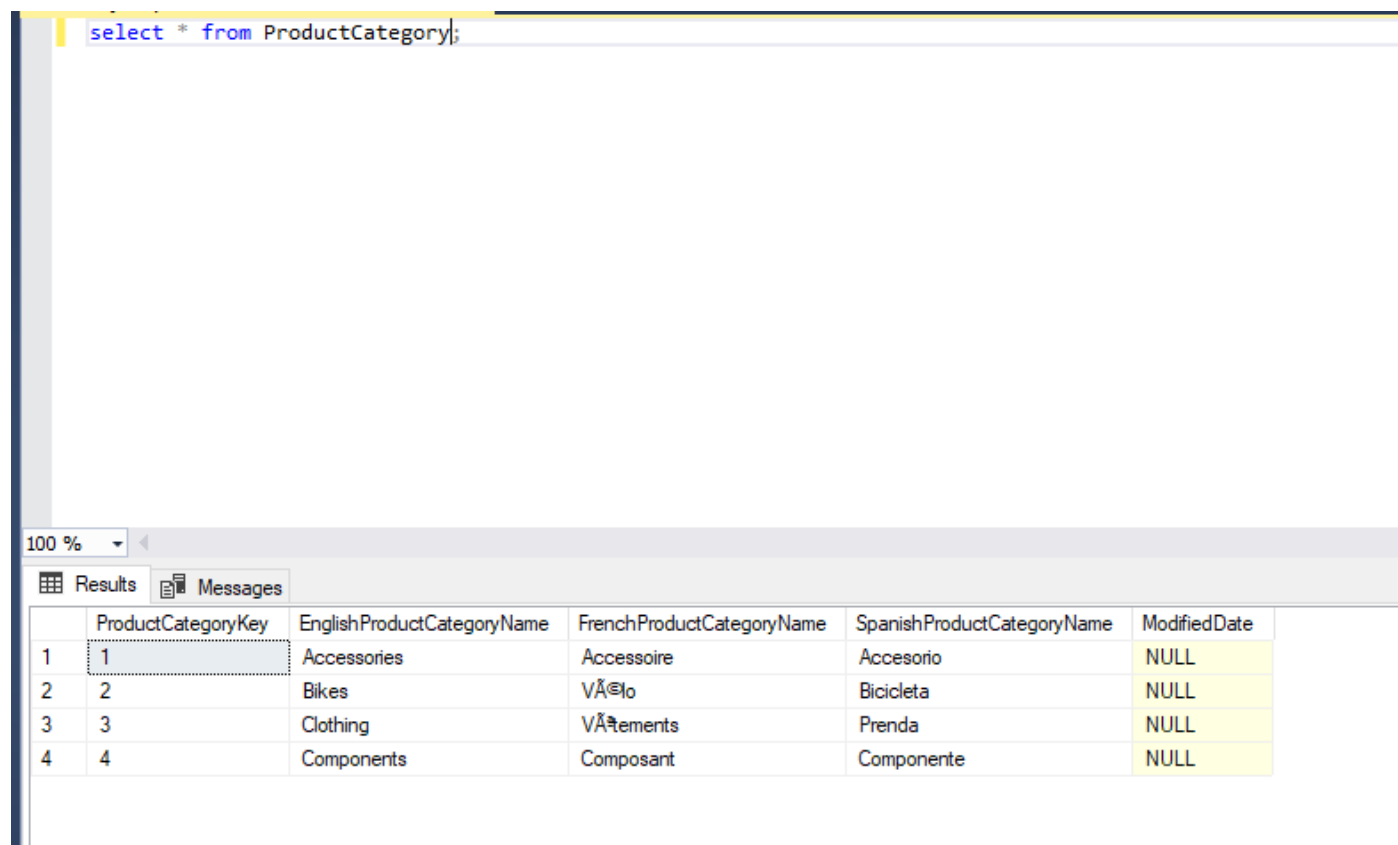
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela ProductCategory. Entre em contato com o suporte.', 1;
    END CATCH
END;

--executar o procedimento
exec dbo.MigrateProductCategoryData;
```

Figure 13 - Percedimento MigrateProductCategoryData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a SQL query execution environment. At the top, a query editor shows the command: `select * from ProductCategory;`. Below the editor, a results pane shows the data returned by the query. The results are presented in a table with the following columns: ProductCategoryKey, EnglishProductCategoryName, FrenchProductCategoryName, SpanishProductCategoryName, and ModifiedDate. The table contains four rows of data, each with a row number in the first column. The ModifiedDate column for all rows contains the value NULL.

	ProductCategoryKey	EnglishProductCategoryName	FrenchProductCategoryName	SpanishProductCategoryName	ModifiedDate
1	1	Accessories	Accessoire	Accesorio	NULL
2	2	Bikes	Vélo	Bicicleta	NULL
3	3	Clothing	Vêtements	Prenda	NULL
4	4	Components	Composant	Componente	NULL

Figure 14 - Tabela ProductCategory

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
-- Criacao da tabela SalesTerritory
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)
);

-- Criar a stored procedure para inserir dados em 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();

        -- Registrar o erro em um log de erros (se ja existir a stored procedure de log de erros)
        -- EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

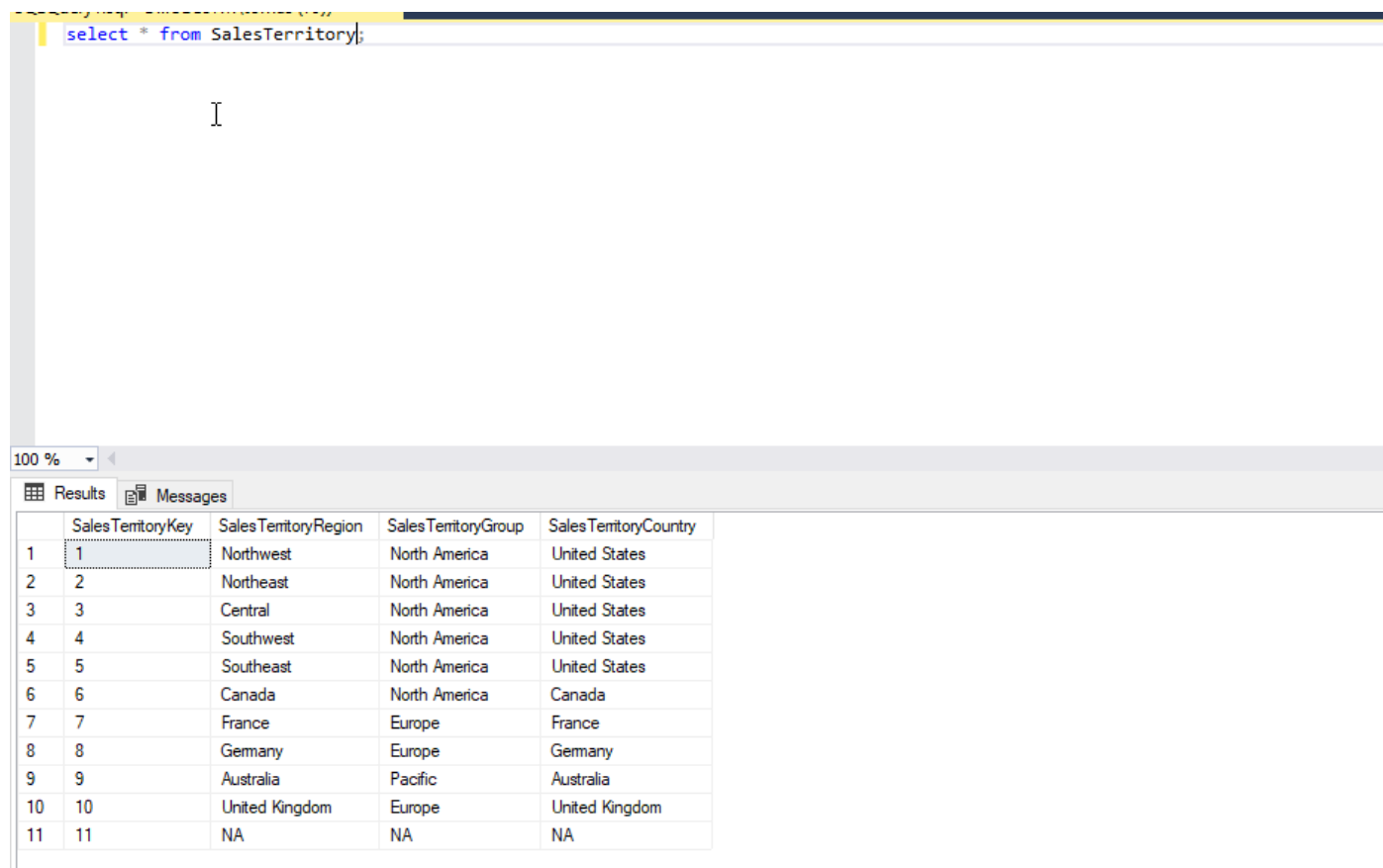
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela SalesTerritory. Entre em contato com o suporte.', 1;
    END CATCH
END;

EXEC dbo.InsertSalesTerritoryData;
```

