



SQL 2016 - Programação em T-SQL (online)

Gabarito



1828_Gabarito01
Abril/ 2017

Mãos à obra!

Laboratório 1 da Aula 1

--A - Criação de banco de dados

--3. Crie um banco de dados de nome DB_Aula_Impacta;
`CREATE DATABASE DB_Aula_Impacta`

--4. Coloque o banco de dados em uso;

`USE DB_Aula_Impacta`

--B - Consulta de objetos do Servidor

--1. Através de funções, procedures, views ou tabelas, retorne as informações abaixo:

--- Nome do banco

`SELECT DB_NAME() AS NOME_DO_BANCO`

--- Lista dos arquivos do banco de dados

`EXEC SP_HELPFILE`

--- Liste os objetos do banco de dados

`SELECT * FROM SYSOBJECTS`

--- Lista dos logins

`EXEC SP_HELPLOGINS`

Laboratório 3 da Aula 1

--A - Utilizando os objetos de catálogo

--1. Através de funções, procedures, views ou tabelas, retorne as informações abaixo:

--- Liste as tabelas de usuário do banco de dados

`SELECT * FROM SYSOBJECTS WHERE XTYPE = 'U'`

--OU

`SELECT * FROM SYS.TABLES`

--- Liste os campos da tabela TB_CLIENTE

`SELECT * FROM SYSCOLUMNS WHERE OBJECT_NAME(ID) = 'TB_CLIENTE'`

--ou

`EXEC SP_HELP TB_CLIENTE`

-- Apresente os objetos do Tipo = 'V'

`SELECT * FROM SYSOBJECTS WHERE XTYPE='V'`

-- Verifique a estrutura da tabela TB_Pedido

`EXEC SP_HELP TB_Pedido`

Laboratório 1 das Aulas 2, 3, 4, 5, 6 e 7

```
-- 1. Coloque em uso o banco de dados PEDIDOS;
USE PEDIDOS
GO
-- 2. Liste todos os pedidos (TB_PEDIDO) do vendedor 'MARCELO' em
janeiro de 2014;
SELECT P.*, V.NOME
FROM TB_PEDIDO P JOIN TB_VENDEDOR V ON P.CODVEN = V.CODVEN
WHERE V.NOME = 'MARCELO' AND
      P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
-- OU
SELECT P.*, V.NOME
FROM TB_PEDIDO P JOIN TB_VENDEDOR V ON P.CODVEN = V.CODVEN
WHERE V.NOME = 'MARCELO' AND
      YEAR(P.DATA_EMISSAO) = 2014 AND
      MONTH(P.DATA_EMISSAO) = 1

-- 3. Liste todos os pedidos de janeiro de 2014
-- mostrando o nome do cliente e do vendedor em cada pedido;
SELECT P.*, C.NOME AS CLIENTE, V.NOME AS VENDEDOR
FROM TB_PEDIDO P
      JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
      JOIN TB_VENDEDOR V ON P.CODVEN = V.CODVEN
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'

/*
4. Liste todos os itens de TB_PEDIDO de janeiro de 2014 com
desconto superior a 7%.
Devem ser mostrados NUM_PEDIDO, DESCRICAO do produto, NOME
do cliente, nome do VENDEDOR e QUANTIDADE vendida;
*/
SELECT
      I.NUM_PEDIDO, I.DESCONTO, I.QUANTIDADE, PE.DATA_EMISSAO,
      PR.DESCRICAO, C.NOME AS CLIENTE, V.NOME AS VENDEDOR
FROM TB_PEDIDO PE
      JOIN TB_CLIENTE C ON PE.CODCLI = C.CODCLI
      JOIN TB_VENDEDOR V ON PE.CODVEN = V.CODVEN
      JOIN TB_ITENSPEDIDO I ON PE.NUM_PEDIDO = I.NUM_PEDIDO
      JOIN TB_PRODUTO PR ON I.ID_PRODUTO = PR.ID_PRODUTO
WHERE PE.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31' AND
      I.DESCONTO > 7

-- 5. Calcule a quantidade de pedidos cadastrados em janeiro de
2014
-- e o maior e o menor valor de pedido (VLR_TOTAL);
SELECT COUNT(*) AS QTD_PEDIDOS,
      MAX(VLR_TOTAL) AS MAIOR_PEDIDO,
      MIN(VLR_TOTAL) AS MENOR_PEDIDO
FROM TB_PEDIDO
WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'

/*
```

6. Calcule o valor total vendido (soma de PEDIDOS.VLR_TOTAL) e o valor da comissão

(soma de PEDIDOS.VLR_TOTAL * VENDEDORES.PORC_COMISSAO/100) de cada vendedor em janeiro de 2014;

*/

```
SELECT V.CODVEN, V.NOME,
       SUM(P.VLR_TOTAL) AS TOT_VENDIDO,
       SUM(P.VLR_TOTAL * V.PORC_COMISSAO / 100) AS COMISSAO
FROM TB_PEDIDO P
JOIN TB_VENDEDOR V ON P.CODVEN = V.CODVEN
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY V.CODVEN, V.NOME
```

-- 7. Liste os nomes e o total comprado pelos 10 clientes que mais compraram em janeiro de 2014;

```
SELECT TOP 10
       C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P
JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY C.CODCLI, C.NOME
ORDER BY TOT_COMPRADO DESC
```

-- 8. Liste os nomes dos clientes que não compraram em janeiro de 2014;

```
SELECT * FROM TB_CLIENTE
WHERE CODCLI NOT IN
(
  SELECT DISTINCT CODCLI FROM TB_PEDIDO
  WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
)
ORDER BY NOME
```

-- 9. Reajuste os preços de venda de todos os produtos com COD_TIPO = 5 de modo que fiquem 20% acima do preço de custo;

```
UPDATE TB_PRODUTO SET PRECO_VENDA = PRECO_CUSTO * 1.2
WHERE COD_TIPO = 5
```

/*

10. Reajuste os preços de venda de todos os produtos com descrição do tipo igual a REGUA, de modo que fiquem 40% acima do preço de custo;

*/

```
UPDATE TB_PRODUTO SET PRECO_VENDA = PRECO_CUSTO * 1.4
FROM TB_PRODUTO P JOIN TB_TIPOPRODUTO T ON P.COD_TIPO = T.COD_TIPO
WHERE T.TIPO = 'REGUA'
-- OU
UPDATE TB_PRODUTO SET PRECO_VENDA = PRECO_CUSTO * 1.4
WHERE COD_TIPO = (SELECT COD_TIPO FROM TB_TIPOPRODUTO
                  WHERE TIPO = 'REGUA')
```

