

# Banco de Dados – MS SQL SERVER

Prof. Washington Almeida, MSc, ISF 27002  
@profwashington.almeida - Instagram

# INTRODUÇÃO

- O **SQL Server (MS SQL)** é um Sistema Gerenciador de Banco de Dados Relacional (SGBD) desenvolvido pela **Sybase** em parceria com a **Microsoft**.
- A parceria entre a **Sybase** e a **Microsoft** durou até 1994, com o lançamento do **Windows NT** e desde então a **Microsoft** assumiu o suporte e manutenção do produto.
- Existem diversas edições do SQL Server destinadas a diferentes públicos e diferentes cargas de trabalho.
- Suas linguagens de consulta primárias são **Transact-SQL (T-SQL)** e **ANSI SQL**.

# INTRODUÇÃO

- Permite a **criação de tabelas relacionadas**, evitando a necessidade de armazenar dados redundantes em vários locais dentro de um banco de dados.
- Fornece **integridade referencial** e outras restrições de integridade para manter a precisão dos dados.
- Suporta **transações**.
- Está alinhado aos princípios de **atomicidade, consistência, isolamento e durabilidade**.



# INTRODUÇÃO

- O SQL Server permite bases de dados com tabelas e manipular seus respectivos dados, dentre as operações que podemos realizar com as tabelas podemos citar aquelas conhecidas como **CRUD**.
- **CREATE**: criar ou incluir novos dados.
- **READ**: ler ou consultar dados.
- **UPDATE**: atualizar ou alterar os dados.
- **DELETE**: apagar ou excluir os dados.



# EDIÇÕES

- Atualmente o SQL Server está disponível em **cinco edições**:
- **Enterprise**: usado em ambientes de larga escala e missão crítica, fornece segurança avançada, análises avançadas, machine learning, etc.
- **Standard**: adequado para aplicativos mais intermediários e data marts, inclui relatórios e análises básicas.
- **WEB**: opção de baixo custo total de propriedade para provedores web, fornece recursos de escalabilidade, acessibilidade e capacidade de gerenciamento de propriedades Web de pequena a grande escala.
- **Developer**: semelhante a edição Enterprise para ambientes que não são de produção (homologação).
- **Express**: para aplicações de pequena escala e livre para uso.

# COMPONENTES-CHAVE

- **Database Engine:** responsável por lidar com o armazenamento, transações rápidas, processamento e proteção de dados.
- **SQL Server:** é o serviço responsável por iniciar, parar, pausar e continuar a instância do SQL Server.
- **SQL Server Agent:** desempenha o papel de agendador de tarefas, pode ser acionado por qualquer evento ou conforme necessário.
- **SQL Server Browser:** responsável por ficar ouvindo as solicitações recebidas e se conectar à instância do SQL Server requisitada.
- **SQL Server Full-Text Search:** permite que o usuário possa executar consultas com apenas texto em caracteres de dados de tabelas SQL.

# COMPONENTES-CHAVE

- **SQL VSS Writer:** permite que seja realizado backup e restauração de arquivos de dados quando o SQL Server não estiver em execução.
- **SQL Server Analysis Services (SSAS):** responsável por oferecer recursos de análise de dados, mineração de dados e machine learning.
- **SQL Server Reporting Services (SSRS):** responsável por fornecer recursos de relatório e capacidade de tomada de decisão.
- **SQL Server Integration Services (SSIS):** responsável por fornecer recursos de extração, transformação e carregamento de diferentes tipos de dados de uma fonte para outra.

# INSTÂNCIAS

- O **SQL Server** permite que sejam **executados vários serviços de uma só vez**, cada um com seu login, portas e banco de dados separados.
- São divididos em dois tipos de instâncias:
  - **Primary Instances** (instâncias primárias)
  - **Named Instances** (instâncias nomeadas)
- As **instâncias primárias** são acessadas utilizando o **nome do servidor ou o endereço IP**.
- As **instâncias nomeadas** são acessadas adicionando uma **barra invertida e o nome da instância**.
- Apesar de poder possuir várias instâncias, é **necessário que apenas uma seja a default** (padrão), enquanto as demais precisam ser instâncias nomeadas.



# INSTÂNCIAS

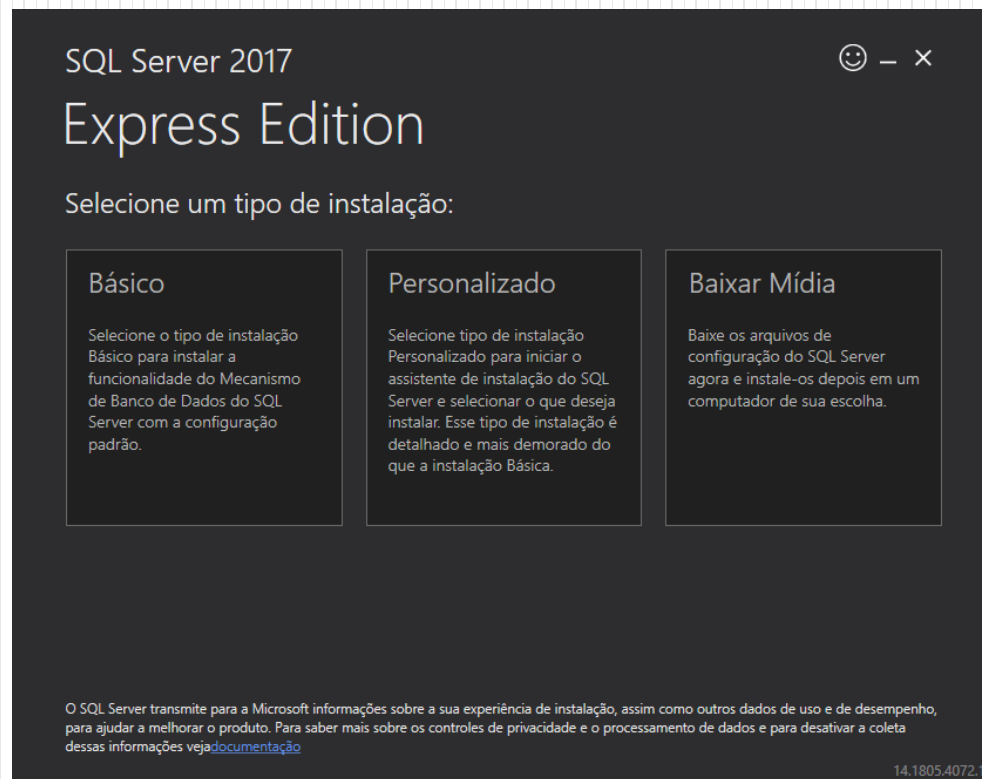
- O **SQL Server** permite que sejam **executados vários serviços de uma só vez**, cada um com seu login, portas e banco de dados separados.
- São divididos em dois tipos de instâncias:
  - **Primary Instances** (instâncias primárias)
  - **Named Instances** (instâncias nomeadas)
- As **instâncias primárias** são acessadas utilizando o **nome do servidor ou o endereço IP**.
- As **instâncias nomeadas** são acessadas adicionando uma **barra invertida e o nome da instância**.
- Apesar de poder possuir várias instâncias, é **necessário que apenas uma seja a default** (padrão), enquanto as demais precisam ser instâncias nomeadas.

# INSTÂNCIAS

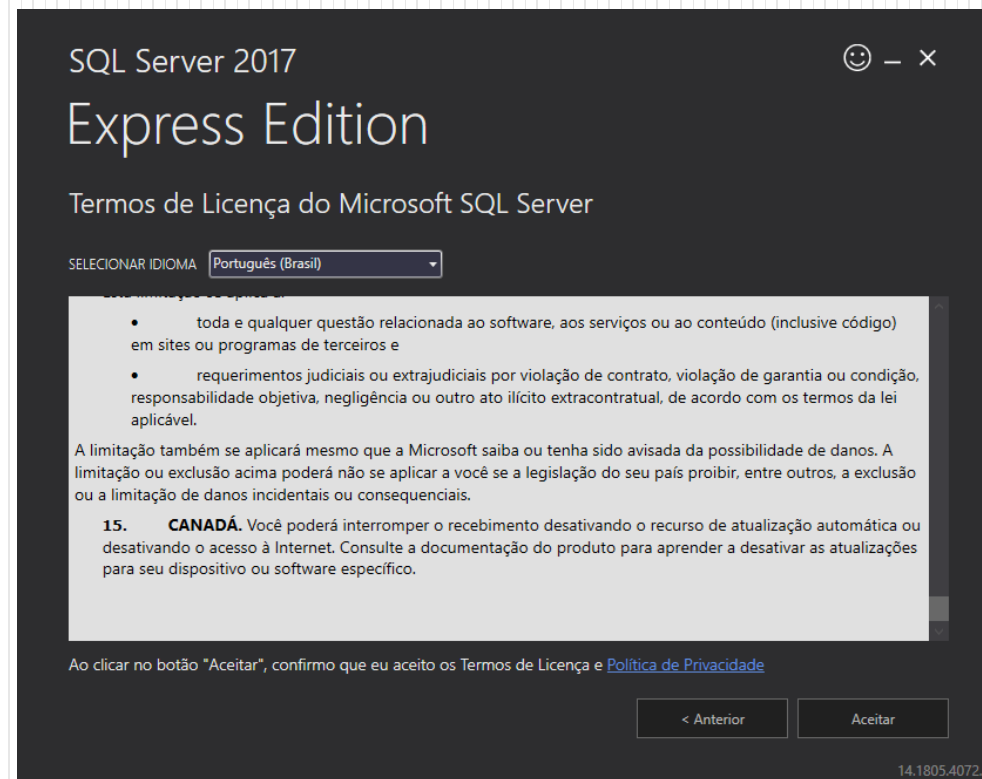
- Segue algumas **vantagens de utilizar instâncias** no SQL Server:
  - É possível **realizar a instalação de diferentes versões** em uma única máquina.
  - Ajuda a **reduzir custos**.
  - Facilita a **manutenção dos ambientes de desenvolvimento, produção e testes** separadamente.
  - Ajuda a **reduzir problemas temporários** no banco de dados.
  - **Separar privilégios** de segurança.
  - Manter um **servidor stand-by** (reserva).