Figure 15 - Percedimento InsertSalesTerritoryData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a SQL query window with the following text:

```
select * from SalesTerritory;
```

Below the query window, the 'Results' pane shows the data returned by the query. The results are organized into a table with the following columns: SalesTerritoryKey, SalesTerritoryRegion, SalesTerritoryGroup, and SalesTerritoryCountry. The data is as follows:

	SalesTerritoryKey	SalesTerritoryRegion	SalesTerritoryGroup	SalesTerritoryCountry
1	1	Northwest	North America	United States
2	2	Northeast	North America	United States
3	3	Central	North America	United States
4	4	Southwest	North America	United States
5	5	Southeast	North America	United States
6	6	Canada	North America	Canada
7	7	France	Europe	France
8	8	Germany	Europe	Germany
9	9	Australia	Pacific	Australia
10	10	United Kingdom	Europe	United Kingdom
11	11	NA	NA	NA

Figure 16 - Tabela SalesTerritory

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
--Criação da tabela Currency
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)
);

-- Crie uma stored procedure para realizar a migração de dados da tabela Currency
CREATE PROCEDURE dbo.MigrateCurrencyData
AS
BEGIN
    BEGIN TRY
        -- Inicie a transação para garantir consistência
        BEGIN TRANSACTION;

        -- Insira dados na tabela Currency
        INSERT INTO Currency (CurrencyAlternateKey, CurrencyName)
        SELECT CurrencyAlternateKey, CurrencyName
        FROM AdventureOldData.dbo.Currency;

        -- 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();

        -- Registre o erro em um log de erros (se tiver um)
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

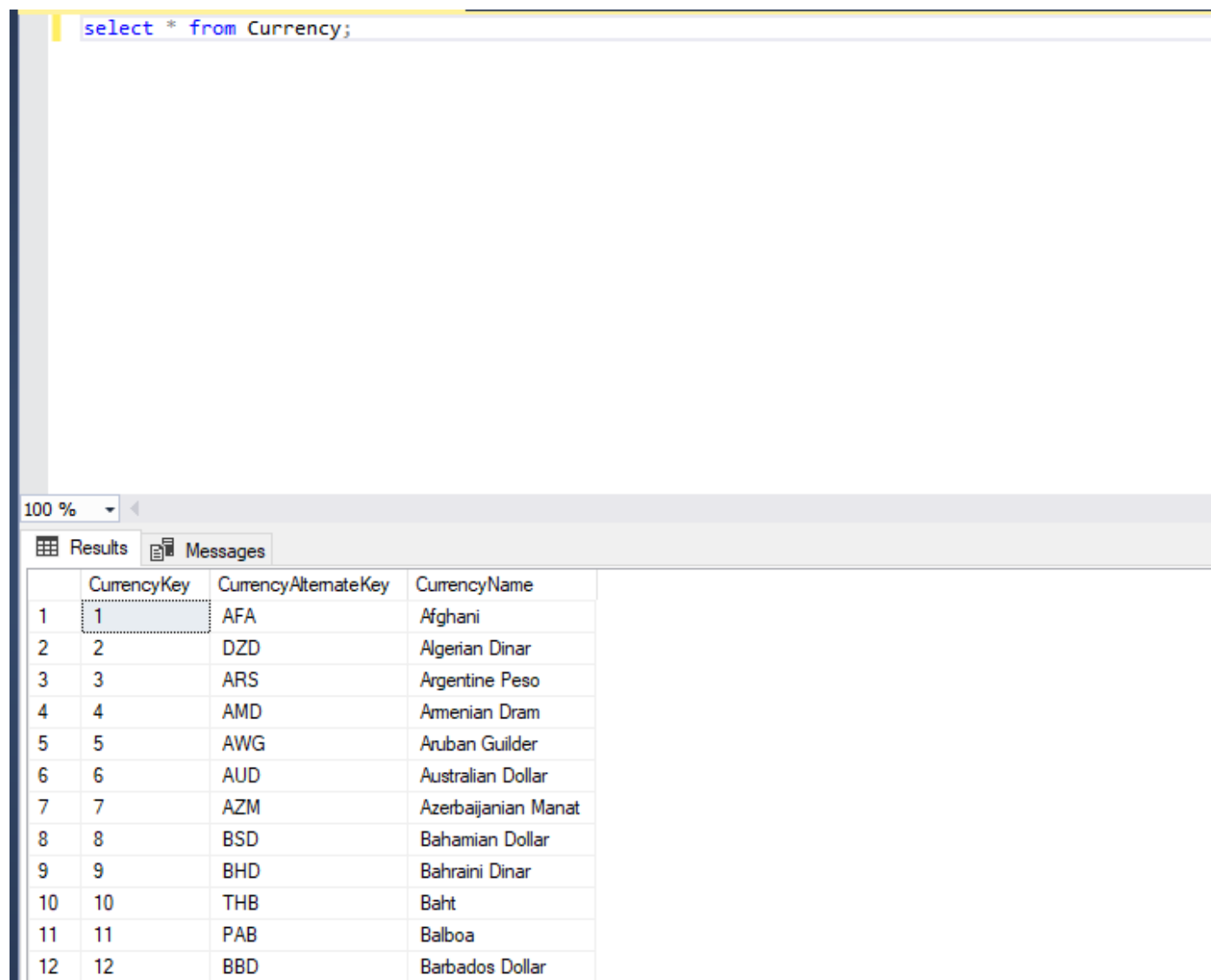
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Color. Entre em contato com o suporte.', 1;
    END CATCH
END;

