SQL Server

Readme.rmd

Sergio Pedro R Oliveira

2022-06-27

Contents

1 Objetivo

Estudo dirigido de SQL Server.

2 Referência

Vídeo aulas “O curso completo de Banco de Dados e SQL, sem mistérios” - Udemy.

3 Modulo 24 - Instalação e delimitador GO

3.1 Instalação

3.1.1 Instalar SQL-server

- Versão:
  Versão usada é a express 2019, por ser a versão mais completa gratuita.

- Ubuntu
  https://docs.microsoft.com/pt-br/sql/linux/quickstart-install-connect-ubuntu?view=sql-server-ver15
  Basta seguir o passo a passo do site, ou pesquisar por pesquisar por "SQL Server Ubuntu" no youtube e seguir alguns tutoriais.

- Windows

3.1.2 Instalar Azure Data Studio

- Gerenciador de banco de dados usado para SQL-server, que estou usando no Ubuntu.

- Onde baixar:
  https://docs.microsoft.com/pt-br/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15

3.2 Acessando SQL Server pelo terminal

- Execute o sqlcmd com parâmetros para o nome do SQL Server (-S), o nome de usuário (-U) e a senha (-P). Neste tutorial, você está se conectando localmente, portanto, o nome do servidor é localhost. O nome de usuário é SA (system administrator, equivalente ao root do MySQL) e a senha é a mesma fornecida para a conta SA durante a instalação.

sqlcmd -S localhost -U SA -P '*YourPassword*'

- É possível omitir a senha na linha de comando para receber uma solicitação para inseri-la.

sqlcmd -S localhost -U SA

3.3 Bancos do sistema

- São os bancos de dados do sistema que armazenam os dicionarios de dados.
- Bancos de dados do sistema:
  - **master**
    - É o banco de dados principal do sistema.
    - Todas as informações dos outros bancos de dados criados ficam armazenados nele.
  - **model**
    - São modelos de tabelas e bancos de dados, que ficam armazenados nesse banco de dados.
    - Pode servir de modelo automatico na criação de uma nova tabela ou banco de dados.
  - **msdb**
    - Armazenamento de rotinas.
    - Integrations Services, área de BI (ferramenta de **ETL**).
  - **tempdb**
    - Bancos de dados temporarios, ele é apagado todo vez que fecha e abre o sistema do banco de dados.
    - Muito utilizado para agilizar o teste de aplicações.

3.4 Detalhes básicos do SQL SERVER

3.4.1 Inserindo comentarios

- Um comentário é uma seqüência arbitrária de caracteres começando por dois hífens ("--") e prosseguindo até o fim da linha.

- Como alternativa, podem ser utilizados blocos de comentários no estilo C (/**bloco de comentarios*/). Utilizado para comentar mais de uma linha.

3.4.2 Extensão de arquivo script SQL

- O arquivo com o script SQL é salvo em ".sql".

- As três formas recomendadas de escrever os script's são:

  - **SQL server Management Studio** (**SSMS**)
    É um gerenciador de bancos de dados oferecidos pela microsoft, ótimo para gerenciar e trabalhar com banco de dados e arquivos ".sql".

  - **Azure**
    É um gerenciador de banco de dados e oferece ferramentas para o melhor entendimento e programação de um script ".sql".

  - Num arquivo de texto
    Preferencialmente o programa "**Sublime Text**", pois oferece a opção de escrever e salvar arquivos ".sql" com todas as ferramentas que envolve o processo.

3.4.3 Abrindo uma "nova consulta"

- Para começar a escrever um script no **SSMS** é necessario iniciar uma "nova consulta", abrir uma pagina que serve para escrever os comandos SQL.

- As duas formas de iniciar essa pagina são:

  - Clickar em nova consulta, parte superior da pagina.

  - Atalho **CRTL + N**

3.5 Uso do delimitador GO

- O **SQL Server** funciona da seguinte forma com seus script's:
  - Não precisa do delimitador para compilar o código, *processamento assincrono*.
  - Quando é pedido para compilar todo o script (sem seleções do código e sem uso de delimitador), o **SQL Server** executa o que for mais rapido primeiro, fora de ordem, por conta do *processamento assincrono*.
  - Ao selecionar uma parte do código ele compila apenas aquela parte do código.
  - Usando o delimitador **GO** executar o código por partes.
- Demilitador **GO**
  - O uso do GO ao final de cada instrução serve como delimitador.
  - O **GO** quebra o codigo em pequenos pacotes que são enviados para o servidor executar.
  - Colocando o **GO** no código ao final de cada instrução, o servidor não faz o *processamento assincrono*, assim quebrando o grande pacote que é o script inteiro, em pequenos pacotes para serem executados na ordem de envio.
- Modo de usar:
  - Colocar o **GO** ao final de cada instrução.
  - Inserir o **GO** na linha de baixo a instrução.
  - Sintaxe:
    CREATE DATABASE *nome_database*
    GO
    USE *nome_database*
    GO
    CREATE TABLE *nome_tabela*(
    campo tipo
)
    GO

4 Modulo 25 - Arquitetura do SQL Server

4.1 Acessando arquivos de banco de dados

- Primeiro clickando com botão direito no banco de dados desejado.
  - Propriedades > Arquivos.

4.2 Arquitetura do SQL Server na maquina

- No Ubuntu os dados de arquitetura ficam gravados no caminho:
  '/var/opt/mssql/data'

- No Windows os dados de arquitetura ficam gravados no caminho:
  'C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA'

4.3 Tipos de arquivos

- **MDF** (*master data file*)
  - Armazena dados do sistema (dicionario de dados).
  - Criação automatica pelo sistema.
  - Recomenda-se que use o MDF apenas para dados do sistema (mudança manual).
  - Arquivos **MDF** sempre vão dentro do grupo **PRIMARY**.
- **LDF** (*log data file*)
  - Armazena log's, transações, conjuntos de instruções.
  - Criação automatica pelo sistema.
  - É apagado quando explicitado (**BEGIN**) a transação, ao finalizada com **COMMIT** (confirmando a transação) ou **ROLLBACK** (desfazendo a transação).
- **NDF** (*not master data file*)
  - Não é criado automaticamente pelo sistema (criação manual), diferente dos outros.
  - Utilizado para armazenar dados.
  - Podendo armazenar dados atraves de grupos dados (*GP*), para melhor organizar os dados, assim fazendo a separação dos dados por assunto.
  - Possibilita a separação fisica dos dados em HD.

4.4 TRANSACTION - Transação

- É uma instrução que só executa as instruções dentro dela, no caso (**INSERT**, **UPDATE**, **DELETE**, ...), apenas se todas as instruções sejam concluidas com sucesso.

- Caso alguma instrução dentro dela dê ERRO, tudo é desfeito.

- Muito util para fazer operações de transação financeira entre contas.

  - Exemplo de transação financeira, transferencia de dinheiro entre contas:

    * Subtrair dinheiro de uma conta.

    * Somar dinheiro em outra conta.

- **COMMIT** ou **ROLLBACK**: Comandos que finalizam a transação onde o '**COMMIT**' confirma o conjunto de comandos e o '**ROLLBACK**' desfaz todo o processo executado pelo corpo de comandos caso tenha ocorrindo algum evento contrario ao desejado.

- Sintaxe:
  BEGIN TRANSACTION (ou **BEGIN**)
  UPDATE tabela SET coluna1_a_modificar = expressão1
  WHERE tabela IN (lista_dos_registros_a_modificar)
  UPDATE tabela SET coluna2_a_modificar = expressão2
  WHERE tabela IN (lista_dos_registros_a_modificar)
  COMMIT (ou **ROLLBACK**)

- Observação: Pode usar **BEGIN TRANSACTION** ou apenas **BEGIN**.