Laboratório 2 das Aulas 2, 3, 4, 5, 6 e 7

```
-- Parte A

-- 1. Coloque em uso o banco de dados PEDIDOS;
USE PEDIDOS

-- 2. Gere uma cópia da tabela PRODUTOS chamada PRODUTOS_COPIA;
IF OBJECT_ID('PRODUTOS_COPIA', 'U') IS NOT NULL
    DROP TABLE PRODUTOS_COPIA;

SELECT * INTO PRODUTOS_COPIA FROM TB_PRODUTO;
GO

-- 3. Exclua da tabela PRODUTOS_COPIA os produtos que sejam do tipo
'CANETA',
-- exibindo os registros que foram excluídos (OUTPUT);
DELETE FROM PRODUTOS_COPIA
OUTPUT DELETED.*
FROM PRODUTOS_COPIA P JOIN TB_TIPOPRODUTO T ON P.COD_TIPO = T.COD_TIPO
WHERE T.TIPO = 'CANETA'

/*
4. Aumente em 10% os preços de venda dos produtos do tipo REGUA,
mostrando com OUTPUT as seguintes colunas:
    ID_PRODUTO, DESCRICAO, PRECO_VENDA_ANTIGO e PRECO_VENDA_NOVO;
*/
UPDATE PRODUTOS_COPIA SET PRECO_VENDA = PRECO_VENDA * 1.10
OUTPUT INSERTED.ID_PRODUTO, INSERTED.DESCRICAO,
        DELETED.PRECO_VENDA AS PRECO_VENDA_ANTIGO,
        INSERTED.PRECO_VENDA AS PRECO_VENDA_NOVO
FROM PRODUTOS_COPIA P JOIN TB_TIPOPRODUTO T ON P.COD_TIPO = T.COD_TIPO
WHERE T.TIPO = 'REGUA'

/*
5. Utilizando o comando MERGE, faça com que a tabela PRODUTOS_COPIA
volte a ser idêntica à tabela PRODUTOS, ou seja,
    o que foi deletado de PRODUTOS_COPIA deve ser reinserido, e os
produtos que tiveram seus preços alterados devem ser alterados novamente
para que voltem a ter o preço anterior.
    O MERGE deve possuir uma cláusula OUTPUT que mostre as seguintes
colunas: ação executada pelo MERGE (DELETE, INSERT, UPDATE), ID_PRODUTO,
PRECO_VENDA_ANTIGO, PRECO_VENDA_NOVO;
*/
SET IDENTITY_INSERT PRODUTOS_COPIA ON;

MERGE PRODUTOS_COPIA PC
USING TB_PRODUTO P
ON PC.ID_PRODUTO = P.ID_PRODUTO
WHEN MATCHED AND PC.PRECO_VENDA <> P.PRECO_VENDA THEN
    UPDATE SET PC.PRECO_VENDA = P.PRECO_VENDA
WHEN NOT MATCHED THEN
    INSERT (ID_PRODUTO, COD_PRODUTO, DESCRICAO, COD_UNIDADE,
            COD_TIPO, PRECO_CUSTO, PRECO_VENDA, QTD_ESTIMADA,
            QTD_REAL, QTD_MÍNIMA, CLAS_FISC, IPI, PESO_LIQ)
    VALUES (ID_PRODUTO, COD_PRODUTO, DESCRICAO, COD_UNIDADE,
            COD_TIPO, PRECO_CUSTO, PRECO_VENDA, QTD_ESTIMADA,
            QTD_REAL, QTD_MÍNIMA, CLAS_FISC, IPI, PESO_LIQ)
OUTPUT $ACTION, INSERTED.ID_PRODUTO,
        DELETED.PRECO_VENDA AS PRECO_VENDA_ANTIGO,
        INSERTED.PRECO_VENDA AS PRECO_VENDA_NOVO;

SET IDENTITY_INSERT PRODUTOS_COPIA OFF;
```

```
-- Parte B
--1
USE PEDIDOS
GO

--2
SELECT ESTADO, VLR_TOTAL, MONTH(DATA_EMISSAO) AS MES
FROM TB_PEDIDO
JOIN TB_CLIENTE ON TB_CLIENTE.CODCLI = TB_PEDIDO.CODCLI
WHERE YEAR(DATA_EMISSAO) = 2006

--3
SELECT ESTADO, [1] AS MES1, [2] AS MES2, [3] AS MES3, [4] AS MES4, [5] AS
MES5,
[6] AS MES6, [7] AS MES7, [8] AS MES8, [9] AS MES9, [10]
AS MES10,
[11] AS MES11, [12] AS MES12
FROM (SELECT ESTADO, VLR_TOTAL, MONTH(DATA_EMISSAO) AS MES
FROM TB_PEDIDO
JOIN TB_CLIENTE ON TB_CLIENTE.CODCLI = TB_PEDIDO.CODCLI
WHERE YEAR(DATA_EMISSAO) = 2006) P
PIVOT( SUM(VLR_TOTAL) FOR MES IN ([1],[2],[3],[4],[5],[6],[7],[8],[9],
[10],[11],[12])) AS PVT
ORDER BY 1

--4
SELECT ESTADO, CIDADE, [1] AS MES1, [2] AS MES2, [3] AS MES3, [4] AS
MES4, [5] AS MES5,
[6] AS MES6, [7] AS MES7, [8] AS MES8,
[9] AS MES9, [10] AS MES10,
[11] AS MES11, [12] AS MES12
FROM (SELECT ESTADO, CIDADE, VLR_TOTAL, MONTH(DATA_EMISSAO) AS MES
FROM TB_PEDIDO
JOIN TB_CLIENTE ON TB_CLIENTE.CODCLI = TB_PEDIDO.CODCLI
WHERE YEAR(DATA_EMISSAO) = 2006
) P
PIVOT( SUM(VLR_TOTAL) FOR MES IN ([1],[2],[3],[4],[5],[6],[7],[8],[9],
[10],[11],[12])) AS PVT
ORDER BY 1

-- Parte C
--1
USE PEDIDOS
GO

--2
WITH CTE( MES, ANO, MAIOR_PEDIDO )
AS
(
-- Membro âncora
SELECT MONTH( DATA_EMISSAO ) AS MES,
YEAR( DATA_EMISSAO ) AS ANO,
MAX( VLR_TOTAL ) AS MAIOR_PEDIDO
FROM TB_PEDIDO
WHERE YEAR(DATA_EMISSAO) = 2006
GROUP BY MONTH(DATA_EMISSAO), YEAR(DATA_EMISSAO)
)

-- Utilização da CTE fazendo JOIN com a tabela PEDIDOS
SELECT CTE.MES, CTE.ANO, CTE.MAIOR_PEDIDO, P.NUM_PEDIDO , C.NOME
FROM CTE JOIN TB_PEDIDO P ON CTE.MES = MONTH(P.DATA_EMISSAO) AND
CTE.ANO = YEAR(P.DATA_EMISSAO) AND
CTE.MAIOR_PEDIDO = P.VLR_TOTAL
JOIN TB_CLIENTE C
ON C.CODCLI = P.CODCLI
```

```
-- Parte D - Utilizando APPLY
--1. Coloque em uso o banco de dados PEDIDOS;
USE PEDIDOS

/*2. Realize uma consulta que apresente as informações abaixo:

- Código, nome, número do pedido, valor total e estado do cliente;
- Quantos pedidos os cliente realizou;
- A soma do valor total dos pedidos do cliente;
- A quantidade dos pedidos do estado do cliente;
- A soma do valor total dos pedidos do estado do cliente;
- O maior e menor valor dos pedidos do estado do cliente;
- A data da última compra do pedido do estado do cliente,
- Percentual da compra sobre o total do mês:
(VLR_Total / Total do mês *100);
- Quantidade de dias entre data de emissão com a última compra;
Os registros devem ser apenas de Janeiro de 2014;
Ordenar pelo nome do cliente e número de pedido.*/

SELECT      C.CODCLI, C.NOME, P.NUM_PEDIDO, C.ESTADO,P.DATA_EMISSAO,P.
VLR_TOTAL,

    (      SELECT COUNT(*)

    FROM TB_PEDIDO AS PE

    JOIN TB_CLIENTE AS CL ON PE.CODCLI = CL.CODCLI

    WHERE CL.ESTADO = C.ESTADO AND

    YEAR(PE.DATA_EMISSAO) =2014 ) AS QTD_ESTADO,

    CR.QTD_PED, CR.TOTAL, CR.MAIOR_VALOR , CR.MENOR_VALOR , CR.DT_ULT_
PEDIDO ,

    P.VLR_TOTAL/CR.TOTAL *100 AS Perc_Pedido,

    FLOOR(CAST(CR.DT_ULT_PEDIDO - P.DATA_EMISSAO AS FLOAT)) AS QTD_
DIAS
FROM TB_CLIENTE AS C
JOIN TB_PEDIDO AS P ON P.CODCLI = C.CODCLI
CROSS APPLY
    ( SELECT      COUNT(*) AS QTD_PED,

    SUM(VLR_TOTAL) AS TOTAL,

    MAX(VLR_TOTAL) AS MAIOR_VALOR,

    MIN(VLR_TOTAL) AS MENOR_VALOR,

    MAX(DATA_EMISSAO) AS DT_ULT_PEDIDO
FROM TB_PEDIDO AS PC
WHERE C.CODCLI = PC.CODCLI AND YEAR(PC.DATA_EMISSAO) = 2014 )AS CR
WHERE YEAR(P.DATA_EMISSAO) = 2014
ORDER BY C.NOME
```

Laboratório 1 das Aulas 8, 9 e 10

USE PEDIDOS

-- 1. Crie os seguintes UDDTs

```
/*
    TIPO_CODIGO            INT            NOT NULL
    TIPO_ENDERECO          VARCHAR(60)     NULL
    TIPO_FONE              CHAR(14)        NULL
    TIPO_SEXO              CHAR(1)         NOT NULL
    TIPO_ALIQUOTA          NUMERIC(4,2)    NULL
    TIPO_PRAZO             INT            NOT NULL
*/
```

-- Resposta:

```
CREATE TYPE TIPO_CODIGO
FROM INT NOT NULL;

CREATE TYPE TIPO_ENDERECO
FROM VARCHAR(60) NOT NULL;

CREATE TYPE TIPO_FONE
FROM CHAR(14) NOT NULL;

CREATE TYPE TIPO_SEXO
FROM CHAR(1) NOT NULL;

CREATE TYPE TIPO_ALIQUOTA
FROM NUMERIC(4,2) NULL;

CREATE TYPE TIPO_PRAZO
FROM INT NOT NULL;
```

-- 2. Exiba os UDDTs que você acabou de criar

-- Resposta:

```
SELECT * FROM SYSTYPES WHERE UID = 1
```

-- 3. Crie as regras de validação

```
/*
    R_SEXO                  Aceita somente 'F' e 'M'
    R_ALIQUOTA              Números não negativos
    R_PRAZO                 Números no intervalo de 1 até 60
*/
```

-- Resposta:

```
GO
CREATE RULE R_SEXO AS @S IN ('F', 'M')
GO
CREATE RULE R_ALIQUOTA AS @A >= 0
GO
CREATE RULE R_PRAZO AS @P BETWEEN 1 AND 60
GO
```