exec dbo.MigrateCurrencyData;
```

Figure 17 - Percedimento MigrateCurrencyData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a database query interface. At the top, a SQL query is entered in a text box: `select * from Currency;`. Below the query box, there is a toolbar with a zoom dropdown set to '100 %' and two buttons: 'Results' (active) and 'Messages'. The 'Results' button is selected, and the query results are displayed in a table below. The table has four columns: 'CurrencyKey', 'CurrencyAlternateKey', 'CurrencyName', and an unlabeled index column. The results show 12 rows of currency data.

	CurrencyKey	CurrencyAlternateKey	CurrencyName
1	1	AFA	Afghani
2	2	DZD	Algerian Dinar
3	3	ARS	Argentine Peso
4	4	AMD	Amenian Dram
5	5	AWG	Aruban Guilder
6	6	AUD	Australian Dollar
7	7	AZM	Azerbaijani Manat
8	8	BSD	Bahamian Dollar
9	9	BHD	Bahraini Dinar
10	10	THB	Baht
11	11	PAB	Balboa
12	12	BBD	Barbados Dollar

Figure 18 - Tabela Currency

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
--Criação da tabela ProductSubCategory
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)
);

-- Crie uma stored procedure para realizar a migração de dados para a tabela ProductSubCategory
CREATE PROCEDURE dbo.MigrateProductSubCategoryData
AS
BEGIN
    BEGIN TRY
        -- Inicie a transação para garantir a consistência dos dados
        BEGIN TRANSACTION;

        -- Insere dados na tabela ProductSubCategory
        INSERT INTO ProductSubCategory (EnglishProductSubCategoryName, SpanishProductSubCategoryName, FrenchProductSubCategoryName)
        SELECT EnglishProductSubCategoryName, SpanishProductSubCategoryName, FrenchProductSubCategoryName
        FROM AdventureOldData.dbo.ProductSubCategory;

        -- 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();

        -- Registre o erro em um log de erros (se disponível)
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

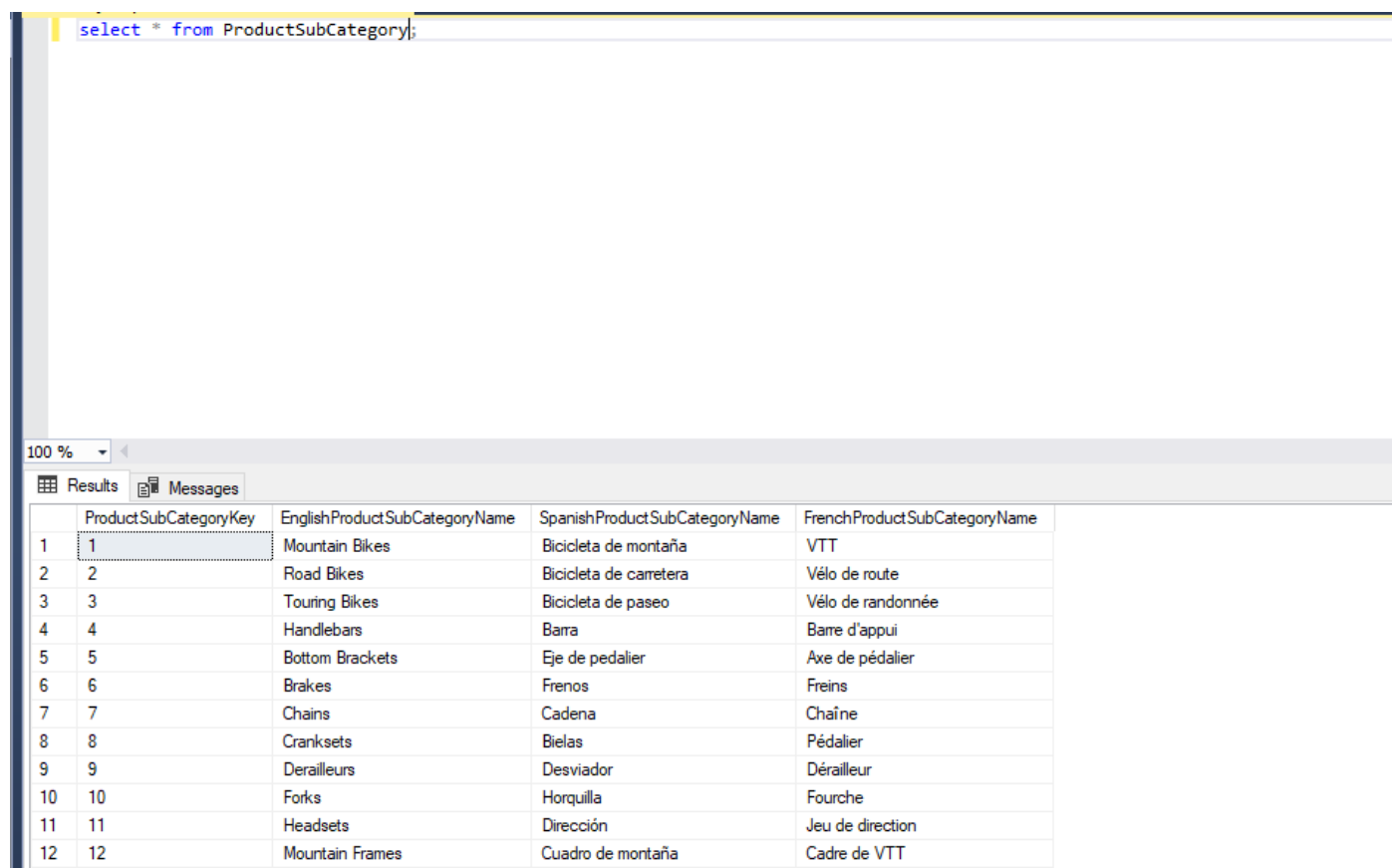
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela ProductSubCategory. Entre em contato com o suporte.', 1;
    END CATCH
END;

exec dbo.MigrateProductSubCategoryData;
```

Figure 19 - Percedimento MigrateProductSubCategoryData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



```
select * from ProductSubCategory;
```

	ProductSubCategoryKey	EnglishProductSubCategoryName	SpanishProductSubCategoryName	FrenchProductSubCategoryName
1	1	Mountain Bikes	Bicicleta de montaña	VTT
2	2	Road Bikes	Bicicleta de carretera	Vélo de route
3	3	Touring Bikes	Bicicleta de paseo	Vélo de randonnée
4	4	Handlebars	Barra	Barre d'appui
5	5	Bottom Brackets	Eje de pedalier	Axe de pédalier
6	6	Brakes	Frenos	Freins
7	7	Chains	Cadena	Chaîne
8	8	Cranksets	Bielas	Pédalier
9	9	Deraileurs	Desviador	Dérailleur
10	10	Forks	Horquilla	Fourche
11	11	Headsets	Dirección	Jeu de direction
12	12	Mountain Frames	Cuadro de montaña	Cadre de VTT

Figure 20 - Tabela ProductSubCategory

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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”.

```
-- Criação da tabela CategoryList
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),
);

-- 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;

        -- Inserir 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();

        -- Registre o erro em um log de erros
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

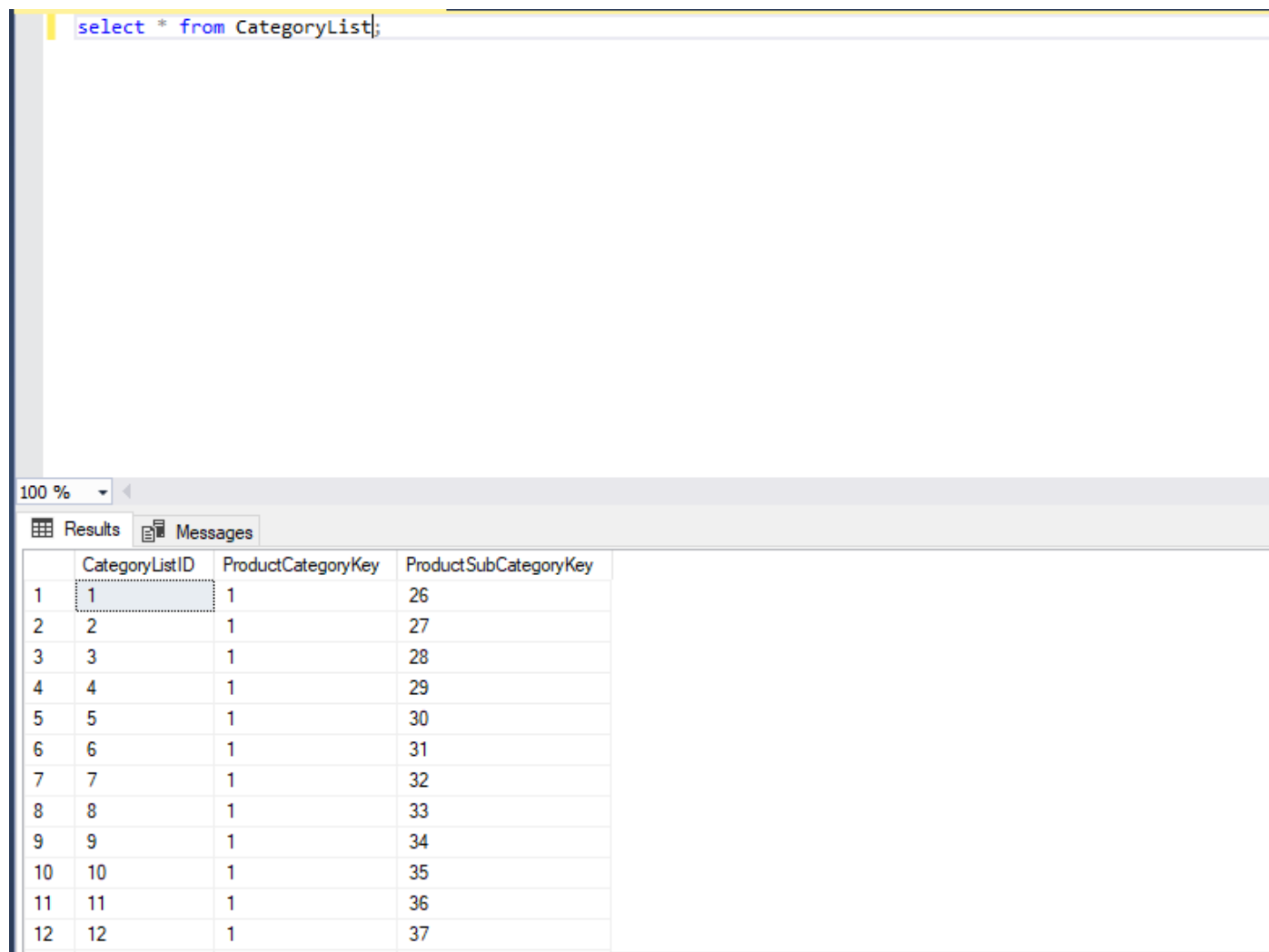
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela CategoryList. Entre em contato com o suporte.', 1;
    END CATCH
END;

exec dbo.MigrateCategoryListData;
```