4.5 Função ERRO

- No **SQL Server** temos uma função de sistema que faz a endentificação de um erro dentro de uma transação chamada de '**@@ERROR**' função essa que por padrão recebe o valor 0 (zero) caso não ocorra nem um erro , no caso de algum erro ela assume o valor 1 (um).

- Uso da função '**@@ERROR**' dentro de um **IF**, para determinar uma *transação* (**TRANSACTION**) pode se mostrar uma boa solução.

- Sintaxe:
  BEGIN TRANSACTION
  UPDATE FROM *tabela*
  SET *campo_1* = 10.000
  WHERE *campo_1* < 50
  IF @@ERROR = 0
  COMMIT
  ELSE
  ROLLBACK
  END

4.6 Criando Grupos de arquivos

4.6.1 Criando grupos de dados apartir de um novo banco de dados

- Clickar com o botão direito no “Banco de dados”, na aba “Pesquisador de objetos”.
- Opção “novo banco de dados”.
- Na aba “Geral” pode dar nome para o banco de dados criado.
- Na aba “Grupos de arquivos” é onde é criado os grupos de arquivos.
  - O botão “adicionar grupo de arquivos” cria um novo grupo de arquivo.
  - Podemos nomear esse novo grupo, a boa pratica indica sempre começar com o prefixo “GA_” (abreviação de “grupo de arquivo”), depois o nome grupo.
  - A opção “PADRÃO” indica que todos os arquivos não especificados o grupo, vai cair nesse grupo. Não deixar ele marcado em **PRIMARY**, pois esse grupo é para pertencer apenas os **MDF** - *dicionarios do sistema*.
- Criar arquivos de banco de dados **NDF**:
  - Na aba “Geral”, clickar no botão “adicionar” para criar um novo “arquivo de banco de dados”.
  - Em “Nome” nomear esse novo arquivo pela categoria (setor, . . .).
  - Em “Nome do Arquivo” repetir o nome + o sufixo “.ndf”.
  - Em “Grupo de Arquivos” direcionar para o grupo desejado (criado anteriormente).
  - Em “Aumento Automático/Tamanho Máximo” podemos determinar o crescimento do banco de dados cada vez que ele atinge o limite, e determinar um tamanho máximo para o banco de dados (quando ele atinge o tamanho máximo, o banco de dados para).
- “OK” para confirmar as criações no final.

4.6.2 Criando grupos de dados em bancos de dados existentes

- Clickar com o botão direito no Banco de dados desejado, na aba “Pesquisador de objetos”.
- Na opção “Propriedades”.
- Na aba “Grupos de arquivos” é onde é criado os grupos de arquivos.
  - O botão “adicionar grupo de arquivos” cria um novo grupo de arquivo.
  - Podemos nomear esse novo grupo, a boa pratica indica sempre começar com o prefixo “GA_” (abreviação de “grupo de arquivo”), depois o nome grupo.
  - A opção “PADRÃO” indica que todos os arquivos não especificados o grupo, vai cair nesse grupo. Não deixar ele marcado em **PRIMARY**, pois esse grupo é para pertencer apenas os **MDF** - *dicionarios do sistema*.
- Criar arquivos de banco de dados **NDF**:
  - Na aba “Arquivos”, clickar no botão “adicionar” para criar um novo “arquivo de banco de dados”.

  - Em “Nome” nomear esse novo arquivo pela categoria (setor, . . .).

  - Em “Nome do Arquivo” repetir o nome + o sufixo “.ndf”.

  - Em “Grupo de Arquivos” direcionar para o grupo desejado (criado anteriormente).

  - Em “Aumento Automático/Tamanho Máximo” podemos determinar o crescimento do banco de dados cada vez que ele atinge o limite, e determinar um tamanho máximo para o banco de dados (quando ele atinge o tamanho máximo, o banco de dados para).

- “OK” para confirmar as criações no final.

4.7 Direcionando dados de tabela para grupos de arquivos (NDF)

- Clickar com o botão direito na tabela desejada.

- Na opção”design”.

- Na aba de “Propriedades”.

  - Dentro de “Identidade”, em “Nome” podemos alterar o nome da tabela.

  - Dentro de “Designer de tabela”, dentro de “Especificação de Espaço de Dados Regular”, em “Nome do Esquema de Partição ou Grupo de Arquivos” podemos selecionar um grupo de arquivos criado anteriormente para enviar a tabela.

  - Dentro de “Designer de tabela”, em “Grupo de Arquivos de Texto/Imagem” caso o banco de dados salve arquivos do tipo texto (.doc ou .odt) e imagem (pdf, jpeg ou .png) pode direcionar para ser salvo dentro de um grupo de arquivos criado anteriormente, podendo ser um grupo diferente do tópico anterior.

5 Modulo 26 PARTE 1 - Comandos básicos, CONSTRAINTS e descrição de tabelas

5.1 Comandos básicos

- **CREATE DATABASE**
  - Criação de banco de dados.
  - Sintaxe:
    CRIATE DATABASE *nome_database*
    GO
- **DROP DATABASE**
  - Apaga um banco de dados e tudo esta contido dentro.
  - Sintaxe:
    DROP DATABASE *nome_database*
    GO
- **USE**
  - Conectando a um banco de dados.
  - Sintaxe:
    USE *nome_database*
    GO
- **CREATE TABLE**
  - Criação de banco de dados.
  - Sintaxe:
    CREATE TABLE *nome_tabela*(
    coluna1 tipo regras,
    ...
)
    GO
- **DROP TABLE**
  - Apaga uma tabela.
  - Sintaxe:
    DROP TABLE *nome_da_tabela*
    GO
- **ALTER TABLE**
  - Adiciona regras (**CONSTRAINT**) a tabelas.
  - É uma boa prática o uso de **ALTER TABLE** para normalizar o nome salvo das regras no sistema. Facilita a pesquisa posteriormente.

- Sintaxe:
  ALTER TABLE *nome_tabela*
  ADD CONSTRAINT *nome_regra*
  [*regra a ser implementada*]
  GO

- Observação: o *nome_regra* (nome da regra) é o nome que fica salvo no **dicionario de dados** (sistema).

- **INSERT**

  - Inserindo novos registros numa tabela.

  - No **SQL Server**, diferente do **MySQL**, nas colunas **PK** (**PRIMARY KEY**) com **IDENTITY** não precisa passar nenhum valor (nem **NULL**), o sistema já entende que vai haver preenchimento e incrementação automatica dos valores.

  - Os valores que vão ser inseridos do *tipo String* (*char*, *varchar*, ...) ou *DATE* devem ser passados entre aspas simples('*valor*').

  - Sintaxe:
    INSERT INTO *nome_tabela* **VALUES**
    (*valor_coluna1*, *valor_coluna2*,*valor_coluna3*,*valor_coluna4*, ...), ...
    (*valor_coluna1*, *valor_coluna2*,*valor_coluna3*,*valor_coluna4*, ...) **GO**

- **DELETE**

  - Apaga os registro de uma tabela.

  - Quando usado em conjunto com WHERE, apaga apenas os registros determinados por uma condição.

  - Sintaxe:
    DELETE FROM *nome_tabela*
    WHERE *condição*
    GO

- **UPDATE**

  - Altera os registros de uma tabela.

  - Quando usado em conjunto com WHERE, altera apenas determinados registros definidos por uma condição

  - Sintaxe:
    UPDATE *tabela_nome*
    SET
    coluna_1 = valor_1,
    coluna_2 = valor_2,
    ...
    WHERE *condição*
    GO

5.2 Regras/Restrições - CONSTRAINTS

- Uma boa prática é criar as **CONSTRAINTS** por fora da tabela, para ter o controle dos nomes das *restrições* que ficaram salvas no dicionario de dados (sistema).
  - Normalmente com uso de **ALTER TABLE**.