-- 4. Exiba as regras de validação que você acabou de criar

-- Resposta:

```
SELECT * FROM SYSOBJECTS WHERE XTYPE = 'R'
```



```
-- 5. Associe as regras aos seus UDDTs
/*
    R_SEXO          ao UDDT TIPO_SEXO
    R_ALIQUOTA      ao UDDT TIPO_ALIQUOTA
    R_PRAZO         ao UDDT TIPO_PRAZO
*/
-- Resposta:
EXEC SP_BINDRULE 'R_SEXO', 'TIPO_SEXO'
EXEC SP_BINDRULE 'R_ALIQUOTA', 'TIPO_ALIQUOTA'
EXEC SP_BINDRULE 'R_PRAZO', 'TIPO_PRAZO'

-- 6. Crie os seguintes objetos DEFAULT
/*
    D_SEXO          "M"
    D_ALIQUOTA      0 (ZERO)
    D_PRAZO
1
*/
-- Resposta:
GO
CREATE DEFAULT D_SEXO AS 'M'
GO
CREATE DEFAULT D_ALIQUOTA AS 0
GO
CREATE DEFAULT D_PRAZO AS 1
GO

-- 7. Exiba os DEFAULTs que você acabou de criar
-- Resposta:
SELECT * FROM SYSOBJECTS WHERE XTYPE = 'D'

-- 8. Associe os defaults aos UDDTs
/*
    D_SEXO a TIPO_SEXO
    D_ALIQUOTA a TIPO_ALIQUOTA
    D_PRAZO a TIPO_PRAZO
*/
-- Resposta:
EXEC SP_BINDEFULT 'D_SEXO', 'TIPO_SEXO'
EXEC SP_BINDEFULT 'D_ALIQUOTA', 'TIPO_ALIQUOTA'
EXEC SP_BINDEFULT 'D_PRAZO', 'TIPO_PRAZO'

-- 9. Crie as tabelas
/*
    PESSOAS

    COD_PESSOA TIPO_CODIGO      autonumeração      chave primária
    NOME       VARCHAR(30)
    ENDERECO   TIPO_ENDERECO
    SEXO       TIPO_SEXO
    -----
*/
-- Resposta:
CREATE TABLE PESSOAS
( COD_PESSOA          TIPO_CODIGO          IDENTITY          PRIMARY KEY,
  NOME               VARCHAR(30),
  ENDERECO           TIPO_ENDERECO,
  SEXO               TIPO_SEXO )
```

```
-- Crie tabela Contas
/*
    CONTAS

    COD_CONTA    TIPO_CODIGO    autonumeração    chave primária
    VALOR        NUMERIC(10,2)
    QTD_PARCELAS    TIPO_PRAZO
    PORC_MULTA    TIPO_ALIQUOTA
*/
-- Resposta:
CREATE TABLE CONTAS
( COD_CONTA            TIPO_CODIGO            IDENTITY    PRIMARY KEY,
  VALOR                NUMERIC(10,2),
  QTD_PARCELAS         TIPO_PRAZO,
  PORC_MULTA           TIPO_ALIQUOTA )

-- 11. Insira 5 registros na tabela PESSOAS
-- Resposta:
INSERT INTO PESSOAS VALUES ('MAGNO', 'RUA A', 'M')
INSERT INTO PESSOAS VALUES ('PEDRO', 'RUA B', 'M')
INSERT INTO PESSOAS VALUES ('SONIA', 'RUA C', 'F')
INSERT INTO PESSOAS VALUES ('LUIZA', 'RUA D', 'F')
INSERT INTO PESSOAS VALUES ('JULIO', 'RUA E', 'M')

-- 12. Exiba os registros da tabela PESSOAS
-- Resposta:
SELECT * FROM PESSOAS

--13. Crie a tabela Funcionario, seguindo o modelo adiante:
/*
Funcionario
COD_FUNC        TIPO_CODIGO        chave primária,
NOME            VARCHAR(30)
ENDereco        VARCHAR(80)
SEXO            TIPO_SEXO
*/

CREATE TABLE FUNCIONARIO (
COD_FUNC        TIPO_CODIGO        PRIMARY KEY,
NOME            VARCHAR(30),
ENDereco        VARCHAR(80),
SEXO            TIPO_SEXO )

--14. Crie um sinônimo de nome tb_Funcionario para a tabela
FUNCIONARIO.
CREATE SYNONYM TB_FUNCIONARIO FOR DBO.FUNCIONARIO

--15. Crie uma SEQUENCE de nome SQ_FUNCIONARIO, que inicie em 100
com incremento 2.
CREATE SEQUENCE SQ_FUNCIONARIO
START WITH 100
INCREMENT BY 1;

--16. Insira um registro na tabela FUNCIONARIO utilizando a
SEQUENCE SQ_FUNCIONARIO e o sinônimo Funcionario.
INSERT INTO TB_FUNCIONARIO (COD_FUNC , NOME , ENDereco ,SEXO)
VALUES (NEXT VALUE FOR DBO.SQ_FUNCIONARIO, 'ANTONIO DA SILVA', 'AV
PAULISTA, 1009' , 'M');
```

Laboratório 2 das Aulas 8, 9 e 10

```

--A - Trabalhando com objetos binários

/*1. Crie a tabela TB_Documento com as características:
- ID_DOCUMENTO inteiro auto numerável e chave primária
- Descrição do documento - Texto livre com até 100 caracteres
- Data do Cadastro - Deve possuir valor padrão (Data e Hora do servidor)
- Documento - Campo binário
*/
GO
CREATE TABLE TB_Documento
(
    ID_DOCUMENTO INT
        IDENTITY PRIMARY KEY,
    DESCRICAO
        VARCHAR(100),
    DATA_CADASTRO DATETIME DEFAULT (GETDATE()),
    Documento
        VARBINARY(MAX)
)

GO
--2. Insira 2 documentos na tabela TB_DOCUMENTO

Insert Into TB_Documento(DESCRICAO, DOCUMENTO)
Select 'Planilha Excel', BulkColumn
from Openrowset (Bulk 'C:\DADOS\PESSOA.XLS', Single_Blob) as
Image

Insert Into TB_Documento(DESCRICAO, DOCUMENTO)
Select 'Arquivo Texto', BulkColumn
from Openrowset (Bulk 'C:\DADOS\ArqTXT.txt', Single_Blob) as
Image

--3. Consulte a tabela TB_DOCUMENTO.
SELECT * FROM TB_Documento

--B - Habilitando FILETABLE
-- No SQL Server Management Studio execute o comando:
-- Enable Filestream
EXEC sp_configure filestream_access_level, 2
RECONFIGURE

--6. Para criar um banco com FILESTREAM execute o comando abaixo:
CREATE DATABASE Banco_LAB3
ON PRIMARY
(Name = FG_Filestream_PRIMARY,
FILENAME = 'C:\DADOS\LAB_Filestream_DATA3.mdf'),
FILEGROUP FG_Filestream_FS CONTAINS FILESTREAM
(NAME = Filestream_ARQ,
FILENAME='C:\DADOS\LAB_Filestream_ARQ3')
LOG ON
(Name = Filestream_log,
FILENAME = 'C:\DADOS\LAB_Filestream_log3.ldf')
WITH FILESTREAM (NON TRANSACTED ACCESS = FULL,
DIRECTORY_NAME = N'Filestream_ARQ3');
GO

```

```
--C- Inserindo e visualizando arquivos
--1.Coloque o banco BANCO_LAB3 em uso;
USE BANCO_LAB3

--2.Crie uma tabela FILETABLE de nome FT_Documento;
CREATE TABLE FT_Documento AS FileTable

--3.Insira 2 documentos nesta tabela;
Insert Into FT_Documento (name, file_stream)
Select 'Planilha Excel', BulkColumn
from Openrowset (Bulk 'C:\DADOS\PESSOA.XLS', Single_Blob) as Image

Insert Into FT_Documento(name, file_stream)
Select 'Arquivo Texto', BulkColumn
from Openrowset (Bulk 'C:\DADOS\ArqTXT.txt', Single_Blob) as Image

--5. Visualize os documentos com comando TSQL.
SELECT * FROM FT_Documento

--OU

SELECT Tab.Name as Nome,
IIF(Tab.is_directory=1, 'Diretório', 'Arquivo') as Tipo,
Tab.file_type as Extensao,
Tab.cached_file_size/1024.0 as Tamanho_KB,
Tab.creation_time as Data_Criacao,
Tab.file_stream.GetFileNamespacePath(1,0) as Caminho,
ISNULL(Doc.file_stream.GetFileNamespacePath(1,0), 'Root Directory') [Parent
Path]
FROM FT_Documento as Tab
LEFT JOIN FT_Documento as Doc
ON Tab.path_locator.GetAncestor(1) = Doc.path_locator
```

Laboratório 3 das Aulas 8, 9 e 10

```
---A - Trabalhando com Colunas computadas

--1.Coloque o banco PEDIDOS em uso;
USE PEDIDOS

/*2.Crie a tabela TB_FUNC_IDADE com os campos:

- Id funcionario          inteiro, auto numerável e chave primária
- Nome do funcionário    alfanumérico com 50 caracteres
- Data de Nascimento      Campo data
- Idade                   Campo calculado
*/
GO
CREATE TABLE TB_FUNC_IDADE
(
    Id_funcionario
        INT
        IDENTITY          PRIMARY KEY,

    FUNCIONARIO
        VARCHAR(50) ,
    Data_Nascimento
        DATETIME,
    Idade
        AS CAST(FLOOR(CAST(GETDATE()- data_nascimento AS FLOAT)/365.25) AS
INT )
)
GO
```

```
--3. Insira os dados da tabela de empregados para a tabela TB_FUNC_IDADE;
INSERT INTO TB_FUNC_IDADE
SELECT NOME, DATA_NASCIMENTO FROM TB_EMPREGADO

--4. Consulte as informações e verifique o campo calculado.
SELECT * FROM TB_FUNC_IDADE

/*5. Adicione o campo VLR_ITEM na tabela TB_ITENSPEDIDO, com o
cálculo abaixo:

PR_UNITARIO * QUANTIDADE * (1 - DESCONTO /100)*/

ALTER TABLE TB_ITENSPEDIDO ADD
    VLR_ITEM AS PR_UNITARIO * QUANTIDADE * (1 - DESCONTO /100)

--6. Faça uma consulta na tabela e verifique o resultado.
SELECT * FROM TB_ITENSPEDIDO
```