Figure 21 - Percedimento MigrateCategoryListData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a SQL query execution environment. At the top, a text box contains the query: `select * from CategoryList;`. Below the query, there are tabs for "Results" and "Messages". The "Results" tab is active, showing a table with 12 rows and 4 columns. The columns are labeled "CategoryListID", "ProductCategoryKey", and "ProductSubCategoryKey". The first column, "CategoryListID", is highlighted with a dotted border. The data in the table is as follows:

	CategoryListID	ProductCategoryKey	ProductSubCategoryKey
1	1	1	26
2	2	1	27
3	3	1	28
4	4	1	29
5	5	1	30
6	6	1	31
7	7	1	32
8	8	1	33
9	9	1	34
10	10	1	35
11	11	1	36
12	12	1	37

Figure 22 - Tabela CategoryList

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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.

```
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)
);
```

Figure 23 - criação de nova Tabela Products

```
-- Crie 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 AdventureWorks.dbo.Products p
        INNER JOIN AdventureWorks.dbo.Color c ON p.Color = c.Color
        INNER JOIN AdventureWorks.dbo.Model b ON p.ModelName = 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;

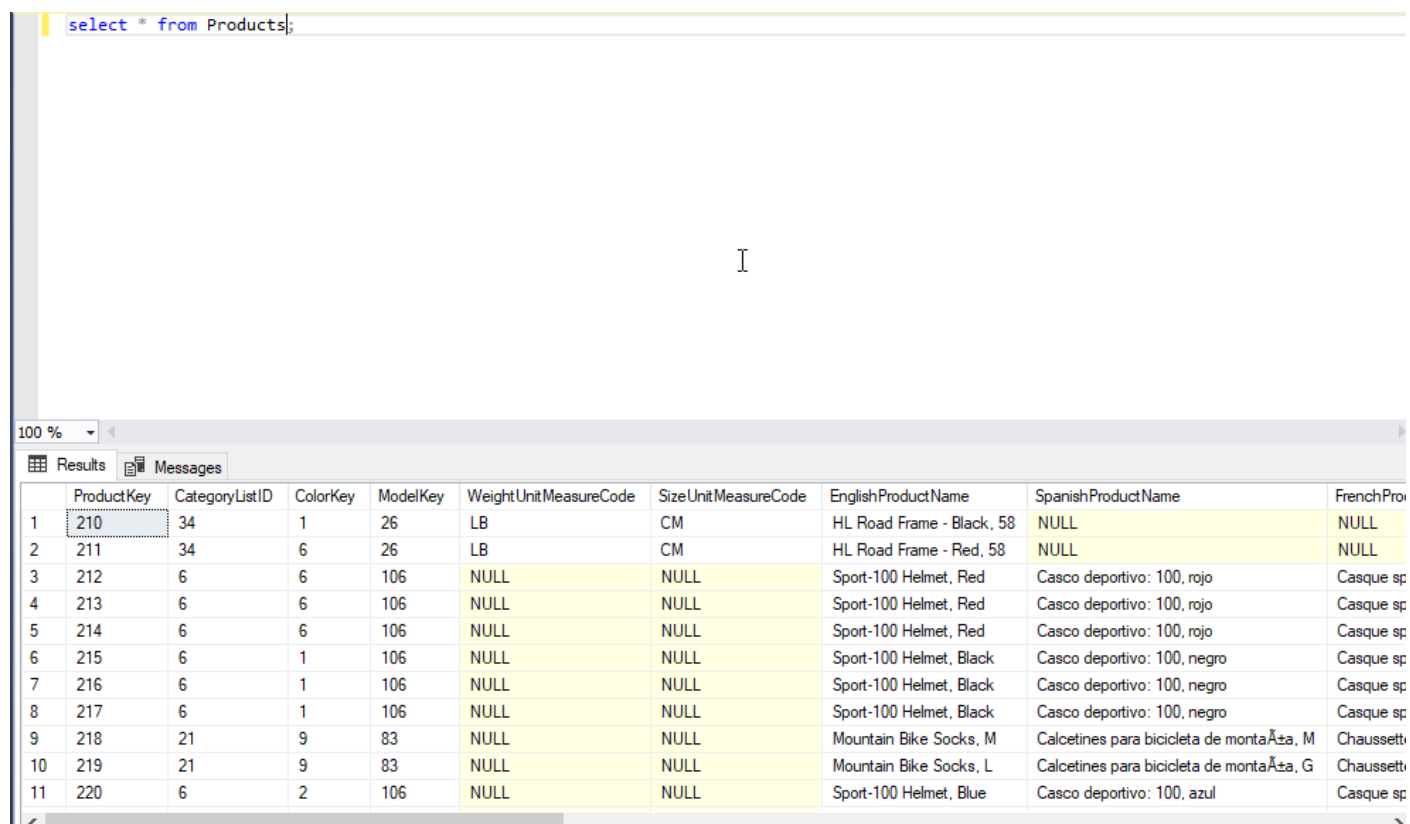
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Products. Entre em contato com o suporte.', 1;
    END CATCH
END;

--Executar o procedimento
exec dbo.MigrateProductsData;
```

Figure 24 - Percedimento MigrateProductsData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



The screenshot displays a SQL query execution window. At the top, the query `select * from Products;` is entered. Below the query, a cursor is visible. The results are shown in a table with 11 rows and 10 columns. The columns are: ProductKey, CategoryListID, ColorKey, ModelKey, WeightUnitMeasureCode, SizeUnitMeasureCode, EnglishProductName, SpanishProductName, and FrenchProductName. The first two rows (ProductKey 210 and 211) have NULL values for Spanish and French product names. The remaining rows have specific product names in all three languages.

	ProductKey	CategoryListID	ColorKey	ModelKey	WeightUnitMeasureCode	SizeUnitMeasureCode	EnglishProductName	SpanishProductName	FrenchProductName
1	210	34	1	26	LB	CM	HL Road Frame - Black, 58	NULL	NULL
2	211	34	6	26	LB	CM	HL Road Frame - Red, 58	NULL	NULL
3	212	6	6	106	NULL	NULL	Sport-100 Helmet, Red	Casco deportivo: 100, rojo	Casque sp
4	213	6	6	106	NULL	NULL	Sport-100 Helmet, Red	Casco deportivo: 100, rojo	Casque sp
5	214	6	6	106	NULL	NULL	Sport-100 Helmet, Red	Casco deportivo: 100, rojo	Casque sp
6	215	6	1	106	NULL	NULL	Sport-100 Helmet, Black	Casco deportivo: 100, negro	Casque sp
7	216	6	1	106	NULL	NULL	Sport-100 Helmet, Black	Casco deportivo: 100, negro	Casque sp
8	217	6	1	106	NULL	NULL	Sport-100 Helmet, Black	Casco deportivo: 100, negro	Casque sp
9	218	21	9	83	NULL	NULL	Mountain Bike Socks, M	Calcetines para bicicleta de monta	Chaussette
10	219	21	9	83	NULL	NULL	Mountain Bike Socks, L	Calcetines para bicicleta de monta	Chaussette
11	220	6	2	106	NULL	NULL	Sport-100 Helmet, Blue	Casco deportivo: 100, azul	Casque sp