5.2.1 IDENTITY

- Exerce a mesma função que **AUTO_INCREMENT** no **MySQL**, incrementar automaticamente a coluna determinada.
- Trás de novo a opção de argumentos "**IDENTITY**(*1º_nº*,*2º_nº*)":
  - O primeiro número é onde começa.
  - O segundo numero é quanto incrementa a cada vez.
- É possivel suprimir os argumentos, onde "**IDENTITY** = **IDENTITY**(1,1)".
- No **SQL Server**, diferente do **MySQL**, quando feito o **INSERT** de dados no campo onde tem **IDENTITY** não precisa entrar com valor nenhum (nem **NULL**), basta ignorar este campo, o **SQL Server** entende automaticamente que é para preencher ele.
- Sintaxe:
  CREATE TABLE *nome_tabela*(
  coluna1 int **PRIMARY KEY IDENTITY**(100,10),
  ...
)
  GO
- Observação: A *coluna1* começa em 100 e recebe o incremento de 10 em 10 a cada novo dado.

5.2.2 CONSTRAINTS

- **PRIMARY KEY** (**PK**)
  - Toda tabela necessita de pelo menos um campo que identifique todo registro como sendo único (é o que chamamos de "*Chave Primaria*" ou "**Primary Key**").
  - Sintaxe:
    ALTER TABLE *nome_tabela*
    ADD CONSTRAINT PK__nome_tabela_
    PRIMARY KEY (*coluna*)
    GO
- **FOREIGN KEY** (**FK**)
  - Cria uma relação entre duas tabelas, atraves de uma *chave estrangeira* na tabela.
  - Sintaxe:
    ALTER TABLE *nome_tabela*
    ADD CONSTRAINT FK__tabela-recebe_tabelare-referenciada_
    FOREIGN KEY (*coluna_FK*)
    REFERENCES *tabela_referenciada*(*coluna_referenciada*)
    GO
  - Observações:
    * A *tabela-recebe* é a tabela que vai receber a regra **FK**.
    * A *coluna_FK* é a coluna especifica na tabela, que recebe a regra, que vai servir para fazer a ligação (relação).
    * As *tabela_referenciada* e *coluna_referenciada* é respectivamente referente a tabela e coluna que serão referenciadas pela ligação (relação) **FK**, ou seja, são as que não recebem a regra.
- **CHECK** (**CK**)
  - Verifica (checa) se determinada coluna os valores dos dados são iguais aos especificados em uma lista.
  - Um substituto no **SQL Server** ao **ENUM** no **MySQL**.
  - Pode ser usado tanto na criação de tabela (**CREATE TABLE**) quanto na alteração de tabelas (**ALTER TABLE**).
  - Sintaxe:
    ALTER TABLE *nome_tabela*
    ADD CONSTRAINT CK__nome_tabela_
    CHECK (*nome_coluna* **IN** ('*valor1*','*valor2*'))
    GO
- **UNIQUE** (**UQ**)
  - A *restrição* **UNIQUE** garante que todos os valores em uma coluna sejam diferentes.
  - Sintaxe:
    ALTER TABLE *nome_tabela*

ADD CONSTRAINT UQ__*nome_tabela*
UNIQUE (*coluna_recebe_UQ*)
GO

5.3 Comandos de descrição tabelas - SP_

- No **SQL Server** a descriação de uma tabela é atraves de **PROCEDURES** (funções).
- **PROCEDURES** já criadas e armazenadas no sistema, "**STORAGE PROCEDURES**" (**SP**).

5.3.1 SP_COLUMNS

- **SP_COLUMNS** é igual a **DESC**, no **MySQL**.
- Faz uma descrição da tabela:
  - Nome das colunas
  - *Tipo* de cada coluna
  - *Regras* em cada coluna
  - ...
- Sintaxe:
  SP_COLUMNS *nome_tabela*
  GO

5.3.2 SP_HELP

- **SP_HELP** é igual ao **SHOW CREATE TABLE**, no **MySQL**.
- Faz uma descrição mais detalha da tabela que **SP_COLUMNS**:
  - Quem criou a tabela.
  - Permissões.
  - Datas importantes (criação e modificação).
  - ...
- Sintaxe:
  SP_HELP *nome_tabela*
  GO

6 Modulo 26 PARTE 2 - Funções, Projeções, Seleções e Junções

6.1 Funções

6.1.1 Funções usuais

- **ISNULL**()
  - Trata os valores **nulos**, na coluna especificada, na consulta.
  - Equivalente do **IFNULL**() do **MySQL**.
  - Dentro do **ISNULL**(), os argumentos são:
    - *Nome da coluna* a ser avaliada.
    - *Texto* se o valor for **nulo**.
  - Dentro do **ISNULL**() usar aspas simples (' ').
  - Sintaxe:
    SELECT
    A.*coluna1*, **ISNULL**(**T**.*coluna2*, 'SEM') **AS** "*alias1*",
    ISNULL(**T**.*coluna3*, 'SEM_NUMERO') **AS** "*alias2*", **FROM** *tabela1* **A**
    LEFT JOIN *tabela3* **T**
    ON A.*coluna***PK** = **T**.*coluna***FK**
    GO

- **PRINT** '*mensagem*'
  - Imprime na tela uma mensagem, colocado entre aspas simples (' ').
  - Pode ser usado dentro de **TRIGGERS** e **PROCEDURES** para passar alguma informação importante ao usuário.
  - Sintaxe com exemplo:
    CREATE TRIGGER *nome_da_trigger*
    ON *DBO.tabela*
    FOR UPDATE
    AS
    ...
    PRINT '*TRIGGER EXECUTADO COM SUCESSO*'
    GO

6.1.2 Funções de auditoria

- **SUSER_NAME**()
  - Função que retorna o usuario logado no banco de dados no momento.
  - Útil para usar dentro de **TRIGGERS** para salvar o usuario reponsavel por alguma alteração numa tabela (audutoria).
  - Sintaxe:
    SELECT SUSER_NAME()
    GO
- **GETDATE**()
  - Pega a data no sistema (data e horário).
  - Util para usar dentro de **TRIGGERS** para salvar a data e horario de alguma alteração numa tabela (auditoria).
  - Formato:
    "aaaa-mm-dd hh:mm:ss.mmm"
  - Sintaxe:
    SELECT GETDATE()
    GO

6.1.3 Funções de datas

- **GETDATE**()
  - Pega a data no sistema (data e horário).
  - Formato:
    "aaaa-mm-dd hh:mm:ss.mmm"
- **DATEDIFF**()
  - Calcula a diferença entre duas datas.
    - Retorna um valor inteiro (*INT*), dia (*DAY*), ou mês (*MONTH*), ou ano (*YEAR*), ou dia da semana (*WEEKDAY*).
    - Sintaxe:
      DATEDIFF(*intervalo*, *data_inicio*, *data_termino*)
      - *intervalo*, indica a função com que parametro estou trabalhando (*day*, *month*, *year*, *weekday*)
  - Outras funções podem se usadas em conjunto, como parametros, para ajudar a fazer os cálculos.
    - Comummente usada em conjunto com **GETDATE**() para cálcular idade.
    - Sintaxe:
      DATEDIFF(*intevalo*,*data_inicio*, **GETDATE**())
      - a função **GETDATE**(), data atual, entra no lugar do parametro *data de termino*, cálculando assim a idade atual.
- **DATENAME**()
  - Retorna o nome da parte da data em questão. (ex.: nome do mês)
  - Retorna uma *string*.
  - Sintaxe:
    DATENAME(*intervalo*, *data*)
    - *intervalo*, indica a função com que parametro estou trabalhando (*day*, *month*, *year*, *weekday*)
- **DATEPART**()
  - Função parecida com **DATENAME**(), porem retorna um inteiro (*INT*).
  - Retorna uma parte da data.
  - Sintaxe:
    DATEPART(*intervalo*, *data*)
    - *intervalo*, indica a função com que parametro estou trabalhando (*day*, *month*, *year*, *weekday*)

- **DATEADD**()
  - Retorna uma data somada a outra data.
  - Sintaxe:
    DATEADD (*intervalo*, *incremento_INT*, *data*)
    * *intervalo*, indica a função com que parametro estou trabalhando (*day*, *month*, *year*, *weekday*)
    * *incremento_INT*, com base no parametro informado pelo *intervalo*, quanto deve ser somado (valor *INT*).
    * *data*, data a ser incrementada.
- **DAY**()
  - Recebe como argumento uma data, formato do sistema.
  - Retorna o dia (**DAY**) contido na data.
  - Sintaxe:
    DAY(*data*)
- **MONTH**()
  - Recebe como argumento uma data, formato do sistema.
  - Retorna o mês (**MONTH**) contido na data.
  - Sintaxe:
    MONTH(*data*)
- **YEAR**()
  - Recebe como argumento uma data, formato do sistema.
  - Retorna o ano (**YEAR**) contido na data.
  - Sintaxe:
    YEAR(*data*)

6.2 Projeção, seleção e Junção - SELECT, WHERE e JOIN

Principais passos de uma consulta.