Laboratório 1 das Aulas 11 e 12

--Laboratório A - Índices

```
--1. Coloque o banco PEDIDOS em uso;
USE PEDIDOS

--2. Verifique se a tabela TB_CLIENTE possui índices;
EXEC SP_HELPINDEX TB_CLIENTE

--3. Crie os índices para a tabela TB_CLIENTE, campos:
--- Nome
CREATE INDEX IX_TB_CLIENTE_NOME ON TB_CLIENTE (NOME)

--- Fantasia
CREATE INDEX IX_TB_CLIENTE_FANTASIA ON TB_CLIENTE (FANTASIA)

--- Estado e Cidade
CREATE INDEX IX_TB_CLIENTE_ESTADO_CIDADE ON TB_CLIENTE (ESTADO,
CIDADE)

--- Nome, e inclua os campos: Estado e Cidade
CREATE INDEX IX_TB_CLIENTE_NOME_INCLUDE ON TB_CLIENTE (NOME)
INCLUDE (ESTADO, CIDADE)

--4. Crie os índices para a tabela TB_Pedido, campos:
--- Data emissao
CREATE INDEX IX_TB_Pedido_DATA_EMISSAO ON TB_PEDIDO (DATA_EMISSAO)

--- CODCLI
CREATE INDEX IX_TB_Pedido_CODCLI ON TB_PEDIDO (CODCLI)

--- CODVEN
CREATE INDEX IX_TB_Pedido_CODVEN ON TB_PEDIDO (CODVEN)
```

```
--Laboratório B - Customizando consultas

--1. Execute uma consulta que apresente os clientes. Utilize um
HINT que force a utilização de um índice criado no laboratório
anterior;
SELECT CODCLI,NOME FROM TB_CLIENTE WITH (INDEX = IX_TB_CLIENTE_
NOME)

--2. Execute o comando:

--BEGIN TRAN
BEGIN TRAN
--UPDATE TB_PRODUTO SET PRECO_VENDA *=PRECO_VENDA *1.2
--WHERE COD_TIPO=3;

UPDATE TB_PRODUTO SET PRECO_VENDA *=PRECO_VENDA *1.2
WHERE COD_TIPO=3;

--3. Abra uma nova consulta;

--4. Execute o comando:

--SET LOCK_TIMEOUT 5000
SET LOCK_TIMEOUT 5000

--5. Faça uma consulta apresentando todos os produtos;
SELECT * FROM TB_PRODUTO

--6. Execute a mesma consulta com um HINT que permita a leitura de
dados não confirmados;
SELECT * FROM TB_PRODUTO WITH (NOLOCK)

--7. Apresente os produtos que não estão bloqueados;
SELECT * FROM TB_PRODUTO WITH (READPAST)

--8. Abra uma nova consulta;

--9. Customize a seção para leitura de dados não confirmados;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

--10. Sem a utilização de um HINT, realize uma consulta dos
produtos;
SELECT * FROM TB_PRODUTO
--11. Volte para a primeira consulta e efetue o ROLLBACK da
transação;
ROLLBACK
```

Laboratório 1 das Aulas 13, 14 e 15

```
-- A - Acessando bancos de dados via OLE DB

-- 1. Execute os comandos abaixo para habilitar consultas
distribuídas:
-- Habilitar a visibilidade das opções avançadas
EXEC sp_configure 'show advanced option', '1';

reconfigure

-- Habilitar a utilização de OPENROWSET

EXEC sp_configure 'Ad Hoc Distributed Queries',1

reconfigure
```

--B Consultas distribuídas

--1. Faça uma consulta na tabela Produtos do banco ACCESS (Pedidos.accdb) que esta na pasta C:\Dados.

```
SELECT *
FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',
'C:\Dados\Pedidos.accdb';
'admin';'', PRODUTOS) P
```

--2. Utilizando o OPENROWSET, realize um consulta na tabela CLIENTES.

```
SELECT *
FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',
'C:\Dados\Pedidos.accdb';
'admin';'', CLIENTES) C
```

/*3. Ainda utilizando o OPENROWSET, realize uma consulta na tabela CLIENTES do banco PEDIDOS.ACCDB e relacione com a tabela PEDIDOS do banco PEDIDOS. Apresente as informações: Num_pedido, Nome, VLR_TOTAL, DATA_EMISSAO, dos pedidos de Janeiro de 2014.*/

```
SELECT Num_pedido, Nome, VLR_TOTAL, DATA_EMISSAO
FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',
'C:\Dados\Pedidos.accdb';
'admin';'', CLIENTES) C
join PEDIDOS.DBO.TB_PEDIDO as P on P.CODCLI = C.codcli
WHERE YEAR(P.DATA_EMISSAO) = 2014 and MONTH(DATA_EMISSAO)=1
```

--C - Trabalhando com BULK INSERT

--1. Crie a tabela TESTE_BULK_INSERT

```
CREATE TABLE TESTE_BULK_INSERT
( CODIGO
INT,
NOME
VARCHAR(40),
DATA_NASCIMENTO DATETIME )
```

--2. Através do comando BULK INSERT faça a carga na tabela TESTE_BULK_INSERT com o arquivo BULK_INSERT.txt

```
BULK INSERT TESTE_BULK_INSERT
FROM 'C:\DADOS\BULK_INSERT.txt'
WITH
(
FIELDTERMINATOR = ';',
ROWTERMINATOR = '\n',
codepage = 'acp'
)
```

--3. Faça uma consulta na tabela TESTE_BULK_INSERT e verifique se as informações foram carregadas.

```
SELECT * FROM TESTE_BULK_INSERT
```

Laboratório 2 das Aulas 13, 14 e 15

```
--A - Trabalhando com XML

-- 1. Coloque o Banco PEDIDOS em uso.
USE PEDIDOS

-- 2. Realize uma consulta, apresentando as informações: Número de
pedido, Nome do Cliente, Nome do Vendedor,
-- Data de emissão e Valor Total, dos pedidos de Janeiro de 2014 e
ordenado pelo número de pedido.
SELECT      p.NUM_PEDIDO , C.NOME AS CLIENTE , V.NOME AS VENDEDOR ,
P.DATA_EMISSAO, P.VLR_TOTAL
FROM PEDIDOS
      AS P
JOIN CLIENTES      AS C ON C.CODCLI = P.CODCLI
JOIN VENDEDORES AS V ON V.CODVEN = P.CODVEN
WHERE YEAR(P.DATA_EMISSAO) = 2014 and MONTH(DATA_EMISSAO)=1
ORDER BY p.NUM_PEDIDO

-- 3. Execute a consulta anterior, exportando para XML conforme
modelo a seguir:
SELECT      p.NUM_PEDIDO , C.NOME AS CLIENTE , V.NOME AS VENDEDOR ,
P.DATA_EMISSAO, P.VLR_TOTAL
FROM PEDIDOS
      AS P
JOIN CLIENTES      AS C ON C.CODCLI = P.CODCLI
JOIN VENDEDORES AS V ON V.CODVEN = P.CODVEN
WHERE YEAR(P.DATA_EMISSAO) = 2014 and MONTH(DATA_EMISSAO)=1
ORDER BY p.NUM_PEDIDO
FOR XML RAW
```

Laboratório 3 das Aulas 13, 14 e 15

```
--A - Trabalhando com JSON

--1. Faça uma consulta apresentando o código e nome dos CLIENTES.
SELECT CODCLI, NOME FROM TB_CLIENTE

--2. Utilizando a consulta anterior execute uma saída com JSON.
SELECT CODCLI, NOME FROM TB_CLIENTE FOR JSON AUTO

--3. Gere um arquivo no padrão JSON para a consulta do item 1.

DECLARE @CMD VARCHAR(4000)

SET @CMD =
'BCP "SELECT CODCLI, NOME FROM TB_CLIENTE FOR JSON AUTO" ' +
' QUERYOUT "C:\DADOS\SAIDAJSON.XML" -SINSTRUTOR -t -w -t -T'

EXEC MASTER..XP_CMDSHELL @CMD
```