Figure 25 - 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.

```
-- Tabela Customer
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 VARCHAR(60),
    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,
    PRIMARY KEY (CustomerKey),
    FOREIGN KEY (SalesTerritoryKey) REFERENCES SalesTerritory(SalesTerritoryKey),
    UNIQUE (CustomerKey)
);

-- Adiciona a coluna ResponseID à tabela Customer
ALTER TABLE Customer
ADD ResponseID INT;

-- Adiciona a chave estrangeira para a coluna ResponseID
ALTER TABLE Customer
ADD CONSTRAINT FK_Customer_Response
FOREIGN KEY (ResponseID) REFERENCES Response(ResponseID);

UPDATE Customer
SET ResponseID = 1;
```

Figure 26 - criação de uma nova tabela Customer

1ª Fase Relatório Técnico – Complementos de Bases de Dados

```
-- Crie uma stored procedure para realizar a migração de dados da tabela Customer
CREATE PROCEDURE dbo.MigrateCustomerData
AS
BEGIN
    BEGIN TRY
        -- Inicie a transação para garantir consistência
        BEGIN TRANSACTION;

        SET IDENTITY_INSERT Customer ON;

        -- Insira os 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 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();

        -- Registre o erro em um log de erros (se tiver um)
        EXEC dbo.LogError @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;

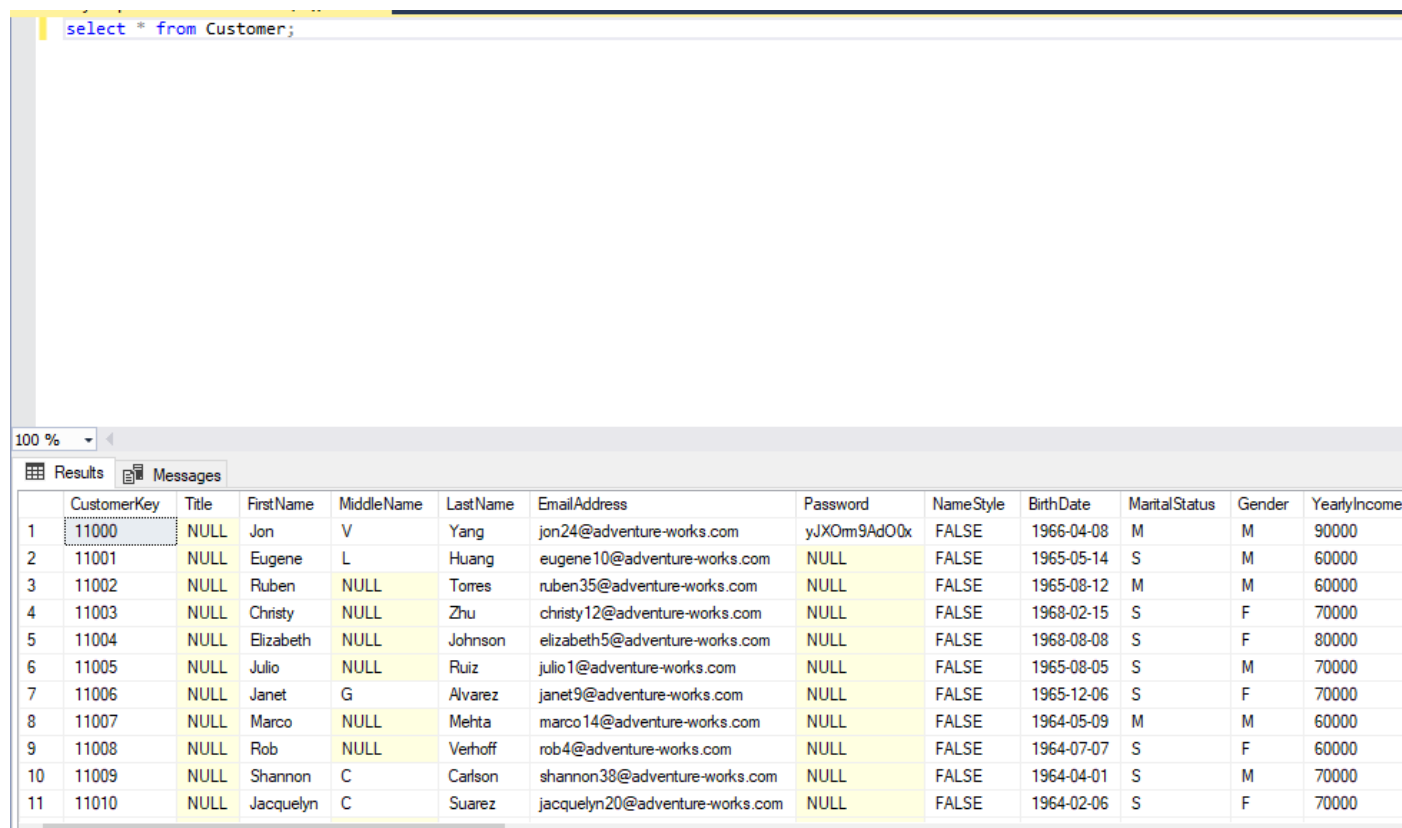
        -- Exiba uma mensagem amigável para o utilizador
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Customer. Entre em contato com o suporte.', 1;
    END CATCH
END;

--Executar o procedimento
exec dbo.MigrateCustomerData;
```

Figure 27 - Percedimento MigrateCustomerData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

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



```
select * from Customer;
```

	CustomerKey	Title	FirstName	MiddleName	LastName	EmailAddress	Password	NameStyle	BirthDate	MaritalStatus	Gender	YearlyIncome
1	11000	NULL	Jon	V	Yang	jon24@adventure-works.com	yJXOm9Ad00x	FALSE	1966-04-08	M	M	90000
2	11001	NULL	Eugene	L	Huang	eugene10@adventure-works.com	NULL	FALSE	1965-05-14	S	M	60000
3	11002	NULL	Ruben	NULL	Torres	ruben35@adventure-works.com	NULL	FALSE	1965-08-12	M	M	60000
4	11003	NULL	Christy	NULL	Zhu	christy12@adventure-works.com	NULL	FALSE	1968-02-15	S	F	70000
5	11004	NULL	Elizabeth	NULL	Johnson	elizabeth5@adventure-works.com	NULL	FALSE	1968-08-08	S	F	80000
6	11005	NULL	Julio	NULL	Ruiz	julio1@adventure-works.com	NULL	FALSE	1965-08-05	S	M	70000
7	11006	NULL	Janet	G	Alvarez	janet9@adventure-works.com	NULL	FALSE	1965-12-06	S	F	70000
8	11007	NULL	Marco	NULL	Mehta	marco14@adventure-works.com	NULL	FALSE	1964-05-09	M	M	60000
9	11008	NULL	Rob	NULL	Verhoff	rob4@adventure-works.com	NULL	FALSE	1964-07-07	S	F	60000
10	11009	NULL	Shannon	C	Carlson	shannon38@adventure-works.com	NULL	FALSE	1964-04-01	S	M	70000
11	11010	NULL	Jacquelyn	C	Suarez	jacquelyn20@adventure-works.com	NULL	FALSE	1964-02-06	S	F	70000

Figure 28 - Tabela Customer

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Para fazer a migração de dados da base de dados antiga para a mais recente, tivemos de criar uma nova tabela “Sales2” na nova base de dados, para fazer a migração dos dados da base de dados antiga para a mais recente.

```
-- Tabela Sales2
CREATE TABLE Sales2
(
    ProductKey INT NOT NULL,
    OrderDateKey INT NOT NULL,
    DueDateKey INT NOT NULL,
    ShipDateKey INT NOT NULL,
    CustomerKey INT NOT NULL,
    PromotionKey INT NOT NULL,
    CurrencyKey INT NOT NULL,
    SalesTerritoryKey INT NOT NULL,
    SalesOrderNumber VARCHAR(255) NOT NULL,
    SalesOrderLineNumber INT NOT NULL,
    RevisionNumber INT NOT NULL,
    OrderQuantity INT NOT NULL,
    UnitPrice INT NOT NULL,
    ExtendedAmount INT NOT NULL,
    UnitPriceDiscountPct INT NOT NULL,
    DiscountAmount INT NOT NULL,
    ProductStandardCost INT NOT NULL,
    TotalProductCost INT NOT NULL,
    SalesAmount INT NOT NULL,
    TaxAmt INT NOT NULL,
    Freight INT NOT NULL,
    CarrierTrackingNumber NVARCHAR(255),
    CustomerPONumber NVARCHAR(255),
    OrderDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ShipDate DATE,
    PRIMARY KEY (ProductKey, CustomerKey, SalesTerritoryKey),
    FOREIGN KEY (ProductKey) REFERENCES Products(ProductKey),
    FOREIGN KEY (CustomerKey) REFERENCES Customer(CustomerKey),
    FOREIGN KEY (SalesTerritoryKey) REFERENCES SalesTerritory(SalesTerritoryKey),
    FOREIGN KEY (CurrencyKey) REFERENCES Currency(CurrencyKey)
);
```