6.2.1 PROJEÇÃO

- O primeiro passo de uma consulta é montar o que quer ver na tela - **SELECT**.

- É tudo que você quer ver na tela.

- Sintaxe comentada:
  SELECT *coluna_1* (PROJEÇÃO)
  FROM *tabela* (ORIGEM)
  GO
  ou
  SELECT 2+2 **AS** *alias* (PROJEÇÃO)
  GO

- É possivel mostrar mais de uma consulta ao mesmo tempo.

  - Sintaxe comentada:
    SELECT * **FROM** *tabela_1* (PROJEÇÃO 1) **SELECT** * **FROM** *tabela_2* (PROJEÇÃO 2)
    GO

Obs.: o que esta entre parênteses é comentario.

6.2.2 SELEÇÃO

- O segundo passo de uma consulta é a seleção dos dados de uma consulta - **WHERE**.
- É filtrar.
- Trazer um subconjunto do conjunto total de registros de uma tabela.
- Sintaxe comentada:
  SELECT *coluna_1*, *coluna_2*, *coluna_3* (PROJEÇÃO)
  FROM *tabela* (ORIGEM)
  WHERE *critero* = *valor_do_criterio* (SELEÇÃO)
  GO
  Obs.: o que esta entre parênteses é comentario.

6.2.3 JUNÇÃO

6.2.3.1 Junção forma errada - gambiarra

- Usa seleção como uma forma de juntar tabelas.
- Como conseguencia:
  - Uso de operadores lógicos para mais criterios de seleção - **WHERE**.
  - Ineficiencia na pesquisa, maior custo computacional.
- Sintaxe comentada:
  SELECT *coluna1_tab1*, *coluna2_tab1*, *coluna1_tab2* (PROJEÇÃO)
  FROM *tabela1*, *tabela2* (ORIGENS)
  WHERE *chave_primaria_tab1* = *chave_estrangeira_tab2*(JUNÇÃO)
  GO
  ou
  SELECT *coluna1_tab1*, *coluna2_tab1*, *coluna1_tab2* (PROJEÇÃO)
  FROM *tabela1*, *tabela2* (ORIGENS)
  WHERE *chave_primaria_tab1* = *chave_estrangeira_tab2* (JUNÇÃO)
  AND *criterio* = *valor*(SELEÇÃO com operador lógico)
  GO
  Obs.: o que esta entre parênteses é comentario.

6.2.3.2 Junção forma certa - JOIN

- Junção **JOIN**, junta duas ou mais tabelas apartir das colunas de *chaves primarias* e *chaves estrangeiras*.
- Admite seleção - **WHERE** - sem maiores custos computacionais.

1. **INNER**

- Exclui os registros sem par (orfans) na outra tabela - **INNER**.
- Consulta com duas tabelas.
  - Sintaxe comentada:
    SELECT *coluna1_tab1*, *coluna2_tab1*, *coluna1_tab2* (PROJEÇÃO)
    FROM *tabela1* (ORIGEM)
    INNER JOIN *tabela2* (JUNÇÃO)
    ON *chave_primaria_tab1* = *chave_estrangeira_tab2*
    WHERE *criterio* = *valor* (SELEÇÃO)
    GO

2. **LEFT**

- Mostra ate os registros sem par (nulos) - **LEFT**.
  - Comum usar a função *ISNULL*() para tratar os valores nulos.

- Consulta com duas tabelas.

  - Sintaxe comentada:
    SELECT *coluna1_tab1*, *coluna2_tab1*, *coluna1_tab2* (PROJEÇÃO)
    FROM *tabela1* (ORIGEM)
    LEFT JOIN *tabela2* (JUNÇÃO)
    ON *chave_primaria_tab1* = *chave_estrangeira_tab2*
    WHERE *criterio* = *valor* (SELEÇÃO)
    GO

6.2.3.3 Cláusulas ambíguas e Ponteiramento

- Consulta com mais de duas tabelas.

  - Pode apresentar colunas/campos com o mesmo nome, de tabelas diferentes. Caso comum das *chaves estrangeiras* (**FK**).

  - Indicar de onde vem cada coluna atraves de "*nome_da_tabela.nome_da_coluna*".

  - Sintaxe comentada:
    SELECT
    tabela1.coluna1_tab1,
    tabela1.coluna2_tab1,
    tabela2.coluna1_tab2,
    tabela3.coluna1_tab3 (PROJEÇÃO)
    FROM *tabela1* (ORIGEM)
    LEFT JOIN *tabela2* (JUNÇÃO)
    ON *tabela1.chave_primaria_tab1* = *tabela2.chave_estrangeira_tab2*
    INNER JOIN *tabela3* (JUNÇÃO)
    ON *tabela1.chave_primaria_tab1* = *tabela3.chave_estrangeira_tab3*
    WHERE *criterio* = *valor* (SELEÇÃO)
    GO
    Obs.: o que esta entre parênteses é comentario.

- Ponteiramento (alias para tabelas)

  - Melhora a performance da consulta.