Laboratório 1 das Aulas 16, 17 e 18

```
-- 1. Colocar em uso o banco de dados PEDIDOS
-- Resp.:
USE PEDIDOS

-- 2. Criar VIEW (VIE_TOT_VENDIDO) para mostrar o total vendido
(soma de TB_PEDIDO.VLR_TOTAL)
-- em cada mês do ano. Deve mostrar o mês, o ano e o total vendido
-- Resp.:
GO
CREATE VIEW VIE_TOT_VENDIDO AS
SELECT MONTH( DATA_EMISSAO ) AS MES,
       YEAR( DATA_EMISSAO ) AS ANO,
       SUM( VLR_TOTAL ) AS TOT_VENDIDO
FROM TB_PEDIDO
GROUP BY MONTH(DATA_EMISSAO), YEAR(DATA_EMISSAO)
GO

-- 3. Faça uma consulta VIE_TOTAL_VENDIDO no ano de 2014. Deve
ordenar os dados por mês
-- Resp.:
SELECT * FROM VIE_TOT_VENDIDO
WHERE ANO = 2014
ORDER BY MES

-- 4. Criar VIEW (VIE_MAIOR_PEDIDO) para mostrar valor do maior
pedido (MAX de TB_PEDIDO.VLR_TOTAL)
-- vendido em cada mês do ano. Deve mostrar o mês, o ano e o maior
pedido
GO
CREATE VIEW VIE_MAIOR_PEDIDO AS
SELECT MONTH( DATA_EMISSAO ) AS MES,
       YEAR( DATA_EMISSAO ) AS ANO,
       MAX( VLR_TOTAL ) AS MAIOR_PEDIDO
FROM TB_PEDIDO
WHERE YEAR(DATA_EMISSAO) = 2014
GROUP BY MONTH(DATA_EMISSAO), YEAR(DATA_EMISSAO)
GO

-- 5. Faça uma consulta VIE_MAIOR_PEDIDO no ano de 2014. Deve
ordenar os dados por mês
-- Resp.:
SELECT * FROM VIE_MAIOR_PEDIDO
WHERE ANO = 2014
ORDER BY 1

-- 6. Faça um JOIN, utilizando VIE_MAIOR_PEDIDO e PEDIDOS que
mostre também o número
-- do pedido (TB_PEDIDO.NUM_PEDIDO) de maior valor em cada mês.
Deve filtrar o ano
-- de 2014 e ordenar por mês.
-- Resp.:
SELECT V.MES, V.ANO, V.MAIOR_PEDIDO, P.NUM_PEDIDO
FROM VIE_MAIOR_PEDIDO V
JOIN TB_PEDIDO P ON V.MES = MONTH(P.DATA_EMISSAO) AND
                  V.ANO = YEAR(P.DATA_EMISSAO) AND
                  V.MAIOR_PEDIDO = P.VLR_TOTAL
WHERE V.ANO = 2014
ORDER BY MES
```

```
-- 7. Idem ao anterior, desta vez mostrando também o nome do
cliente que comprou esse pedido
SELECT V.MES, V.ANO, V.MAIOR_PEDIDO, P.NUM_PEDIDO, C.NOME AS
CLIENTE
FROM VIE MAIOR_PEDIDO V
JOIN TB_PEDIDO P ON V.MES = MONTH(P.DATA_EMISSAO) AND
V.ANO = YEAR(P.DATA_EMISSAO) AND
V.MAIOR_PEDIDO = P.VLR_TOTAL
JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE V.ANO = 2014
ORDER BY MES

-- 8. Criar uma VIEW (VIE_ITENS_PEDIDO) que mostre todos os campos
da tabela TB_ITENSPEDIDO, mais
-- DATA EMISSÃO do pedido, DESCRIÇÃO do produto, NOME do cliente
que comprou e NOME do
-- vendedor que vendeu
GO
CREATE VIEW VIE_ITENSPEDIDO AS
SELECT
I.NUM_PEDIDO, I.NUM_ITEM, I.ID_PRODUTO, I.COD_PRODUTO,
I.QUANTIDADE, I.PR_UNITARIO, I.DESCONTO, I.DATA_ENTREGA,
PE.DATA_EMISSAO, PR.DESCRICAO, C.NOME AS CLIENTE,
V.NOME AS VENDEDOR
FROM TB_PEDIDO PE
JOIN TB_CLIENTE C ON PE.CODCLI = C.CODCLI
JOIN TB_VENDEDOR V ON PE.CODVEN = V.CODVEN
JOIN TB_ITENSPEDIDO I ON PE.NUM_PEDIDO = I.NUM_PEDIDO
JOIN TB_PRODUTO PR ON I.ID_PRODUTO = PR.ID_PRODUTO
GO
-- 9. Execute VIE_ITENS_PEDIDO filtrando apenas pedidos de Jan/2014
-- Resp.:
SELECT * FROM VIE_ITENSPEDIDO
WHERE YEAR(DATA_EMISSAO) = 2014

/*10. Crie a tabela tb_CLIENTE_VIEW com os campos:
- ID Inteiro auto numerável e PRIMARY KEY
- Nome Alfanumérico de 50
- Estado Alfanumérico de 2
*/
GO
CREATE TABLE tb_CLIENTE_VIEW
(
ID
INT IDENTITY PRIMARY KEY,
NOME VARCHAR(50),
ESTADO CHAR(2)
)
GO

--11. Crie uma view de nome vW_Clientes_VIEW para consulta e
atualização da tabela tb_CLIENTE_VIEW.
CREATE VIEW vW_Clientes_VIEW AS
SELECT * FROM tb_CLIENTE_VIEW
GO

--12. Faça a inserção de 2 registros através da view vW_Clientes_VIEW.
INSERT INTO vW_Clientes_VIEW VALUES
('Antonio da Silva', 'SP'),
('Margarida Antunes', 'RJ')

--13. Realize a consulta através da view vW_Clientes_VIEW.
select * from vW_Clientes_VIEW
```

Laboratório 1 das Aulas 19, 20 e 21

--1. Complete o código a seguir de modo a mostrar o maior dos três números sorteados:

```
DECLARE @A INT, @B INT, @C INT;
DECLARE @MAIOR INT;
SET @A = 50 * RAND();
SET @B = 50 * RAND();
SET @C = 50 * RAND();
-- Aqui você colocará os IFs ---
```

-- SOLUÇÃO 1

```
IF @A > @B
    IF @A > @C
        SET @MAIOR = @A
    ELSE
        SET @MAIOR = @C
ELSE
    IF @B > @C
        SET @MAIOR = @B
    ELSE
        SET @MAIOR = @C
```

-- SOLUÇÃO 2

```
/*IF @A > @B AND @A > @C
    SET @MAIOR = @A
ELSE
    IF @B > @C
        SET @MAIOR = @B
    ELSE
        SET @MAIOR = @C
*/
```

-- SOLUÇÃO 3

```
/*SET @MAIOR = @A;
IF @B > @MAIOR SET @MAIOR = @B;
IF @C > @MAIOR SET @MAIOR = @C;
*/
```

```
PRINT @A;
PRINT @B;
PRINT @C;
PRINT 'MAIOR = ' + CAST(@MAIOR AS VARCHAR(2));
```

/*
2. Complete o código a seguir (que sorteará quatro números no intervalo de 0 a 10, os quais representarão as quatro notas de um aluno) de acordo com o que o comentário pede:
*/

```
DECLARE @N1 NUMERIC(4,2), @N2 NUMERIC(4,2), @N3 NUMERIC(4,2), @N4 NUMERIC(4,2);
DECLARE @MEDIA NUMERIC(4,2);
SET @N1 = 10 * RAND();
SET @N2 = 10 * RAND();
SET @N3 = 10 * RAND();
SET @N4 = 10 * RAND();
```

```
-- Imprimir as 4 notas
PRINT 'Nota 1: ' + CAST( @N1 AS VARCHAR(5) );
PRINT 'Nota 2: ' + CAST( @N2 AS VARCHAR(5) );
PRINT 'Nota 3: ' + CAST( @N3 AS VARCHAR(5) );
PRINT 'Nota 4: ' + CAST( @N4 AS VARCHAR(5) );
-- Calcular e imprimir a média das 4 notas
SET @MEDIA = (@N1 + @N2 + @N3 + @N4) / 4;
PRINT 'Média: ' + CAST( @MEDIA AS VARCHAR(5) );
-- Imprimir REPROVADO se média menor que 5, caso contrário
APROVADO
IF @MEDIA < 5
    PRINT 'REPROVADO'
ELSE
    PRINT 'APROVADO'
-- Dependendo da média, imprimir uma das classificações abaixo
-- Média até 2.....PÉSSIMO
-- Acima de 2 até 4.....RUIM
-- Acima de 4 até 6.....REGULAR
-- Acima de 6 até 8.....BOM
-- Acima de 8.....ÓTIMO
IF @MEDIA <= 2 PRINT 'PÉSSIMO'
ELSE IF @MEDIA <= 4 PRINT 'RUIM'
ELSE IF @MEDIA <= 6 PRINT 'REGULAR'
ELSE IF @MEDIA <= 8 PRINT 'BOM'
ELSE PRINT 'ÓTIMO'
-----
--
-- 3. Escreva um código que gere e imprima os números pares de 0
até 100;
DECLARE @CONT INT = 0;
WHILE @CONT <= 100
    BEGIN
        PRINT @CONT;
        SET @CONT += 2
    END
PRINT 'FIM'

-- 4. Escreva um código que gere e imprima os números ímpares
entre 0 e 100;
DECLARE @CONT INT = 1;
WHILE @CONT <= 100
    BEGIN
        PRINT @CONT;
        SET @CONT += 2
    END
PRINT 'FIM'

/*
5. Complete o código de modo a calcular a soma de todos os números
inteiros de 0 até @N;
*/

DECLARE @N INT, @CONT INT = 1, @SOMA INT = 0;
SET @N = CAST( 20 * RAND() AS INT );
-- Complete o código -----

WHILE @CONT <= @N
    BEGIN
        SET @SOMA += @CONT;
        SET @CONT += 1;
    END
-----
PRINT 'A SOMA DE 1 ATÉ ' + CAST(@N AS VARCHAR(2)) +
    ' É ' + CAST(@SOMA AS VARCHAR(4));

/*
```

6. Complete o código de modo a calcular o fatorial de @N. Por exemplo, o fatorial de 5 é $1 * 2 * 3 * 4 * 5 = 120$;

```

/*
DECLARE @N INT, @CONT INT = 1, @FAT INT = 1;
SET @N = CAST( 10 * RAND() AS INT );
-- Complete o código -----
WHILE @CONT <= @N
BEGIN
    SET @FAT *= @CONT;
    SET @CONT += 1;
END
-----
PRINT 'O FATORIAL DE ' + CAST(@N AS VARCHAR(2)) +
      ' É ' + CAST(@FAT AS VARCHAR(10));

```

/*

7. Insira os comandos de acordo com os comentários adiante, de modo que o código gere todos os números primos de 1 até 1000.

Números primos são números inteiros divisíveis apenas por 1 e por ele próprio.

*/

```

-- 1. Declarar as variáveis @N, @I (inteiras) e @SN_PRIMO do tipo CHAR(1)
DECLARE @N INT, @I INT, @SN_PRIMO CHAR(1);
-- 2. Imprimir os números 1, 2 e 3 que já sabemos serem primos
PRINT 1;
PRINT 2;
PRINT 3;
-- 3. Iniciar a variável @N com 4
SET @N = 4;
-- 4. Enquanto @N for menor ou igual a 1000
WHILE @N <= 1000
BEGIN
    -- 4.1. Iniciar a variável @I com 2
    SET @I = 2;
    -- 4.2. Iniciar a variável @SN_PRIMO com 'S'
    SET @SN_PRIMO = 'S';
    -- 4.3. Enquanto @I for menor ou igual a @N / 2
    WHILE @I <= @N / 2
    BEGIN
        -- 4.3.1. Se o resto da divisão de @N por @I for zero (é divisível)
        IF @N % @I = 0
        BEGIN
            -- 4.3.1.1. Colocar 'N' na variável @SN_PRIMO sinalizando assim
            -- que @N não é um número primo
            SET @SN_PRIMO = 'N';
            -- 4.3.1.2. Abandonar este Loop (4.3)
            BREAK;
        END
        -- 4.3.2. Somar 1 na variável @I
        SET @I += 1;
        -- Final do loop 4.3.
    END
    -- 4.4. Se @SN_PRIMO for 'S', imprimir @N porque ele é primo
    IF @SN_PRIMO = 'S' PRINT @N;
    -- 4.5. Somar 1 na variável @N
    SET @N = @N + 1;
    -- Final do loop (4)
END