Figure 29 - Criação de um novo Sales2

1ª Fase Relatório Técnico – Complementos de Bases de Dados

```
-- Criar a stored procedure para mover os dados
CREATE PROCEDURE dbo.MigrateSalesData
AS
BEGIN
    BEGIN TRY
        INSERT INTO Sales2
        (
            ProductKey, OrderDateKey, DueDateKey, ShipDateKey, CustomerKey, PromotionKey, CurrencyKey, SalesTerritoryKey,
            SalesOrderNumber, SalesOrderLineNumber, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct,
            DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight, CarrierTrackingNumber,
            CustomerPONumber, OrderDate, DueDate, ShipDate
        )
        SELECT
            ProductKey, OrderDateKey, DueDateKey, ShipDateKey, CustomerKey, PromotionKey, CurrencyKey, SalesTerritoryKey,
            SalesOrderNumber, SalesOrderLineNumber, RevisionNumber, OrderQuantity, UnitPrice, ExtendedAmount, UnitPriceDiscountPct,
            DiscountAmount, ProductStandardCost, TotalProductCost, SalesAmount, TaxAmt, Freight, CarrierTrackingNumber,
            CustomerPONumber, 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;

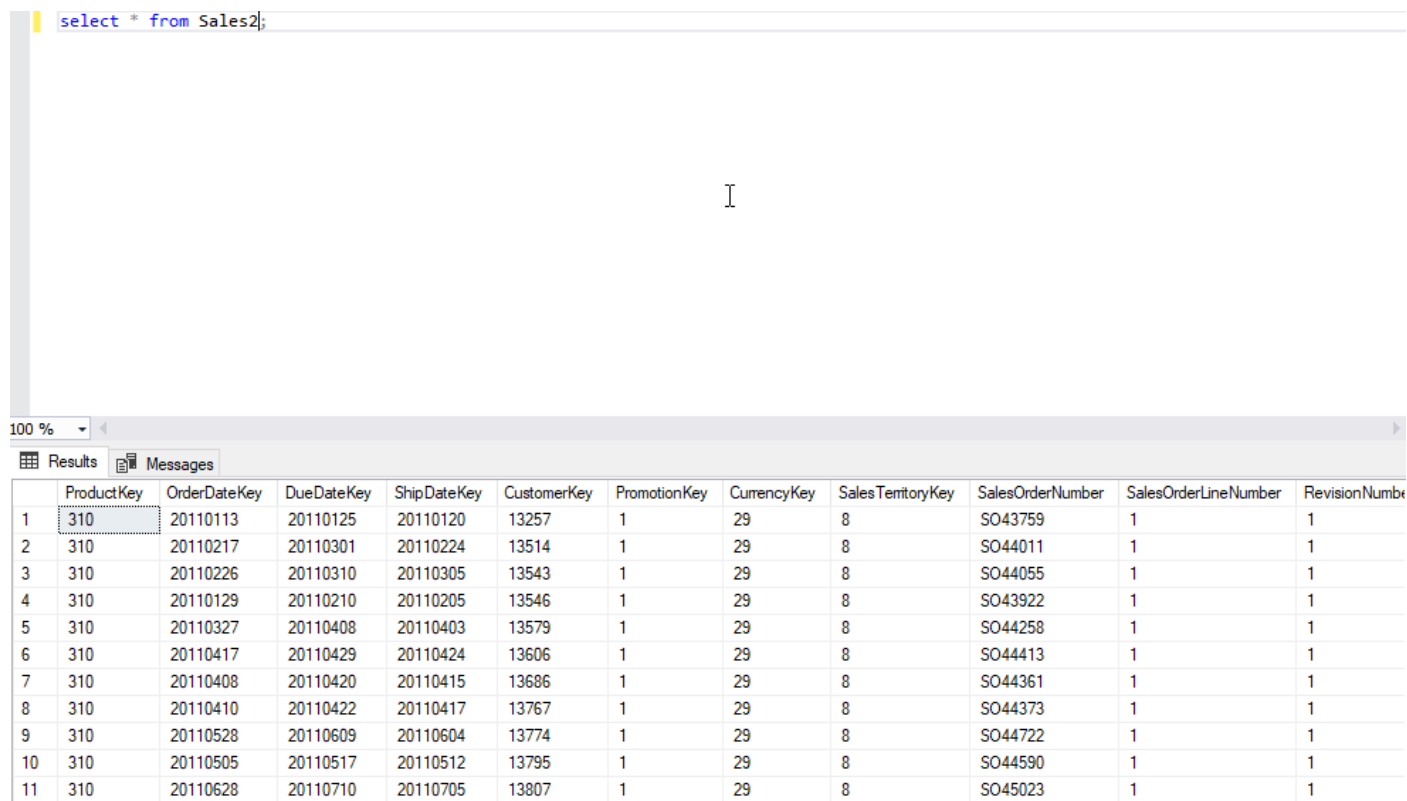
        THROW 50000, 'Ocorreu um erro durante a migração de dados da tabela Sales2. Entre em contato com o suporte.', 1;
    END CATCH
END;

--Executar o procedimento
EXEC dbo.MigrateSalesData;
```

Figure 30 - Procedimento MigrateSalesData

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Na seguinte imagem mostra a consulta da tabela “Sales2”.



The screenshot shows a SQL query editor with the query `select * from Sales2;` entered. Below the query, the results are displayed in a table. The table has 12 columns: ProductKey, OrderDateKey, DueDateKey, ShipDateKey, CustomerKey, PromotionKey, CurrencyKey, SalesTerritoryKey, SalesOrderNumber, SalesOrderLineNumber, and RevisionNumber. The results show 11 rows of data, all with ProductKey 310.

	ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	PromotionKey	CurrencyKey	SalesTerritoryKey	SalesOrderNumber	SalesOrderLineNumber	RevisionNumber
1	310	20110113	20110125	20110120	13257	1	29	8	SO43759	1	1
2	310	20110217	20110301	20110224	13514	1	29	8	SO44011	1	1
3	310	20110226	20110310	20110305	13543	1	29	8	SO44055	1	1
4	310	20110129	20110210	20110205	13546	1	29	8	SO43922	1	1
5	310	20110327	20110408	20110403	13579	1	29	8	SO44258	1	1
6	310	20110417	20110429	20110424	13606	1	29	8	SO44413	1	1
7	310	20110408	20110420	20110415	13686	1	29	8	SO44361	1	1
8	310	20110410	20110422	20110417	13767	1	29	8	SO44373	1	1
9	310	20110528	20110609	20110604	13774	1	29	8	SO44722	1	1
10	310	20110505	20110517	20110512	13795	1	29	8	SO44590	1	1
11	310	20110628	20110710	20110705	13807	1	29	8	SO45023	1	1

Figure 31 - Tabela Sales2

6. Programação

6.1 Views

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

6.2 Functions

Nome	Atributos	Requisito	Descrição
<i>dbo.udf_getUtilizador</i>	<i>@id_user INT</i>	<i>RO#</i>	<i>Permite obter informação detalhada sobre um utilizador</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