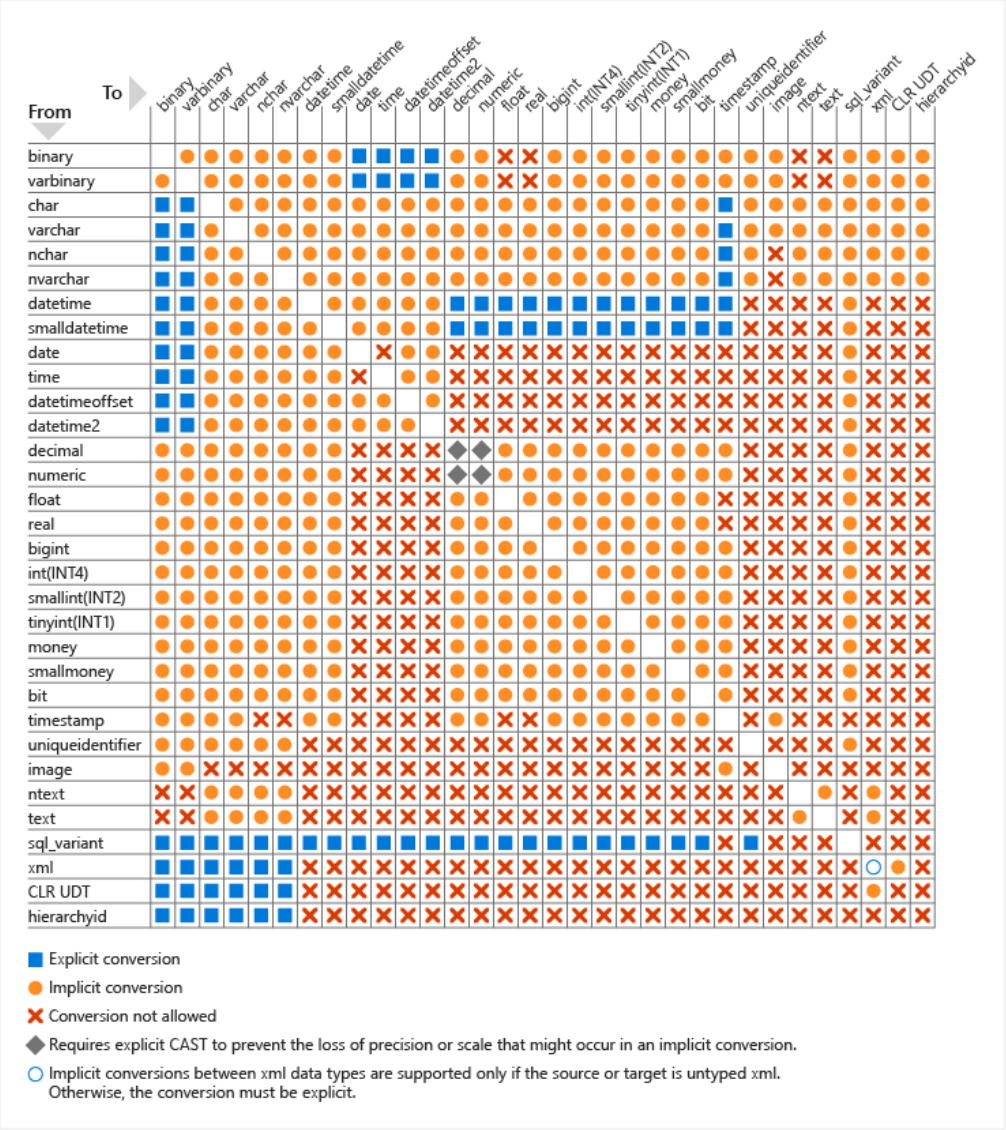
  - Sintaxe comentada:
    SELECT
    A.coluna1_tab1,
    A.coluna2_tab1,
    B.coluna1_tab2,
    C.coluna1_tab3
    FROM *tabela1 A* (PONTEIRAMENTO DA TABELA 1)
    LEFT JOIN *tabela2 B* (PONTEIRAMENTO DA TABELA 2)
    ON *A.chave_primaria_tab1* = *B.chave_estrangeira_tab2*
    INNER JOIN *tabela3 C* (PONTEIRAMENTO DA TABELA 3)
    ON *A.chave_primaria_tab1* = *C.chave_estrangeira_tab3*
    WHERE *criterio* = *valor*
    GO

7 Modulo 26 PARTE 3 - Conversão de tipo de dados

7.1 Conversão de dados automatica pelo sistema

- Conversões de *TIPO* que o **SQL Serve** faz automaticamente pelo sistema.
- Existe um direcionamento em que o sistema costuma forçar de **STRING** para **INT**, nunca o contrario de maneira automatica.
- O simbolo '+', alem de operador matemático, funciona como concatenador.
  - Sintaxe:
    SELECT '1' + '1'
    GO
    (Retorno '11')

7.2 Tabela de conversões de dados automatico pelo sistema



- *Conversão implicita.*
  - São conversões automaticas pelo sistema.
- *Conversão explicita.*
  - São conversões por meio de funções. (Ex.: **CAST**())

7.3 Funções de conversão

- **CAST**()
  - A função **CAST**() converte um valor (de qualquer tipo) em um tipo de dados especificado.
  - O tipo de dados para converter a expressão. Pode ser um dos seguintes:
    * **BIGINT**
    * **INT**
    * **SMALLINT**
    * **TINYINT**
    * **BIT**
    * **DECIMAL**
    * **NUMERIC**
    * **MONEY**
    * **SMALLMONEY**
    * **FLOAT**
    * **REAL**
    * **DATETIME**
    * **SMALLDATETIME**
    * **CHAR**
    * **VARCHAR**
    * **TEXT**
    * **NCHAR**
    * **NVARCHAR**
    * **NTEXT**
    * **BINARY**
    * **VARBINARY**
    * **IMAGE**
  - *Expressões* do tipo **STRING** devem entrar entre aspas simples (' ').
  - Sintaxe:
    SELECT
    CAST(*expressão* **AS** *TIPO_especificado*)
    GO
- **CHARINDEX**()

- Retorna um numero inteiro de acordo com a posição de determinada caracter num **VARCHAR**.
  - As posições no VARCHAR começam a ser contadas a partir da posição 1.
  - O retorno 0, é caso não tenha achado nenhum caracter procurado.
- Os argumento do CHARINDEX(*o que procurar?*, *onde procurar?*, *a partir de tal posição?*)
  - *O que procurar?*
    O caracter que deve ser encontrado.
  - *onde procurar?*
    O **VARCHAR** que deve ser percorrido procurando o caracter.
  - *a partir de tal posição?*
    A partir de qual posição a busca deve começar. As posições do VARCHAR começam a ser contadas a partir da posição 1.
    Pode omitir esse ultimo argumento, a função entenderá como começando da posição 1 (a inicial).
- Sintaxe:
  SELECT
  CHARINDEX('*caracter*',*sting*, *numero_da_posição_inicial_procura*) **AS** '*alias*'
  FROM *tabela*
  GO

8 Modulo 26 PARTE 4 - Importação de arquivo de dados

8.1 Aspacetos importantes da importação de Arquivos

- Além da função de importação de arquivo (**BULK INSERT**), é necessario antes, montar uma estrutura preparada para receber os dados do arquivo (criação de **BANCO DE DADOS** e **TABELAS** para recerber os dados).

  - Sintaxe:
    CREATE DATABASE *nome_database*
    GO
    CREATE TABLE *tabela*(
    campo1 tipo regra,
    campo2 tipo regra,
    ...
)
    GO

- Outro aspecto importante é como esta organizado os dados no arquivo importados.

  - A organização dos dados, no arquivo, interfere diretamente no processo de importação do arquivo.

  - Partes em branco, dentro do arquivo, provavelmente resultarão em registros *nulos* (**NULL**), quando não em erro.

  - É importante para importação conhecer os caracteres de comando da tabela **ASCII**, são necessarios como argumentos da função **BULK INSERT**.

```
##   Nome_na_ASC                Descricao Representacao_em_C
## 1         nul      null byte/byte nulo                \\0*
## 2         bel     bell character/apito                  \a
## 3          bs                 backspace                  \b
## 4          ht horizontal tab/tabulação                  \t
## 5          np   formfeed/fim da pagina                  \f
## 6          nl       newline/nova linha                  \n
## 7          cr          carriage return                  \r
## 8          vt             vertical tab                  \v
```

* É uma barra invertida só.

8.2 Função de importação de arquivos BULK INSERT

- A função **BULK INSERT** serve para importação dos dados, de um arquivo qualquer, para dentro do **SQL Server**.

- Antes de qualquer coisa, deve ser criado anteriormente um estrutura para receber esses dados no **SQL Server**, ou seja, a criação do banco de dados e da tabela que vai receber esses dados.

- Sintaxe:
  BULK INSERT *tabela_importação*
  FROM '*caminho*'
  WITH(
  FIRSTROW = 2,
  DATAFILETYPE = '*char*',
  FIELDTERMINATOR = '\t',
  ROWTERMINATOR = '\n'
)
  GO

- Argumentos do **BULK INSERT**:

  - *tabela_importação*
    A tabela a qual os dados importatos serão direcionados.

  - *caminho*
    O caminho no sistema do computador onde o arquivo esta locado. O caminho é colocado entre aspas simples, pois é uma **string**.
    Ex.: 'C:/SPB_Data/github_bkp/SQL-Server/Arquivos_importacao/CONTAS.txt'

- Argumentos do **WITH**:

  - **FIRSTROW**
    É um numero inteiro que indica a partir de qual linha começa os dados, começando na linha 1. Normalmente exclui-se o cabeçalho, começando assim a partir da linha 2, ou seja o valor 2.