# INSTALAÇÃO

- Para realizar uma instalação básica do SQL Server (neste caso, será utilizado o SQL Server Express por ser gratuito) observe as imagens a seguir:



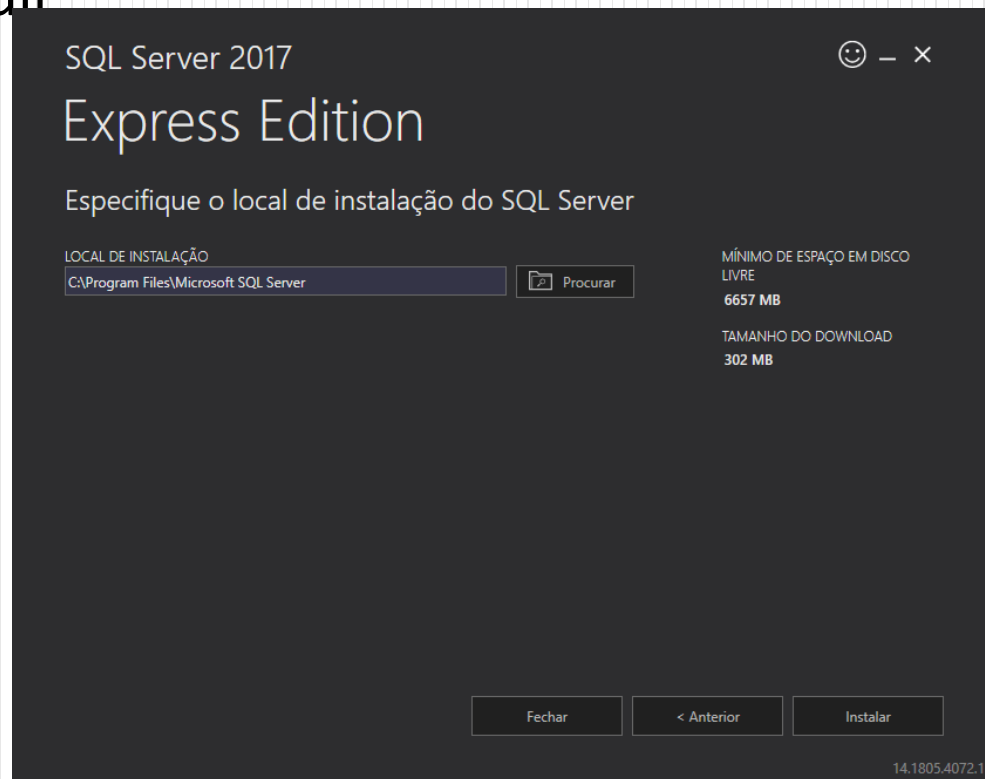
Nesta tela inicial, clique no tipo de instalação básica



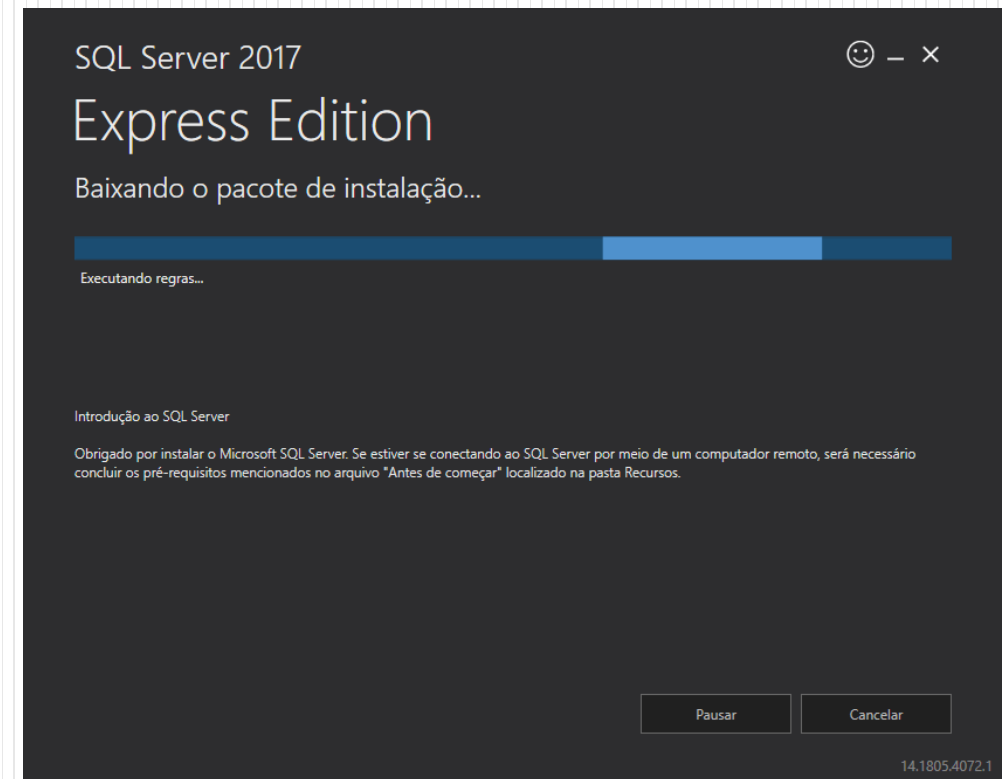
Na próxima tela, aceite os termos de licença

# INSTALAÇÃO

- Seguindo com os procedimentos de instalação do SQL Server Express, observe as imagens a seguir:



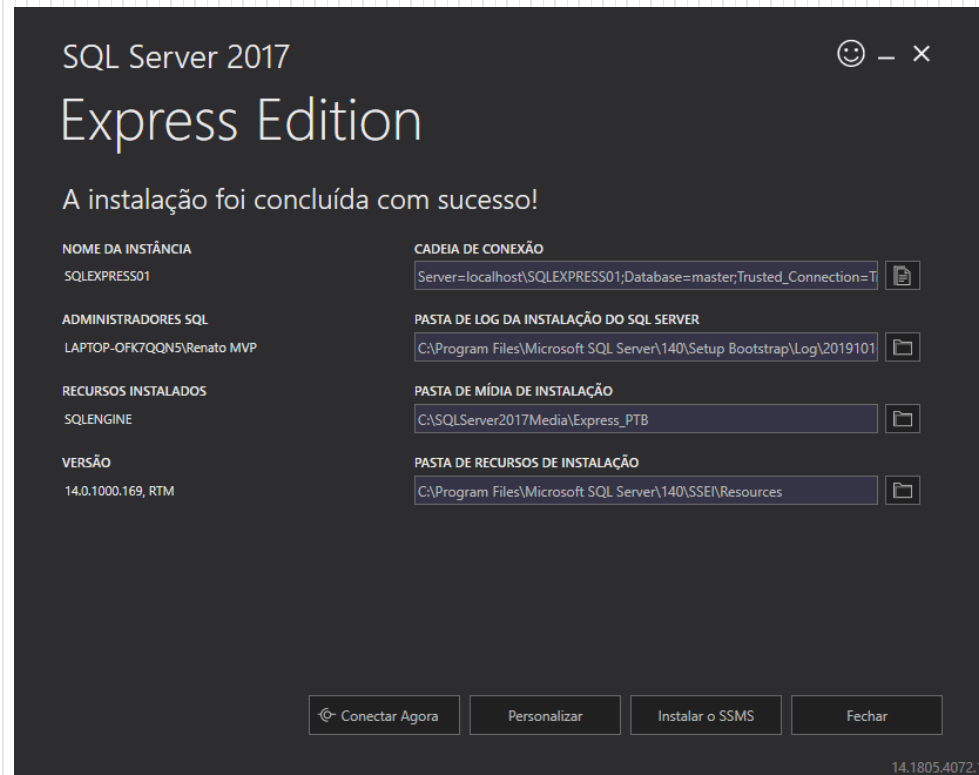
Nesta tela inicial, é possível alterar o local de instalação, ou manter o padrão



Na próxima tela, o processo de instalação começou e podemos acompanhar seu progresso

# INSTALAÇÃO

- Ao término da instalação temos um resumo com diversas informações a respeito da instalação do SQL Server, observe a imagem a seguir:



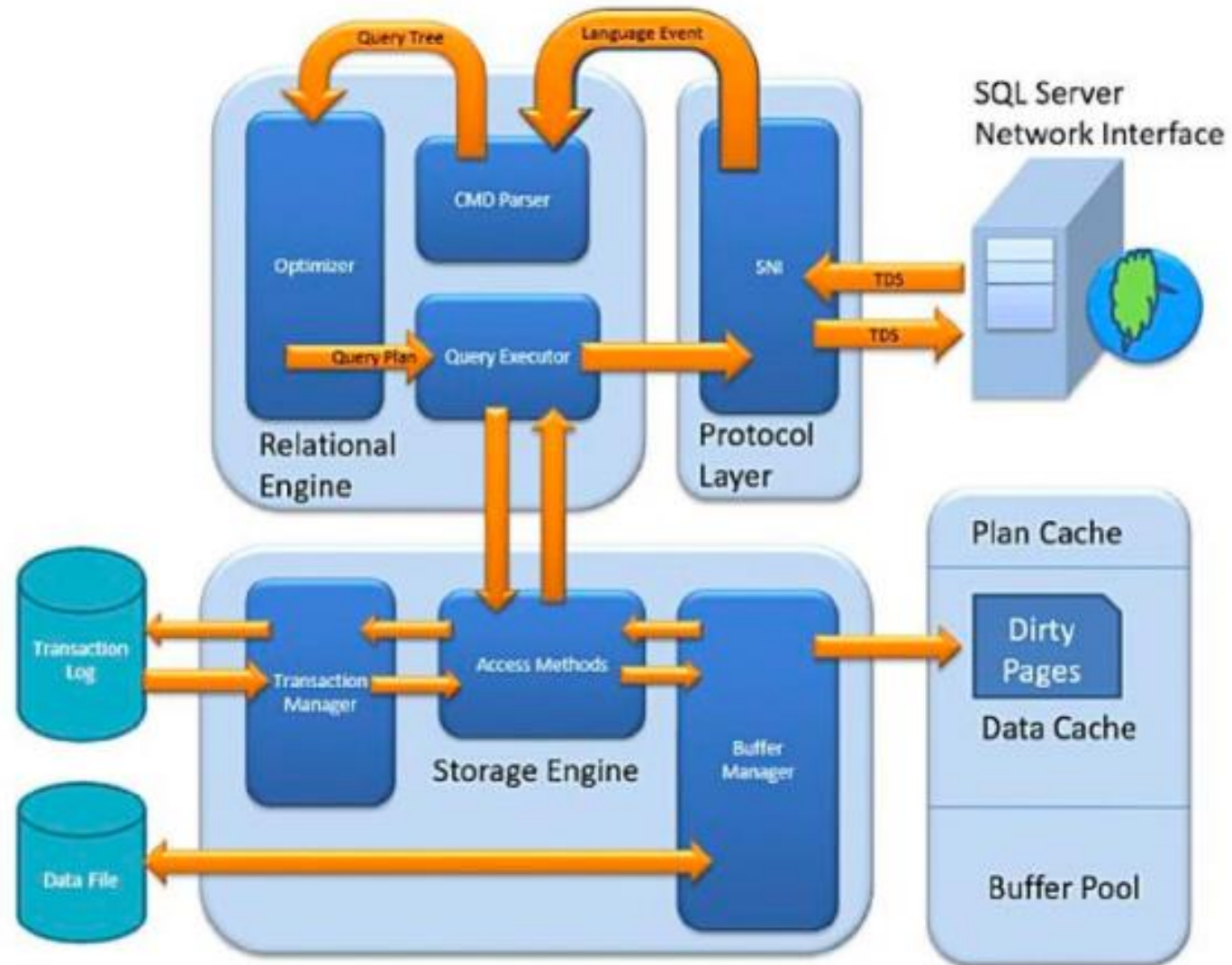
# INSTALAÇÃO

- **Cadeia de conexão** (ou `ConnectionString`): que pode ser utilizada no desenvolvimento de aplicações .NET, basta clicar no botão copiar e colar a cadeia de conexão no código.
- **Conectar agora**: ao clicar neste botão, o prompt de comando abre e automaticamente executa a ferramenta de linha de comando do **SQL Server**, conectando ao servidor.
- **Instalar o SSMS**: este botão vai realizar a instalação do **SQL Server Management Studio** para ser utilizado com o **SQL Server** sem a linha de comando, uma interface gráfica de gerenciamento
- **Nome da instância**: para cada instalação de **SQL Server** há uma instância, é possível ter diversas instâncias de **SQL Server** que podem ser diferentes umas das outras.

# ARQUITETURA

- **SQL Server** é uma arquitetura **Cliente-Servidor**.
- O processo do **SQL Server** começa o aplicativo cliente enviando uma solicitação.
- O **SQL Server** **aceita, processa e responde a solicitação** com os dados processados.
- O **SQL Server** possui três componentes principais em sua arquitetura;
  - **Protocol Layer**
  - **Relational Engine**
  - **Storage Engine**

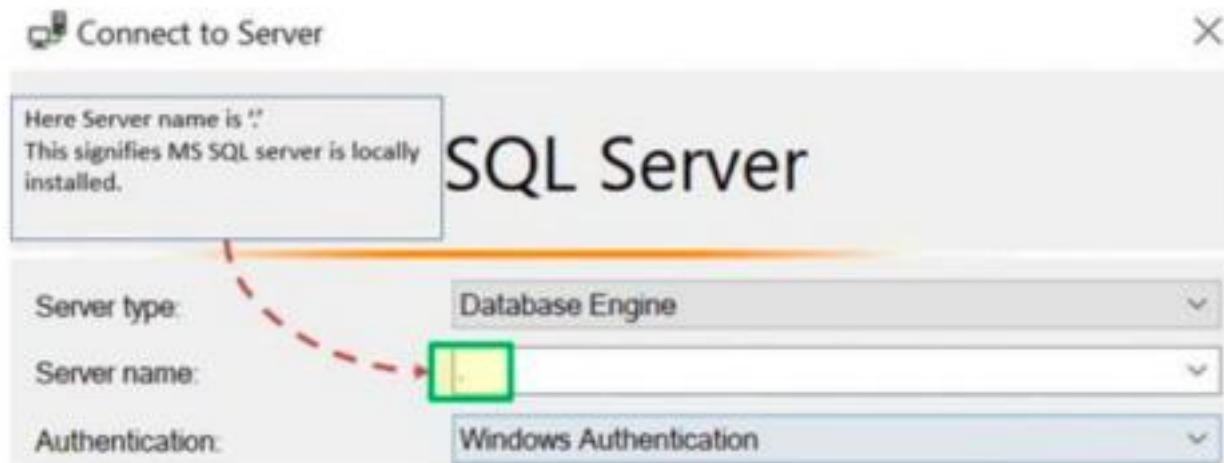
# ARQUITETURA





# ARQUITETURA - Protocol Layer

- **SQL Server Protocol Layer** suporta **3 tipos** de arquitetura Cliente-Servidor.
- **Shared Memory**
  - **SQL Server** fornece um protocolo de **Shared Memory** (memória compartilhada). O **cliente** e o **SQL Server** são **executados no mesmo servidor**, ambos podem se comunicar usando **Shared Memory**.
  - Para realizar a conexão através do Management Studio usando **Shared Memory**, basta usar umas das seguintes opções: **“.”**, **“localhost”**, **“127.0.0.1”** e **“Machine\Instance”**.



# ARQUITETURA - Protocol Layer

## • TCP/IP

- **SQL Server** fornece a capacidade de **interagir usando o protocolo TCP/IP**, quando o **cliente** e o **SQL Server** são remotos entre si e **instalados em servidores/estações separadas**.
- **SQL Server**, por padrão, utiliza a porta **TCP 1433**.
- Para realizar a conexão através do Management Studio usando **TCP/IP**, basta usar a opção **“Machine\Instance”**.



# ARQUITETURA - Protocol Layer

- **Named Pipes**

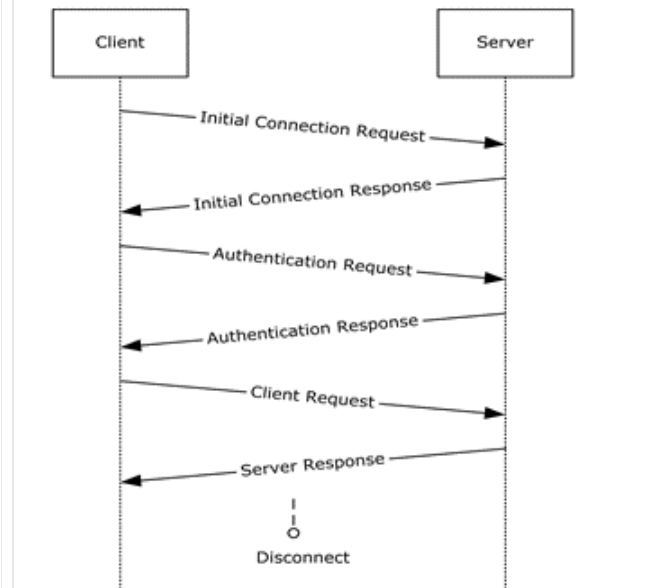
- **SQL Server** fornece a capacidade de **conexão entre o cliente e o SQL Server através de uma rede LAN**, interagindo pelo protocolo Named Pipes.
- Desenvolvido para redes locais (LAN).
- Fornece uma forma para que a **comunicação dentro dos processos** (inter-process) ocorra entre os **processos em execução no mesmo servidor**.
- Oferece uma maneira de **enviar seus dados sem perder desempenho** por envolver uma camada de rede.
- Uma **parte da memória é usada por um processo a fim de passar informações para outro processo**, para que a saída de um seja a entrada de outro.

# ARQUITETURA - Tabular Data Stream

- O **Tabular Data Stream (TDS)** é um protocolo a nível de aplicativo usado na **transferência de solicitações e respostas entre clientes e sistemas de servidores de banco de dados**.
- Ambos os 3 protocolos (**Shared Memory, TCP/IP e Named Pipes**) utilizam pacotes **TDS**.
- O cliente normalmente estabelece uma conexão de longa duração com o servidor, com a conexão estabelecida usando um protocolo em nível de transporte, **as mensagens TDS são usadas para se comunicar entre o cliente e o servidor**.
- O **TDS é encapsulado em pacotes de rede**, habilitando a transferência de dados entre cliente e servidor.
- A conexão **TDS** persiste até que a conexão no nível de transporte seja encerrada.

# ARQUITETURA - Tabular Data Stream

- O **TDS** inclui recursos para autenticação e identificação, negociação de criptografia de canal, emissão de lotes SQL, chamadas de procedimento armazenado, retorno de dados e solicitações do gerenciador de transações.
- O **fluxo de dados** descreve os nomes, tipos e descrições opcionais das linhas que estão sendo retornadas.

