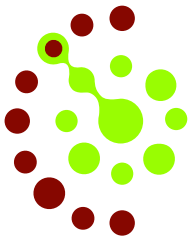


IPBeja

Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistema de aluguer de habitações

Beja, 21 de Dezembro de 2024

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistema de aluguer de habitações

Sara Soares de Oliveira - 24845
Amanda Breatriz Martinez Rodriguez - 25608

Lecionado por :

Elsa da Piedade Chinita Soares Rodrigues, IPBeja

Relatório de Trabalho Prático

2024

Resumo

Sistema de aluguer de habitações

Este relatório tem como objetivo apresentar de forma detalhada todas as etapas do desenvolvimento do banco de dados que suporta o sistema, desde a estruturação das tabelas e suas relações até à implementação de stored procedures, triggers, estratégias de segurança, cópias de segurança (backups) e práticas de manutenção e automatização.

...

...

Palavras-chave: banco de dados, stored procedures, triggers, segurança de dados, backups, manutenção de banco de dados, automatização, estruturação de tabelas, relações entre tabelas, desenvolvimento de sistemas.

Abstract

Sistema de aluguer de habitações

This report aims to detail all stages of the database development process that supports the system, from the structuring of tables and their relationships to the implementation of stored procedures, triggers, security strategies, backups, and maintenance and automation practices.

...

...

Keywords: database, stored procedures, triggers, data security, backups, database maintenance, automation, table structuring, table relationships, system development.

Conteúdo

Lista de Figuras

List of Listings

Capítulo 1

Introdução

Nos últimos anos, o mercado de aluguer de habitações para férias sofreu uma transformação significativa devido ao surgimento de plataformas digitais inovadoras que aproximaram proprietários e clientes, facilitando o processo de oferta e procura desses serviços. Plataformas como Airbnb, Booking e Trivago revolucionaram a forma como os imóveis são apresentados e reservados, oferecendo aos clientes uma experiência mais prática, transparente e segura. A disponibilização de fotos, vídeos, avaliações e comentários de outros utilizadores permitiu que os clientes tivessem uma visão mais clara do que estão a contratar, enquanto os proprietários conseguiram gerir os seus imóveis com maior autonomia e eficiência.

Além de melhorar a experiência do utilizador final, essas plataformas também desempenham um papel crucial na normalização do mercado, trazendo segurança e confiança para transações financeiras online. Atualmente, a simplicidade e rapidez com que um imóvel pode ser alugado, reservado e pago refletem a necessidade de sistemas robustos e eficientes que suportem um alto volume de operações, mantendo a integridade e segurança dos dados em um ambiente globalizado.

Neste contexto, o presente projeto tem como objetivo desenvolver uma base de dados relacional para um sistema de aluguer de habitações, concebido para abranger todas as funcionalidades essenciais desse tipo de serviço.

Capítulo 2

Criação da Base de Dados e seus Elementos

A base de dados, denominada "DREAMBOOKINGS", foi projetada utilizando SQL Server, onde foram implementados elementos fundamentais, como tabelas, índices e stored procedures, que interagem de forma coesa para garantir uma gestão eficaz dos dados.

2.1 FileGroups

A base de dados foi organizada com a separação de suas tabelas em Filegroups distintos, os quais foram estruturados de forma estratégica, separando tabelas e índices conforme a necessidade, volume de dados e frequência de acesso.

2.1.1 Estrutura dos Filegroups

1. PRIMARY: Contém as tabelas essenciais e críticas para o funcionamento do sistema, que possuem dados centrais e frequentemente consultados.

2. FG_GERAL: Armazena tabelas que contêm dados compartilhados ou com acesso secundário em diversas funcionalidades.

3. FG_USUARIOS: Filegroup dedicado às tabelas diretamente relacionadas aos utilizadores e sua gestão.

4. FG_PROPRIEDADES: Armazena informações relacionadas às propriedades anunciadas pelos proprietários.

5. FG_RESERVAS: Filegroup dedicado à gestão das reservas realizadas na plataforma.

6. FG_INDICES: Dedicado exclusivamente ao armazenamento de índices das principais tabelas do sistema.

Código para a criação dos filegroups

```
--FILEGROUPS--
CREATE DATABASE DREAMBOOKINGS
ON
PRIMARY
(
NAME = 'DREAMBOOKINGS_PRIMARY',
FILENAME = 'C:\Databases1\DREAMBOOKINGS_PRIMARY.mdf',
SIZE = 10MB,
MAXSIZE = UNLIMITED,
FILEGROWTH = 10MB
),
FILEGROUP FG_GERAL
(
NAME = 'DREAMBOOKINGS_FG_GERAL',
FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_GERAL.ndf',
SIZE = 10MB,
MAXSIZE = UNLIMITED,
FILEGROWTH = 10MB
),
FILEGROUP FG_USUARIOS
(
NAME = 'DREAMBOOKINGS_FG_USUARIOS',
FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_USUARIOS.ndf',
SIZE = 10MB,
MAXSIZE = UNLIMITED,
FILEGROWTH = 10MB
),
FILEGROUP FG_PROPRIEDADES
(
NAME = 'DREAMBOOKINGS_FG_PROPRIEDADES',
FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_PROPRIEDADES.ndf',
SIZE = 10MB,
MAXSIZE = UNLIMITED,
FILEGROWTH = 10MB
),
FILEGROUP FG_RESERVAS
(
NAME = 'DREAMBOOKINGS_FG_RESERVAS',
FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_RESERVAS.ndf',
SIZE = 10MB,
MAXSIZE = UNLIMITED,
FILEGROWTH = 10MB
),
FILEGROUP FG_INDICES
(
NAME = 'DREAMBOOKINGS_FG_INDICES',
```

```
47  FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_INDICES.ndf',
48  SIZE = 10MB,
49  MAXSIZE = UNLIMITED,
50  FILEGROWTH = 10MB
51  )
52  LOG ON
53  (
54  NAME = 'DREAMBOOKINGS_FG_Log',
55  FILENAME = 'C:\Databases1\DREAMBOOKINGS_FG_Log.ndf',
56  SIZE = 10MB,
57  MAXSIZE = UNLIMITED,
58  FILEGROWTH = 10MB
59  );
```

Listing 2.1: Código FILEGROUPS

Exemplo de funcionamento

2.2 Tabelas

Essas tabelas estão interligadas por chaves primárias e estrangeiras para garantir a integridade referencial e facilitar a navegação entre os dados, de forma que seja possível fazer consultas específicas como preço consoante à época do ano, avaliação e comentários.

2.2.1 Organização das Tabelas

1. PAIS: Armazena informações sobre os países e suas respetivas taxas de turismo.
2. CIDADE: Contém cidades associadas aos países.

3. PROPRIEDADES: Armazena informações das propriedades disponíveis para aluguer.
4. IMAGEM_VIDEO: Contém URLs de imagens e vídeos das propriedades.
5. CALENDARIO_DISPONIBILIDADE: Gerencia as datas de disponibilidade das propriedades.
6. PRECO_EPOCA: Define ajustes de preço para períodos específicos (baixa, meia e alta temporada).
7. CARACTERISTICAS_EXTRAS: Lista características adicionais que uma propriedade pode possuir.
8. CARACTERISTICAS_PROPRIEDADE: Relaciona propriedades com características adicionais.
9. PROPRIEDADES_DESTAQUE: Define propriedades destacadas na plataforma.
10. DESCONTOS: Gerencia descontos gerais ou baseados em códigos promocionais.
11. DESCONTO_PROPRIEDADE: Relaciona descontos a propriedades específicas.
12. RESERVA: Armazena todas as reservas realizadas no sistema.
13. CANCELAMENTO: Registra cancelamentos de reservas, seus motivos e valores reembolsados.
14. ESTADO_RESERVA: Gerencia os estados da reserva, como "PENDENTE", "CONFIRMADA"e "CANCELADA".
15. HISTORICO_RESERVA: Mantém um histórico de eventos associados às reservas.
16. PAGAMENTO: Registra os pagamentos associados às reservas.
17. AVALIACAO: Armazena avaliações feitas pelos clientes sobre as propriedades.

Código para a criação das tabelas

```
1          -- Criar Tabelas
2  -- Tabela PAIS
3  CREATE TABLE PAIS (
4      ID_PAIS INT PRIMARY KEY IDENTITY(1,1),
5      NOME VARCHAR(100) NOT NULL,
6      TAXA_TURISTA DECIMAL(10, 2) DEFAULT 0
7  ) ON FG_GERAL;
8
9  -- Tabela CIDADE
```

```
10  CREATE TABLE CIDADE (
11      ID_CIDADE INT PRIMARY KEY IDENTITY(1,1),
12      NOME VARCHAR(100) NOT NULL,
13      ID_PAIS INT NOT NULL,
14      FOREIGN KEY (ID_PAIS) REFERENCES PAIS(ID_PAIS)
15  ) ON FG_GERAL;
16
17  -- Tabela USUARIO
18  CREATE TABLE USUARIO (
19      ID_USER INT PRIMARY KEY IDENTITY(1,1),
20      NOME VARCHAR(100) NOT NULL,
21          DATA_NACIMENTO DATE NULL,
22      TELEFONE VARCHAR(20),
23          ID_CIDADE INT NOT NULL,
24      ENDERECO VARCHAR(255),
25          EMAIL VARCHAR(100) UNIQUE NOT NULL,
26      SENHA_HASH VARCHAR(64) NOT NULL,
27      TIPO_USUARIO VARCHAR(20)
28      CHECK (TIPO_USUARIO IN ('CLIENTE', 'PROPRIETARIO'))
29          FOREIGN KEY (ID_CIDADE) REFERENCES CIDADE(ID_CIDADE)
30  ) ON FG_USUARIOS;
31
32  -- Tabela CLIENTE
33  CREATE TABLE CLIENTE (
34      ID_CLIENTE INT PRIMARY KEY IDENTITY(1,1),
35      ID_USER INT NOT NULL,
36      FOREIGN KEY (ID_USER) REFERENCES USUARIO(ID_USER)
37  ) ON FG_USUARIOS;
38
39  -- Tabela PROPRIETARIO
40  CREATE TABLE PROPRIETARIO (
41      ID_PROPRIETARIO INT PRIMARY KEY IDENTITY(1,1),
42      ID_USER INT NOT NULL,
43      FOREIGN KEY (ID_USER) REFERENCES USUARIO(ID_USER)
44  ) ON FG_USUARIOS;
45
46  -- Tabela PROPRIEDADE
47  CREATE TABLE PROPRIEDADE (
48      ID_PROPRIEDADE INT PRIMARY KEY IDENTITY(1,1),
49      DESCRICAO VARCHAR(200) NOT NULL,
50      ID_CIDADE INT NOT NULL,
51      ID_PROPRIETARIO INT NOT NULL,
52      PRECO_BASE DECIMAL(10, 2) NOT NULL,
53      TIPO VARCHAR(50),
54      FOREIGN KEY (ID_CIDADE) REFERENCES CIDADE(ID_CIDADE),
55      FOREIGN KEY (ID_PROPRIETARIO)
56      REFERENCES PROPRIETARIO(ID_PROPRIETARIO)
```

```
57  ) ON FG_PROPRIEDADES;
58
59  -- Tabela IMAGEM_VIDEO
60  CREATE TABLE IMAGEM_VIDEO (
61      ID_IMAGEM_VIDEO INT PRIMARY KEY IDENTITY(1,1),
62      ID_PROPRIEDADE INT NOT NULL,
63      TIPO VARCHAR(20) CHECK (TIPO IN ('IMAGEM', 'VIDEO')),
64      URL VARCHAR(255),
65      DESCRICAO VARCHAR(255),
66      FOREIGN KEY (ID_PROPRIEDADE) REFERENCES PROPRIEDADE(ID_PROPRIEDADE)
67  ) ON FG_PROPRIEDADES;
68
69  -- Tabela CALENDARIO_DISPONIBILIDADE
70    CREATE TABLE CALENDARIO_DISPONIBILIDADE (
71      ID_CALENDARIO INT PRIMARY KEY IDENTITY(1,1),
72      ID_PROPRIEDADE INT NOT NULL,
73      DATA_INICIO DATE NOT NULL,
74      DATA_FIM DATE NOT NULL,
75      STATUS VARCHAR(20) DEFAULT 'INDISPONIVEL',
76      FOREIGN KEY (ID_PROPRIEDADE)
77      REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE
78  ) ON FG_PROPRIEDADES;
79
80  -- Tabela PRECO_EPOCA
81  CREATE TABLE PRECO_EPOCA (
82      ID_PRECO INT PRIMARY KEY IDENTITY(1,1),
83      ID_PROPRIEDADE INT NOT NULL,
84      NOME_EPOCA VARCHAR(100) NOT NULL,
85      DATA_INICIO DATE NOT NULL,
86      DATA_FIM DATE NOT NULL,
87      PERCENTUAL_AJUSTE DECIMAL(5,2) NOT NULL,
88      FOREIGN KEY (ID_PROPRIEDADE) REFERENCES PROPRIEDADE(ID_PROPRIEDADE)
89  ) ON FG_PROPRIEDADES;
90
91  -- Tabela CARACTERISTICAS_EXTRAS
92  CREATE TABLE CARACTERISTICAS_EXTRAS (
93      ID_CARACTERISTICAS INT PRIMARY KEY IDENTITY(1,1),
94      DESCRICAO VARCHAR(100) NOT NULL UNIQUE
95  ) ON FG_PROPRIEDADES;
96
97  -- Tabela CARACTERISTICAS_PROPRIEDADE
98  CREATE TABLE CARACTERISTICAS_PROPRIEDADE (
99      ID_PROPRIEDADE INT NOT NULL,
100     ID_CARACTERISTICAS INT NOT NULL,
101     PRIMARY KEY (ID_PROPRIEDADE, ID_CARACTERISTICAS),
102     FOREIGN KEY (ID_PROPRIEDADE)
103     REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE,
```