  - **DATAFILETYPE**
    Tipo do arquivo do arquivo (dados).

  - **FIELDTERMINATOR**
    Determina onde termina cada dado.
    Usar o caracter de comando da tabela ASCII. Ou o que seja que faça a separação dos dados no arquivo, muito comum o uso do ";".
    O caracter de comando é entre aspas simples ' '.

  - **ROWTERMINATOR**
    Determina onde termina cada registro/linha.
    Usar o caracter de comando da tabela ASCII.
    O caracter de comando é entre aspas simples ' '.

9 Modulo 26 PARTE 5 - Técnica de “flag-ar” coluna (SELECT)

- Técnica usada para criar, numa consulta (**SELECT**), uma espécie de tabela verdade com os resultados possiveis de uma coluna.

- Essa técnica se baseia no uso da função **CHARINDEX**() para achar determinados resultados e a partir dele criar novas colunas, na consulta (**SELECT**).

- Sendo cada nova coluna, um dos resultados possiveis.

- E os resultados são valores de “0” ou “1”, em cada coluna nova.

  - Resultado “0”, na nova coluna, significa que a consulta daque dado, no registro correspondente, não corresponde aquele resultado.

  - Resultado “1”, na nova coluna, significa que a consulta daque dado, no registro correspondente, corresponde aquele resultado.

- Outra possibilidade de continuação da técnica é a partir dessas novas colunas, criar um multiplicador para interagir com os dados e transformar ele.

- Sintaxe exemplo, técnica em duas partes:

  - Parte 1:
    SELECT
    CONTA,
    VALOR,
    DEB_CRED,
    CHARINDEX(‘*D*’,*DEB_CRED*) **AS** *DEBITO*,
    CHARINDEX(‘*C*’,*DEB_CRED*) **AS** *CREDITO*,
    ((**CHARINDEX**(‘*C*’,*DEB_CRED*)*2)-1) **AS** *MULTIPLICADOR*
    FROM *LANCAMENTO_CONTABIL*
    GO

    * Cria duas colunas DEBITO e CREDITO.

    * Na nova coluna *CREDITO*, se na coluna *DEB_CRED* o valor é credito (‘C’) a coluna leva “1”, senão leva “0”.

    * Na nova coluna *DEBITO*, se na coluna *DEB_CRED* o valor é debito (‘D’) a coluna leva “1”, senão leva “0”.

    * Por último, cria uma coluna *MULTIPLICADOR*, onde se é credito leva “1”, se é debito leva “-1”.

  - Parte 2:
    SELECT
    CONTA,
    SUM((*VALOR**(**CHARINDEX**(‘*C*’,*DEB_CRED*)*2)-1)) **AS** *SALDO*
    FROM *LANCAMENTO_CONTABIL*
    GROUP BY *CONTA*
    ORDER BY *CONTA*
    GO

    * Cria uma coluna *SALDO* que é a soma dos créditos e debitos.

* Sendo debito negativo e crédito positivo.

* agrupando os dados pela coluna *CONTA* e ordenando pela coluna *CONTA*.

10 Modulo 27 - TRIGGER (Gatilho) DML (Data Manipulation Language)

- A **TRIGGER** é um gatilho de programação, que dispara toda vez que algo predeterminado acontecer.
- Exemplos de gatilhos disparadores de uma **TRIGGER** são:
  - **INSERT**
  - **UPDATE**
  - **DELETE**
- Após os gatilhos (**TRIGGERS**) disparados, são executados blocos de programação.
- Os comandos a seguir são de *criação* (**CREATE TRIGGER**), *modificação* (**ALTER TRIGGER**) e *pagar* (**DROP TRIGGER**) **TRIGGERS**.

10.1 Conceitos Préliminares - Argumentos temporais (INSERTED/DELETED) e Declaração de variáveis (DECLARE)

10.1.1 Argumentos temporais - INSERTED e DELETED

- São áreas do sistema que guardam dados.
- Comparando com MySQL:
  - **INSERTED** = **AFTER** (depois)
  - **DELETED** = **BEFORE** (antes)

10.1.1.1 INSERTED

- A área **INSERTED** guarda os dados novos inseridos, ou seja, ao usar o **INSERTED** pega os novos dados ("depois" de) inseridos na tabela.
- Os **DML** que se valem desse artificio, normalmente, são:
  - **INSERT**
    Usado normalmente para guardar os novos dados inseridos na tabela, guarda os dados "depois" de inseridos.
  - **UPDATE**
    Usado para guardar os novos dados modificados na tabela, guarda os dados "depois" de modificados.

10.1.1.2 DELETED

- A área **DELETED** guarda os dados antigos, ou seja, ao usar o **DELETED** pega os antigos dados ("antes" de) modificados na tabela.

- Os **DML** que se valem desse artificio, normalmente, são:
  - **DELETE**
    Usado normalmente para guardar os antigos dados, “antes” de deletados da tabela.
  - **UPDATE**
    Usado para guardar os antigos dados modificados na tabela, guarda os dados “antes” de modifica-los.

10.1.2 Declaração de variáveis - DECLARE

10.2 CREATE TRIGGER

10.3 ALTER TRIGGER

10.4 DROP TRIGGER

10.5 Boas Práticas

10.5.1 Blocos de Programação

- São os blocos de programação (instruções **SQL**) dentro do **TRIGGER**.

10.5.1.1 Primeiro Bloco - declaração de variaveis (DECLARE)

- Espaço usado para declarar todas as variáveis que serão usadas dentro do **TRIGGER**.

- Sintaxe:
  DECLARE *@nome_variavel1 tipo*
  DECLARE *@nome_variavel2 tipo*
  ...

10.5.1.2 Segundo Bloco - Atribuindo valor em variáveis via SELECT

- Insere nas variáveis valores vindos de tabelas, que são inseridos pelo comando "**SELECT**".

- Sintaxe:
  SELECT *@nome_variavel1 = nome_coluna/campo* **FROM INSERTED** (ou **DELETED**)
  ...

10.5.1.3 Terceiro Bloco - Atribuindo valor em variáveis via funções

- Insere nas variáveis valores vindos de funções ou literais, que são inseridos pelo comando "**SET**".

- Sintaxe:
  SET *@nome_variavel1 = função_qualquer*()
  SET *@nome_variavel2 =* '*texto*'
  ...
  Obs.: *texto* inserido dentro de variável atraves de aspas simples.

10.5.1.4 Quarto Bloco - INSERT dados na tabela do TRIGGER

- Inserindo dados das variáveis na tabela de armazenamento do **TRIGGER**, tabela normalmente de auditoria dos dados.

- Sintaxe:
  INSERT INTO *tabela_auditoria*
  (*coluna1*, *coluna2*, *coluna3*, *coluna4*, *coluna5*, *coluna6*, *coluna7*)
  VALUES
  (*@nome_variavel1*, *@nome_variavel2*, *@nome_variavel3*, *@nome_variavel4*, *@nome_variavel5*, *@nome_variavel6*, *@nome_variavel7*)

10.5.1.5 Mensagem ao usuário - PRINT

- A função **PRINT** imprime uma mensagem na tela.

- Este é o espaço no bloco de programação para deixar alguma mensagem para o usuário ao final da aplicação da **TRIGGER**.

- A mensagem a ser passada pela função **PRINT**, fica entre aspas simples(' '), aspas duplas retorna **ERRO**.

- Sintaxe:
PRINT '*TRIGGER EXECUTADO COM SUCESSO*'

11 Categorias de comandos

11.1 DML - *Data Manipulation Language* (Linguagem de Manipulação de Dados)

É um conjunto de instruções usada nas consultas e modificações dos dados armazenados nas tabelas do banco de dados.