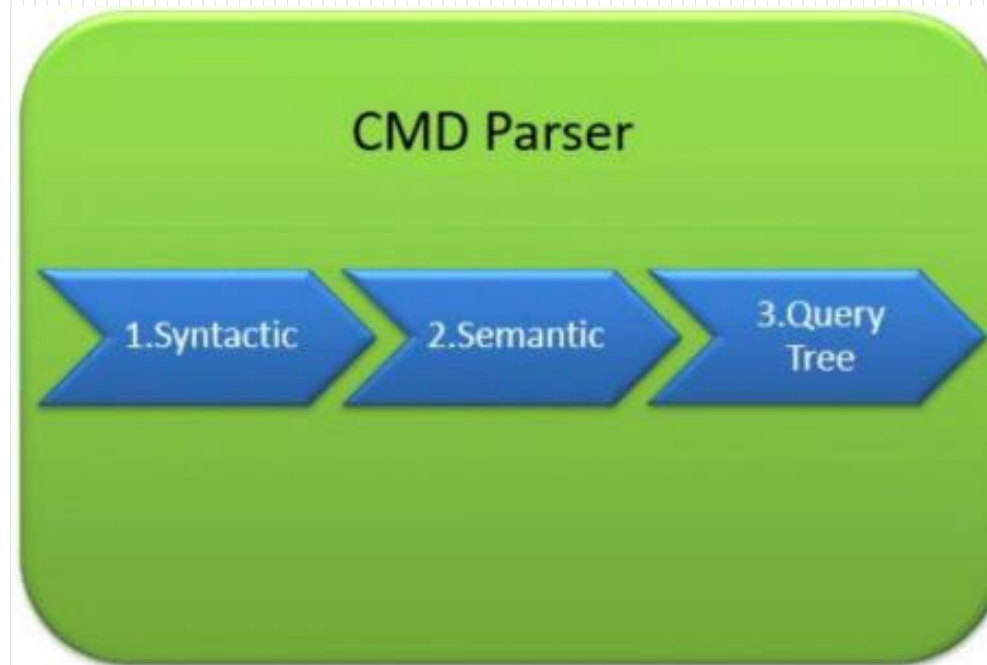


# ARQUITETURA - Relational Engine

- **Relational Engine** é conhecido como **Query Processor** (processador de consultas).
- Possui os **componentes do SQL Server que determinam o que exatamente uma consulta precisa fazer** e como ela pode ser melhor executada.
- **Responsável pela execução das consultas do usuário** solicitando dados de mecanismo de armazenamento e processando os resultados retornados.
- Existem **três componentes principais** dentro do **Relational Engine**:
  - **CMD Parser**
  - **Optimizer**
  - **Query Executor**

# ARQUITETURA - CMD Parser

- Os **dados recebidos** da camada de protocolo são passados para o **Relational Engine**.
- **CMD Parser** é o **primeiro componente** do **Relational Engine** a **receber os dados de consulta**.
- A **principal tarefa** do **CMD Parser** é **verificar se há erros sintáticos e semânticos**.
- Gera uma **árvore de consulta**.



# ARQUITETURA - CMD Parser

- **Verificação Sintática**

- Assim como qualquer outra linguagem de programação, o **SQL Server** também possui o **conjunto predefinido de palavras-chave**. O **SQL Server** tem sua **própria gramática**.
- **SELECT, INSERT, UPDATE** e muitas outras pertencem às **listas de palavras-chave predefinidas do SQL Server**.
- **CMD Parser** faz **verificação sintática**, se a entrada dos usuários não seguir as **regras de sintaxe ou gramática**, será retornado um erro.

- **Verificação Semântica**

- Tarefa realizada pelo **normalizador** (Normalizer).
- Em sua forma mais simples, **verifica se o nome da coluna e o nome da tabela que está sendo consultado existem no esquema** (schema). E, se existir, **vincula à consulta realizada**. Isso também é conhecido como **ligação** (Binding).
- A **complexidade aumenta quando as consultas do usuário contêm VIEW**. O **normalizador** (Normalizer) executa a substituição com a definição de visualização armazenada internamente e muito mais.

- **Árvore de Consulta**

- Essa etapa gera uma **árvore de execução diferente na qual a consulta pode ser executada**.
- Observe que todas as árvores diferentes têm a mesma saída desejada.

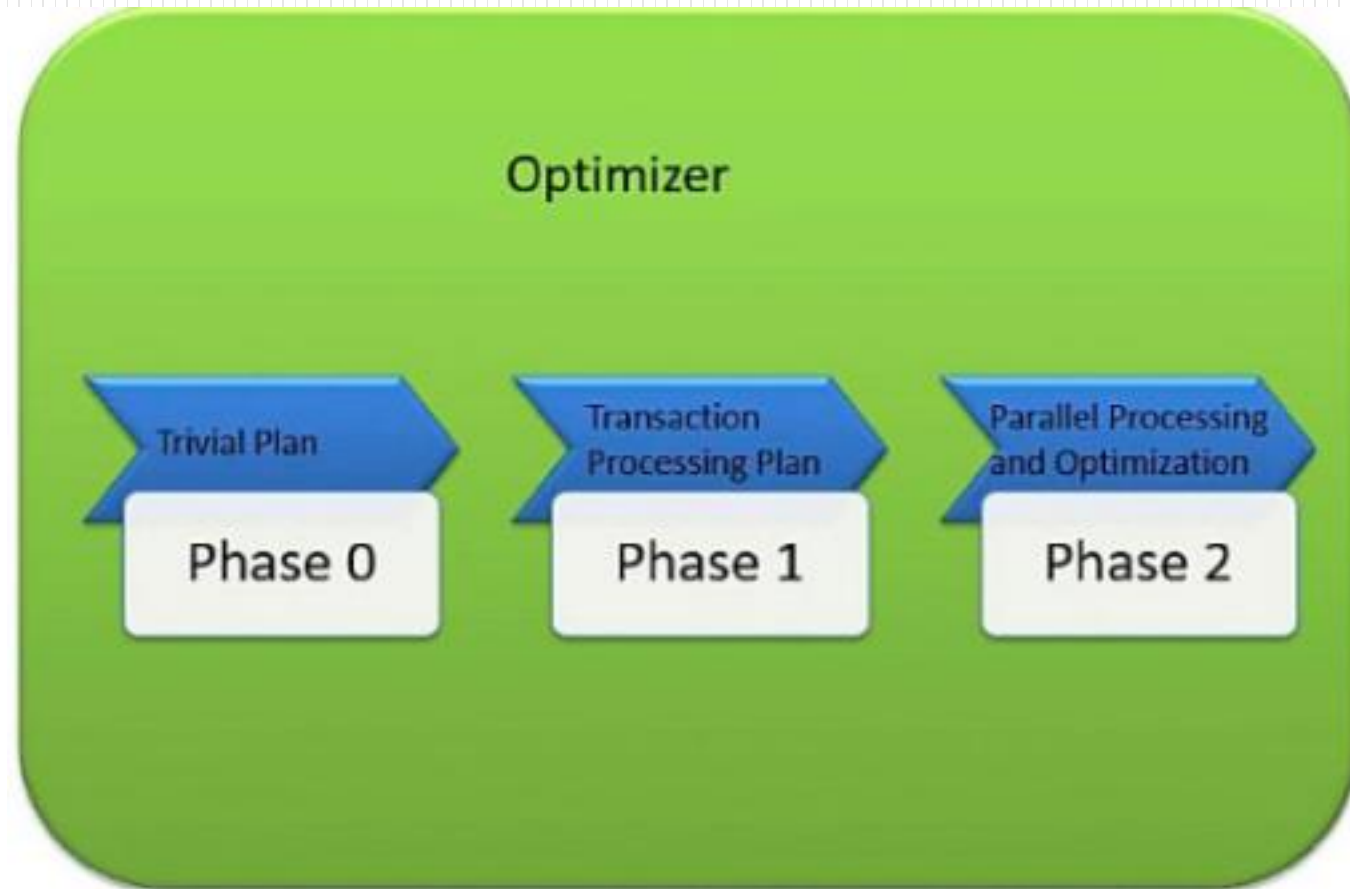


# ARQUITETURA - Optimizer

- O trabalho do **optimizer** (otimizador) é **criar um plano de execução para a consulta** do usuário. Este plano irá determinar como a consulta será executada.
- Nem todas as consultas são otimizadas.
- A otimização é feita para **comandos DML (Data Modification Language)** como **SELECT, INSERT, DELETE e UPDATE**. Essas consultas são **marcadas primeiro e depois enviadas para o otimizador**.
- **Comandos DDL (Data Definition Language)** como **CREATE e ALTER** não são otimizados, mas são compilados em um formulário interno.
- O **custo da consulta** é calculado com base em fatores como **uso da CPU, uso da memória e necessidades de entrada/saída**.
- A função do **optimizer** é **encontrar o plano de execução mais barato e econômico**, e não o melhor.
- O **SQL Server optimizer** funciona em **algoritmos exaustivos/heurísticos embutidos**.
- O objetivo é **minimizar o tempo de execução da consulta**.

# ARQUITETURA - Optimizer

- As pesquisas de otimização seguem **três fases**.

