SQL Server

Readme.rmd

Sergio Pedro R Oliveira

2022-06-04

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 especificiados 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 especificiados 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

- **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
- **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**

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

- **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
- **GETDATE**()
  - Pega a data no sistema (data e horário).
  - Formato:
    "aaaa-mm-dd hh:mm:ss.mmm"

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)
  ou
  SELECT 2+2 **AS** *alias*; (PROJEÇÃO)
  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)
  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)
  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)
  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)

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)

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)
    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*;

7 Observações

7.1 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*.

7.2 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**"

8 Andamento dos Estudos

8.1 Assunto em andamento

Atualmente estou estudando Módulo 26 - AULA 100.