6.3 Stored procedures

Nome	Atributos	Requisito	Descrição
<i>dbo.MigrateCategoryListData</i>	@nome VARCHAR(50)	RO#	<i>Este procedimento dbo.MigrateCategoryListData move dados da tabela CategoryList de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateColorData</i>			<i>Este procedimento dbo.MigrateColorData move dados da tabela Color de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateCurrencyData</i>			<i>Este procedimento dbo.MigrateCurrencyData move dados da tabela Currency de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateCustomerData</i>			<i>Este procedimento dbo.MigrateCustomerData move dados da tabela Customer de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateModelData</i>			<i>Este procedimento dbo.MigrateModelData move dados da tabela Model de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateProductCategory Data</i>			<i>Este procedimento dbo.MigrateProductCategoryData move dados da tabela ProductCategory de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateProductsData</i>			<i>Este procedimento dbo.MigrateProductsData move dados da tabela Products de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>
<i>dbo.MigrateProductSubCategoryData</i>			<i>Este procedimento dbo.MigrateProductsSubCategoryData move dados da tabela ProductSubCategory de uma base de dados chamado AdventureOldData para a base de dados AdventureWorks.</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

<i>dbo.MigrateSalesData</i>			<i>Este procedimento <code>dbo.MigrateSalesData</code> move dados da tabela <code>Sales2</code> de uma base de dados chamado <code>AdventureOldData</code> para a base de dados <code>AdventureWorks</code>.</i>
<i>dbo.InsertSalesTerritoryData</i>			<i>Este procedimento <code>dbo.InsertSalesTerritory</code> move dados da tabela <code>SalesTerritory</code> de uma base de dados chamado <code>AdventureOldData</code> para a base de dados <code>AdventureWorks</code>.</i>
<i>dbo.CreateErrorTrigger</i>	<i>@TableName NVARCHAR(255)</i>		<i>O procedimento <code>dbo.CreateErrorTrigger</code> 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 <code>dbo.ErrorLog</code>. Essa abordagem oferece flexibilidade na criação de gatilhos personalizados para diferentes tabelas da base de dados.</i>
<i>GetSalesByYear</i>			<i>Este procedimento armazenado chamado <code>GetSalesByYear</code> 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 <code>Sales2</code>, agrupando os resultados por ano. Se ocorrer um erro durante a execução da consulta, o bloco <code>CATCH</code> captura o erro e utiliza <code>RAISERROR</code> para sinalizar a ocorrência do erro, exibindo a mensagem de erro</i>
<i>GetSalesByProductCategory</i>			<i>Este procedimento <code>GetSalesByProductCategory</code> busca obter informações de vendas agregadas por categoria de produto. Ele realiza uma consulta que junta tabelas como <code>Sales2</code>, <code>Products</code>, <code>CategoryList</code> e <code>ProductCategory</code> para calcular a soma total dos valores de vendas (<code>SalesAmount</code>) para cada categoria de produto (<code>EnglishProductCategoryName</code>). Se ocorrer algum erro durante a execução da consulta, o bloco <code>CATCH</code> captura e sinaliza o erro, exibindo a mensagem de erro, sua severidade e estado.</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

<i>GetSalesByProductSubCategory</i>			<i>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.</i>
<i>GetProductsSoldByCategoryAndYear</i>			<i>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.</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

6.4 Triggers

Nome	Tipo	Tabela	Requisito	Descrição
<i>dbo.trg_Customer_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo.Customer</i>	<i>R0#</i>	<i>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.</i>
<i>dbo.trg_CategoryList_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo. CategoryList</i>	<i>R0#</i>	<i>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.</i>
<i>dbo.trg_Color_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo. Color</i>	<i>R0#</i>	<i>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.</i>
<i>dbo.trg_Model_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo. Model</i>	<i>R0#</i>	<i>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.</i>
<i>dbo.trg_ProductCategory_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo. ProductCategory</i>	<i>R0#</i>	<i>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.</i>
<i>dbo.trg_Products_Error</i>	<i>AFTER UPDATE, INSERT, DELETE</i>	<i>dbo. Products</i>	<i>R0#</i>	<i>Esse Trigger está destinado a capturar e registar erros específicos relacionados às operações na tabela Products, mas</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

				atualmente está configurado apenas para simular um erro.
dbo.trg_ProductSubCategory_Error	AFTER UPDATE, INSERT, DELETE	dbo. ProductSubCategory	RO#	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	RO#	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	RO#	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	RO#	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	RO#	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	RO#	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.

7. Catálogo/Metadados

7.1 Monitorização

Nome	Atributos	Descrição
<i>dbo.MetadadosTabelas</i>	<i>TabelaID INT IDENTITY(1,1) PRIMARY KEY, NomeTabela NVARCHAR(128), NomeColuna NVARCHAR(128), TipoDados NVARCHAR(128), Tamanho INT, Restricoes NVARCHAR(512), DataAlteracao DATETIME</i>	<i>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.</i>
<i>dbo.MetadadosTabelas_ View</i>	<i>NomeTabela, NomeColuna, TipoDados, Tamanho, Restricoes, DataAlteracao</i>	<i>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.</i>

1ª Fase Relatório Técnico – Complementos de Bases de Dados

<i>dbo.EstatisticasTabelas</i>	<i>TabelaID INT IDENTITY(1,1) PRIMARY KEY, NomeTabela NVARCHAR(128), NomeColuna NVARCHAR(128), NumRegistros INT, EspacoOcupadoKB DECIMAL(10, 2), DataRegistro DATETIME</i>	<i>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</i>
--------------------------------	---	---

8. Descrição da Demonstração

Nesta 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.

8.1 Script de demonstração

Com o decorrer do projeto, tivemos de escrever vários códigos de SQL, mas para isso tivemos de pensar em que tipo de estrutura utilizar, ou seja, foi a partir daí que começamos a criar um modelo de entidade relação e um modelo relacional para podermos basear em uma estrutura já pensada.

Começamos por criar os requisitos adicionais, que pedia no enunciado, mas para isso tínhamos de ter um campo de password ligado os clientes (Customer). Com este passo feito, focamo-nos em fazer o que enunciado nos pedia que era caso o user fizesse login ou pedisse uma nova password, nós tínhamos de criar um corpo de email e enviar-lhe o email com a nova password.

Para elaborar esta tarefa declaramos um novo procedimento onde tem o nome de ‘sp_SendPasswordResetEmail’ este vai gerar uma nova password no e com essa fazer um corpo de email para enviar ao email do cliente, mas para isso ele vai colocar o email no “SendEmail”.

```
-- 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
    -- 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 (Mensaje, 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;
```

Figure 32 - Perocedimento sp_SendPasswordResetEmail

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Após efetuar a alteração de password de um cliente, começamos a pensar em adicionar acessos, editar acessos e removê los, assim sendo, nós criamos procedimentos para todos eles onde para adicionar tem como nome “AdicionarAcesso”, este procedimento permite colocar o id do cliente mais a data que acedeu na tabela Acessos, por fim, imprime “Acesso adicionado com sucesso.” caso tenha adicionado, caso contrário imprime “Erro ao adicionar o acesso.” Já o Editar criamos um procedimento com o nome de “EditarAcessos”, onde com o id do cliente alteramos a data de acesso do cliente, se tudo correr bem imprime ‘Acesso atualizado com sucesso.’, mas caso contraio imprime ‘Utilizador não encontrado ou acesso não atualizado.’ assim dizendo que não existe utilizador na tabela de Acessos. Para remover um acesso criamos o procedimento “RemoverAcesso” que vai receber o id do cliente e remover todos os acessos efetuados por esse cliente na tabela Acessos após executar vai imprimir um feedback de sucesso ou Erro.

```
-- Criar um procedimento RemoverAcesso
CREATE PROCEDURE RemoverAcesso
    @UserID INT
AS
BEGIN
    DELETE FROM Acessos
    WHERE CustomerKey = @UserID;

    -- Verificar se a exclusão foi bem-sucedida
    IF @@ROWCOUNT > 0
    BEGIN
        PRINT 'Acesso removido com sucesso.';
    END
    ELSE
    BEGIN
        PRINT 'Erro ao remover o acesso. Utilizador não encontrado.';
    END
END

-- Criar procedimento para EditarAcessos
CREATE PROCEDURE EditarAcessos
    @UserID INT
AS
BEGIN
    UPDATE Acessos
    SET DataAcesso = CURRENT_TIMESTAMP
    WHERE CustomerKey = @UserID;

    -- Verificar se a atualização foi bem-sucedida
    IF @@ROWCOUNT > 0
    BEGIN
        PRINT 'Acesso atualizado com sucesso.';
    END
    ELSE
    BEGIN
        PRINT 'Utilizador não encontrado ou acesso não atualizado.';
    END
END

-- Criar procedimento AdicionarAcesso
CREATE PROCEDURE AdicionarAcesso
    @UserID INT
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.';
    END
    ELSE
    BEGIN
        PRINT 'Erro ao adicionar o acesso.';
    END
END
```