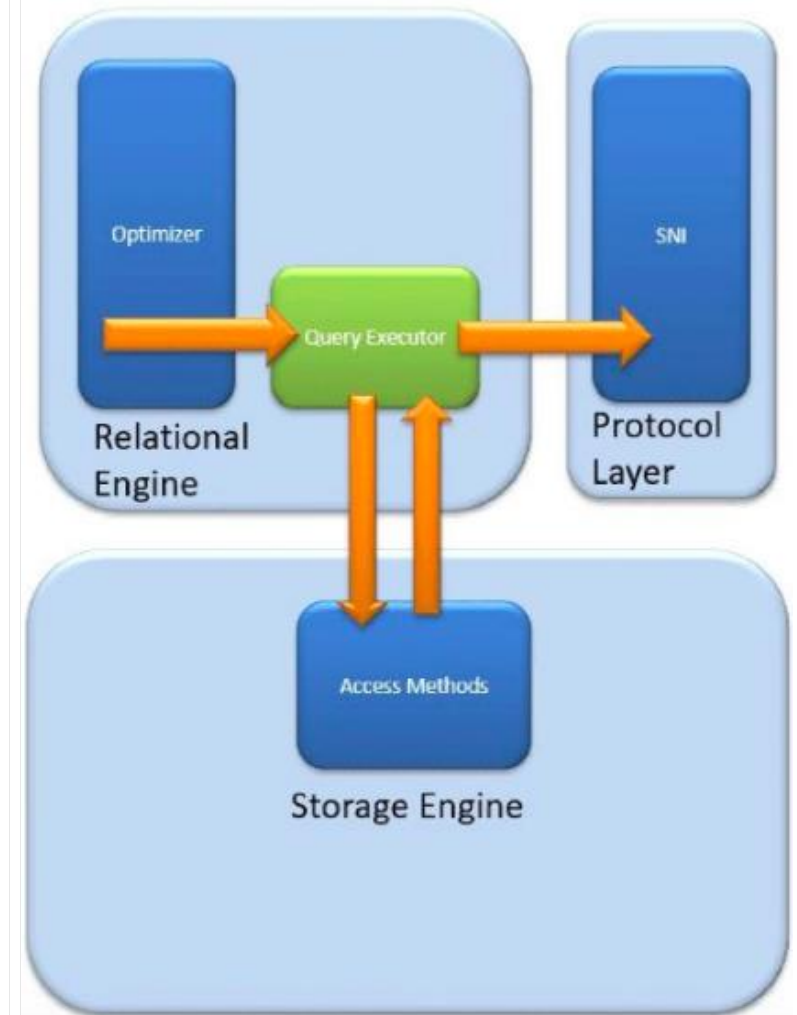


# ARQUITETURA - Optimizer

- **Fase 0: Procurar um plano trivial**
  - Também conhecido como **estágio de pré-otimização**.
  - Para alguns casos, **pode haver apenas um plano prático e viável**, conhecido como **plano trivial**, não havendo necessidade de criar um plano otimizado. O motivo é que, se **pesquisar mais resultaria na localização do mesmo tempo de execução do plano**, o que gera **custo extra**, procurar um plano otimizado que não era necessário.
  - Se nenhum plano trivial for encontrado, a **fase 1 será iniciada**.
- **Fase 1: Procurar planos de processamento de transações**
  - Isso inclui a **busca de um plano simples e complexo**.
  - **Pesquisa de plano simples**: os **dados antigos da coluna e do índice envolvidos na consulta serão utilizados para análise estatística**. Isso geralmente consiste, mas não está restrito, a um índice por tabela.
  - Ainda assim, se o plano não for encontrado, o **plano mais complexo será pesquisado**, o que envolve vários índices por tabela.
- **Fase 2: Processamento e otimização paralelos**
  - Se nenhuma das estratégias acima funcionar, o otimizador **procurará as possibilidades de processamento paralelo**, o que **depende das capacidades e configuração de processamento da máquina**.
  - Se isso ainda não for possível, a **fase final de otimização é iniciada**. Agora, o **objetivo final da otimização é encontrar todas as outras opções possíveis para executar a consulta da melhor maneira**.

# ARQUITETURA - Query Executor

- O **Query Executor** (executor de consultas) chama o método de acesso (**Method Access**).
- Ele fornece um plano de execução para a lógica de busca de dados necessária para a execução.
- Uma vez os dados são recebidos do **Storage Engine**, o resultado é publicado no **Protocol Layer**.
- Por fim, os dados são enviados ao usuário final.

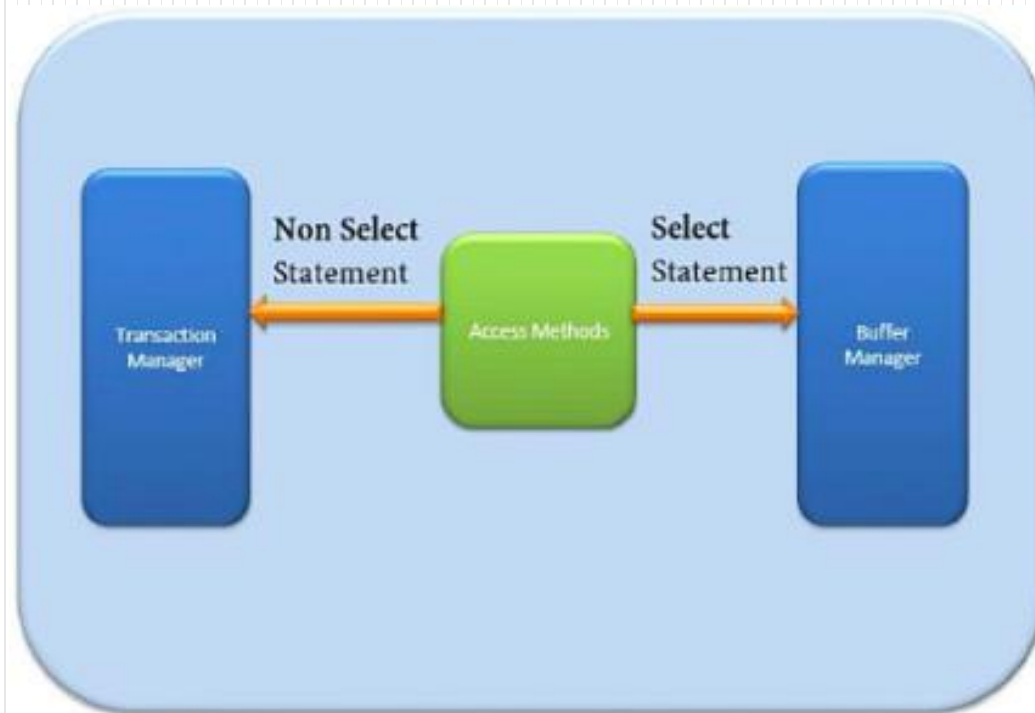


# ARQUITETURA - Storage Engine

- O trabalho do **Storage Engine** (mecanismo de armazenamento) é **armazenar dados em um sistema de armazenamento como disco ou SAN e recuperar os dados quando necessário.**

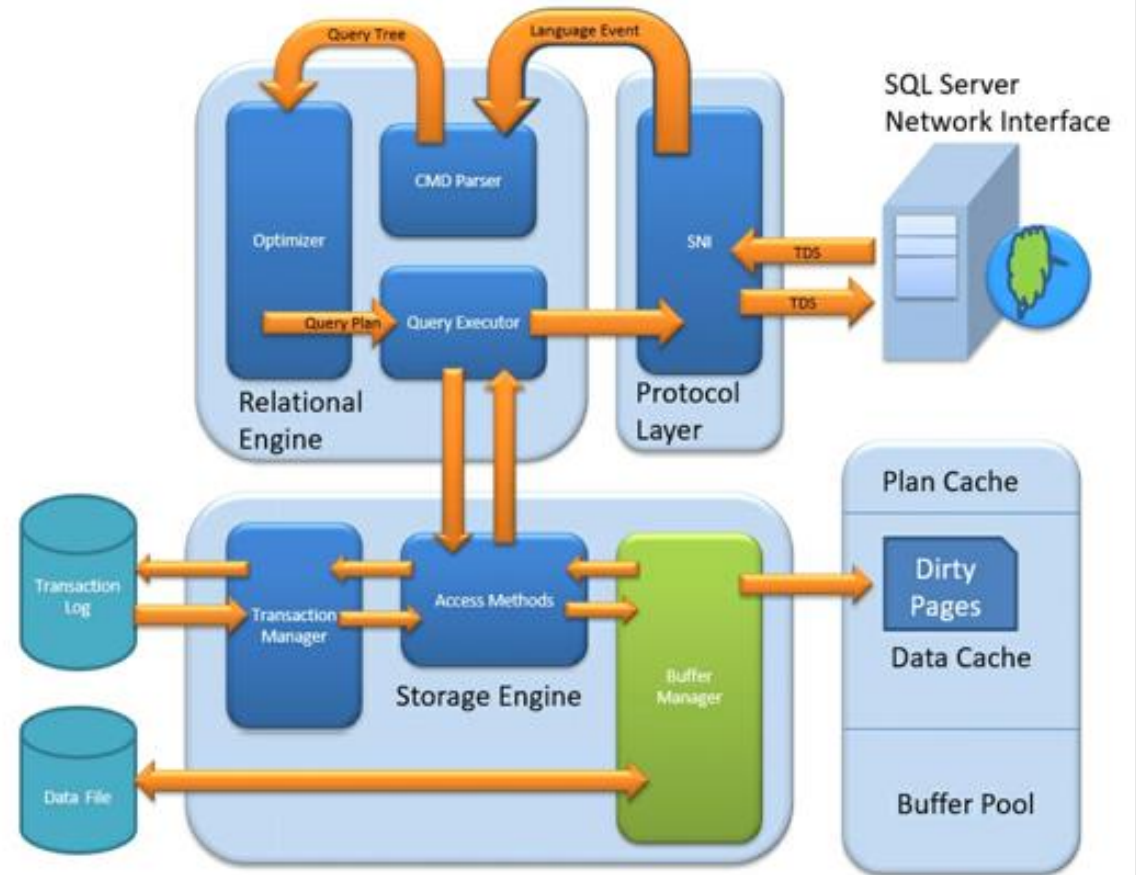
# ARQUITETURA - Access Method

- Atua como uma **interface** entre o **Query Executor** (executor de consultas) e o **Buffer Manager** (gerenciador de buffer)/ **Transaction Logs** (logs de transações).
- O **Access Method** (método de acesso) **não realiza nenhuma execução**.
- A primeira ação é **determinar se a consulta é**:
  - **SELECT** (DDL)
  - **NON-SELECT** (DDL e DML)
- Dependendo do resultado, o **Access Method** executa as seguintes etapas:
  - Se a consulta for DDL, **SELECT**, a consulta é passada para o **Buffer Manager** para processamento posterior.
  - Se a consulta for uma instrução DDL, **NON-SELECT**, a consulta será passada para o **Transaction Manager**. Isso inclui principalmente a instrução **UPDATE**.



# ARQUITETURA - Buffer Manager

- O gerenciador de buffers gerencia as principais funções dos módulos abaixo:
  - **Plan Cache**
  - **Data Parsing: Buffer Cache & Data Storage**
  - **Dirty Page**



# ARQUITETURA - Buffer Manager

- **Plan Cache**

- **Existing Query plan** (plano de consulta existente): o gerenciador de buffer verifica se o plano de execução está armazenado no **Plan Cache**. Se Sim, o **cache do plano de consulta e seu cache de dados associado serão usados**.
- **First time Cache plan** (primeiro plano de cache): se o **plano de execução da consulta inicial estiver sendo executado e for complexo, faz sentido armazená-lo no Plan Cache**. Isso garantirá uma disponibilidade mais rápida quando o próximo servidor SQL receber a mesma consulta.