- **SELECT**
  - Recupera linhas do banco de dados e permite a seleção de uma ou várias linhas ou colunas de uma ou várias tabelas.
  - Projeção do que quer ter de visualização na tela.
  - Sintexa:
    SELECT *
    FROM *tabela*
    GO
- **INSERT**
  - Adiciona registros numa tabela.
  - Sintaxe:
    INSERT INTO *nome_da_tabela*
    VALUES
    (*valor_na_coluna_1_registro1*, *valor_na_coluna_2_registro1*,...),
    (*valor_na_coluna_1_registro2*, *valor_na_coluna_2_registro2*,...),
    ...
    GO
- **UPDATE**
  - Altera os dados de um ou mais registros em uma tabela.
  - Sintaxe:
    UPDATE *tabela* **SET** *coluna_a_atualizar* = *valor_atualizado*
    WHERE *condição* = *valor*
    GO
- **DELETE**
  - Remove um ou mais registros de uma tabela.
  - Sintaxe:
    DELETE FROM *tabela*
    WHERE *criterio_do_que_se_quer_deletar* = *valor*
    GO
- **BULK INSERT***
  - Importa um arquivo de dados em uma tabela ou exibição do banco de dados em um formato especificado pelo usuário.

– Sintaxe:
BULK INSERT *tabela_importação*
FROM '*caminho*'
WITH(
FIRSTROW = 2,
DATAFILETYPE = '*char*',
FIELDTERMINATOR = '\t',
ROWTERMINATOR = '\n'
)
GO
* Mais detalhes no "*Modulo 26 PARTE 4 - Importação de arquivo de dados*".

11.2 DDL - *Data Definition Language* (Linguagem de definição de dados)

É um conjunto de instruções usado para criar e modificar as estruturas dos objetos armazenados no banco de dados.

- **CREATE**
  Utilizada para construir um novo banco de dados, tabela, índice ou consulta armazenada.
  - **DATABESE**
    - Criação de banco de dados.
    - Sintaxe:
      CREATE DATABASE *nome_banco_de_dados*
      GO
  - **TABLE**
    - Criação de tabela.
    - Sintaxe:
      CREATE TABLE *nome_tabela* (
      coluna1 tipo regra retrições,
      coluna2 tipo regra retrições,
      ...
)
      GO
- **DROP**
  Remove um banco de dados, tabela, índice ou visão existente.
  - **DATABESE**
    - Remove banco de dados.
    - Sintaxe:
      DROP DATABASE *nome_do_banco_de_dados*
      GO
  - **TABLE**
    - Remove tabela.
    - Sintaxe:
      DROP TABLE *nome_da_tabela*
      GO
- **ALTER**
  - Modifica um objeto existente do banco de dados.

– É possível incluir, eliminar e alterar colunas.

– Para alterar uma tabela existente, é necessario que os registros existentes já sejam compativeis com a alteração.

  * **ALTER COLUMN**

    · Altera o *tipo* e *regras* de uma coluna/campo.

    · Sintaxe:
    ALTER TABLE *nome_tabela*
    ALTER COLUMN *nome_coluna modificação_tipo*
    GO

  * **ADD**

    · Adiciona chaves (primaria ou estrangeira) a uma coluna.

    · Não é possivel adicionar "*auto_increment*".

    · Sintaxe:
    ALTER TABLE *tabela*
    ADD PRIMARY KEY(*coluna*)
    GO
    ou
    ALTER TABLE *tabela*
    ADD FOREING KEY(*coluna_da_tabela*)
    REFERENCES (*coluna_chave_primaria_de_outra_tabela*)
    GO

    · O comando **ADD** funciona para adicionar nova coluna.
    Sintaxe:
    ALTER TABLE *tabela*
    ADD *nova_coluna tipo*
    GO

  * **DROP COLUMN**

    · Deleta uma determinada coluna de uma tabela.

    · Sintaxe:
    ALTER TABLE [nome_database.]*nome_tabela*
    DROP COLUMN *nome_coluna*
    GO

  * **ALTER DATABASE**

    · Alterar nome de uma database.

    · Sintaxe:
    ALTER DATABASE *nome_database*

MODIFY NAME = *novo_nome_database*
GO

- **SP_RENAME**

  - Mudar nome da tabela e/ou coluna.

  - Sintaxe:

    * Mudar nome da *tabela*:
      SP_RENAME '*NomeTabelaAntigo*', '*NomeTabelaNovo*'
      GO
      ou

    * Mudar nome da *coluna*:
      SP_RENAME '*NomeTabela.NomeColunaAntigo*', '*NovoNomeColuna*', '*COLUMN*'
      GO

- **TRUNCATE**

  - Esvazia imediatamente todo o conteúdo de uma tabela ou objeto que contenha dados.

  - É muito mais rápido que um comando DELETE, pois, ao contrário deste, não armazena os dados sendo removidos no log de transações. Por esse motivo, em vários SGBDs é um comando não-transacional e irrecuperável, não sendo possível desfazê-lo com **ROLLBACK**.

  - Sintaxe:
    TRUNCATE TABLE *nome_tabela*
    GO

11.3 DCL - *Data Control Language* (Linguagem de Controle de Dados)

São usados para controle de acesso e gerenciamento de permissões para usuários em no banco de dados. Com eles, pode facilmente permitir ou negar algumas ações para usuários nas tabelas ou registros (segurança de nível de linha).

11.3.1 Login

- **CREATE LOGIN**
  - A instrução **CREATE LOGIN** cria uma identidade usada para se conectar a uma instância do SQL Server.
  - O login é então mapeado para um usuário do banco de dados (portanto, antes de criar um usuário no SQL Server, você deve primeiro criar um login).
  - Quatro tipos de logons que você pode criar no SQL Server:
    - Pode-se criar um login usando a autenticação do **Windows**.
      CREATE LOGIN *test_domain/techonthenet*
      FROM WINDOWS
      GO
      Obs.: Este exemplo de **CREATE LOGIN** criaria um novo Login chamado *test_domain/techonthenet* que usa a autenticação do Windows.
    - Pode-se criar um login usando a autenticação do **SQL Server**.
      CREATE LOGIN *nome_login*
      WITH PASSWORD = '*senha*'
      GO
    - Pode-se criar um Login a partir de um **certificate**.
      CREATE LOGIN *nome_login*
      FROM CERTIFICATE *nome_certificação*
      GO
    - Pode-se criar um Login a partir de um **asymmetric key**.
      CREATE LOGIN *nome_login*
      FROM ASYMMETRIC KEY *nome_asym_key*
      GO
- **ALTER LOGIN**
  - A instrução **ALTER LOGIN** modifica uma identidade usada para se conectar a uma instância do **SQL Server**.
  - Você pode usar a instrução **ALTER LOGIN** para:
    - Alterar uma senha
      ALTER LOGIN *nome_login*
      WITH PASSWORD = '*nova_senha*'
      GO
    - Forçar uma alteração de senha
      ALTER LOGIN *nome_login*

WITH PASSWORD = '*nova_senha*' **MUST_CHANGE**, **CHECK_EXPIRATION** = **ON**
GO
Obs.: Força a alteração da senha após o primeiro logon usando a instrução **ALTER LOGIN** no **SQL Server** (Transact-SQL).

* Desabilitar um login
ALTER LOGIN *nome_login* **DISABLE**
GO

* Habilitar um login
ALTER LOGIN *nome_login* **ENABLE**
GO

* Desbloquear um login
ALTER LOGIN *nome_login*
WITH PASSWORD = '*senha*'
UNLOCK
GO

* Renomear um login
ALTER LOGIN *nome_login*
WITH NAME = *novo_nome*
GO

* Etc.

- **DROP LOGIN**

  - A instrução **DROP LOGIN** é usada para remover uma identidade (ou seja: Login) usada para conectar a uma instância do **SQL Server**.

  - Sintaxe:
  DROP LOGIN *nome_login*
  GO

- *Find logins* no **SQL Server**

  - No SQL Server, há uma exibição de catálogo (ou seja: exibição do sistema) chamada sys.sql_logins.