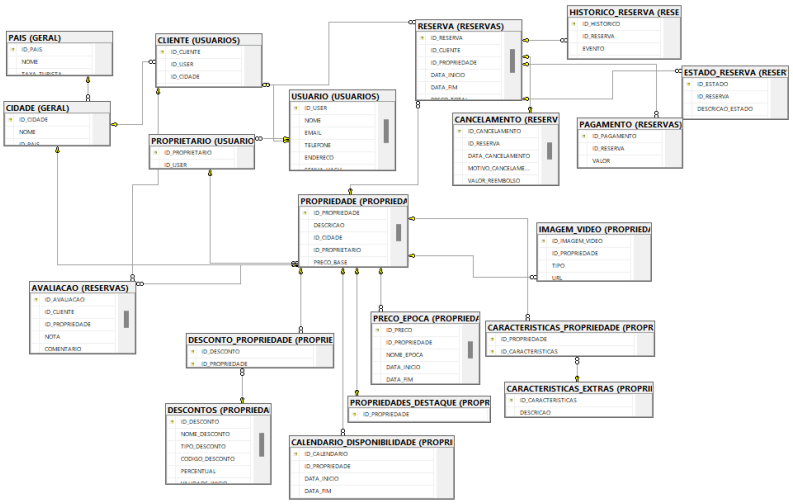
```
104     FOREIGN KEY (ID_CARACTERISTICAS)
105     REFERENCES CARACTERISTICAS_EXTRAS(ID_CARACTERISTICAS) ON DELETE CASCADE
106 ) ON FG_PROPRIEDADES;
107 
108 -- Tabela PROPRIEDADES_DESTAQUE
109 CREATE TABLE PROPRIEDADES_DESTAQUE (
110     ID_PROPRIEDADE INT PRIMARY KEY,
111     FOREIGN KEY (ID_PROPRIEDADE)
112     REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE
113 ) ON FG_PROPRIEDADES;
114 
115 -- Tabela DESCONTO
116 CREATE TABLE DESCONTOS (
117     ID_DESCONTO INT PRIMARY KEY IDENTITY(1,1),
118     NOME_DESCONTO VARCHAR(100) NOT NULL,
119     TIPO_DESCONTO VARCHAR(20) CHECK (TIPO_DESCONTO IN ('GERAL', 'CODIGO')),
120     CODIGO_DESCONTO VARCHAR(50),
121     PERCENTUAL DECIMAL(5,2) NOT NULL,
122     VALIDADE_INICIO DATE NOT NULL,
123     VALIDADE_FIM DATE NOT NULL
124 ) ON FG_PROPRIEDADES;
125 
126 -- Tabela DESCONTO_PROPRIEDADE
127 CREATE TABLE DESCONTO_PROPRIEDADE (
128     ID_DESCONTO INT NOT NULL,
129     ID_PROPRIEDADE INT NOT NULL,
130     PRIMARY KEY (ID_DESCONTO, ID_PROPRIEDADE),
131     FOREIGN KEY (ID_DESCONTO)
132     REFERENCES DESCONTOS(ID_DESCONTO) ON DELETE CASCADE,
133     FOREIGN KEY (ID_PROPRIEDADE)
134     REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE
135 ) ON FG_PROPRIEDADES;
136 
137 -- Tabela RESERVA
138 CREATE TABLE RESERVA (
139     ID_RESERVA INT PRIMARY KEY IDENTITY(1,1),
140     ID_CLIENTE INT NOT NULL,
141     ID_PROPRIEDADE INT NOT NULL,
142     DATA_INICIO DATE NOT NULL,
143     DATA_FIM DATE NOT NULL,
144     PRECO_TOTAL DECIMAL(10,2) DEFAULT 0,
145     DATA_RESERVA DATETIME DEFAULT GETDATE(),
146     FOREIGN KEY (ID_CLIENTE)
147     REFERENCES CLIENTE(ID_CLIENTE) ON DELETE CASCADE,
148     FOREIGN KEY (ID_PROPRIEDADE)
149     REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE
150 ) ON FG_RESERVAS;
```

```
151
152 -- Tabela CANCELAMENTO
153 CREATE TABLE CANCELAMENTO (
154     ID_CANCELAMENTO INT IDENTITY(1,1) PRIMARY KEY ,
155     ID_RESERVA INT UNIQUE NOT NULL,
156     DATA_CANCELAMENTO DATETIME DEFAULT GETDATE(),
157     MOTIVO_CANCELAMENTO VARCHAR(255),
158     VALOR_REEMBOLSO DECIMAL(10, 2),
159     FOREIGN KEY (ID_RESERVA) REFERENCES RESERVA(ID_RESERVA)
160 )ON FG_RESERVAS;
161
162 -- Tabela ESTADO_RESERVA
163 CREATE TABLE ESTADO_RESERVA (
164     ID_ESTADO INT PRIMARY KEY IDENTITY(1,1),
165     ID_RESERVA INT NOT NULL,
166     DESCRICAO_ESTADO VARCHAR(50)
167     CHECK (DESCRICAO_ESTADO IN ('PENDENTE', 'CONFIRMADA', 'CANCELADA')),
168     DATA_ALTERACAO DATETIME DEFAULT GETDATE(),
169     FOREIGN KEY (ID_RESERVA) REFERENCES RESERVA(ID_RESERVA)
170 ) ON FG_RESERVAS;
171
172 -- Tabela HISTORICO_RESERVA
173 CREATE TABLE HISTORICO_RESERVA (
174     ID_HISTORICO INT PRIMARY KEY IDENTITY(1,1),
175     ID_RESERVA INT NOT NULL,
176     EVENTO VARCHAR(255) NOT NULL,
177     DATA_EVENTO DATETIME DEFAULT GETDATE(),
178     FOREIGN KEY (ID_RESERVA) REFERENCES RESERVA(ID_RESERVA)
179 ) ON FG_RESERVAS;
180
181 -- Tabela PAGAMENTO FEITO
182 CREATE TABLE PAGAMENTO (
183     ID_PAGAMENTO INT PRIMARY KEY IDENTITY(1,1),
184     ID_RESERVA INT NOT NULL,
185     VALOR DECIMAL(10,2) ,
186     METODO_PAGAMENTO VARCHAR(50) CHECK (METODO_PAGAMENTO IN
187     ('CARTAO', 'TRANSFERENCIA BANCARIA', 'MBWAY', 'DINHEIRO')),
188     TIPO_PAGAMENTO VARCHAR(20) DEFAULT 'PAGAMENTO',
189     DATA_PAGAMENTO DATETIME DEFAULT GETDATE(),
190     FOREIGN KEY (ID_RESERVA) REFERENCES RESERVA(ID_RESERVA)
191 ) ON FG_RESERVAS;
192
193 -- Tabela AVALIACAO
194 CREATE TABLE AVALIACAO (
195     ID_AVALIACAO INT PRIMARY KEY IDENTITY(1,1),
196     ID_CLIENTE INT NOT NULL,
197     ID_PROPRIEDADE INT NOT NULL,
```

```
198      NOTA INT NOT NULL CHECK (NOTA BETWEEN 1 AND 5),
199      COMENTARIO TEXT,
200      DATA_AVALIACAO DATETIME DEFAULT GETDATE(),
201      FOREIGN KEY (ID_CLIENTE)
202      REFERENCES CLIENTE(ID_CLIENTE) ON DELETE CASCADE,
203      FOREIGN KEY (ID_PROPRIEDADE)
204      REFERENCES PROPRIEDADE(ID_PROPRIEDADE) ON DELETE CASCADE
205  ) ON FG_RESERVAS;
206
```

Listing 2.2: Código SQL para criação das tabelas

Exemplo de funcionamento



Capítulo 3

Método de Preenchimento da Base de Dados

Existem diversas maneiras de inserir dados em um banco de dados, como por meio de inserções manuais, interfaces gráficas ou processos automatizados. Neste caso, optamos por utilizar um método automatizado através de scripts desenvolvidos em Python devido à sua eficiência, flexibilidade e simplicidade na geração e manipulação de dados.

3.1 Funcionamento Geral do Preenchimento Automatizado com Python

O preenchimento automatizado do banco de dados segue quatro etapas principais:

1. Geração de Dados Fictícios ou Realista ou Importação de Dados Reais de Arquivos: Utiliza-se bibliotecas como Faker para criar dados fictícios, como nomes, e-mails, endereços e telefones. Esses dados simulam informações reais que podem ser usadas para testes ou povoamento inicial da base. Quando dados reais estão disponíveis em formatos como CSV ou TXT, realizamos a leitura e processamento desses arquivos para inserção no banco.

2. Tratamento e Manipulação dos Dados: Antes de inserir os dados, é comum realizar algumas operações, como: Remoção de caracteres especiais para garantir compatibilidade com o banco. Estruturação dos dados no formato exigido pelo banco de dados.

3. Conexão com o Banco de Dados: A conexão é estabelecida utilizando bibliotecas de interface com bancos de dados. No caso de SQL Server, a biblioteca pyodbc é amplamente utilizada. Ela permite enviar comandos SQL diretamente para o banco.

4. Inserção dos Dados: Após a conexão ser estabelecida, os dados são inseridos nas tabelas desejadas utilizando comandos SQL como INSERT INTO. Um loop percorre os registros gerados e executa as inserções. Ao final, é feita a confirmação da transação (commit), garantindo que os dados sejam salvos.

Vantagens da Abordagem Utilizada

1. Automatização: Gera e insere grandes volumes de dados em um curto espaço de tempo.
2. Escalabilidade: Pode ser facilmente adaptado para inserir dados em múltiplas tabelas ou sistemas diferentes.
3. Flexibilidade: Bibliotecas como Faker permitem personalizar os dados gerados conforme a necessidade do sistema.
4. Segurança: O uso de técnicas de criptografia, como hashes para senhas, garante a proteção das informações sensíveis.
5. Testes Realistas: Dados simulados aproximam-se de informações reais, facilitando testes de funcionalidades no sistema.

Bibliotecas Utilizadas no Processo

1. Faker: Geração de dados fictícios como nomes, e-mails, endereços, etc.
2. Unicodedata: Tratamento de strings para remover caracteres especiais.
3. Hashlib: Criação de hashes para senhas, garantindo segurança.
4. PyODBC: Interface para conectar e executar comandos SQL em bancos de dados.

Códigos para preenchimentos

```
import pyodbc
# Configuração do banco de dados SQL Server
server = 'AMANDABMR'  # Nome do seu servidor
database = 'DREAMBOOKINGS'  # Nome do seu banco de dados
username = 'AMANDABMR\\amand'  # Nome do usuário (com o domínio, se necessário)
password = 'SUA_SENHA'  # Senha do usuário

# Conectar ao banco de dados
conn = pyodbc.connect(f'DRIVER={{SQL Server}};SERVER={server};
DATABASE={database};Trusted_Connection=yes')
cursor = conn.cursor()

# Função para ler o arquivo TXT
```

```
14  def ler_arquivo_txt(caminho_arquivo):
15      with open(caminho_arquivo, 'r', encoding='utf-8') as arquivo:
16          linhas = arquivo.readlines()
17      return [linha.strip().split(",") for linha in linhas]
18
19  # Função para inserir os dados na tabela GERAL.PAIS
20  def inserir_dados_pais(paises):
21      query = """
22      INSERT INTO GERAL.PAIS (NOME, TAXA_TURISTA)
23      VALUES (?, ?)
24      """
25      for pais in paises:
26          try:
27              cursor.execute(query, pais)
28              print(f"País {pais[0]} inserido com sucesso!")
29          except Exception as e:
30              print(f"Erro ao inserir o país {pais[0]}: {e}")
31      conn.commit()
32
33  # Função para inserir os dados na tabela GERAL.CIDADE
34  def inserir_dados_cidade(cidades):
35      query = """
36      INSERT INTO GERAL.CIDADE (NOME, ID_PAIS)
37      VALUES (?, ?)
38      """
39      for cidade in cidades:
40          try:
41              cursor.execute(query, cidade)
42              print(f"Cidade {cidade[0]} inserida com sucesso!")
43          except Exception as e:
44              print(f"Erro ao inserir a cidade {cidade[0]}: {e}")
45      conn.commit()
46
47  # Caminho do arquivo TXT
48  caminho_arquivo_pais = "Dados_Pais.txt"
49  caminho_arquivo_cidade = "Dados_Cidade.txt"
50
51  # Ler os arquivos e realizar os inserts
52  paises = ler_arquivo_txt(caminho_arquivo_pais)
53  inserir_dados_pais(paises)
54
55  cidades = ler_arquivo_txt(caminho_arquivo_cidade)
56  inserir_dados_cidade(cidades)
57
58  # Fechar conexão
59  cursor.close()
60  conn.close()
```

```
61
```

Listing 3.1: Código para inserir Cidades através dum arquivo de texto

Exemplo de funcionamento

	ID_CIDADE	NOME	ID_PAIS
1	1	Paris	1
2	2	Marseille	1
3	3	Lyon	1
4	4	Toulouse	1
5	5	Nice	1
6	6	Nantes	1
7	7	Strasbourg	1
8	8	Bordeaux	1
9	9	Lille	1
10	10	Rennes	1
11	11	Madrid	2
12	12	Barcelona	2
13	13	Sevilha	2
14	14	Valencia	2
15	15	Bilbao	2