- **Data Parsing: Buffer Cache & Data Storage**

- O **Buffer Manager** fornece acesso aos dados necessários. Duas abordagens são possíveis, dependendo se existe dados no cache.
- **Buffer Cache - Soft Parsing**: O **Buffer Manager procura dados no buffer no cache de dados**. Se presente, esses dados são usados pelo Query Executor, melhorando o desempenho.
- **Data Storage - Hard Parsing**: Se os dados não estiverem presentes no **Buffer Manager**, os **dados necessários serão pesquisados no Data Storage** (armazenamento de dados).

- **Dirty Page**

- É armazenado como uma lógica de processamento do **Transaction Manager**.



# ARQUITETURA - Transaction Manager

- O **Transaction Manager** (gerenciador de transações) é chamado quando o **método de acesso determina que a consulta é uma instrução do tipo NON-SELECT**.
- **Log Manager** (gerenciador de log)
  - Mantém um **controle de todas as atualizações feitas no sistema** por meio de logs nos logs de transações.
  - Possuem um número de sequencia de logs com o ID de transação e o **registro de modificação de dados**.
  - Usado para **acompanhar a transação confirmada e a reversão da transação**.
- **Lock Manager** (gerenciador de bloqueio):
  - Durante a transação, **os dados associados no armazenamento de dados estão no estado de bloqueio**.
  - Esse **processo garante a consistência e o isolamento dos dados (ACID)**.

# ARQUITETURA - Transaction Manager

- **Execution Process** (processo de execução):
  - O **Log Manager** inicia o log e o **Lock Manager** bloqueia os dados associados.
  - A cópia dos dados é mantida no cache do buffer.
  - A cópia dos dados que devem ser atualizados é mantida no buffer de log e todos os eventos atualizam os dados no buffer de dados.
  - As páginas que armazenam os dados são conhecidas como **Dirty Pages**.
  - **Checkpoint e Write-Ahead Logging** (registro de ponto de verificação e gravação antecipada): esse processo é executado e marca todas as **Dirty Pages** para o disco, mas as **páginas permanece no cache**. Mas a página é primeiro empurrada para uma página de dados do arquivo de log a partir do log do buffer, isso é conhecido como **Write-Ahead Logging**.
  - **Lazy Writer** (gravador lento): a **Dirty Page pode permanecer na memória**. Quando o **SQL Server** observa uma carga enorme e a memória de buffer é necessária para uma nova transação, ele **libera Dirty Pages do cache**.

# DATABASE

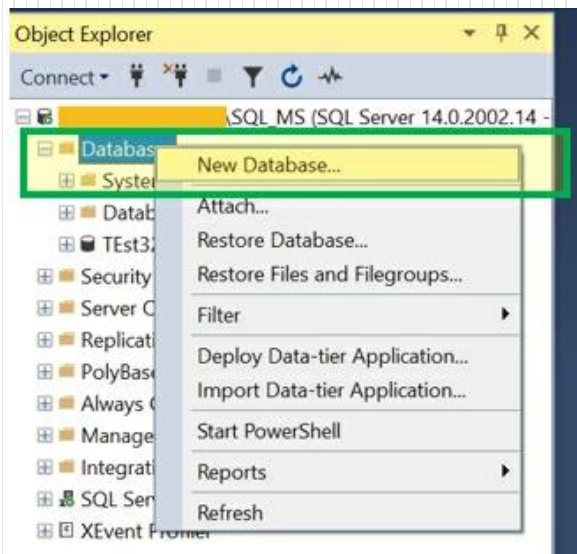
- Um **Database** (banco de dados) é uma **coleção de objetos** como tabelas, views (visualizações), stored procedures (procedimentos armazenados), triggers (gatilhos), funções, etc.
- O **Database** (banco de dados) **armazena os dados de maneira bem organizada** para facilitar o acesso e a recuperação. No **SQL Server**, existem **dois tipos de bancos de dados**:
- **System Databases** (bancos de dados do sistema): são **criados automaticamente para você ao instalar o SQL Server**. Desempenham um papel crucial no servidor, especialmente para garantir que os objetos de banco de dados sejam executados corretamente.
- **User Databases** (bancos de dados do usuário): são **criados por usuários do banco de dados**.

# DATABASE - Regras de Nomes

- Os nomes dos bancos de dados devem ser **exclusivos em uma instância do SQL Server**.
- Os nomes do banco de dados podem ter no máximo **128 caracteres**.
- A instrução **CREATE DATABASE** deve ser executada no modo de confirmação automática.

# DATABASE - Criando um Database

- Parar criar um banco de dados no **SQL Server** é possível utilizar o **SQL Server Management Studio** e o **T-SQL**.
- Criar banco de dados usando o **SQL Server Management Studio**:



- Criar banco de dados usando o **T-SQL**:

```
CREATE DATABASE <Database_name>
```

# DATABASE - Alterando um Database

- Para alterar um banco de dados no **SQL Server** é possível utilizar o **SQL Server Management Studio** e o **T-SQL**.
- É possível **renomear o nome do banco de dados, alterar sua localização e a configuração do arquivo**, etc.
- Regras para alterar o novo banco de dados:
  - A instrução **ALTER DATABASE** deve ser executada no modo de confirmação automática.
  - **ALTER DATABASE** não é permitido em uma transação explícita ou implícita.

```
ALTER DATABASE <Database_name>  
MODIFY NAME = <New Name>
```

# DATABASE - Deletando um Database

- Parar deletar um banco de dados no **SQL Server** é possível utilizar o **SQL Server Management Studio** e o **T-SQL**.
- Use o comando **DROP DATABASE** para remover o banco de dados.

```
DROP DATABASE <Database_name>
```

# DATABASE - Backup

- **BACKUP** é basicamente uma **cópia de segurança**. A razão do procedimento é **assegurar que o usuário ou organização se resguardem de uma possível perda de arquivos e dados originais**, que pode ocorrer por diversos motivos conforme descrito anteriormente.
- **BACKUPS realizados periodicamente**, com o procedimento correto e armazenados de forma inteligente minimizam e até neutralizam os impactos a serem sofridos na perda de dados e arquivo

- Realizar um **BACKUP Full**.

```
BACKUP DATABASE { database_name | @database_name_var }  
    TO <backup_device> [ ,...n ]  
    [ <MIRROR TO clause> ] [ next-mirror-to ]  
    [ WITH { DIFFERENTIAL  
            | <general_WITH_options> [ ,...n ] } ]  
[;]
```

- Realizar **BACKUP** de arquivos específicos.

```
BACKUP DATABASE { database_name | @database_name_var }  
    <file_or_filegroup> [ ,...n ]  
    TO <backup_device> [ ,...n ]  
    [ <MIRROR TO clause> ] [ next-mirror-to ]  
    [ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] } ]  
[;]
```

- Realizar um **BACKUP Parcial/Incremental**.

```
BACKUP DATABASE { database_name | @database_name_var }  
    READ_WRITE_FILEGROUPS [ , <read_only_filegroup> [ ,...n ] ]  
    TO <backup_device> [ ,...n ]  
    [ <MIRROR TO clause> ] [ next-mirror-to ]  
    [ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] } ]  
[;]
```

- Realizar **BACKUP** de Logs Transacionais.

```
BACKUP LOG  
    { database_name | @database_name_var }  
    TO <backup_device> [ ,...n ]  
    [ <MIRROR TO clause> ] [ next-mirror-to ]  
    [ WITH { <general_WITH_options> | \<log-specific_optionspec> } [ ,...n ] ]  
[;]
```



# DATABASE - Restaurando um Database

- **RESTORE** é um procedimento utilizado para **recuperar a condição do banco de dados ou repositório de arquivos**, utilizando o **BACKUP** realizado. Neste procedimento, os dados **estarão na condição exata do momento em que o backup foi realizado**.

- Restaurar um banco de dados inteiro de um backup de banco de dados completo (uma restauração completa).

```
RESTORE DATABASE { database_name | @database_name_var }  
[ FROM <backup_device> [ ,...n ] ]  
[ WITH  
  {  
    [ RECOVERY | NORECOVERY | STANDBY =  
      {standby_file_name | @standby_file_name_var }  
    ]  
    , <general_WITH_options> [ ,...n ]  
    , <replication_WITH_option>  
    , <change_data_capture_WITH_option>  
    , <FILESTREAM_WITH_option>  
    , <service_broker_WITH_options>  
    , \<point_in_time_WITH_options-RESTORE_DATABASE>  
  } [ ,...n ]  
]  
[;]
```

- Restaurar parte de um banco de dados (uma restauração parcial).

```
RESTORE DATABASE { database_name | @database_name_var }  
  <files_or_filegroups> [ ,...n ]  
[ FROM <backup_device> [ ,...n ] ]  
WITH  
  PARTIAL, NORECOVERY  
  [ , <general_WITH_options> [ ,...n ]  
    , \<point_in_time_WITH_options-RESTORE_DATABASE>  
  ] [ ,...n ]  
[;]
```

- Restaurar arquivos ou grupos de arquivos específicos para um banco de dados (uma restauração de arquivo).

```
RESTORE DATABASE { database_name | @database_name_var }  
  <file_or_filegroup> [ ,...n ]  
[ FROM <backup_device> [ ,...n ] ]  
WITH  
  {  
    [ RECOVERY | NORECOVERY ]  
    , <general_WITH_options> [ ,...n ] ]  
  } [ ,...n ]  
[;]
```