```

Laboratório 1 da Aula 22

```
-----
-- Laboratório A
-- 1. Crie uma função chamada FN_MENOR que receba 2 números
-- inteiros e
-- retorne o MENOR deles
-----

GO
CREATE FUNCTION FN_MENOR( @N1 INT, @N2 INT )
    RETURNS INT
AS BEGIN
    DECLARE @RET INT;
    IF @N1 < @N2
        SET @RET = @N1
    ELSE
        SET @RET = @N2;

    RETURN ( @RET )
END
-- Testando
SELECT DBO.FN_MENOR( 5, 3 )
SELECT DBO.FN_MENOR( 7, 11 )

GO

-- Testando
SELECT DBO.FN_MENOR( 5, 3 )
SELECT DBO.FN_MENOR( 7, 11 )

-----
-- 2. Criar função FN_NOME_MES que retorne o nome do MÊS
-- MONTH( DATA ) -> retorna o número do mês
-----

CREATE FUNCTION FN_NOME_MES ( @DT DATETIME )
    RETURNS VARCHAR(15)
AS BEGIN
    RETURN CASE MONTH( @DT )
        WHEN 1 THEN 'JANEIRO'
        WHEN 2 THEN 'FEVEREIRO'
        WHEN 3 THEN 'MARÇO'
        WHEN 4 THEN 'ABRIL'
        WHEN 5 THEN 'MAIO'
        WHEN 6 THEN 'JUNHO'
        WHEN 7 THEN 'JULHO'
        WHEN 8 THEN 'AGOSTO'
        WHEN 9 THEN 'SETEMBRO'
        WHEN 10 THEN 'OUTUBRO'
        WHEN 11 THEN 'NOVEMBRO'
        WHEN 12 THEN 'DEZEMBRO'
    END
END

GO

-- Testando
SELECT NOME, DATA_ADMISSAO, DATENAME( MONTH, DATA_ADMISSAO ),
    DBO.FN_NOME_MES( DATA_ADMISSAO )
FROM TB_EMPREGADO

-----
-- 3. Criar função que retorna a ÚLTIMA data do mês
-- Dica: Como a data é um número em que cada dia corresponde a
--       1 unidade, podemos concluir que a última data de um mês
--       é igual à primeira data do mês seguinte menos 1
-----
```

```

GO
CREATE FUNCTION FN_ULT_DATA ( @DT DATETIME )
    RETURNS DATETIME
AS BEGIN
    DECLARE @MES INT, @ANO INT , @DATA DATETIME;
    SET @MES = MONTH(@DT) + 1;
    SET @ANO = YEAR(@DT);

    IF @MES > 12
        BEGIN

            SET @MES = 1;

            SET @ANO = @ANO + 1;
        END
    SET @DATA = DATEFROMPARTS ( @ANO, @MES, 1 )
    RETURN @DATA - 1 ;
END

GO
-- Testando
SELECT NOME, DATA_ADMISSAO, DBO.FN_ULT_DATA(DATA_ADMISSAO)
FROM TB_EMPREGADO

-- OU
SELECT NOME, DATA_ADMISSAO, EOMONTH( DATA_ADMISSAO, 1 )
FROM TB_EMPREGADO

-----
-- 4. Criar função que retorne a quantidade de
-- dias úteis existentes entre 2 datas (inclusive as 2)
-- Parâmetros:
--         @DATA_INI      DATETIME
--         @DATA_FIM DATETIME

GO
CREATE TABLE FERIADOS
( DATA DATETIME, MOTIVO VARCHAR(40) )

GO
CREATE FUNCTION FN_DIAS_UTEIS( @DATA_INI DATETIME, @DATA_FIM
DATETIME )
    RETURNS INT
AS BEGIN
    DECLARE @RET INT;
    DECLARE @DT DATETIME;

    SET @RET = 0;
    SET @DT = @DATA_INI;

    WHILE @DT < @DATA_FIM
        BEGIN
            SET @DT = @DT + 1;
            IF DATEPART( WEEKDAY, @DT ) IN (1,7) OR
                EXISTS( SELECT * FROM FERIADOS
                    WHERE DATA = @DT ) CONTINUE;
            SET @RET = @RET + 1;
        END
    RETURN (@RET);
END

GO

-- Testando
SELECT DBO.FN_DIAS_UTEIS('01.01.2008' , '01.01.2009')

```

```
-- 5. Criar uma função tabular (FN VENDAS POR PRODUTO)
-- que receba as datas inicial (@DT1) e final (@DT2) de um período
-- e retorne o total vendido de cada produto neste período
-----
/*
    SELECT

        Pr.ID_PRODUTO, Pr.DESCRICAO, SUM( I.QUANTIDADE ) AS QTD_
TOTAL,

        SUM( I.QUANTIDADE * I.PR_UNITARIO ) AS VALOR_TOTAL
    FROM ITENSPEDIDO I

    JOIN PRODUTOS Pr ON I.ID_PRODUTO=Pr.ID_PRODUTO

    JOIN PEDIDOS Pe ON I.NUM_PEDIDO=Pe.NUM_PEDIDO
    WHERE
        Pe.DATA_EMISSAO BETWEEN @DT1 AND @DT2
    GROUP BY Pr.ID_PRODUTO, Pr.DESCRICAO
*/
GO
CREATE FUNCTION FN_VENDAS_POR_PRODUTO ( @DT1 DATETIME,
                                         @DT2 DATETIME )
RETURNS TABLE
AS
RETURN (SELECT

        Pr.ID_PRODUTO, Pr.DESCRICAO, SUM( I.QUANTIDADE ) AS QTD_
TOTAL,

        SUM( I.QUANTIDADE * I.PR_UNITARIO ) AS VALOR_TOTAL
    FROM TB_ITENSPEDIDO I

    JOIN TB_PRODUTO Pr ON I.ID_PRODUTO=Pr.ID_PRODUTO

    JOIN TB_PEDIDO Pe ON I.NUM_PEDIDO=Pe.NUM_PEDIDO
    WHERE
        Pe.DATA_EMISSAO BETWEEN @DT1 AND @DT2
    GROUP BY Pr.ID_PRODUTO, Pr.DESCRICAO)
GO
-- Testando a função
SELECT * FROM FN_VENDAS_POR_PRODUTO( '2006.1.1', '2006.12.31')
ORDER BY 3
--
SELECT * FROM FN_VENDAS_POR_PRODUTO( '2006.1.1', '2006.12.31')
ORDER BY DESCRICAO
--
SELECT TOP 10 *
FROM FN_VENDAS_POR_PRODUTO( '2006.1.1', '2006.12.31')
ORDER BY 4 DESC
-----

--B - Funções de classificação
--1. Faça uma consulta que apresente o nome do cliente, o número
do pedido e o valor total.
-- Além desses campos, crie uma coluna que seja numerada
automaticamente.

SELECT      ROW_NUMBER() OVER(ORDER BY NUM_PEDIDO) AS ORDER_ROW_
NUMBER,

        C.NOME , P.NUM_PEDIDO , P.VLR_TOTAL
FROM TB_PEDIDO AS P
JOIN TB_CLIENTE AS C ON C.CODCLI = P.CODCLI
```



```
--2. Utilizando a mesma consulta, adicione uma coluna que
apresente o Ranking
--      das vendas dos clientes

SELECT      ROW_NUMBER() OVER(ORDER BY NUM_PEDIDO) AS ORDER_ROW_
NUMBER,

            RANK() OVER (ORDER BY VLR_TOTAL DESC) AS ORDER_RANK,

            C.NOME , P.NUM_PEDIDO , P.VLR_TOTAL
FROM TB_PEDIDO AS P
JOIN TB_CLIENTE AS C ON C.CODCLI = P.CODCLI

--C - Campos calculados com função

--1. Crie uma função que apresente o total de funcionários de um
departamento.
GO
CREATE FUNCTION FN_QTD_EMPREGADOS (@DEPTO INT ) RETURNS INT
AS
BEGIN

    DECLARE @RET INT

    SELECT @RET = COUNT(*) FROM TB_EMPREGADO
    WHERE COD_DEPTO = @DEPTO

    RETURN (@RET )
END
GO
--Teste
SELECT DBO.FN_QTD_EMPREGADOS ( 1)

--2. Adicione uma coluna calculada na tabela TB DEPARTAMENTO. Esta
coluna deve utilizar a função criada no exercício anterior.
GO
ALTER TABLE TB DEPARTAMENTO ADD
QTD_FUNC AS DBO.FN_QTD_EMPREGADOS (COD_DEPTO)

GO

--3. Realize uma consulta na tabela TB DEPARTAMENTO que apresente
os departamentos e a quantidade de funcionários.
SELECT * FROM TB_DEPARTAMENTO
```

Laboratório 1 das Aulas 23 e 24

--A - Criando procedures

--1. Crie uma procedure que retorne os clientes (código, nome, valor e número do pedido), com parâmetro ANO e ordenado pelo nome do cliente.

```
CREATE PROCEDURE SP_Retorna_vendas @ANO int AS

BEGIN
    SELECT      C.CODCLI , C.NOME , P.VLR_TOTAL , P.NUM_PEDIDO
    FROM TB_PEDIDO AS P
    JOIN TB_CLIENTE AS C ON C.CODCLI = P.CODCLI
    WHERE YEAR(P.DATA_EMISSAO)= @ANO

    ORDER BY C.NOME
END

GO
```

--2. Teste a procedure criada com os seguintes anos: 2012, 2013 e 2014

```
EXEC SP_Retorna_vendas 2012
EXEC SP_Retorna_vendas 2013
EXEC SP_Retorna_vendas 2014
```