Figura 3.1: Tabela Cidades

```python
import random
from faker import Faker
import unicodedata
import hashlib
import pyodbc
from datetime import datetime, timedelta

# Instanciar o gerador de dados
fake = Faker("pt_PT")
# Função para remover caracteres especiais
def remover_caracteres_especiais(texto):
    return unicodedata.normalize('NFKD', texto).encode
    ('ASCII', 'ignore').decode('ASCII')
# Função para gerar hash da senha
def gerar_senha_hash(senha):
    return hashlib.sha256(senha.encode('utf-8')).hexdigest()
# Função para gerar uma data de nascimento aleatória
def gerar_data_nascimento():
    idade = random.randint(18, 65)
    today = datetime.today()
    data_nascimento = today - timedelta(days=idade * 365)
```

```
22      return data_nascimento.date()
23
24  # Função para buscar um ID_CIDADE aleatório na tabela CIDADE
25  def buscar_id_cidade():
26      cursor.execute("SELECT ID_CIDADE FROM GERAL.CIDADE")
27      cidades = cursor.fetchall()
28      if cidades:
29          cidade_escolhida = random.choice(cidades)  # Escolhe uma cidade
30          aleatoriamente
31          return cidade_escolhida[0]  # Retorna o ID_CIDADE
32      else:
33          print("Nenhuma cidade encontrada na tabela CIDADE.")
34          return None
35  # Função para gerar os dados dos usuários
36  def gerar_dados_usuarios(num_usuarios):
37      usuarios = []
38      for _ in range(num_usuarios):
39          nome = remover_caracteres_especiais(fake.first_name() + " " +
40          fake.last_name())
41          email = remover_caracteres_especiais(fake.unique.email())
42          telefone = fake.phone_number()
43          endereco = remover_caracteres_especiais(fake.street_name())
44          senha_hash = gerar_senha_hash(fake.password(length=8))
45          tipo_usuario = random.choice(["CLIENTE", "PROPRIETARIO"])
46          data_nascimento = gerar_data_nascimento()
47          # Buscar o ID_CIDADE da tabela CIDADE
48          id_cidade = buscar_id_cidade()
49          if id_cidade:
50              # Convertendo a data de nascimento para string no formato YYYY-MM-DD
51              data_nascimento_str = data_nascimento.strftime('%Y-%m-%d')
52              usuarios.append((nome, email, telefone, endereco, senha_hash,
53              tipo_usuario, id_cidade, data_nascimento_str))
54      return usuarios
55  # Configuração do banco de dados SQL Server
56  server = 'AMANDABMR'
57  database = 'DREAMBOOKINGS'
58  username = 'AMANDABMR\\amand'
59  password = 'SUA_SENHA'
60  # Conectar ao banco de dados
61  conn = pyodbc.connect(f'DRIVER={{SQL Server}};SERVER={server};DATABASE=
62  {database};Trusted_Connection=yes')
63  cursor = conn.cursor()
64  # Função para inserir os dados no banco de dados
65  def inserir_dados(usuarios):
66      query = """
67      INSERT INTO USUARIOS.USUARIO (NOME, EMAIL, TELEFONE, ENDERECO, SENHA_HASH,
68      TIPO_USUARIO, ID_CIDADE, DATA_NACIMENTO)
```

```
69          VALUES (?, ?, ?, ?, ?, ?, ?, ?)
70          """
71          for usuario in usuarios:
72              if usuario[6] is not None:  # Verifica se o ID_CIDADE foi encontrado
73                  try:
74                      cursor.execute(query, usuario[0], usuario[1], usuario[2],
75                      usuario[3], usuario[4], usuario[5], usuario[6], usuario[7])
76                      print(f"Usuário {usuario[0]} inserido com sucesso!")
77                  except Exception as e:
78                      print(f"Erro ao inserir o usuário {usuario[0]}: {e}")
79              else:
80                  print(f"Não foi possível inserir o usuário {usuario[0]} devido à
81                  cidade não encontrada.")
82          conn.commit()
83      # Configurações
84      total_usuarios = 500
85      # Gerar os usuários e inserir diretamente no banco de dados
86      usuarios = gerar_dados_usuarios(total_usuarios)
87      inserir_dados(usuarios)
88      # Fechar conexão
89      cursor.close()
90      conn.close()
91      print("Todos os usuários foram inseridos com sucesso!")
```

Listing 3.2: Código para gerar e inserir usuários

Exemplo de funcionamento

97 %

Results | Messages

	ID_USER	NOME	EMAIL	TELEFONE	ENDERECO	SENHA_HASH	TIPO_USUARIO
1	1	Josue Camara	maria-eduardajesus@example.org	81 7310-1508	Esplanada de Cassiano	34cee60e5f96d144bd7322aa66a4116caa46fe343a1cc06ee7...	CLIENTE
2	2	Caio Pereira	xpereira@example.org	61 4652-3336	Residencial Augusto Carvalho	ba3d7f66f12fa9e3d7c04ac4e7c26542e25f9990eabfb0d7212...	CLIENTE
3	3	Benicio da Rocha	zoeviana@example.com	+55 (061) 1726-9946	Conjunto Almeida	c5e1207225a7c69b7aaa2ce15da6447484bb72a2ed30d14c8...	PROPRIETARIO
4	4	Gael Rezende	davi-luizcaldeira@example.com	(061) 6279 5210	Via Cardoso	a85d3669b9ef08c1dd740adcfbe260d765a59be18628f97f04b...	PROPRIETARIO
5	5	Maria Luiza Gomes	joanacamargo@example.com	11 9602-0580	Distrito Cavalcanti	b6c2d1873c28f60e6d1d4b59a8289b724365789abdcaf48b1b...	PROPRIETARIO
6	6	Maria Laura Vasconcelos	crios@example.com	+55 (031) 6332-5722	Vereda de Pimenta	36d0b7790011e9679a12f09596572f2bd6ecbf80791e91f156...	PROPRIETARIO
7	7	Arthur Gabriel Cavalcante	isis78@example.net	81 6586 5760	Viela Sampaio	51ba6bc03b38eadc57b33d47db943e89b09ade30c5104929...	PROPRIETARIO
8	8	Maria Alice Silveira	meloheloisa@example.org	+55 (084) 9653-1406	Avenida Pastor	5da5ce22f14273601fb43ea034f8c010bbead6f35b243ef5203...	PROPRIETARIO
9	9	Heitor Moreira	nina38@example.org	+55 (084) 9753-6671	Ladeira Otavio Gomes	b02ae2185da6cd4db19583062d10f21a1f01ae7f7a49a02a3e...	PROPRIETARIO
10	10	Bella Nascimento	bruno61@example.com	(071) 7109-7572	Feira Cardoso	54294c4cead0875673b866a6bb679cb717ab53279fed49c2d...	CLIENTE
11	11	Theo Moreira	lferreira@example.net	0300-917-1700	Conjunto Pietra Nunes	2e139a1deededbda03a88050391e6b23f158883d574dfb765...	CLIENTE
12	12	Otto Jesus	das-neveslara@example.com	0500 483 4113	Campo de da Mota	919b6e8e20812924259b4cf446d8e154e3dfecbd8d9f24ddd8...	PROPRIETARIO
13	13	Leo Moura	diego14@example.net	71 7373-6443	Vale Maria Alice Casa Grande	42843921c7418317692220 48c9cb2b5b042009ee3f0fbabf84f...	CLIENTE
14	14	Joao Vitor da Mota	anna-liznogueira@example.org	+55 31 9096 9645	Praca Thales Abreu	3a26e3e2b3d21b8f8eac1b902cd585cee9d6773616db04cc9...	CLIENTE
15	15	Nicole Moraes	afarias@example.org	51 1939 3022	Esplanada Marcos Vinicius Duarte	255237538bb2085ac3f47bcde3d029a6e26edbb39700f817a...	PROPRIETARIO
16	16	Maria Julia Fogaca	isadoraandrade@example.net	(061) 4009 8704	Lagoa Carolina Vieira	0672d19ef306f98e91b5b26d1d15a942822ec35f7cabf7f9143...	CLIENTE
17	17	Isabella Sales	ceciliacamargo@example.com	31 8837 0291	Lago Camargo	76e8b38135e7a9b280033cc75c3f80a72703603e3a68ca781...	CLIENTE
18	18	Alexia Novaes	vcastro@example.net	0300 849 2470	Praca da Cunha	b42841460cc9bb5113acc66e959c654caf4144e99a52e2934c...	PROPRIETARIO
19	19	Brayan da Rocha	vda-cruz@example.net	+55 (071) 9856-9899	Aeroporto Maysa Ribeiro	b5dd680c3674dd33225bcd3103c7f94e2c309c0e8ecafb5a6b...	PROPRIETARIO
20	20	Davi Luiz Barros	renannovaes@example.com	41 7754-5945	Alameda Elisa Novais	1ff86a73516bdfadedcfecfb6489b462a646a45b7903d05ad76...	PROPRIETARIO
21	21	Joao Vitor Aparecida	iteixeira@example.com	(061) 1370-7764	Patio de Camara	e2be9ee78bb20d5367d1f8d1209fa0464be0d25dc0489f9bb4...	CLIENTE
22	22	Rebeca Teixeira	yuricastro@example.com	(051) 6633-3759	Nucleo Elisa Albuquerque	da0b1d302fea80be1dbec6fbc52f12bc361b16c8a76ff66f18b6...	CLIENTE
23	23	Levi Novaes	qnogueira@example.net	0800 201 9812	Aeroporto de Costa	d6c985cf75dbab42a6538bd0f57bb6e59a803c35b981fa9c5ef...	PROPRIETARIO
24	24	Davi Luiz Garcia	gomesotavio@example.net	(021) 7704 8364	Chacara de da Cunha	25513bee3f93377bf69469663823efff6809a1f4bd8b59174df7...	PROPRIETARIO
25	25	Jose Peixoto	pnascimento@example.com	41 7504-7268	Aeroporto de Aragao	6a22df2bf13154371bc4b4446b0eec8ac5fdb24662d31f88445...	CLIENTE
26	26	Leticia Vasconcelos	machadomaysa@example.com	51 6244 4078	Estrada Joao Miguel Gomes	d57b1d9438723cdf2aa1274039457ce6b59ed6783a32e92f9	CLIENTE

Figura 3.2: Tabela Usuarios

Capítulo 4

Store Procedures e Triggers

No contexto de bancos de dados modernos, automação, eficiência e controle são indispensáveis. Stored Procedures e Triggers emergem como ferramentas cruciais para otimizar o processamento de dados e preservar a integridade das informações.

4.1 Stored procedures

Stored Procedures são blocos de código SQL pré-compilados, armazenados no banco e executados sob demanda. Promovem reutilização, organização e melhor desempenho ao centralizar lógicas complexas, evitando redundâncias e reduzindo a sobrecarga de processamento.

No sistema DREAMBOOKINGS, foram utilizadas para automatizar notificações por e-mail em eventos como confirmações de reservas e pagamentos.

A função sp_send_dbmail do SQL Server possibilita o envio de e-mails. O código a seguir configura o Database Mail, ativando as opções avançadas e o recurso Extended Procedures (XPs), essencial para o envio de e-mails pelo servidor.

4.1.1 Implementação de e-mails automáticos

Código para ativar Database Mail

```
1     --EMAIL
2          --VERIFICACAO
3  EXEC sp_configure 'show advanced options', 1;
4  RECONFIGURE;
5  EXEC sp_configure 'Database Mail XPs', 1;
6  RECONFIGURE;
```

Listing 4.1: Código para ativar os mails

sp_EnviarEmailConfirmacaoReserva:

Esta Stored Procedure é acionada sempre que uma reserva é confirmada. Ela utiliza o recurso Database Mail do SQL Server para enviar um e-mail ao cliente contendo os detalhes da reserva, como número da reserva e agradecimento.

```
1  -- Procedimento para enviar um e-mail de confirmação de reserva
2  CREATE PROCEDURE sp_EnviarEmailConfirmacaoReserva
3      @ID_RESERVA INT
4  AS
5  BEGIN
6      DECLARE @EmailCliente NVARCHAR(255), @Assunto NVARCHAR(255),
7      @Mensagem NVARCHAR(MAX);
8
9      -- Obter o e-mail do cliente associado à reserva
10     SELECT @EmailCliente = usuario.EMAIL
11     FROM USUARIOS.usuario
12     JOIN USUARIOS.CLIENTE cliente ON usuario.ID_USER = cliente.ID_USER
13     JOIN RESERVAS.RESERVA reserva ON cliente.ID_CLIENTE = reserva.ID_CLIENTE
14     WHERE reserva.ID_RESERVA = @ID_RESERVA;
15
16     -- Definir o assunto e a mensagem do e-mail
17     SET @Assunto = 'Confirmação de Reserva';
18     SET @Mensagem = 'Sua reserva foi confirmada com sucesso. Detalhes: ' +
19                     'Número da Reserva: ' + CAST(@ID_RESERVA AS NVARCHAR) +
20                     '. Agradecemos por escolher os serviço DREAMBOOKINGS!.';
21
22     -- Enviar o e-mail
23     EXEC msdb.dbo.sp_send_dbmail
24         @profile_name = 'DREAMBOOKINGS',
25         @recipients = @EmailCliente,
26         @subject = @Assunto,
27         @body = @Mensagem;
28 END;
29 GO
```

Listing 4.2: Código sp_EnviarEmailConfirmacaoReserva

sp_EnviarEmailPagamentoRealizado

Quando o pagamento de uma reserva é processado, esta Stored Procedure envia um e-mail de confirmação ao cliente, informando que o pagamento foi recebido.

```sql
-- Procedimento para enviar um e-mail de pagamento realizado
CREATE PROCEDURE sp_EnviarEmailPagamentoRealizado
    @ID_RESERVA INT
AS
BEGIN
    DECLARE @EmailCliente NVARCHAR(255),
    @Assunto NVARCHAR(255), @Mensagem NVARCHAR(MAX);

    -- Obter o e-mail do cliente associado à reserva
    SELECT @EmailCliente = usuario.EMAIL
    FROM USUARIOS.USUARIO usuario
    JOIN USUARIOS.CLIENTE cliente ON usuario.ID_USER = cliente.ID_USER
    JOIN RESERVAS.RESERVA reserva ON cliente.ID_CLIENTE = reserva.ID_CLIENTE
    WHERE reserva.ID_RESERVA = @ID_RESERVA;

    -- Definir o assunto e a mensagem do e-mail
    SET @Assunto = 'Pagamento Realizado';
    SET @Mensagem = 'Seu pagamento foi recebido com sucesso
    para a reserva número: ' +
                    CAST(@ID_RESERVA AS NVARCHAR) +
                                    '. Agradecemos por escolher
                        os serviço DREAMBOOKINGS!.';

    -- Enviar o e-mail
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'DREAMBOOKINS',
        @recipients = @EmailCliente,
        @subject = @Assunto,
        @body = @Mensagem;
END;
GO
```

Listing 4.3: Código sp_EnviarEmailPagamentoRealizado

No momento não conseguimos testar de forma prática estes códigos, pois são criados com e-mail fictício.

4.2 Triggers

Triggers operam de forma reativa, sendo ativados automaticamente por eventos como INSERT, UPDATE ou DELETE. São adequados para implementar regras de negócio, auditar dados e assegurar a consistência das informações sem intervenção manual.