# DATABASE - Restaurando um Database

- Restaurar páginas específicas para um banco de dados (uma restauração de página).
- Restaurar um log de transações em um banco de dados (uma restauração de log de transações).

```
RESTORE DATABASE { database_name | @database_name_var }  
    PAGE = 'file:page [ ,...n ]'  
[ , <file_or_filegroups> ] [ ,...n ]  
[ FROM <backup_device> [ ,...n ] ]  
WITH  
    NORECOVERY  
    [ , <general_WITH_options> [ ,...n ] ]  
[;]
```

```
RESTORE LOG { database_name | @database_name_var }  
[ <file_or_filegroup_or_pages> [ ,...n ] ]  
[ FROM <backup_device> [ ,...n ] ]  
[ WITH  
    {  
        [ RECOVERY | NORECOVERY | STANDBY =  
            {standby_file_name | @standby_file_name_var }  
        ]  
        | , <general_WITH_options> [ ,...n ]  
        | , <replication_WITH_option>  
        | , \<point_in_time_WITH_options-RESTORE_LOG>  
    } [ ,...n ]  
]  
[;]
```

# DIRETO DO CONCURSO

## Questão (FCC/SEFAZ-BA/AUDITOR/2019)

No SQL Server, em condições ideais, para criar um backup diferencial do banco de dados sanasa para o arquivo D:\backups\backupSANASA.bak utiliza-se o comando

- a) RMAN DATABASE sanasa TO DISK='D:\backups\backupSANASA.bak' WITH DIFFERENTIAL;
- b) BACKUP DATABASE sanasa TO DISK='D:\backups\backupSANASA.bak' WITH DIFFERENTIAL;
- c) SQLDUMP DATABASE sanasa PARAMETERS(DISK='D:\backups\ backupSANASA.bak', TYPE=DIFFERENTIAL);
- d) BKP\_DUMP DATABASE sanasa TO 'D:\backups\backupSANASA.bak' TYPE DIFFERENTIAL.
- e) BACKUP DATABASE sanasa TO 'D:\backups\backupSANASA.bak' TYPE DIFFERENTIAL;

# GABARITO

## LETRA B.

No **SQL Server**, para executar um backup diferencial de uma database com os parâmetros passados, que será salvo em um local específico e um database específico utilize a instrução **BACKUP DATABASE** com os parâmetros **TO DISK** (para definir o local onde será salvo) e **WITH DIFFERENTIAL** (para definir que é um backup diferencial).

# DATATYPE

- Um **Data Type** (tipo de dados) é definido como **o dado que qualquer coluna ou variável pode armazenar no SQL Server**. Ao criar qualquer tabela ou variável, além de especificar o nome, você também define o tipo de dados que ele será armazenado.
- A determinação do tipo de dados também **restringe o usuário a inserir dados inesperados ou inválidos**.
- Cada **tipo de dados possui requisitos de memória diferentes**, faz mais sentido definir a coluna ou variável com o tipo de dados que ele manterá para uso eficiente da memória.
- O **SQL Server** suporta as seguintes categorias de tipo de dados:
  - **Exact numeric** (numeral exato)
  - **Approximate numeric** (numeral aproximado)
  - **Date e Time** (data e hora)
  - **Character strings** (cadeias de caracteres)
  - **Unicode character strings** (cadeias de caracteres unicode)
  - **Binary strings** (cadeias binárias)

# DATATYPE - Exatic Numeric

- O numeral exato possui nove tipos de sub-dados.

Data Type	Description	Lower limit	Upper limit	Memory
bigint	It stores whole numbers in the range given	$-2^{63}$ (-9,223,372,036,854,775,808)	$2^{63}-1$ (-9,223,372,036,854,775,807)	8 bytes
int	It stores whole numbers in the range given	$-2^{31}$ (-2,147,483,648)	$2^{31}-1$ (-2,147,483,647)	4 bytes
smallint	It stores whole numbers in the range given	$-2^{15}$ (-32,767)	$2^{15}$ (-32,768)	2 bytes
tinyint	It stores whole numbers in the range given	0	255	1 byte
bit	It can take 0, 1, or NULL values.	0	1	1 byte/8bit column
decimal	Used for scale and fixed precision numbers	$-10^{38}+1$	$10^{38}-1$	5 to 17 bytes
numeric	Used for scale and fixed precision numbers	$-10^{38}+1$	$10^{38}-1$	5 to 17 bytes
money	Used monetary data	-922,337,203,685,477.5808	+922,337,203,685,477.5807	8 bytes
smallmoney	Used monetary data	-214,478.3648	+214,478.3647	4 bytes

# DATATYPE - Approximate Numeric

- O **numeral aproximado** inclui ponto flutuante e valores reais. Eles são usados principalmente em cálculos científicos.

Data Type	Description	Lower limit	Upper limit	Memory	Precision
<b>float(n)</b>	Used for a floating precision number	-1.79E+308	1.79E+308	Depends on the value of n	7 Digit
<b>real</b>	Used for a floating precision number	-3.40E+38	3.40E+38	4 bytes	15 Digit

# DATATYPE – Date e Time

- Ele armazena dados do tipo **data e hora**.

Data Type	Description	Storage size	Accuracy	Lower Range	Upper Range
<b>DateTime</b>	Used for specifying a date and time from January 1, 1753 to December 31, 9999. It has an accuracy of 3.33 milliseconds.	8 bytes	Rounded to increments of .000, .003, .007	1753-01-01	9999-12-31
<b>smalldatetime</b>	Used for specifying a date and time from January 1, 0001 to December 31, 9999. It has an accuracy of 100 nanoseconds	4 bytes, fixed	1 minute	1900-01-01	2079-06-06
<b>date</b>	Used to store only date from January 1, 0001 to December 31, 9999	3 bytes, fixed	1 day	0001-01-01	9999-12-31
<b>time</b>	Used for storing only time only values with an accuracy of 100 nanoseconds.	5 bytes	100 nanoseconds	00:00:00.0000000	23:59:59.9999999
<b>datetimeoffset</b>	Similar to datetime but has a time zone offset	10 bytes	100 nanoseconds	0001-01-01	9999-12-31
<b>datetime2</b>	Used for specifying a date and time from January 1, 0001 to December 31, 9999	6 bytes	100 nanoseconds	0001-01-01	9999-12-31



# DATATYPE - Character Strings

- Este tipo de dado está relacionada aos **caracteres**. É possível definir o tipo de dado de caractere que pode ser de tamanho fixo e variável. Possui quatro tipos de dados.

Data Type	Description	Lower limit	Upper limit	Memory
<b>char</b>	It is a character string with a fixed width. It stores a maximum of 8,000 characters.	0 chars	8000 chars	n bytes
<b>varchar</b>	This is a character string with variable width	0 chars	8000 chars	n bytes + 2 bytes
<b>varchar (max)</b>	This is a character string with a variable width. It stores a maximum of 1,073,741,824 characters.	0 chars	2 <sup>31</sup> chars	n bytes + 2 bytes
<b>text</b>	This is a character string with a variable width. It stores a maximum 2GB of text data.	0 chars	2,147,483,647 chars	n bytes + 4 bytes

# DATATYPE - Unicode Character Strings

- Esta categoria armazena toda a gama de **caracteres Unicode** que usam a codificação de caracteres UTF-16.

Data Type	Description	Lower limit	Upper limit	Memory
nchar	It is a Unicode string of fixed width	0 chars	4000 chars	2 times n bytes
nvarchar	It is a unicode string of variable width	0 chars	4000 chars	2 times n bytes + 2 bytes
ntext	It is a unicode string of variable width	0 chars	1,073,741,823 char	2 times the string length

# DATATYPE - Binary String

- Esta categoria contém uma **cadeia binária de comprimento fixo e variável**.

Data Type	Description	Lower limit	Upper limit	Memory
binary	It is a fixed width binary string. It stores a maximum of 8,000 bytes.	0 bytes	8000 bytes	n bytes
varbinary	This is a binary string of variable width. It stores a maximum of 8,000 bytes	0 bytes	8000 bytes	The actual length of data entered + 2 bytes
image	This is a binary string of variable width. It stores a maximum of 2GB.	0 bytes	2,147,483,647 bytes	

# DATATYPE - Other Datatypes

- Existem outros tipos de dados no **SQL Server**.

Data Type	Description
Cursor	Its output is a column of <code>sp_cursor_list</code> and <code>sp_describe_cursor</code> . It returns the name of the cursor variable.
Row version	It version stamps table rows.
Hierarchyid	This datatype represents a position in the hierarchy
Uniqueidentifier	Conversion from a character expression.
Sql_variant	It stores values of SQL server supported Datatypes.
XML	It stores XML data in a column.
Spatial Geometry type	It represents data in a flat coordinate system.
Spatial Geography type	It represents data in the round-earth coordinate system.
table	It stores a result set for later processing.

# DIRETO DO CONCURSO

## **Questão (CESPE /ME/Desenvolvimento de Software/2020)**

Acerca de sistemas gerenciadores de banco de dados, julgue o item subsequente.

O SQL Server tem um tipo de dado denominado `sql_variant`, que armazena dados de imagem e texto de tamanho grande, maior que 10 kMbytes.

# GABARITO

**ERRADO.**

Uma coluna do tipo **sql\_variant** pode conter linhas de tipos de dados diferentes. Por exemplo, uma coluna definida como **sql\_variant** pode armazenar valores **int**, **binary** e **char**.

# VARIÁVEIS

- No **SQL Server**, variáveis são os **objetos que atuam como um espaço reservado** para um local de memória.
- Variáveis **retêm um valor de dado único**.
- O **SQL Server** possui dois tipos de variáveis:
  - Variável Local
  - Variável Global.
- O **usuário pode criar apenas variáveis locais**.



# VARIÁVEIS

- **Variável Local**
  - Usuário declara a variável local
  - Uma variável local começa com @.
  - Todo escopo de variável local possui restrição ao lote ou procedimento atual em qualquer sessão.
- **Variável Global**
  - Sistema mantém a variável global.
  - Um usuário não pode declará-los.
  - A variável global começa com @@.
  - Ele armazena informações relacionadas à sessão.