--3. Crie uma procedure para inserir departamentos na tabela TB_DEPARTAMENTO.

```
GO
CREATE PROCEDURE SP_INSERE_DEPARTAMENTO @DEPTO VARCHAR(25) AS
BEGIN

    INSERT INTO TB_DEPARTAMENTO VALUES (@DEPTO)

    SELECT 'REGISTRO INSERIDO COM SUCESSO' AS MSG
END
GO
```

--4. Insira os departamentos Mensageria e TI

```
EXEC SP_INSERE_DEPARTAMENTO 'Mensageria'
EXEC SP_INSERE_DEPARTAMENTO 'TI'
```

--5. Crie uma procedure para inserir tipo de produto (TB_TIPOPRODUTO).

```
GO
CREATE PROCEDURE SP_INSERE_TIPOPRODUTO @TIPO VARCHAR(30) AS
BEGIN
    INSERT INTO TB_TIPOPRODUTO VALUES ( @TIPO )

    SELECT 'OK' AS MSG
END

GO
```

--6. Insira os tipos TESTE e TESTE2.

```
EXEC SP_INSERE_TIPOPRODUTO 'TESTE'
EXEC SP_INSERE_TIPOPRODUTO 'TESTE2'
```

--7. Crie uma procedure que exclua um tipo de produto (TB_TIPOPRODUTO). Antes de excluir é necessário que seja verificado se o tipo de produto é utilizado em produtos.
 --O parâmetro deve ser a descrição do Tipo e não o código.
 --O retorno deve ser um OK ou NOK para tipos que são utilizados por produtos.

```
GO
CREATE PROCEDURE SP_EXCLUI_TIPOPRODUTO @TIPO VARCHAR(25) AS
BEGIN
    IF EXISTS (
        SELECT *
        FROM TB_PRODUTO AS PR
        JOIN TB_TIPOPRODUTO AS T ON T.COD_TIPO = PR.COD_TIPO
        WHERE TIPO = @TIPO)
    BEGIN
        SELECT 'NOK' AS MSG
    END
    ELSE
    BEGIN
        DELETE FROM TB_TIPOPRODUTO WHERE TIPO=@TIPO
        SELECT 'OK' AS MSG
    END
END
GO
```

--8. Exclua o tipo de produto TESTE.
 EXEC SP_EXCLUI_TIPOPRODUTO 'TESTE'

--9. Exclua o tipo de produto REGUA.
 EXEC SP_EXCLUI_TIPOPRODUTO 'REGUA'

--10. Crie a tabela TB_Resumo com os seguintes campos:
 --ID_Resumo INT auto numerável chave primária
 --Ano INT
 --MÊS INT
 --Valor DECIMAL(10,2)
 GO
 CREATE TABLE TB_Resumo
 (
 ID_Resumo INT IDENTITY PRIMARY KEY,
 Ano INT,
 Mes INT,
 Valor DECIMAL(10,2)
)
 GO

/*11. Crie uma procedure que carregue as informações da tabela de pedidos. Utilize os parâmetros @ANO INT para filtrar as informações.

- Não insira valores duplicados;
- Exclua os registros do ano antes de realizar a carga;
- Utilize Transações;
- Faça o tratamento de erros com o TRY CATCH;
- Retorne a quantidade de registros carregados;
- Caso ocorra erro, retorne a mensagem.

*/

GO

```
CREATE PROCEDURE SP_CARREGA_TB_RESUMO @ANO INT AS
BEGIN
    BEGIN TRAN
    BEGIN TRY

        --Apaga registros do Ano e mês
        DELETE FROM TB_Resumo
        WHERE ANO =@ANO

        INSERT INTO TB_Resumo
        SELECT YEAR(DATA_EMISSAO), MONTH(DATA_EMISSAO), SUM(VLR_
TOTAL)
        FROM TB_PEDIDO
        WHERE YEAR(DATA_EMISSAO) = @ANO
        GROUP BY YEAR(DATA_EMISSAO), MONTH(DATA_EMISSAO)

        SELECT CAST(@@ROWCOUNT AS VARCHAR(10)) AS MSG

        COMMIT

    END TRY
    BEGIN CATCH

        ROLLBACK

        SELECT ERROR_MESSAGE() AS MSG
    END CATCH
END
GO
--
--12. Faça o teste carregando os seguintes anos: 2012, 2013 e 2014.
EXEC SP_CARREGA_TB_RESUMO 2012
EXEC SP_CARREGA_TB_RESUMO 2013
EXEC SP_CARREGA_TB_RESUMO 2014
```

--13. Faça a consulta na tabela TB_RESUMO e verifique se as informações estão corretas.
`SELECT * FROM TB_RESUMO`

Laboratório 2 das Aulas 23 e 24

```

/*
Laboratório A. Criar uma stored procedure que crie um campo novo
em todas as tabelas do banco de dados.

Sabendo que
    SELECT ID, NAME FROM SYSOBJECTS WHERE XTYPE = 'U': Lista os
nomes de todas as
    tabelas do banco de dados.
    EXEC( @COMANDO ): Executa uma instrução SQL contida em uma
variável.

Crie uma stored procedure que receba como parâmetro uma variável
@CAMPO VARCHAR(200) que conterá o nome do campo e @TIPO
VARCHAR(200),
o tipo e outras características de uma campo,

por exemplo: @CAMPO -> 'COD_USUARIO'
              @TIPO  -> 'INT NOT NULL DEFAULT 0'

*/

-- Resposta:
-- 1. Criar procedure
USE PEDIDOS

CREATE PROCEDURE STP_CRIA_CAMPO @CAMPO VARCHAR(200), @TIPO
VARCHAR(200)
AS BEGIN
-- 2. Declarar variável @COMANDO VARCHAR(200), @TABELA
VARCHAR(200) e
--    @ID INT.
DECLARE @COMANDO VARCHAR(200), @TABELA VARCHAR(200),
        @ID INT;
-- 3. Declarar cursor para SELECT ID, NAME FROM SYSOBJECTS WHERE
XTYPE = 'U'
DECLARE CR_TABELAS CURSOR KEYSET
FOR SELECT ID, NAME FROM SYSOBJECTS WHERE XTYPE = 'U'
-- 4. Abrir o cursor
OPEN CR_TABELAS;
-- 5. Ler a primeira linha do cursor
FETCH FIRST FROM CR_TABELAS INTO @ID, @TABELA;
-- 6. Enquanto não chegar no final dos dados
WHILE @@FETCH_STATUS = 0
BEGIN
-- a. Armazenar na variável comando a instrução
--    'ALTER TABLE ' + @TABELA + ' ADD ' + @CAMPO + ' '
+ @TIPO;
SET @COMANDO = 'ALTER TABLE ' + @TABELA + ' ADD ' + @CAMPO + ' '
+ @TIPO;
-- b. Executar o comando contido na variável @COMANDO
EXEC(@COMANDO);
-- c. Imprimir na área de mensagens o comando que foi executado
PRINT @COMANDO

```

```
-- d. Ler a próxima linha da tabela
FETCH NEXT FROM CR_TABELAS INTO @ID, @TABELA;
-- e. Finaliza o loop
END -- WHILE
-- 7. Fechar o cursor
CLOSE CR_TABELAS;
-- 8. Desalocar o cursor da memória
DEALLOCATE CR_TABELAS;
END
-- Testar
EXEC STP_CRIA_CAMPO 'DATA ALTERACAO',
                   'DATETIME NOT NULL DEFAULT GETDATE()'

EXEC STP_CRIA_CAMPO 'COD_USR_ALTEROU',
                   'INT NOT NULL DEFAULT 0'

-----
-- Laboratório B.
--   Alterando a procedure anterior para testar se o campo já
--   existe na tabela e imprimindo-o,
--   caso ele exista.
/*   Obs: Para testar se a tabela produtos tem um campo chamado
      PRECO_VENDA, podemos fazer:

      SELECT ID FROM SYSOBJECTS WHERE NAME = 'PRODUTOS'

      o ID da tabela PRODUTOS é: 357576312

      SELECT * FROM SYSCOLUMNS
      WHERE NAME = 'PRECO_VENDA' AND ID = 357576312

      Adaptando para a procedure, podemos fazer:

      IF EXISTS(SELECT * FROM SYSCOLUMNS
                WHERE NAME = @CAMPO AND ID = @ID)

*/

-- Resposta:

USE PEDIDOS

ALTER PROCEDURE STP_CRIA_CAMPO @CAMPO VARCHAR(200), @TIPO
VARCHAR(200)
AS BEGIN
-- 1. Declarar variável @COMANDO VARCHAR(200), @TABELA
VARCHAR(200) e
--   @ID INT.
DECLARE @COMANDO VARCHAR(200), @TABELA VARCHAR(200),
        @ID INT;
-- 2. Declarar cursor para SELECT ID, NAME FROM SYSOBJECTS WHERE
XTYPE = 'U'
DECLARE CR_TABELAS CURSOR KEYSET
        FOR SELECT ID, NAME FROM SYSOBJECTS WHERE XTYPE = 'U'
-- 3. Abrir o cursor
OPEN CR_TABELAS;
-- 4. Ler a primeira linha do cursor
FETCH FIRST FROM CR_TABELAS INTO @ID, @TABELA;
```

```
-- 5. Enquanto não chegar no final dos dados
WHILE @@FETCH_STATUS = 0
BEGIN
    IF EXISTS(SELECT * FROM SYSCOLUMNS
              WHERE NAME = @CAMPO AND ID = @ID)
        PRINT @CAMPO + ' JÁ EXISTE EM ' + @TABELA;
    ELSE
        BEGIN
            -- 5.1. Armazenar na variável comando a instrução
            -- 'ALTER TABLE ' + @TABELA + ' ADD ' + @CAMPO +
            ' ' + @TIPO;
            SET @COMANDO = 'ALTER TABLE ' + @TABELA + ' ADD ' + @CAMPO
+ ' ' + @TIPO;
            -- 5.2. Executar o comando contido na variável @COMANDO
            EXEC(@COMANDO);
            -- 5.3. Imprimir na área de mensagens o comando que foi
            executado
            PRINT @COMANDO
            END -- Fim do bloco ELSE do IF
            -- 5.4. Ler a próxima linha da tabela
            FETCH NEXT FROM CR_TABELAS INTO @ID, @TABELA;
            -- Fim do loop
            END -- WHILE
        -- 6. Fechar o cursor
        CLOSE CR_TABELAS;
        -- 7. Desalocar o cursor da memória
        DEALLOCATE CR_TABELAS;
    END
```