4.2.1 Triggers Implementados

Para automatizar verificações e atualizações na nossa base de dados, implementámos os seguintes triggers:

TRG_Insert_Cliente_or_Proprietario

Este Trigger insere automaticamente os dados nas tabelas CLIENTE ou PROPRIETARIO, conforme o tipo definido na tabela USUARIO.

```
1  --INSERT CLIENTE/PROPRIETARIO--
2  CREATE TRIGGER TRG_INSERT_CLIENTE_OR_PROPRIETARIO
3  ON USUARIOS.USUARIO
4  AFTER INSERT
5  AS
6  BEGIN
7  -- Insertar en la tabla CLIENTE si el tipo de usuario es CLIENTE
8     INSERT INTO USUARIOS.CLIENTE (ID_USER)
9      SELECT ID_USER
10     FROM inserted
11     WHERE TIPO_USUARIO = 'CLIENTE';
12
13 -- Insertar en la tabla PROPRIETARIO si el tipo de usuario es PROPRIETARIO
14     INSERT INTO USUARIOS.PROPRIETARIO (ID_USER)
15     SELECT ID_USER
16     FROM inserted
17     WHERE TIPO_USUARIO = 'PROPRIETARIO';
18 END;
```

Listing 4.4: Código TRG_Insert_Cliente_or_Proprietario

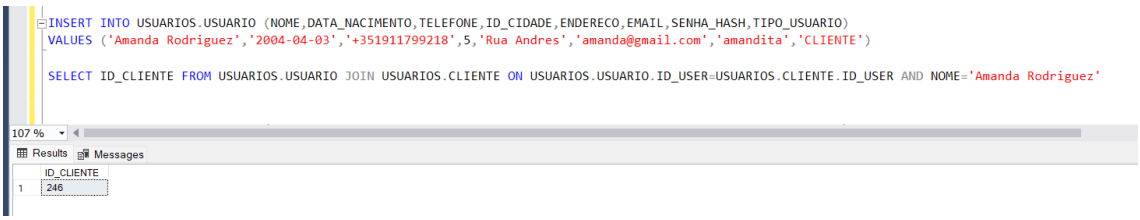


Figura 4.1: TRG_Insert_Cliente_or_Proprietario Exemplo 1 do funcionamento

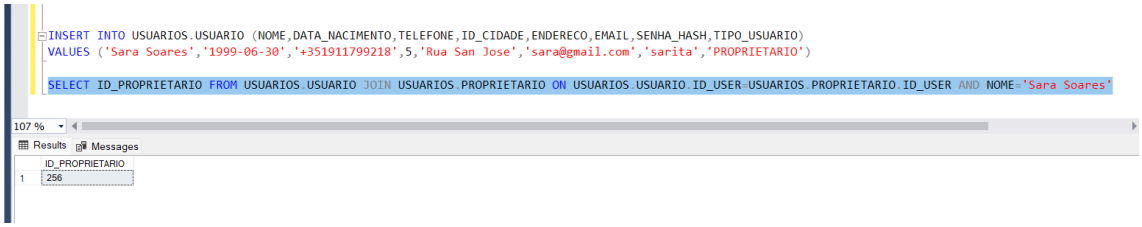


Figura 4.2: TRG_Insert_Cliente_or_Proprietario Exemplo 2 do funcionamento

TGR_AtualizarPrecoReserva

Ativado após a inserção ou atualização de uma reserva, este Trigger calcula o preço total da reserva considerando

1. Ajustes de preço por temporada
2. Aplicação de descontos personalizados
3. Inclui automaticamente a taxa turística

```
--PRECO NA RESERVA--
CREATE TRIGGER TGR_AtualizarPrecoReserva
ON RESERVAS.RESERVA
AFTER INSERT, UPDATE
AS
BEGIN
SET NOCOUNT ON;
UPDATE reservaAtualizada
SET PRECO_TOTAL = (
SELECT TOP 1 (
    -- Preço base ajustado pela época do ano
dadosPropriedade.PRECO_BASE * COALESCE(1 + precoEpoca.PERCENTUAL_AJUSTE / 100.0, 1)
    -- Aplica o desconto
    * (1 - COALESCE(dadosDesconto.PERCENTUAL / 100.0, 0))
    -- Multiplica pelo número de dias da reserva
    * DATEDIFF(DAY, novaReserva.DATA_INICIO, novaReserva.DATA_FIM)
    -- Adiciona a taxa turística
    + CASE
    WHEN clientePais.ID_PAIS = propriedadePais.ID_PAIS THEN 0
    ELSE COALESCE(propriedadePais.TAXA_TURISTA, 0)
    * DATEDIFF(DAY, novaReserva.DATA_INICIO, novaReserva.DATA_FIM) / 100.0
    END)
    FROM INSERTED novaReserva
    JOIN PROPRIEDADES.PROPRIEDADE dadosPropriedade ON
    novaReserva.ID_PROPRIEDADE = dadosPropriedade.ID_PROPRIEDADE
     LEFT JOIN PROPRIEDADES.PRECO_EPOCA precoEpoca
     ON dadosPropriedade.ID_PROPRIEDADE = precoEpoca.ID_PROPRIEDADE
    AND novaReserva.DATA_INICIO BETWEEN precoEpoca.DATA_INICIO
```

```
29      AND precoEpoca.DATA_FIM
30      LEFT JOIN PROPRIEDADES.DESCONTO_PROPRIEDADE descontoProp
31      ON dadosPropriedade.ID_PROPRIEDADE = descontoProp.ID_PROPRIEDADE
32      LEFT JOIN PROPRIEDADES.DESCONTOS dadosDesconto
33      ON descontoProp.ID_DESCONTO = dadosDesconto.ID_DESCONTO
34      AND novaReserva.DATA_INICIO BETWEEN dadosDesconto.VALIDADE_INICIO
35      AND dadosDesconto.VALIDADE_FIM
36  LEFT JOIN USUARIOS.CLIENTE cliente ON novaReserva.ID_CLIENTE=cliente.ID_CLIENTE
37      LEFT JOIN USUARIOS.USUARIO usuario ON cliente.ID_USER = usuario.ID_USER
38      LEFT JOIN GERAL.CIDADE cidadeCliente
39      ON usuario.ID_CIDADE = cidadeCliente.ID_CIDADE
40  LEFT JOIN GERAL.PAIS clientePais ON cidadeCliente.ID_PAIS = clientePais.ID_PAIS
41      LEFT JOIN GERAL.CIDADE cidadePropriedade ON
42      dadosPropriedade.ID_CIDADE = cidadePropriedade.ID_CIDADE
43      LEFT JOIN GERAL.PAIS propriedadePais ON
44      cidadePropriedade.ID_PAIS = propriedadePais.ID_PAIS
45      WHERE novaReserva.ID_RESERVA = reservaAtualizada.ID_RESERVA
46      GROUP BY novaReserva.ID_RESERVA, dadosPropriedade.PRECO_BASE,
47      novaReserva.DATA_INICIO, novaReserva.DATA_FIM,
48      clientePais.ID_PAIS, propriedadePais.ID_PAIS, precoEpoca.PERCENTUAL_AJUSTE,
49      dadosDesconto.PERCENTUAL, propriedadePais.TAXA_TURISTA
50      )
51      FROM RESERVAS.RESERVA reservaAtualizada
52  JOIN INSERTED novaReserva ON reservaAtualizada.ID_RESERVA=novaReserva.ID_RESERVA;
53  END;
```

Listing 4.5: Código TGR_AtualizarPrecoReserva

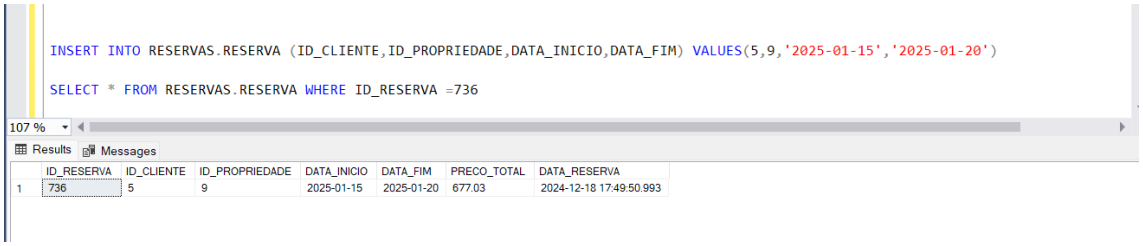


Figura 4.3: TGR_AtualizarPrecoReserva Exemplo de funcionamento

TRG_AtualizarDisponibilidadeCalendario

Antes de permitir a inserção de uma reserva, este Trigger verifica se há conflitos ou sobreposições com datas já registradas. Após a inserção duma reserva, marca as datas como "INDISPONÍVEL"no calendário, garantindo a consistência dos dados e eliminando atualizações manuais

```
--TRIGGER CALENDARIO DE DISPONIBILIDADE ---
CREATE TRIGGER TRG_AtualizarDisponibilidadeCalendario
ON RESERVAS.RESERVA
AFTER INSERT, DELETE
AS
BEGIN
SET NOCOUNT ON;
-- Quando uma nova reserva for inserida, verifica a disponibilidade
e marca as datas como 'INDISPONIVEL'
IF EXISTS (SELECT 1 FROM INSERTED)
 BEGIN
-- Verificar se a reserva se sobrepõe a uma data já marcada como 'INDISPONIVEL'
 IF EXISTS (
 SELECT 1
FROM INSERTED novaReserva
JOIN PROPRIEDADES.CALENDARIO_DISPONIBILIDADE calendario
ON novaReserva.ID_PROPRIEDADE = calendario.ID_PROPRIEDADE
 WHERE calendario.STATUS = 'INDISPONIVEL'
 AND (
-- Verifica se qualquer parte da reserva se sobrepõe ao
calendário de disponibilidade
(novaReserva.DATA_INICIO BETWEEN calendario.DATA_INICIO
  AND calendario.DATA_FIM)
  OR
(novaReserva.DATA_FIM BETWEEN calendario.DATA_INICIO
  AND calendario.DATA_FIM)
   OR
(novaReserva.DATA_INICIO <= calendario.DATA_INICIO
  AND novaReserva.DATA_FIM >= calendario.DATA_FIM)
  OR
(novaReserva.DATA_INICIO >= calendario.DATA_INICIO
  AND novaReserva.DATA_FIM <= calendario.DATA_FIM))
)
 BEGIN
-- Levanta um erro e impede a inserção se a reserva se sobrepuser
a datas já indisponíveis
RAISERROR('Não é possível fazer a reserva, a propriedade
está indisponível para as datas selecionadas.', 16, 1);
ROLLBACK TRANSACTION;  -- Impede a inserção da reserva
RETURN;
 END;
-- Inserir a nova reserva no calendário, marcando as datas como 'INDISPONIVEL'
 INSERT INTO PROPRIEDADES.CALENDARIO_DISPONIBILIDADE
 (ID_PROPRIEDADE, DATA_INICIO, DATA_FIM, STATUS)
 SELECT novaReserva.ID_PROPRIEDADE, novaReserva.DATA_INICIO,
 novaReserva.DATA_FIM, 'INDISPONIVEL'
FROM INSERTED novaReserva;
    END
```

```
49  END;
50  GO
```

Listing 4.6: Código TRG_AtualizarDisponibilidadeCalendario

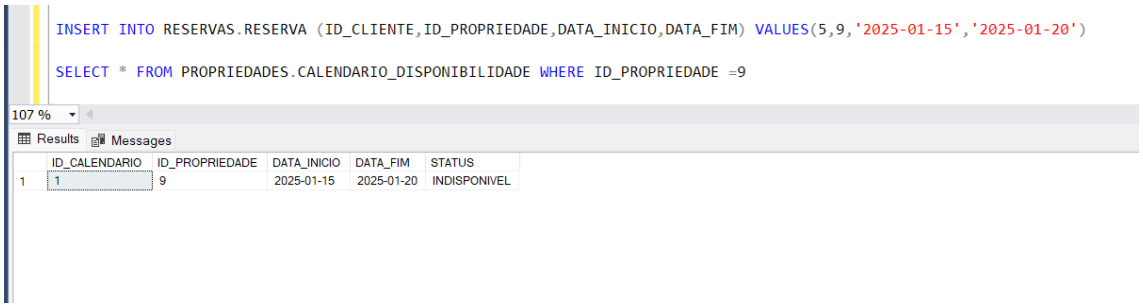


Figura 4.4: TRG_AtualizarDisponibilidadeCalendario Exemplo 1 do funcionamento

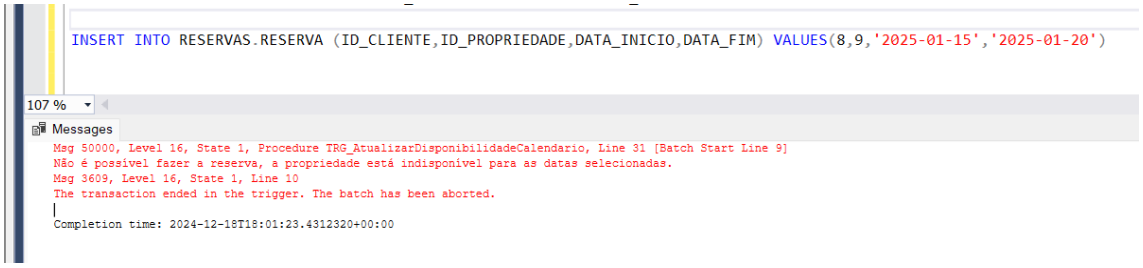


Figura 4.5: TRG_AtualizarDisponibilidadeCalendario Exemplo 2 do funcionamento

TGR_AtualizarEstadoReserva_Pendente

Após a inserção de uma nova reserva, este Trigger realiza as seguintes ações:

1. Define o estado inicial da reserva como 'PENDENTE', indicando que a mesma está aguardando confirmação
2. Regista no histórico que o estado da reserva foi alterado para 'PENDENTE'

```
1   --TRIGGER PARA ATUALIZAR O ESTADO DA RESERVA PARA PENDENTE---
2   CREATE TRIGGER TGR_AtualizarEstadoReserva_Pendente
3   ON RESERVAS.RESERVA
4   AFTER INSERT
5   AS
6   BEGIN
7       SET NOCOUNT ON;
8       -- Registrar no histórico que a reserva foi criada
9       INSERT INTO RESERVAS.HISTORICO_RESERVA (ID_RESERVA, EVENTO, DATA_EVENTO)
10      SELECT
```

```
11          reserva.ID_RESERVA,
12          'Reserva criada',
13          GETDATE()
14      FROM INSERTED reserva;
15      -- Inserir estado 'PENDENTE' para a nova reserva
16  INSERT INTO RESERVAS.ESTADO_RESERVA (ID_RESERVA, DESCRICAO_ESTADO,
17  DATA_ALTERACAO)
18      SELECT
19          reserva.ID_RESERVA,
20          'PENDENTE',
21          GETDATE()
22      FROM INSERTED reserva
23      WHERE NOT EXISTS (
24          SELECT 1
25          FROM RESERVAS.ESTADO_RESERVA estado
26          WHERE estado.ID_RESERVA = reserva.ID_RESERVA
27          AND estado.DESCRICAO_ESTADO = 'PENDENTE'
28      );
29      -- Registrar no histórico o estado inicial PENDENTE
30      INSERT INTO RESERVAS.HISTORICO_RESERVA (ID_RESERVA, EVENTO, DATA_EVENTO)
31      SELECT
32          reserva.ID_RESERVA,
33          'Estado alterado para PENDENTE',
34          GETDATE()
35      FROM INSERTED reserva;
36  END;
37  GO
```