Figure 33 - Procedimentos de Acesso

Ao concluirmos os acessos e as alterações na password do cliente, começamos a tratar do Layout da base de dados onde começamos a ver os registo de cada tabela, seu espaço, tamanho através de um SP (sp_spaceused) e a frente de cada SP colocamos o nome da tabela que queríamos saber.

```
-- 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 'Sales2';  
EXEC sp_spaceused 'SalesTerritory';  
EXEC sp_spaceused 'SentEmails';  
EXEC sp_spaceused 'sysdiagrams';
```

Figure 34 - Espaço ocupado por registo de cada tabela

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Ao ler o enunciado na parte dos Layout's nós começamos a perceber que era bom fornecer uma visão abrangente do espaço ocupado por cada tabela em nossa base de dados, juntamente com o número atual de registos, para isso criamos um Select onde buscava cada tabela da base de dados e colocava ela com as suas devidas informações.

```
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;
```

Figure 35 - Espaço ocupado por cada tabela com o número atual de registo

1ª Fase Relatório Técnico – Complementos de Bases de Dados

No tópico a seguir (No enunciado), pede para que a nossa base de dados faça uma a taxa de crescimento por cada tabela, assim baseando no que tínhamos feito no tópico anterior colocamos dentro de uma tabela temporária.

```
-- Criação de tabela temporaria 'HistoricalGrowth'
WITH HistoricalGrowth AS (
  -- Sua consulta original 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;
```

Figure 36 - Taxa de crescimento por tabela

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Por fim, nos Layout's tínhamos de colocar um limite de acessos, para isso usamos TRIGGER, onde eles vão verificar se tem mais de que 1 pessoa a fazer acesso a tabela, isso faz com que as tabelas não demorem muito tempo a entregar resultados, assim tornando-as eficientes, caso desse erro ele ia mandar o erro para ErrorLog.

```
CREATE PROCEDURE dbo.CreateErrorTrigger
    @TableName NVARCHAR(255)
AS
BEGIN
    DECLARE @TriggerScript NVARCHAR(MAX);

    -- Construir o script do gatilho dinâmico
    SET @TriggerScript = '
    CREATE TRIGGER Trg_' + @TableName + '_Error
    ON ' + @TableName + '
    AFTER INSERT, UPDATE, DELETE
    AS
    BEGIN
        -- Simular um erro (substitua por sua lógica de verificação de erro)
        IF (EXISTS (SELECT 1 FROM inserted) AND EXISTS (SELECT 1 FROM deleted))
        BEGIN
            DECLARE @ErrorMessage NVARCHAR(4000) = 'Erro simulado ocorreu na tabela ' + @TableName + ' durante uma operação de UPDATE.';
            DECLARE @ErrorNumber INT = 50000; -- Número de erro personalizado
            DECLARE @ErrorSeverity INT = 16; -- Severidade do erro
            DECLARE @ErrorState INT = 1; -- Estado do erro

            -- Chamar o procedimento armazenado para registrar o erro
            EXEC dbo.ErrorLog @ErrorMessage, @ErrorNumber, @ErrorSeverity, @ErrorState;
        END;
    END;
    ';

    -- Executar o script do gatilho dinâmico
    EXEC sp_executesql @TriggerScript;
END;
```

Figure 37 - Gatilho dinâmico para erros

1ª Fase Relatório Técnico – Complementos de Bases de Dados

Para colocar este trigger em cada tabela declaramos o procedimento que tem como objetivo tornar a criação de trigger mais dinâmico, assim só precisamos de dizer o nome da tabela ao chamar este procedimento.

```
EXEC dbo.CreateErrorTrigger @TableName = 'Customer';
EXEC dbo.CreateErrorTrigger @TableName = 'Sales2';
EXEC dbo.CreateErrorTrigger @TableName = 'Products';
EXEC dbo.CreateErrorTrigger @TableName = 'ProductSubCategory';
EXEC dbo.CreateErrorTrigger @TableName = 'SalesTerritory';
EXEC dbo.CreateErrorTrigger @TableName = 'Model';
EXEC dbo.CreateErrorTrigger @TableName = 'ProductCategory';
EXEC dbo.CreateErrorTrigger @TableName = 'Questions';
EXEC dbo.CreateErrorTrigger @TableName = 'Response';
EXEC dbo.CreateErrorTrigger @TableName = 'CategoryList';
EXEC dbo.CreateErrorTrigger @TableName = 'Color';
EXEC dbo.CreateErrorTrigger @TableName = 'SentEmails';
```

Figure 38 - Colocação de gatilhos para cada tabela

1ª Fase Relatório Técnico – Complementos de Bases de Dados

No 1º tópico do enunciado de Metadados destaca a implementação de uma stored procedure denominada GerarEntradasTabelaMetadados. Essa stored procedure foi projetada para gerar entradas detalhadas em uma tabela específica, denominada MetadadosTabelas. O objetivo principal é registrar informações cruciais sobre todas as tabelas e colunas presentes no banco de dados.

- O código é projetado para ser executado periodicamente, mantendo um histórico atualizado das alterações no esquema.
- A coluna Restrições inclui informações sobre restrições padrão, como chaves primárias e estrangeiras, proporcionando uma visão abrangente das estruturas de dados.
- Esse processo fornece uma camada de auditoria eficiente para o esquema do banco de dados, permitindo um rastreamento preciso das mudanças ao longo do tempo.

```
-- 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;
```

Figure 39 - histórico do esquema

1ª Fase Relatório Técnico – Complementos de Bases de Dados

No 2º tópico do enunciado dos Metadados destaca a criação de uma view denominada MetadadosTabelas_View, projetada para disponibilizar os dados mais recentes registrados na tabela MetadadosTabelas. utilizamos a MetadadosTabelas para obter informações históricas sobre tabelas e colunas, A cláusula WHERE filtra as entradas da tabela onde a data de alteração é igual à data da alteração mais recente registrada na tabela.

- A view é projetada para fornecer uma visão instantânea dos dados mais recentes na tabela MetadadosTabelas.
- A cláusula WHERE garante que apenas as entradas correspondentes à alteração mais recente sejam retornadas.

```
-- 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);
```

Figure 40 - Visão instantânea dos dados mais recentes

1ª Fase Relatório Técnico – Complementos de Bases de Dados

No último tópico dos Metadados tínhamos de implementar uma stored procedure chamada `RegistrarEstatisticasTabelas`, que tem como objetivo registrar, em uma tabela dedicada chamada `EstatisticasTabelas`, o número de registros e uma estimativa mais fiável do espaço ocupado para cada tabela na base de dados. Para isso usamos os Catálogos do sistema para obter informações sobre o número de registros e o espaço ocupado para cada tabela.

- O código é projetado para ser executado periodicamente, mantendo um histórico das estatísticas para cada tabela.
- As estatísticas incluem uma estimativa mais fiável do espaço ocupado, levando em consideração as partições e alocações de índices.

```
-- Metadados: Stored Procedure para registrar o número de registros e estimativa de espaço ocupado:
CREATE PROCEDURE dbo.RegistrarEstatisticasTabelas
AS
BEGIN
    -- Criar uma tabela de estatísticas (se ainda não existir)
    IF NOT EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'EstatisticasTabelas')
    BEGIN
        CREATE TABLE dbo.EstatisticasTabelas (
            TabelaID INT IDENTITY(1,1) PRIMARY KEY,
            NomeTabela NVARCHAR(128),
            NumRegistros INT,
            EspacoOcupadoKB DECIMAL(10, 2),
            DataRegistro DATETIME
        );
    END

    -- Estatísticas para cada tabela
    INSERT INTO dbo.EstatisticasTabelas (NomeTabela, NumRegistros, EspacoOcupadoKB, DataRegistro)
    SELECT
        t.name AS NomeTabela,
        SUM(p.rows) AS NumRegistros,
        SUM(a.total_pages) * 8 / 1024 AS EspacoOcupadoKB,
        GETDATE() AS DataRegistro
    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;
END;
```

Figure 41 - Histórico das estatísticas

9. 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 e gestão dos dados. 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 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.