Laboratório 1 da Aula 25

```
-----
-- Laboratório A
-- 1. Crie trigger para TB_PRODUTO que seja executado sempre
-- que ocorrer alteração de registro. Devem ser inseridos dados
-- na tabela de histórico se houver alteração de PRECO_VENDA

CREATE TABLE PRODUTOS_HIST_PRECO
( NUM_MOVTO
  INT IDENTITY,
  ID_PRODUTO INT,
  DATA_ALTERACAO DATETIME,
  PRECO_ANTIGO NUMERIC(12,4),

  PRECO_NOVO
  NUMERIC(12,4),
  CONSTRAINT PK_PRODUTOS_HIST_PRECO
  PRIMARY KEY (NUM_MOVTO) )
GO
-----
--
```

```
CREATE TRIGGER TRG_PRODUTOS_HIST_PRECO ON TB_PRODUTO
FOR UPDATE
AS BEGIN
INSERT INTO PRODUTOS_HIST_PRECO
(ID_PRODUTO, DATA_ALTERAÇÃO, PRECO_ANTIGO, PRECO_NOVO)
SELECT I.ID_PRODUTO, GETDATE(), D.PRECO_VENDA, I.PRECO_VENDA
FROM INSERTED I JOIN DELETED D ON I.ID_PRODUTO = D.ID_PRODUTO

WHERE I.PRECO_VENDA <> D.PRECO_VENDA
END

--- TESTANDO
DELETE PRODUTOS_HIST_PRECO

UPDATE TB_PRODUTO SET PRECO_VENDA = PRECO_VENDA * 1.5
WHERE COD_TIPO = 2
--
SELECT * FROM PRODUTOS_HIST_PRECO

-----

-- 2. Crie um trigger que corrija o estoque (campo QTD_REAL da
tabela TB_PRODUTO) toda vez que um item de pedido for incluído,
alterado ou excluído.
GO
CREATE TRIGGER TRG_ITENSPEDIDO_CORRIGE_ESTOQUE ON TB_ITENSPEDIDO
FOR DELETE, INSERT, UPDATE
AS BEGIN
-- .SE o trigger foi executado por "culpa" de DELETE
-- .Somar em TB_PRODUTO.QTD_REAL a QUANTIDADE do
-- item que foi deletado
IF NOT EXISTS(SELECT * FROM INSERTED)
UPDATE TB_PRODUTO
SET QTD_REAL = P.QTD_REAL + D.QUANTIDADE
FROM TB_PRODUTO P
JOIN DELETED D ON P.ID_PRODUTO = D.ID_PRODUTO
-- .SE o trigger foi executado por "culpa" de INSERT
-- .Subtrair de PRODUTOS.QTD_REAL a QUANTIDADE do
-- item que foi inserido
ELSE IF NOT EXISTS(SELECT * FROM DELETED)
UPDATE TB_PRODUTO
SET QTD_REAL = P.QTD_REAL - I.QUANTIDADE
FROM TB_PRODUTO P
JOIN INSERTED I ON P.ID_PRODUTO = I.ID_PRODUTO
-- .SE o trigger foi executado por "culpa" de UPDATE
-- .Somar em PRODUTOS.QTD_REAL o valor resultante
-- de (DELETED.QUANTIDADE - INSERTED.QUANTIDADE)
ELSE
UPDATE TB_PRODUTO
SET QTD_REAL = P.QTD_REAL + ( D.QUANTIDADE - I.QUANTIDADE )
FROM TB_PRODUTO P
JOIN INSERTED I ON P.ID_PRODUTO = I.ID_PRODUTO
JOIN DELETED D ON P.ID_PRODUTO = D.ID_PRODUTO
END
```



```

-----
-- 3. Crie um trigger de DDL para o banco de dados PEDIDOS
--    que registre na tabela criada a seguir todos os eventos
CREATE, ALTER
--    e DROP executados no nível do banco de dados.

GO
CREATE TABLE TAB_LOG_BANCO
(
    ID
    INT IDENTITY PRIMARY KEY,
    EventType
    VARCHAR(100),
    PostTime
    VARCHAR(50),
    UserName
    VARCHAR(100),
    ObjectType
    VARCHAR(100),
    ObjectName
    VARCHAR(300),
    CommandText
    Text
)

GO
CREATE TRIGGER TRG_LOG_BANCO
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS BEGIN
DECLARE @DATA XML;
-- Recupera todas as informações sobre o motivo da
-- execução do trigger
SET @DATA = EVENTDATA();

INSERT INTO TAB_LOG_BANCO
( EventType, PostTime, UserName, ObjectType, ObjectName,
  CommandText )
VALUES
( @DATA.value(' (/EVENT_INSTANCE/EventType) [1]', 'Varchar(100)'),
  @DATA.value(' (/EVENT_INSTANCE/PostTime) [1]', 'Varchar(100)'),
  @DATA.value(' (/EVENT_INSTANCE/UserName) [1]', 'Varchar(100)'),
  @DATA.value(' (/EVENT_INSTANCE/ObjectType) [1]', 'Varchar(100)'),
  @DATA.value(' (/EVENT_INSTANCE/ObjectName) [1]', 'Varchar(200)'),
  @DATA.value(' (/EVENT_INSTANCE/TSQLCommand/CommandText) [1]',
    'Varchar(8000)') )

END
GO
----- TESTANDO
CREATE TABLE TESTE ( COD INT, NOME VARCHAR(30) )
GO
ALTER TABLE TESTE ADD E_MAIL VARCHAR(100)
GO
DROP TABLE TESTE
GO
--
SELECT * FROM TAB_LOG_BANCO
-----

```

```
-- 4. Crie um trigger de DDL para o banco de dados PEDIDOS
-- que registre na tabela criada a seguir todos os eventos
CREATE,
-- ALTER e DROP executados no nível do banco de dados.
GO
USE MASTER

GO
CREATE TABLE TAB_LOG_SERVER
(
    ID
    INT IDENTITY PRIMARY KEY,
    DatabaseName    VARCHAR(100),
    EventType
    VARCHAR(100),
    PostTime
    VARCHAR(50),
    UserName
    VARCHAR(100),
    ObjectType
    VARCHAR(100),
    ObjectName
    VARCHAR(300),
    CommandText
    VARCHAR(max)
)

GO
USE MASTER

GO
CREATE TRIGGER TRG_LOG_SERVER
    ON ALL SERVER
    FOR CREATE DATABASE, DROP DATABASE, ALTER DATABASE,
    DDL_DATABASE_LEVEL_EVENTS
AS BEGIN
    DECLARE @DATA XML;
    -- Recupera todas as informações sobre o motivo da
    -- execução do trigger
    SET @DATA = EVENTDATA();

    INSERT INTO TAB_LOG_SERVER
    (DatabaseName, EventType, PostTime, UserName, ObjectType,
    ObjectName,
    CommandText )
    VALUES
    ( @DATA.value('(/EVENT_INSTANCE/DatabaseName) [1]',
    'Varchar(100)'),
    @DATA.value('(/EVENT_INSTANCE/EventType) [1]', 'Varchar(100)'),
    @DATA.value('(/EVENT_INSTANCE/PostTime) [1]', 'Varchar(100)'),
    @DATA.value('(/EVENT_INSTANCE/UserName) [1]', 'Varchar(100)'),
    @DATA.value('(/EVENT_INSTANCE/ObjectType) [1]', 'Varchar(100)'),
    @DATA.value('(/EVENT_INSTANCE/ObjectName) [1]', 'Varchar(200)'),
    @DATA.value('(/EVENT_INSTANCE/TSQLCommand/CommandText) [1]',
    'Varchar(8000)') )
END
GO
```

```
-- TESTANDO
CREATE DATABASE TESTE_TRIGGER
GO
USE TESTE_TRIGGER

GO
CREATE TABLE TESTE (C1 INT, C2 VARCHAR(30))
GO
USE MASTER

GO
DROP DATABASE TESTE_TRIGGER
GO
SELECT * FROM TAB_LOG_SERVER
--
-- 5.Crie um trigger de logon para bloqueio de acesso de usuários
-- não administrativos e gravação de auditoria para acesso ao
-- servidor. Utilize a tabela:
GO
CREATE TABLE DBA_AuditLogin(
    idPK int IDENTITY(1,1),
    Data datetime ,
    ProcID int ,
    LoginID varchar(128) ,
    NomeHost varchar(128) ,
    App varchar(128) ,
    SchemaAutenticacao varchar(128) ,
    Protocolo varchar(128) ,
    IPcliente varchar(30) ,
    IPServidor varchar(30) ,
    xmlConectInfo xml
)
GO
CREATE TRIGGER DBA_AuditLogin on all server
for logon
as
insert master.dbo.DBA_AuditLogin
(Data , ProcID, LoginID, NomeHost, App, SchemaAutenticacao, Pro
tocolo,
IPcliente, IPServidor, xmlConectInfo)
select getdate(), @@spid, s.login_name, s.[host_name],
s.program_name, c.auth_scheme, c.net_transport,
c.client_net_address, c.local_net_address, eventdata()
from sys.dm_exec_sessions s join sys.dm_exec_connections c
on s.session_id = c.session_id
where s.session_id = @@spid
GO

-- Teste
select * from DBA_AuditLogin
```