Listing 4.7: Código TGR_AtualizarEstadoReserva_Pendente

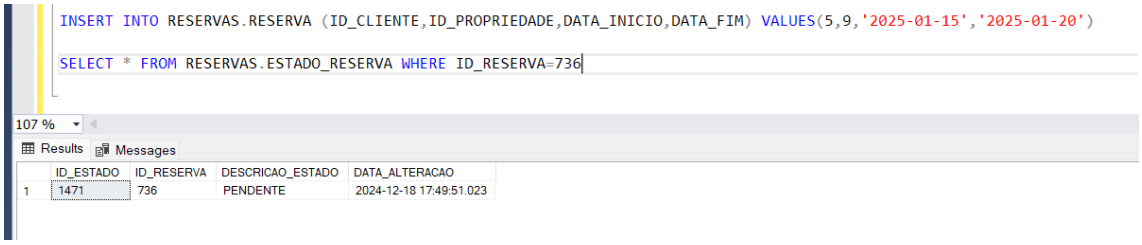


Figura 4.6: TGR_AtualizarEstadoReserva_Pendente Exemplo 1 de funcionamento

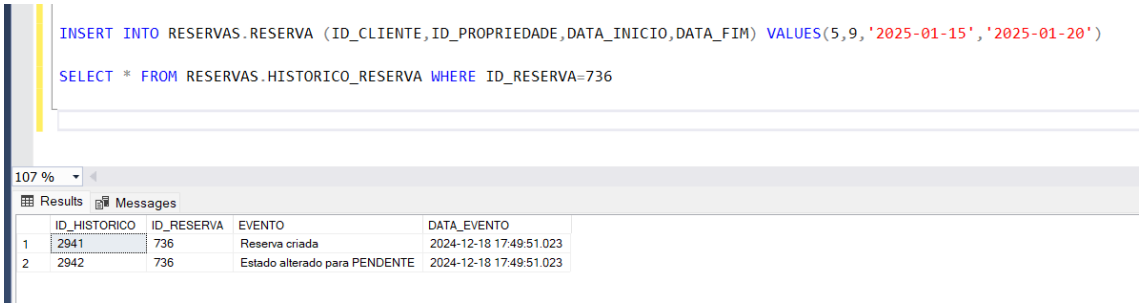


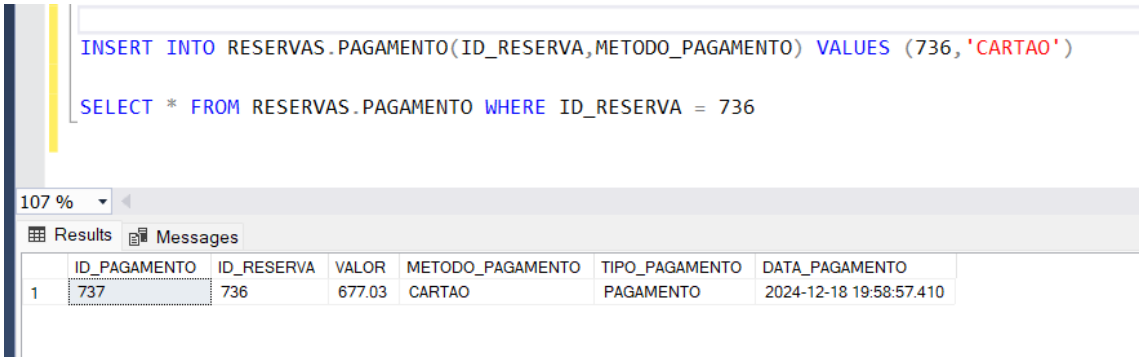
Figura 4.7: TGR_AtualizarEstadoReserva_Pendente Exemplo 2 de funcionamento

TGR_AutomaticValorPagamento

Ativado após a inserção de um pagamento, atualiza o valor com base no tipo de operação. Para reembolsos, calcula 50 porcento do preço total da reserva; para pagamentos normais, aplica o preço integral. Também registra o valor no histórico da reserva.

```
1  --TRIGGER PARA ATUALIZAR O VALOR DO PAGAMENTO/REEMBOLSO-
2  CREATE TRIGGER TGR_AutomaticValorPagamento
3  ON RESERVAS.PAGAMENTO
4  AFTER INSERT
5  AS
6  BEGIN
7  SET NOCOUNT ON;
8  -- Atualiza o valor do pagamento com base no tipo
9  UPDATE PAGAMENTO
10 SET VALOR = (
11     CASE
12 WHEN pagamento.TIPO_PAGAMENTO = 'REEMBOLSO' THEN
13 reserva.PRECO_TOTAL / 2 -- Reembolso (50% do preço total)
14  ELSE reserva.PRECO_TOTAL -- Pagamento normal
15   END )
16 FROM RESERVAS.PAGAMENTO pagamento
17 JOIN INSERTED i ON pagamento.ID_PAGAMENTO = i.ID_PAGAMENTO
18 JOIN RESERVAS.RESERVA reserva ON i.ID_RESERVA = reserva.ID_RESERVA;
19  -- Registrar no histórico o pagamento realizado
20 INSERT INTO RESERVAS.HISTORICO_RESERVA (ID_RESERVA, EVENTO, DATA_EVENTO)
21  SELECT
22  novoPagamento.ID_RESERVA,
23  CONCAT('PAGAMENTO REALIZADO: ',
24 CAST(novoPagamento.VALOR AS VARCHAR(20)),
25    ' VIA ', novoPagamento.METODO_PAGAMENTO),
26   GETDATE()
27  FROM INSERTED novoPagamento;
28 END;
29 GO
```

Listing 4.8: Código TGR_AutomaticValorPagamento



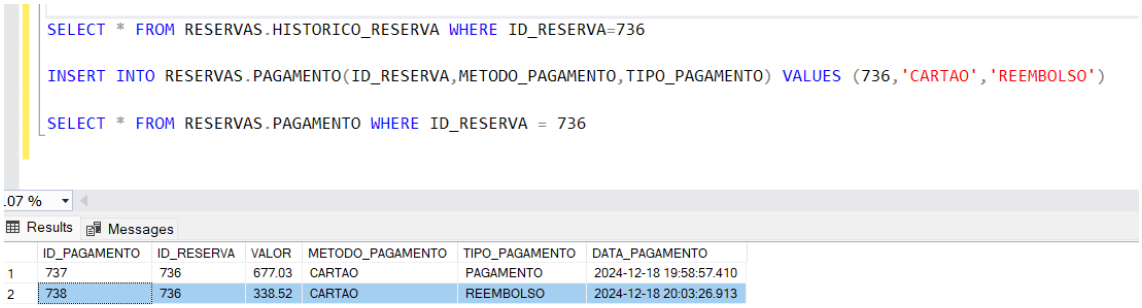
```
INSERT INTO RESERVAS.PAGAMENTO(ID_RESERVA,METODO_PAGAMENTO) VALUES (736,'CARTAO')

SELECT * FROM RESERVAS.PAGAMENTO WHERE ID_RESERVA = 736
```

107 %

Results | Messages

	ID_PAGAMENTO	ID_RESERVA	VALOR	METODO_PAGAMENTO	TIPO_PAGAMENTO	DATA_PAGAMENTO
1	737	736	677.03	CARTAO	PAGAMENTO	2024-12-18 19:58:57.410

Figura 4.8: TGR_AutomaticValorPagamento Exemplo 1 de funcionamento



```
SELECT * FROM RESERVAS.HISTORICO_RESERVA WHERE ID_RESERVA=736

INSERT INTO RESERVAS.PAGAMENTO(ID_RESERVA,METODO_PAGAMENTO,TIPO_PAGAMENTO) VALUES (736,'CARTAO','REEMBOLSO')

SELECT * FROM RESERVAS.PAGAMENTO WHERE ID_RESERVA = 736
```

.07 %

Results | Messages

	ID_PAGAMENTO	ID_RESERVA	VALOR	METODO_PAGAMENTO	TIPO_PAGAMENTO	DATA_PAGAMENTO
1	737	736	677.03	CARTAO	PAGAMENTO	2024-12-18 19:58:57.410
2	738	736	338.52	CARTAO	REEMBOLSO	2024-12-18 20:03:26.913

Figura 4.9: TGR_AutomaticValorPagamento Exemplo 2 de funcionamento

TGR_AtualizarEstadoReserva_Confirmada

Logo de um pagamento ser inserido corretamente , o Trigger adiciona o estado "CONFIRMADA"à tabela de estados, indicando que a reserva foi confirmada.

```
1  CREATE TRIGGER TGR_AtualizarEstadoReserva_Confirmada
2  ON RESERVAS.PAGAMENTO
3  AFTER INSERT
4  AS
5  BEGIN
6  SET NOCOUNT ON;
7  -- Atualizar estado para 'CONFIRMADA' na tabela de estados
8  INSERT INTO RESERVAS.ESTADO_RESERVA (ID_RESERVA,
9  DESCRICAO_ESTADO, DATA_ALTERACAO)
10  SELECT
11  pagamento.ID_RESERVA,
```

```
12  'CONFIRMADA',
13   GETDATE()
14   FROM INSERTED pagamento
15  WHERE NOT EXISTS (
16  SELECT 1
17  FROM RESERVAS.ESTADO_RESERVA estado
18  WHERE estado.ID_RESERVA = pagamento.ID_RESERVA
19  AND estado.DESCRICAO_ESTADO = 'CONFIRMADA'
20  );
21  -- Registrar no histórico o estado CONFIRMADA
22  INSERT INTO RESERVAS.HISTORICO_RESERVA (ID_RESERVA, EVENTO, DATA_EVENTO)
23  SELECT
24   pagamento.ID_RESERVA,
25  'Estado alterado para CONFIRMADA',
26   GETDATE()
27   FROM INSERTED pagamento;
28  END;
29  GO
```

Listing 4.9: Código TGR_AtualizarEstadoReserva_Confirmada

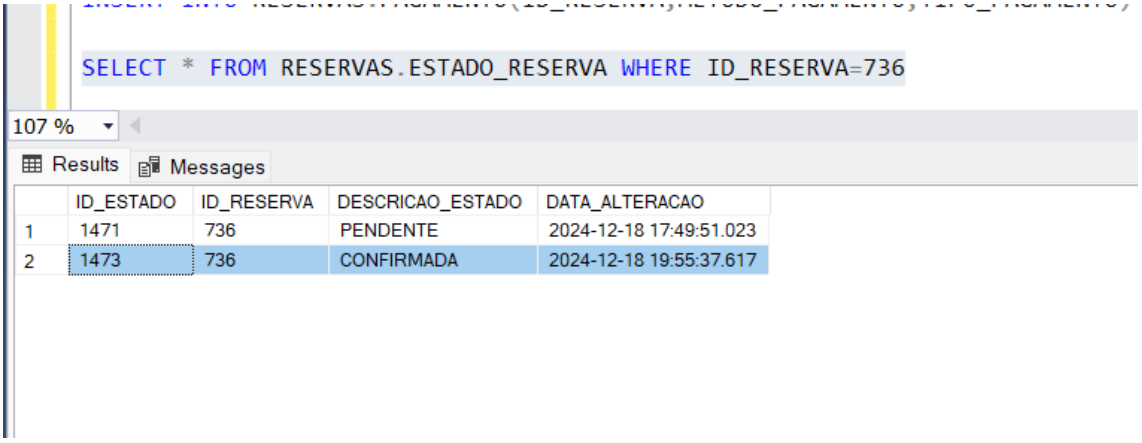
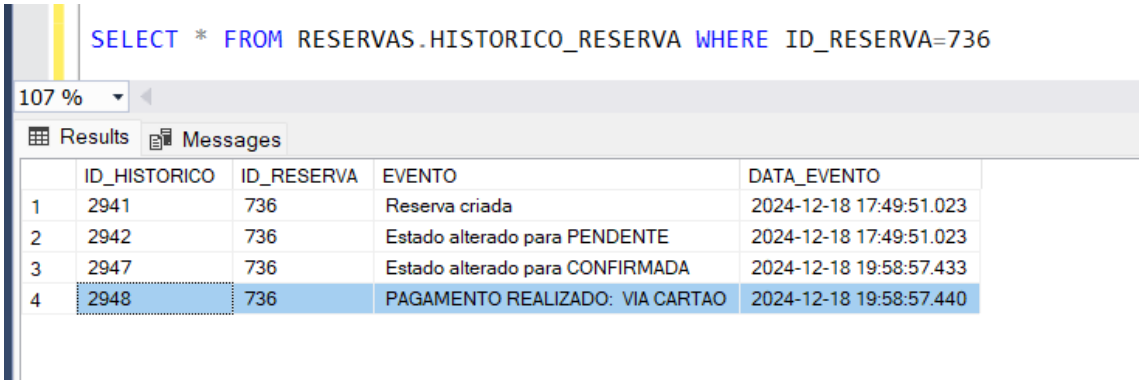


Figura 4.10: TGR_AtualizarEstadoReserva_Confirmada Exemplo 1 de funcionamento



	ID_HISTORICO	ID_RESERVA	EVENTO	DATA_EVENTO
1	2941	736	Reserva criada	2024-12-18 17:49:51.023
2	2942	736	Estado alterado para PENDENTE	2024-12-18 17:49:51.023
3	2947	736	Estado alterado para CONFIRMADA	2024-12-18 19:58:57.433
4	2948	736	PAGAMENTO REALIZADO: VIA CARTAO	2024-12-18 19:58:57.440

Figura 4.11: TGR_AtualizarEstadoReserva_Confirmada Exemplo 2 de funcionamento

TRG_VerificarUnicoPagamentoReembolso

Este Trigger impede duplicações de pagamentos ou reembolsos para a mesma reserva. Durante a tentativa de inserção:

1. Verifica se já há um pagamento registrado e, em caso positivo, gera um erro e cancela a transação.

2. Confirma a inexistência de reembolsos anteriores, revertendo a operação caso um já tenha sido processado.

```
--TRIGGER PARA VERIFICAR UM PAGAMENTO/REEMBOLSO UNICO-
CREATE TRIGGER TRG_VerificarUnicoPagamentoReembolso
ON RESERVAS.PAGAMENTO
INSTEAD OF INSERT
AS
BEGIN
SET NOCOUNT ON;
-- Verifica duplicação de pagamentos no conjunto INSERTED
IF EXISTS (
SELECT 1
FROM RESERVAS.PAGAMENTO P
JOIN INSERTED I
ON P.ID_RESERVA = I.ID_RESERVA
WHERE I.TIPO_PAGAMENTO = 'PAGAMENTO' AND P.TIPO_PAGAMENTO = 'PAGAMENTO'
    )
BEGIN
RAISERROR ('Já existe um pagamento registrado para esta reserva.', 16, 1)
ROLLBACK TRANSACTION;
RETURN;
END
 -- Verifica duplicação de reembolsos no conjunto INSERTED
IF EXISTS (
  SELECT 1
```

```
24     FROM RESERVAS.PAGAMENTO P
25    JOIN INSERTED I
26   ON P.ID_RESERVA = I.ID_RESERVA
27   WHERE I.TIPO_PAGAMENTO = 'REEMBOLSO' AND P.TIPO_PAGAMENTO = 'REEMBOLSO'
28   )
29   BEGIN
30   RAISERROR ('Já existe um reembolso registrado para esta reserva.',
31   16, 1);ROLLBACK TRANSACTION;
32    RETURN;
33   END
34    -- Realiza a inserção para registros válidos
35   INSERT INTO RESERVAS.PAGAMENTO (ID_RESERVA, VALOR, METODO_PAGAMENTO,
36   TIPO_PAGAMENTO, DATA_PAGAMENTO)
37   SELECT ID_RESERVA, VALOR, METODO_PAGAMENTO,TIPO_PAGAMENTO, DATA_PAGAMENTO
38   FROM INSERTED;
39   END;
40   GO
```