  - Você pode executar uma consulta nessa exibição do sistema que retorna todos os logins que foram criados no **SQL Server**, bem como informações sobre esses logins.

  - Sintaxe:
  SELECT *
  FROM *master.sys.sql_logins*
  GO

11.3.2 USER - Usuário

- **CREATE USER**
  - A instrução CREATE USER cria um usuário de banco de dados para fazer logon no SQL Server.
  - Um usuário de banco de dados é mapeado para um **LOGIN**, que é uma identidade usada para se conectar a uma instância do SQL Server.
  - Comando para criação de usuários.
  - Sintaxe:
    CREATE USER *user_nome* **FOR LOGIN** *login_nome*
    GO
    - *user_nome*
      O nome do usuário do banco de dados que você deseja criar.
    - *login_nome*
      O **Login** usado para se conectar à instância do SQL Server.
- Listar usuários:
  - No SQL Server, há uma exibição do sistema chamada *sys.database_principals*. Você pode executar uma consulta nessa exibição do sistema que retorna todos os usuários que foram criados no SQL Server, bem como informações sobre esses usuários.
  - Sintaxe:
    SELECT *
    FROM *master.sys.database_principals*
    GO
- Mostrar usuário conectado atual:
  - Função **SUSER_NAME**().
  - Função que retorna o usuario logado no banco de dados no momento.
  - Útil para usar dentro de **TRIGGERS** para salvar o usuario reponsavel por alguma alteração numa tabela (audutoria).
  - Sintaxe:
    SELECT SUSER_NAME()
    GO
- Removendo usuários:
  - A instrução **DROP USER** é usada para remover um usuário do banco de dados SQL Server.
  - Sintaxe:
    DROP USER *user_nome*
    GO

11.3.3 Permissões

- **GRANT**
  - Permitir que usuários especificados realizem tarefas especificadas.
  - Tambem permite gerenciar permissão para realizar tarefas especificas em database e/ou tabelas especificas.
  - Sintaxe:
    GRANT *lista_de_privilégios* **ON** [*nome_database.*]*nome_tabela* **TO** *nome_usuário*
    GO
    ou para dar permissão de root:
    GRANT ALL ON * . * **TO** *nome_usuário*
    GO
  - Revisar as permissões atuais de um usuário:
    * Para obter informações sobre as permissões dos usuários ou funções, você pode consultar a exibição do catálogo do sistema sys.database_principals.
    * Esta será uma lista enorme. Você também pode personalizar essa consulta para obter as permissões associadas a um usuário ou função adicionando a condição WHERE.
    * Sintaxe:
      SELECT *pri.name* **As** *Username*,
      pri.type_desc **AS** [*User Type*],
      permit.permission_name **AS** [*Permission*],
      permit.state_desc **AS** [*Permission State*],
      permit.class_desc **Class**,
      object_name(*permit.major_id*) **AS** [*Object Name*]
      FROM *sys.database_principals pri*
      LEFT JOIN *sys.database_permissions permit*
      ON *permit.grantee_principal_id* = *pri.principal_id*
      [**WHERE** *name* = '*nome_usuário*']
      GO

- **REVOKE**
  - Cancela/revoga permissões previamente concedidas.
  - Sintaxe:
    REVOKE *lista_de_privilégios* **ON** [*nome_database.*]*nome_tabela* **FROM** *nome_usuário*
    GO
    Obs.: Note que no **REVOKE** é usado **FROM** e no **GRANT** é usado **TO**.

- **DENY**
  - O comando é usado para impedir explicitamente que um usuário receba uma permissão específica.
  - A instrução **DENY** impede que os usuários executem ações. Isso significa que a instrução remove as permissões existentes das contas de usuário ou impede que os usuários obtenham permissões por meio de sua associação de grupo/função que pode ser concedida no futuro.

- Todas as opções da instrução **DENY** têm o mesmo significado lógico que as opções com o mesmo nome na instrução **GRANT**.

- **DENY** tem uma opção adicional, *CASCADE*, que especifica que as permissões serão negadas ao usuário A e a qualquer outro usuário para quem o usuário A passou essa permissão. (Se a opção CASCADE não for especificada na instrução DENY e a permissão de objeto correspondente tiver sido concedida com **WITH GRANT OPTION**, um erro será retornado).

- Sintaxe:
  DENY *lista_privilegios* **ON** *objeto::database.tabela* **TO** *usuário*
  GO

- Privilégios que podem ser CONCEDIDOS à ou REVOCADOS de um usuário:

  - **ALL** — ALL não concede todas as permissões para a tabela. Em vez disso, ele concede as permissões ANSI-92 que são **SELECT**, **INSERT**, **UPDATE**, **DELETE** e **REFERENCES**.

  - **CREATE** — permite criar novas tabelas ou bancos de dados.

  - **SELECT** - permite usar o comando SELECT para ler os bancos de dados.

  - **DROP** — permite deletar tabelas ou bancos de dados.

  - **DELETE** — permite excluir linhas de tabelas.

  - **INSERT** — permite inserir linhas em tabelas.

  - **UPDATE** — permite atualizar linhas de tabelas.

  - **REFERENCES** - Capacidade de criar uma restrição que se refere à tabela.

  - **ALTER** - Capacidade de executar instruções **ALTER TABLE** para alterar a definição da tabela.

  - **GRANT OPTION** — permite conceder ou remover privilégios de outros usuários.

11.4 TCL - *Tool Command Language* (Linguagem de Comandos de Ferramentas)

São usados para gerenciar as mudanças feitas por instruções DML. Ele permite que as declarações a serem agrupadas em transações lógicas.

- **BEGIN TRANSACTION**
  - O comando garante que diversas instruções sejam executadas, porem se alguma for mal sucedida todas falham.
  - É possivel avaliar o processo de implementação das instruções e seus resultados e caso necessario regredir ao estado anterior as instruções ou confirmar sua implementação.
  - Principais instruções que são comuns de serem usadas na transação são as **DML** (**INSERT**, **UPDATE** e **DELETE**).
  - Sintaxe:
    BEGIN TRANSACTION (ou apenas, **BEGIN**)
- **BACKROLL**
  - Regressão para o estado anterior ao inicio da transação (**BEGIN TRANSACTION**).
  - Sintaxe:
    BACKROLL
    GO
- **COMMIT**
  - Confirmação de que as instruções da transação (**BEGIN TRANSACTION**) podem ser implementadas sem problemas.
  - Sintaxe:
    COMMIT
    GO

12 Observações

12.1 Problemas para fazer *login* o SSMS

- Caso o **SSMS** não identifique o usuário "sa" e senha como deveria, seguir os seguintes passos:
  - Desabilitar temporariamente o antivirus do computados.
  - Desabilitar o "**firewall**" do computados.
    "Painel de Controle\Sistema e Segurança\Windows Defender Firewall\Personalizar Configurações"
  - Abrir o *instalador* de **SQL Server** e pedir para "**Reparar**".
  - Ao final da reparação, abrir o **SSMS** novamente e fazer o *login*.

12.2 Abreviações do nome de restrições (CONSTRAINTS) no dicionario de dados - sistema (boas práticas)

- Padronização do nome das restrições salvas no sistema.
- Abreviações do nome das restrições (**CONSTRAINTS**), para salvar no sistema por meio do **ALTER TABLE**.
  - '**PK**' é abreviação de "**PRIMARY KEY**"
  - '**FK**' é abreviação de "**FOERIGN**"
  - '**UQ**' é abreviação de "**UNIQUE**"
  - '**CK**' é abreviação de "**CHECK**"

12.3 Formato da data no sistema

"aaaa-mm-dd hh:mm:ss.mmm"
(ano-mês-dia hora:minuto:segundos.milisegundos)

13 Andamento dos Estudos

13.1 Assunto em andamento

Atualmente estou estudando Módulo 27 - AULA 107 e 108.