Listing 4.10: Código TRG_VerificarUnicoPagamentoReembolso

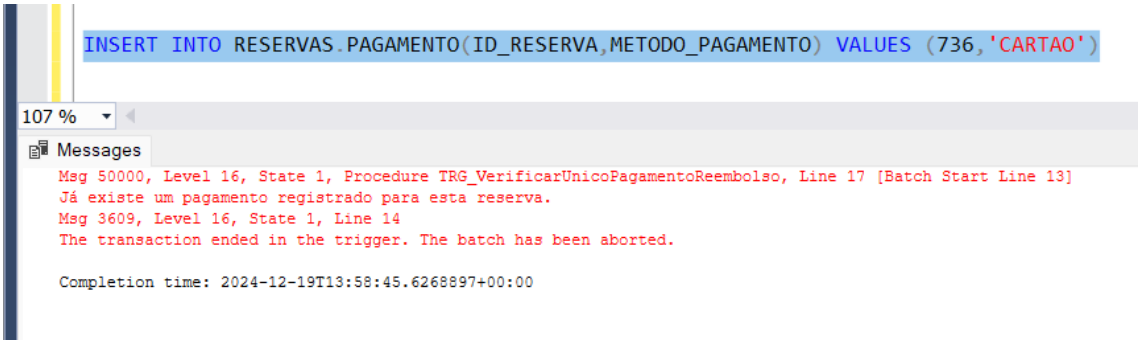


Figura 4.12: TRG_VerificarUnicoPagamentoReembolso Exemplo 1 de funcionamento

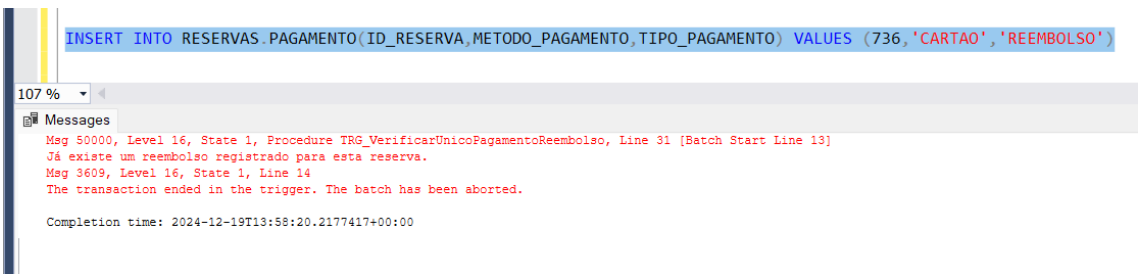


Figura 4.13: TRG_VerificarUnicoPagamentoReembolso Exemplo 2 de funcionamento

TGR_UpdateReservaEstadoCancelada

É ativado automaticamente após a inserção de um cancelamento,adiciona o estado "CANCELADA"à tabela de estados, registra a mudança no histórico e calcula o reembolso como 50 porcento do preço total.

```
1  --TRIGGER PARA ATUALIZAR O ESTADO DA RESERVA PARA CANCELADA-
2  CREATE TRIGGER TGR_UpdateReservaEstadoCancelada
3  ON RESERVAS.CANCELAMENTO
4  AFTER INSERT
5  AS
6  BEGIN
7  SET NOCOUNT ON;
8  -- Inserir estado "CANCELADA" na tabela de estados
9  INSERT INTO RESERVAS.ESTADO_RESERVA (ID_RESERVA, DESCRICAO_ESTADO, DATA_ALTERACAO)
10 SELECT
11   cancelamento.ID_RESERVA,
12    'CANCELADA',
13    GETDATE()
14 FROM INSERTED cancelamento;
15   -- Registrar no histórico o estado CANCELADA
16 INSERT INTO RESERVAS.HISTORICO_RESERVA (ID_RESERVA, EVENTO, DATA_EVENTO)
17 SELECT
18  cancelamento.ID_RESERVA,
19   'Estado alterado para CANCELADA',
20   GETDATE()
21 FROM INSERTED cancelamento;
22 -- Atualizar o valor do reembolso na tabela CANCELAMENTO
23  UPDATE C
24 SET C.VALOR_REEMBOLSO = R.PRECO_TOTAL / 2 -- 50% do preço total da reserva
25   FROM RESERVAS.CANCELAMENTO C
26   INNER JOIN INSERTED i ON C.ID_CANCELAMENTO = i.ID_CANCELAMENTO
27   INNER JOIN RESERVAS.RESERVA R ON i.ID_RESERVA = R.ID_RESERVA;
28 END;
29 GO
30
```

Listing 4.11: Código TGR_UpdateReservaEstadoCancelada

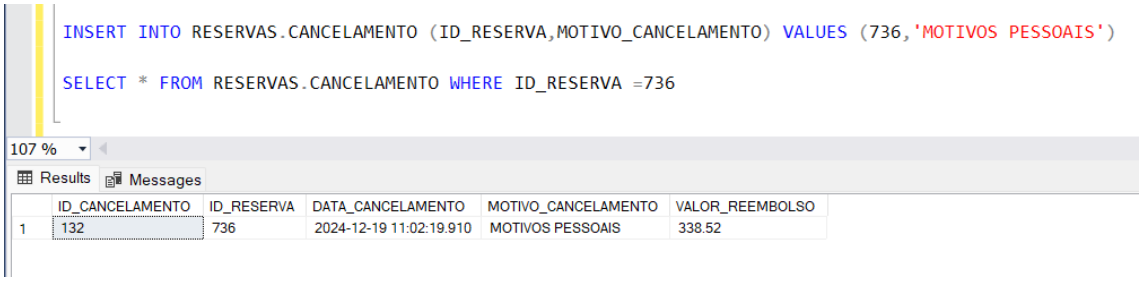


Figura 4.14: TGR_UpdateReservaEstadoCancelada Exemplo 1 de funcionamento

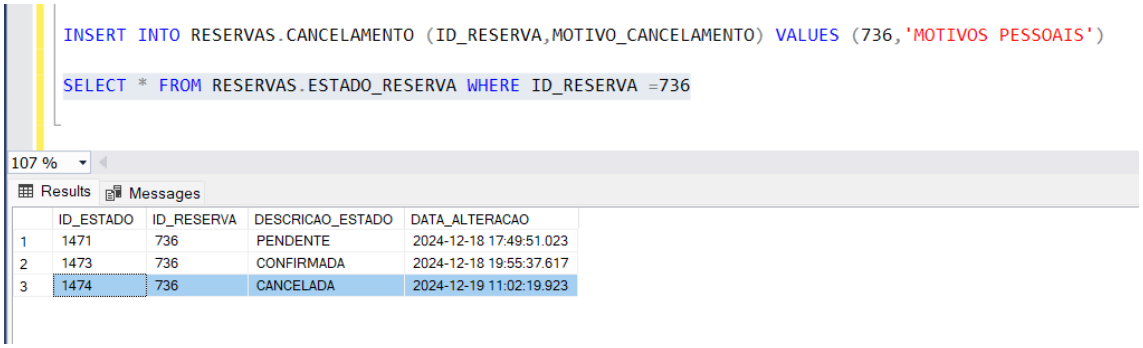


Figura 4.15: TGR_UpdateReservaEstadoCancelada Exemplo 2 de funcionamento

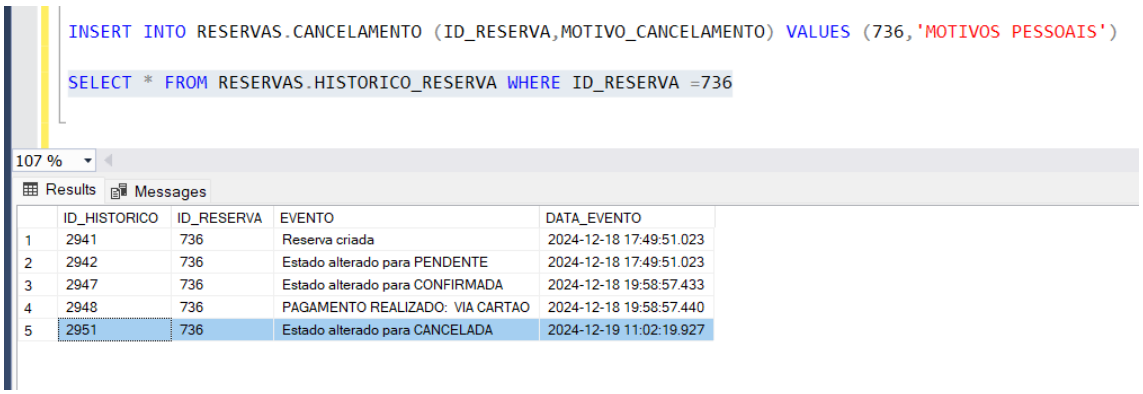


Figura 4.16: TGR_UpdateReservaEstadoCancelada Exemplo 3 de funcionamento

TRG_RemoverDatasCancelamento

Este Trigger é o encargado de remover as datas correspondentes do calendário de disponibilidade após ser inserido um cancelameto , liberando-as para novas reservas.

```
1  --TRIGGER REMOVER DATAS APÓS UM CANCELAMENTO ---
2  CREATE TRIGGER TRG_RemoverDatasCancelamento
3  ON RESERVAS.CANCELAMENTO
4  AFTER INSERT
5  AS
```

```
6  BEGIN
7  SET NOCOUNT ON;
8   -- Remove as datas do calendário associadas à reserva cancelada
9   DELETE FROM PROPRIEDADES.CALENDARIO_DISPONIBILIDADE
10 FROM PROPRIEDADES.CALENDARIO_DISPONIBILIDADE calendario
11  JOIN RESERVAS.RESERVA reserva
12  ON calendario.ID_PROPRIEDADE = reserva.ID_PROPRIEDADE
13  JOIN INSERTED cancelamento
14  ON cancelamento.ID_RESERVA = reserva.ID_RESERVA
15  WHERE calendario.DATA_INICIO = reserva.DATA_INICIO
16  AND calendario.DATA_FIM = reserva.DATA_FIM;
17 END;
18 GO
```

Listing 4.12: Código TRG_RemoverDatasCancelamento

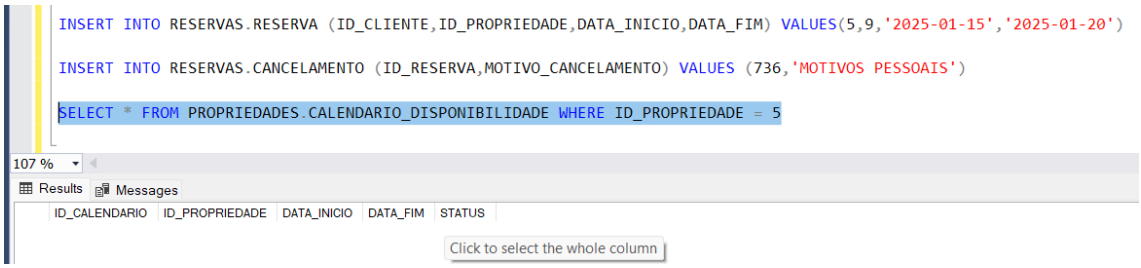


Figura 4.17: TRG_RemoverDatasCancelamento Exemplo de funcionamento

TGR_AtualizarPropriedadesDestaque

Responsável por atualizar a tabela PROPRIEDADES_DESTAQUE, este Trigger recalcula e seleciona as 10 propriedades com as melhores avaliações após inserções ou exclusões

```
1  --TRIGGER PARA ATUALIZAR AS PROPRIEDADES DESTAQUES-
2  CREATE TRIGGER TGR_AtualizarPropriedadesDestaque
3  ON RESERVAS.AVALIACAO
4  AFTER INSERT, DELETE
5  AS
6  BEGIN
7  SET NOCOUNT ON;
8
9  -- Limpar a tabela PROPRIEDADES_DESTAQUE
10 DELETE FROM PROPRIEDADES.PROPRIEDADES_DESTAQUE;
11
12 -- Recalcular as 10 propriedades com as melhores avaliações
13 WITH MediaAvaliacoes AS (
14  SELECT
```

```
15      ID_PROPRIEDADE,
16      AVG(NOTA) AS MediaNota
17       FROM AVALIACAO
18      GROUP BY ID_PROPRIEDADE
19  ),
20   Top10Propriedades AS (
21       SELECT
22       ID_PROPRIEDADE
23       FROM MediaAvaliacoes
24      ORDER BY MediaNota DESC
25  OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
26  )
27  INSERT INTO PROPRIEDADES.PROPRIEDADES_DESTAQUE (ID_PROPRIEDADE)
28  SELECT ID_PROPRIEDADE
29   FROM Top10Propriedades;
30  END;
31  GO
```

Listing 4.13: Código TGR_AtualizarPropriedadesDestaque

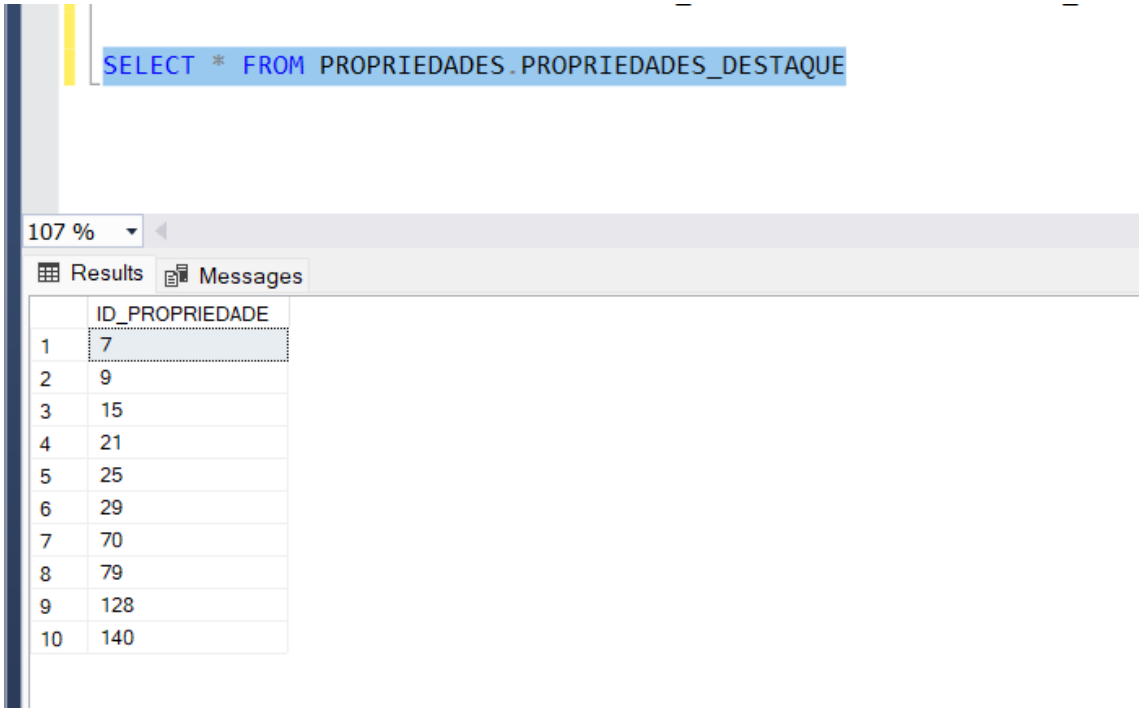


Figura 4.18: TGR_AtualizarPropriedadesDestaque Exemplo de funcionamento

TRG_InserirPrecoEpoca

É ativado após a inserção de uma nova propriedade, para cada nova propriedade inserida, o Trigger cria três registos na tabela PRECO_EPOCA, um para cada época do ano (Alta Temporada, Meia Temporada e Baixa Temporada).

```
1  --TRIGGER PARA INSERIR PRECO DE EPOCA DO ANO EM PROPRIEDADES-
2  CREATE TRIGGER TRG_InserirPrecoEpoca
3  ON PROPRIEDADES.PROPRIEDADE
4  AFTER INSERT
5  AS
6  BEGIN
7  SET NOCOUNT ON;
8  -- Insere valores específicos na tabela PRECO_EPOCA para cada nova propriedade
9  INSERT INTO PRECO_EPOCA (ID_PROPRIEDADE, NOME_EPOCA, DATA_INICIO,
10 DATA_FIM, PERCENTUAL_AJUSTE)
11 SELECT
12 I.ID_PROPRIEDADE,
13  CASE X.NUM
14  WHEN 1 THEN 'Alta Temporada'
15  WHEN 2 THEN 'Meia Temporada'
16   WHEN 3 THEN 'Baixa Temporada'
17  END AS NOME_EPOCA,
18     CASE X.NUM
19   WHEN 1 THEN '2025-06-01'
20   WHEN 2 THEN '2025-09-01'
21   WHEN 3 THEN '2025-01-01'
22 END AS DATA_INICIO,
23  CASE X.NUM
24  WHEN 1 THEN '2025-08-31'
25 WHEN 2 THEN '2025-12-31'
26 WHEN 3 THEN '2025-03-31'
27 END AS DATA_FIM,
28  CASE X.NUM
29  WHEN 1 THEN 0.50
30 WHEN 2 THEN 0.70
31 WHEN 3 THEN 0.90
32  END AS PERCENTUAL_AJUSTE
33 FROM INSERTED I
34 CROSS JOIN (VALUES (1), (2), (3)) X(NUM);
35 END;
36 GO
```

Listing 4.14: Código TRG_InserirPrecoEpoca

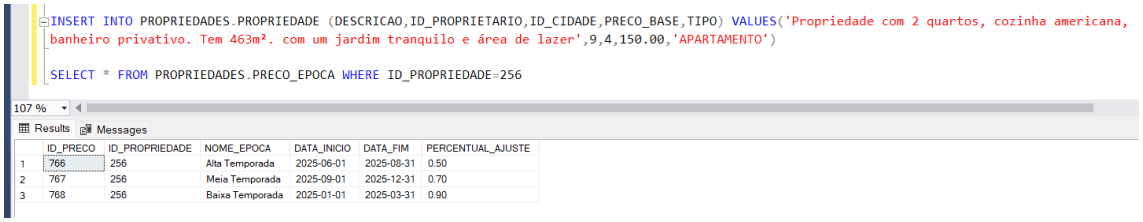


Figura 4.19: TRG_InserirPrecoEpoca Exemplo de funcionamento

Capítulo 5

Segurança

A segurança da base de dados é essencial para proteger as informações armazenadas e garantir que apenas os usuários autorizados possam acessá-las ou modificá-las. No caso desta base de dados, foram adotadas diversas medidas para garantir um controle rigoroso de acessos e permissões, além de uma organização lógica que melhora a segurança. Seguem as principais implementações realizadas

5.1 Criação de Logins e Usuários

O processo iniciou com a criação do login externo login_administrador, protegido por uma senha segura (admin123). Este login foi associado ao usuário interno usuario_administrador, dedicado exclusivamente à base de dados, garantindo acesso seguro ao servidor e ao banco de dados.

```
1  --SEGURANÇA
2          --LOGIN
3          CREATE LOGIN login_administrador
4          WITH PASSWORD = 'admin123';
5          GO
6          --USER
7          USE DREAMBOOKINGS;
8          GO
9          CREATE USER usuario_administrador
10         FOR LOGIN login_administrador;
11         GO
12         --LIGACAO ENTRE USER E ROLES
13         ALTER ROLE ADMIN_dreambookings ADD MEMBER usuario_administrador;
14         GO
```

Listing 5.1: Código Segurança

Esse usuário possui controle total sobre a base, permitindo a administração completa de seus objetos e dados

Implementação do código exposto

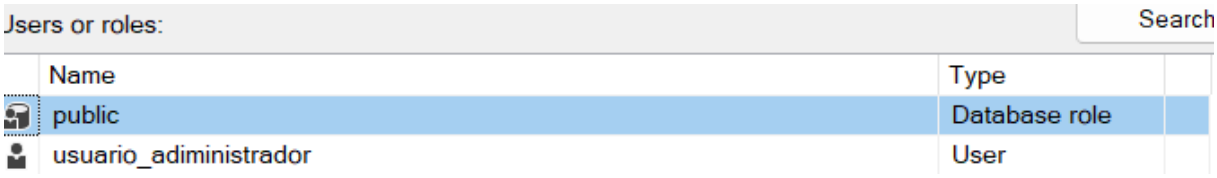


Figura 5.1: Ligação entre user criado e role

5.2 Organização de Schemas

Para otimizar a organização e o controle de acessos, as tabelas foram divididas em quatro schemas:

1. GERAL: Contém tabelas gerais como PAIS e CIDADE.
2. USUARIOS: Contém informações relacionadas a usuários, como USUARIO, CLIENTE e PROPRIETARIO.
3. PROPRIEDADES: Armazena dados sobre propriedades e suas características, como PROPRIEDADE, CALENDARIO_DISPONIBILIDADE e PRECO_EPOCA.
4. RESERVAS: Agrupa tabelas relacionadas a reservas, como RESERVA, AVALIACAO e PAGAMENTO.

Implementação dos Schemas na base de dados

A seguir mostra-se a criação dos schemas e transferência das tabelas para os mesmos.

```
1  --SCHEMAS
2  CREATE SCHEMA GERAL;
3  GO
4  CREATE SCHEMA USUARIOS;
5  GO
6  CREATE SCHEMA PROPRIEDADES;
7  GO
8  CREATE SCHEMA RESERVAS;
9  GO
10 --GERAL
11 ALTER SCHEMA GERAL TRANSFER dbo.PAIS;
12 ALTER SCHEMA GERAL TRANSFER dbo.CIDADE;
13 GO
14 --USUARIOS
```

```
15  ALTER SCHEMA USUARIOS TRANSFER dbo.USUARIO;
16  ALTER SCHEMA USUARIOS TRANSFER dbo.CLIENTE;
17  ALTER SCHEMA USUARIOS TRANSFER dbo.PROPRIETARIO;
18  GO
19  --PROPRIEDADES
20  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.PROPRIEDADE;
21  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.IMAGEM_VIDEO;
22  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.CARACTERISTICAS_EXTRAS;
23  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.CARACTERISTICAS_PROPRIEDADE;
24  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.CALENDARIO_DISPONIBILIDADE;
25  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.PRECO_EPOCA;
26  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.PROPRIEDADES_DESTAQUE;
27  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.DESCONTOS;
28  ALTER SCHEMA PROPRIEDADES TRANSFER dbo.DESCONTO_PROPRIEDADE;
29  GO
30  --RESERVAS
31  ALTER SCHEMA RESERVAS TRANSFER dbo.RESERVA;
32  ALTER SCHEMA RESERVAS TRANSFER dbo.ESTADO_RESERVA;
33  ALTER SCHEMA RESERVAS TRANSFER dbo.HISTORICO_RESERVA;
34  ALTER SCHEMA RESERVAS TRANSFER dbo.PAGAMENTO;
35  ALTER SCHEMA RESERVAS TRANSFER dbo.AVALIACAO;
36  ALTER SCHEMA RESERVAS TRANSFER dbo.CANCELAMENTO;
37  GO
38
```

Listing 5.2: Código Schema

Implementação do código exposto

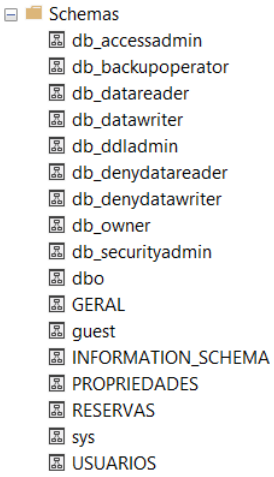


Figura 5.2: Schemas

5.3 Criação de Roles e Atribuição de Permissões

Com o objetivo de garantir que cada usuário tenha acesso apenas aos dados necessários para realizar suas funções, foram criadas roles (funções) para diferentes perfis de usuários. As roles criadas foram:

1. ADMIN_dreambookings: Permissão total nos schemas principais, permitindo o controle completo da base de dados.
2. PROPRIETARIO_propriedades: Acesso total (SELECT, INSERT, UPDATE, DELETE) às tabelas relacionadas a propriedades e reservas.
3. CLIENTE_dreambookings: Permissão para visualizar propriedades e criar reservas ou avaliações.
4. LEITOR: Permissão apenas para leitura em toda a base.

Criação e implementação dos Roles

Segue os comandos SQL implementados para implementar as roles descritas.

```sql
-- Criar os roles
CREATE ROLE ADMIN_dreambookings;
GO
CREATE ROLE PROPRIETARIO_propriedades;
GO
CREATE ROLE CLIENTE_dreambookings;
GO
CREATE ROLE LEITOR;
GO
-- Atribuindo permissões aos roles
GRANT CONTROL ON SCHEMA::GERAL TO ADMIN_dreambookings;
GRANT CONTROL ON SCHEMA::USUARIOS TO ADMIN_dreambookings;
GRANT CONTROL ON SCHEMA::PROPRIEDADES TO ADMIN_dreambookings;
GRANT CONTROL ON SCHEMA::RESERVAS TO ADMIN_dreambookings;
GO

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::PROPRIEDADES TO
PROPRIETARIO_propriedades;
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::RESERVAS TO
PROPRIETARIO_propriedades;
GO

GRANT SELECT ON SCHEMA::GERAL TO CLIENTE_dreambookings;
GRANT SELECT ON SCHEMA::PROPRIEDADES TO CLIENTE_dreambookings;
GRANT SELECT, INSERT ON RESERVAS.RESERVA TO CLIENTE_dreambookings;
GRANT SELECT, INSERT ON RESERVAS.AVALIACAO TO CLIENTE_dreambookings;
```

```
27  GO
28
29  GRANT SELECT ON SCHEMA::GERAL TO LEITOR;
30  GRANT SELECT ON SCHEMA::USUARIOS TO LEITOR;
31  GRANT SELECT ON SCHEMA::PROPRIEDADES TO LEITOR;
32  GRANT SELECT ON SCHEMA::RESERVAS TO LEITOR;
33  GO
```

Listing 5.3: Código Roles

Implementação do código exposto

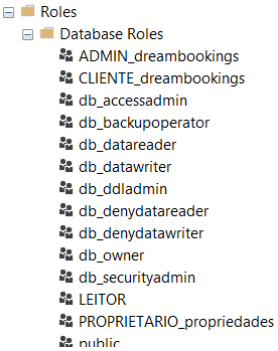


Figura 5.3: Roles

Capítulo 6

Cópias de Segurança (Backups)

Garantir a segurança e a recuperação dos dados é essencial para a continuidade das operações do banco de dados. Nesse sentido, foi implementado um plano abrangente de backups utilizando o recurso de Maintenance Plans do SQL Server e também foram feitos scripts de backups separadamente, caso haja alguma necessidade de execução.

6.1 Estratégias de Backup

Para assegurar a recuperação completa dos dados em diferentes cenários de falhas, optou-se pelas seguintes estratégias de backup:

1. Backup Completo: Este tipo de backup é executado semanalmente e constitui a base para os backups diferenciais e de logs de transação.

2. Backup Diferencial: Esse backup é executado diariamente para reduzir o tempo necessário para recuperação em caso de falha.

3. Backup dos Logs de Transação: É executado várias vezes ao longo do dia, garantindo proteção contra perda de dados recentes.

Comandos para a execução manual dos backups

```
1  --COMPLETO
2  BACKUP DATABASE DREAMBOOKINGS
3  TO DISK = 'C:\Backups\DREAMBOOKINGS_Full.bak'
4  WITH FORMAT,
5       INIT,
6       NAME = 'DREAMBOOKINGS Full Backup',
7       STATS = 10,
8       COMPRESSION;
```

Listing 6.1: Código BACKUP_COMPLETO

```
1  --DIFERENCIAL
2  BACKUP DATABASE DREAMBOOKINGS
3  TO DISK = 'C:\Backups\DREAMBOOKINGS_Diff.bak'
4  WITH DIFFERENTIAL,
5       INIT,
6       NAME = 'DREAMBOOKINGS Differential Backup',
7       STATS = 10,
8       COMPRESSION;
```

Listing 6.2: Código BACKUP_DIFERENCIAL

```
1  --LOG DE TRANSACOES
2  BACKUP LOG DREAMBOOKINGS
3  TO DISK = 'C:\Backups\DREAMBOOKINGS_Log.trn'
4  WITH INIT,
5       NAME = 'DREAMBOOKINGS Log Backup',
6       STATS = 10,
7       COMPRESSION;
```

Listing 6.3: Código BACKUP_LOG

Implementação dos comandos expostos

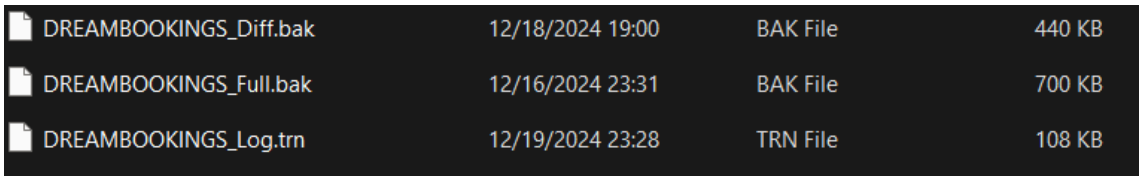


Figura 6.1: Backups

Capítulo 7

Desempenho

O desempenho de bases de dados no SQL Server depende de uma combinação de boas práticas, técnicas otimizadas e configurações adequadas que visam garantir a execução eficiente de consultas, manipulação de dados e tarefas administrativas. A seguir, são apresentados os principais fatores e práticas que impactam o desempenho em um ambiente SQL Server

7.1 Uso de FILEGROUPS

Os FILEGROUPS permitem a segmentação dos dados em diferentes discos físicos, melhorando o desempenho ao possibilitar a paralelização de operações de leitura e gravação.

É recomendável separar logs, índices e dados em FILEGROUPS distintos, para garantir uma melhor distribuição dos dados e reduzir a contenção de recursos.

Esta técnica é especialmente útil em sistemas que requerem alto desempenho em operações simultâneas e grandes volumes de dados.

7.2 Tabelas Particionadas

O particionamento de tabelas consiste em dividir grandes tabelas em partições menores, facilitando o acesso e manipulação de subconjuntos de dados.

A segmentação de dados permite a melhoria do desempenho ao reduzir o volume de dados processados durante consultas específicas.

7.3 Utilização de Índices

Os índices são fundamentais para o desempenho de consultas, pois facilitam a busca e recuperação eficiente de dados. Entretanto, é importante equilibrar o uso de índices para evitar penalizações em operações de inserção e atualização:

1. Índices Excessivos: A criação de muitos índices pode aumentar o custo de manutenção do banco de dados.

2. Manutenção de Índices: Recomenda-se monitorar o uso dos índices, além de realizar operações periódicas de reorganização e reconstrução para evitar a fragmentação.

7.4 Uso de Stored Procedures

As Stored Procedures são blocos de código SQL armazenados e compilados no servidor, oferecendo diversos benefícios em termos de desempenho:

1. Reutilização de Planos de Execução: A execução de consultas armazenadas evita recompilação frequente, economizando tempo de processamento.

2. Redução de Tráfego de Rede: Como o código SQL é executado diretamente no servidor, há uma diminuição no tráfego de dados entre cliente e servidor.

7.5 Manutenção Periódica

A manutenção regular do banco de dados é essencial para garantir um desempenho consistente e prevenir degradação ao longo do tempo:

1. Backups Regulares: Implementar estratégias de backups completos, diferenciais e de logs de transação para proteger os dados e garantir recuperação rápida em caso de falha.

2. Gerenciamento de Tabelas de Log: Remover registros desnecessários ou antigos em tabelas de logs para evitar crescimento excessivo e impactos negativos no desempenho.

7.6 Ferramentas de Monitoramento no SQL Server

O SQL Server disponibiliza diversas ferramentas que auxiliam na análise e monitoramento do desempenho do banco de dados, permitindo a identificação e resolução de gargalos:

1. SQL Server Profiler: Ferramenta utilizada para capturar e analisar consultas e eventos lentos. É útil para identificar gargalos de desempenho e otimizar a execução de consultas.

2. Activity Monitor: Permite monitorar sessões ativas, bloqueios, uso de CPU, memória e outras métricas críticas em tempo real.

3. DMVs (Dynamic Management Views): Fornecem informações detalhadas sobre o desempenho de consultas, uso de índices, bloqueios e outras estatísticas relevantes.

Capítulo 8

Manutenção e automatização do servidor

Para garantir o funcionamento eficiente, seguro e confiável do banco de dados ,foi criado um plano de manutenção e automação abrangente utilizando os Maintenance Plans do SQL Server. Este plano consolida diversas tarefas essenciais de administração do servidor, como integridade, otimização e backups, em uma abordagem automatizada e sistemática. Todos os planos foram configurados para ocorrer em horários de menos fluxo de utilização na base de dados, entre meia noite e cinco da manhã.

8.1 Tarefas configuradas

Verificação de Integridade do Banco de Dados (Check Database Integrity)

Ocorre uma vez por semana, realiza verificações automáticas da consistência lógica e física dos bancos de dados para identificar problemas que possam comprometer a integridade dos dados. Ajuda a detetar e corrigir erros antes que causem falhas críticas no sistema.

Otimização de Índices

Foi agendado para ocorrer uma vez por semana:

1. Reorganize Index: Reorganiza índices fragmentados com fragmentação entre 5 e 30 porcento, garantindo eficiência em consultas.
2. Rebuild Index: Reconstrói índices com fragmentação acima de 30 porcento, corrigindo a desordem para melhorar a performance de leitura e escrita.

Limpeza de Histórico e Arquivos Obsoletos

1. Clean Up History: Remove todos os meses automaticamente entradas desnecessárias de histórico do SQL Server Agent mais antigas que 4 meses, reduzindo o armazena-

mento utilizado.

2. Maintenance Cleanup Task: Exclui backups antigos e arquivos desnecessários toda semama, liberando espaço em disco.

Update Statistics

É executado toda semana para garantir que as estatísticas do banco de dados estejam atualizadas,

Backups Automatizados

1. Backup Database (Full)
2. Backup Database (Differential)
3. Backup Database (Transaction Log)

Capítulo 9

Indices

9.1 Introdução à Implementação de Índices:

A implementação de índices em um banco de dados é uma prática fundamental para otimizar a performance de consultas, especialmente em bases de dados grandes, onde a eficiência na recuperação de dados se torna crucial.

No contexto da base de dados em questão, foram criados diversos índices nas tabelas RESERVAS, PROPRIEDADES, CALENDARIO_DISPONIBILIDADE, AVALIACAO, PAGAMENTO, ESTADO_RESERVA e HISTORICO_RESERVA, com o objetivo de melhorar o desempenho de consultas frequentes, especialmente aquelas que envolvem filtros, junções e ordenações.A seguir, destacam-se as vantagens de cada tipo de índice implementado.

9.1.1 Tipos de Índices Implementados

1. Tabela RESERVA: Índices sobre as colunas ID_PROPRIEDADE, DATA_INICIO e DATA_FIM permitem uma busca rápida por propriedades, datas de início e fim de reservas

2. Tabela PROPRIEDADE: O índice sobre ID_PROPRIEDADE facilita a busca rápida por propriedades específicas, um campo frequentemente utilizado para identificar propriedades em consultas de alto desempenho.

3. Tabela CALENDARIO_DISPONIBILIDADE: Índices sobre ID_PROPRIEDADE, DATA_INICIO e DATA_FIM são essenciais para consultas que verificam a disponibilidade de propriedades em períodos específicos,

4. Tabela AVALIACAO: O índice sobre ID_PROPRIEDADE acelera as buscas por avaliações relacionadas a propriedades, e o índice sobre a coluna NOTA permite uma ordenação rápida ou filtragem de avaliações com base na nota atribuída.

5. Tabela PAGAMENTO: Os índices sobre ID_RESERVA e DATA_PAGAMENTO garantem que consultas relacionadas ao status de pagamento e datas de pagamento sejam realizadas de forma eficiente.

6. Tabela ESTADO_RESERVA: Índices nas colunas ID_RESERVA e DESCRICAO_ESTADO são criados para facilitar a consulta por estados específicos de reservas e permitir que operações de leitura sobre o status de uma reserva sejam rápidas

7. Tabela HISTORICO_RESERVA: Índices sobre ID_RESERVA e DATA_EVENTO são importantes para otimizar consultas que buscam o histórico de eventos de uma reserva em uma data específica.

9.1.2 Codigo para a criação dos indices:

```
1  -- INDICES
2                  --TABELA RESERVA
3  CREATE INDEX idx_reserva_propriedade ON
4  RESERVAS.RESERVA (ID_PROPRIEDADE) ON FG_INDICES;
5  CREATE INDEX idx_reserva_data_inicio ON
6  RESERVAS.RESERVA (DATA_INICIO) ON FG_INDICES;
7  CREATE INDEX idx_reserva_data_fim ON RESERVAS.RESERVA (DATA_FIM) ON FG_INDICES;
8                  --TABELA PROPRIEDADE
9  CREATE INDEX idx_propriedade_id ON
10 PROPRIEDADES.PROPRIEDADE (ID_PROPRIEDADE) ON FG_INDICES;
11                 --TABELA CALENDARIO_DISPONIBILIDADE
12 CREATE INDEX idx_calendario_propriedade ON
13 PROPRIEDADES.CALENDARIO_DISPONIBILIDADE (ID_PROPRIEDADE)ON FG_INDICES;
14 CREATE INDEX idx_calendario_data_inicio ON
15 PROPRIEDADES.CALENDARIO_DISPONIBILIDADE (DATA_INICIO) ON FG_INDICES;
16 CREATE INDEX idx_calendario_data_fim ON
17 PROPRIEDADES.CALENDARIO_DISPONIBILIDADE (DATA_FIM) ON FG_INDICES;
18                 --TABELA AVALIACAO
19 CREATE INDEX idx_avaliacao_propriedade ON
20 RESERVAS.AVALIACAO (ID_PROPRIEDADE) ON FG_INDICES;
21 CREATE INDEX idx_avaliacao_nota ON
22 RESERVAS.AVALIACAO (NOTA) ON FG_INDICES;
23                 --TABELA PAGAMENTO
24 CREATE INDEX idx_pagamento_reserva ON
25 RESERVAS.PAGAMENTO (ID_RESERVA) ON FG_INDICES;
26 CREATE INDEX idx_pagamento_data ON
27 RESERVAS.PAGAMENTO (DATA_PAGAMENTO) ON FG_INDICES;
28                 --TABELA ESTADO_RESERVA
29 CREATE INDEX idx_estado_reserva_reserva ON
30 RESERVAS.ESTADO_RESERVA (ID_RESERVA) ON FG_INDICES;
31 CREATE INDEX idx_estado_reserva_estado ON
32 RESERVAS.ESTADO_RESERVA (DESCRICAO_ESTADO) ON FG_INDICES;
```

```
33                  --TABELA HISTORICO_RESERVA
34  CREATE INDEX idx_historico_reserva_reserva ON
35  RESERVAS.HISTORICO_RESERVA (ID_RESERVA) ON FG_INDICES;
36  CREATE INDEX idx_historico_reserva_data ON
37  RESERVAS.HISTORICO_RESERVA (DATA_EVENTO) ON FG_INDICES;
```

Listing 9.1: Código para Indices

```
39  SELECT t.name AS table_name,
40         i.name AS index_name,
41         i.type_desc AS index_type
42  FROM sys.tables t
43  JOIN sys.indexes i ON t.object_id = i.object_id
44  WHERE i.type_desc <> 'HEAP'
45  ORDER BY t.name, i.name;
46
```

117 %

Results | Messages

	table_name	index_name	index_type
18	ESTADO_RESERVA	idx_estado_reserva_reserva	NONCLUSTERED
19	ESTADO_RESERVA	PK__ESTADO_R__241E2E01424E36D4	CLUSTERED
20	HISTORICO_RESERVA	idx_historico_reserva_data	NONCLUSTERED
21	HISTORICO_RESERVA	idx_historico_reserva_reserva	NONCLUSTERED
22	HISTORICO_RESERVA	PK__HISTORIC__9679E8E0261C3B09	CLUSTERED
23	IMAGEM_VIDEO	PK__IMAGEM_V__1006A4F8576C1FBB	CLUSTERED
24	PAGAMENTO	idx_pagamento_data	NONCLUSTERED
25	PAGAMENTO	idx_pagamento_reserva	NONCLUSTERED
26	PAGAMENTO	PK__PAGAMENT__F3C896DB6ADD8E...	CLUSTERED
27	PAIS	PK__PAIS__B68D33A111DF3340	CLUSTERED
28	PRECO_EPOCA	PK__PRECO_EP__7C6F5D528E15045E	CLUSTERED
29	PROPRIEDADE	idx_propriedade_id	NONCLUSTERED
30	PROPRIEDADE	PK__PROPRIED__075D971AE11AC4A5	CLUSTERED
31	PROPRIEDADES_DESTAQUE	PK__PROPRIED__075D971A843B942C	CLUSTERED
32	PROPRIETARIO	PK__PROPRIET__215415B61752539D	CLUSTERED
33	RESERVA	idx_reserva_data_fim	NONCLUSTERED
34	RESERVA	idx_reserva_data_inicio	NONCLUSTERED

Figura 9.1: Indices criados

Capítulo 10

Conclusão

Este trabalho documentou o desenvolvimento de uma base de dados completa e otimizada, cobrindo desde a estruturação das tabelas até a implementação de triggers e stored procedures, cada elemento foi cuidadosamente projetado para oferecer eficiência, segurança e escalabilidade.

As estratégias de desempenho adotadas, como a utilização de filegroups e índices otimizados, garantem um acesso rápido aos dados, enquanto as práticas de segurança reforçam a proteção das informações. A abordagem automatizada para manutenção e monitoramento do sistema assegura a continuidade operacional, reduzindo custos e aumentando a confiabilidade.

Portanto, o projeto "DREAMBOOKINGS"atingiu plenamente os objetivos estabelecidos, resultando em uma base de dados funcional, escalável e segura, preparada para atender às demandas de sistemas complexos. As técnicas aplicadas e os conhecimentos adquiridos ao longo do trabalho representam uma base sólida para futuros desenvolvimentos e desafios técnicos

Capítulo 11

Referências Bibliográficas

1. Elmasri, R., Navathe, S. B. (2021). Fundamentals of Database Systems. Pearson.
2. Garcia-Molina, H., Ullman, J. D., Widom, J. (2022). Database Systems: The Complete Book. Pearson.
3. Johnson, M. (2021). "Digital Marketplaces and Trust: A Study on Online Rental Platforms."Journal of Digital Innovation, 14(3), 45-59
4. Smith, J. (2023). "The Evolution of Vacation Rentals in the Digital Age."International Journal of Hospitality Technology, 10(2), 101-119.