# VARIÁVEIS - Declarando uma Variável

- Antes de usar qualquer variável no lote ou procedimento, é preciso **declarar a variável necessária**.
- O comando **DECLARE** é usado para declarar uma variável.
- Somente quando a declaração é feita, uma variável pode ser usada na parte subsequente do lote ou procedimento.

```
DECLARE { @LOCAL_VARIABLE[AS] data_type [ = value ] }
```

- Regras para declarar uma variável:
  - A inicialização é uma coisa opcional ao declarar.
  - Por padrão, DECLARE inicializa a variável para NULL.
  - O uso da palavra-chave 'AS' é opcional.
  - Para declarar mais de uma variável local, use uma vírgula após a primeira definição de variável local e defina o próximo nome da variável local e tipo de dados.

# VARIÁVEIS - Atribuindo um Valor

- Você pode **atribuir um valor a uma variável de três maneiras**:

- Durante a declaração da variável usando a palavra-chave **DECLARE**.
- Usando **SET**.
- Usando **SELECT**.

- Usando **DECLARE**: o valor é atribuído durante a declaração da variável.

```
DECLARE { @Local_Variable [AS] Datatype [ = value ] }
```

- Usando **SET**: é possível manter a **declaração e a inicialização separadas**. O **SET** pode ser usado para atribuir valores à variável, após declarar uma variável.

```
DECLARE @Local_Variable <Data_Type>  
SET @Local_Variable = <Value>
```

- É possível usar o **SET** para atribuir valores para mais de uma variável.

```
DECLARE @Local_Variable_1 <Data_Type>, @Local_Variable_2 <Data_Type>,  
SET @Local_Variable_1 = <Value_1>  
SET @Local_Variable_2 = <Value_2>
```

# VARIÁVEIS - Atribuindo um Valor

- Usando **SELECT**: assim como **SET**, também podemos usar **SELECT** para atribuir valores às variáveis, declarando uma variável usando **DECLARE**.

```
DECLARE @LOCAL_VARIABLE <Data_Type>  
SELECT @LOCAL_VARIABLE = <Value>
```

- É possível usar o **SELECT** para atribuir valores para mais de uma variável.

```
DECLARE @Local_Variable _1 <Data_Type>, @Local_Variable _2 <Data_Type>, SELECT  
@Local_Variable _1 = <Value_1>, @Local_Variable _2 = <Value_2>
```

# DIRETO DO CONCURSO

## Questão (FCC/SEFAZ-BA/AUDITOR/2019)

No SQL Server 2017 um Auditor Fiscal da área de Tecnologia da Informação foi chamado para analisar as instruções abaixo.

```
DELETE Producao.HistoricoCustoProduto  
WHERE CustoPadrao BETWEEN 12.00 AND 14.00  
AND DataFim IS_NULL;  
PRINT 'O número de linhas excluídas é ' + CAST(@@ROWCOUNT as char(3));
```

Concluiu corretamente que há um erro na instrução

- a) @@ROWCOUNT, pois deveria conter apenas um caracter @.
- b) IS\_NULL , que deveria ser escrita como IS NULL .
- c) CAST, por ser um comando totalmente desnecessário nesse contexto.
- d) print , pois para exibir a informação na tela deveria ser usada a instrução ECHO .
- e) BETWEEN 12.00 AND 14.00, pois os valores 12.00 AND 14.00 deveriam ser colocados entre parênteses.

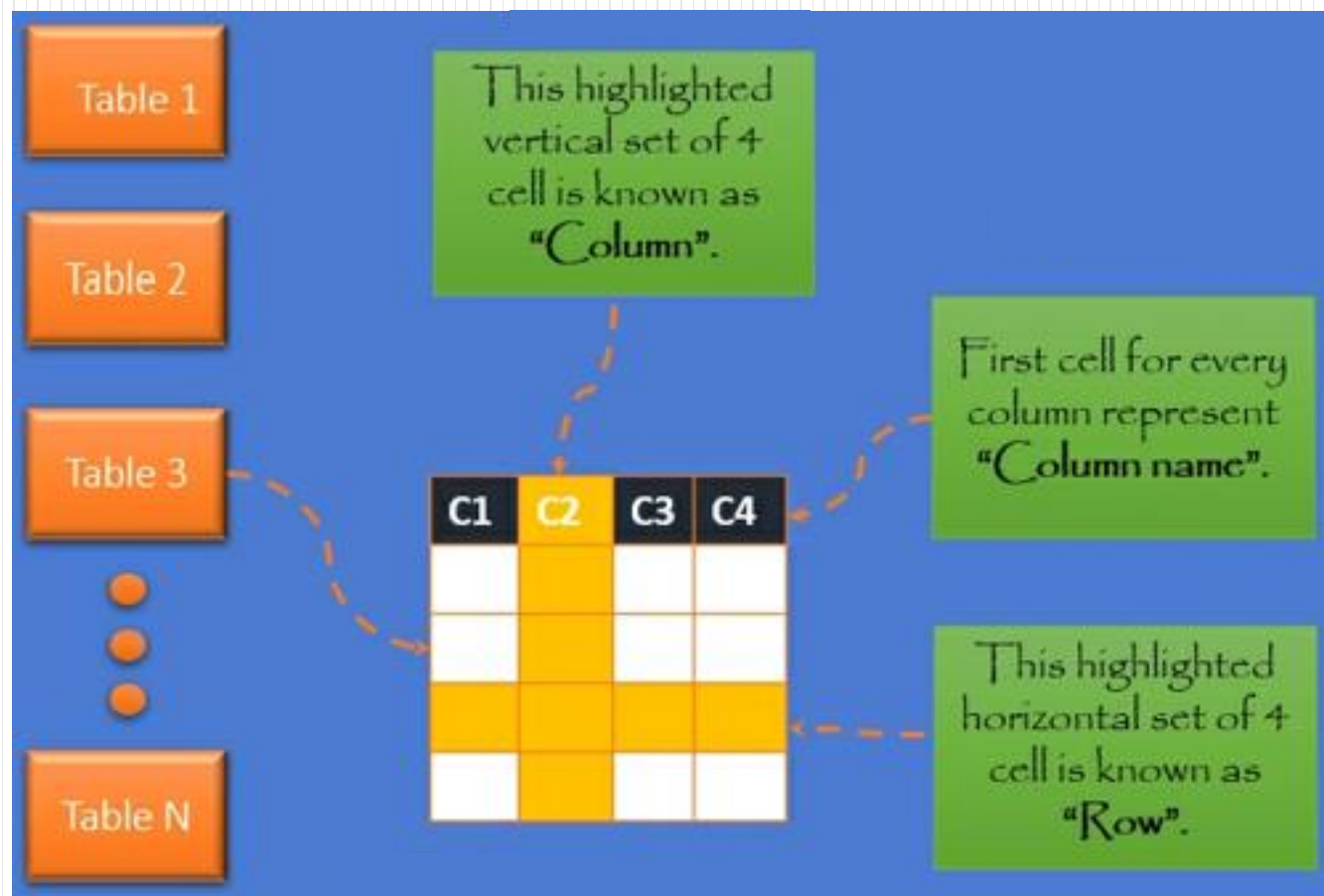
# GABARITO

## LETRA B.

No **SQL Server**, a condição “é vazio” (IS NULL) se escreve sem o underline (\_), estando errado a forma IS\_NULL.

# TABELAS

- Uma **TABLE** (tabela) é um objeto que **armazena dados no formato de linha e coluna**.



# TABELAS - Criando uma Tabela

- A primeira etapa para armazenar dados no banco de dados é **criar uma tabela na qual os dados residam**. Após a criação da tabela, podemos continuar **inserindo as linhas na tabela**.
- **Criando uma nova tabela definindo todas as colunas e seu tipo de dados.**

```
CREATE TABLE tableName  
(  
    column_1 datatype [ NULL | NOT NULL ],  
    column_2 datatype [ NULL | NOT NULL ],  
    ...  
);
```

- **Criando nova tabela usando uma tabela existente.**

```
SELECT (Column 1, ...) INTO <New Table name> FROM <Old Table name>;
```

- O parâmetro **tableName** indica o nome da tabela que será criado.
- Os parâmetros **column\_1, column\_2...** indicam as colunas a serem adicionadas à tabela.
- Uma coluna deve ser especificada como **NULL** ou **NOT NULL**. Se não for especificado, o **SQL Server** usará NULL como padrão.

# TABELAS - Inserindo Dados

- **T-SQL** possui a instrução **INSERT** que pode ser usada para **inserir dados em uma tabela**. Com esta declaração, podemos inserir uma ou mais colunas em uma tabela.

```
INSERT INTO tableName  
(column_1, column_2, ... )  
VALUES  
(expression_1, expression_2, ... ),  
(expression_1, expression_2, ... ),  
...;
```



# TABELAS - Consultando Dados

- Para visualizar os dados no **SQL Server**, é usado a instrução **SELECT**.

```
SELECT expression  
FROM tableName  
[WHERE condition];
```

# TABELAS - Alterando uma Tabela

- Para alterar uma tabela adicionando novas colunas, é usado a instrução **ALTER TABLE**.

```
Alter TABLE <Table name> ADD Column1 datatype, Column2 datatype;
```

# TABELAS - Deletando uma Tabela

- Excluimos a tabela quando ela não é mais necessária.
- Para excluir uma tabela, é usado a instrução **DROP TABLE**.

```
DROP TABLE <tableName>;
```

- Como alternativa, é possível usar o comando **DELETE TABLE**. Mas, serão excluídos apenas linhas (dados). A estrutura da tabela permanecerá intacta. O comando tem como objetivo truncar a tabela para que possa ser usada posteriormente.

```
DELETE TABLE <Table name>;
```

# DIRETO DO CONCURSO

## **Questão (FGV / IMBEL / Analista Especializado - Analista de Sistemas / 2021)**

No contexto do SQL Server, assinale os comandos utilizados para remover registros de uma tabela.

Alternativas:

A ALTER/ERASE

B DELETE/TRUNCATE

C DROP/DUMP

D ERASE/REMOVE

E REWRITE/EXCLUDE

# GABARITO

## LETRA B.

No **SQL Server**, existe o comando TRUNCATE que limpa a tabela e seus registros de memória, é considerado assim com o DELETE um comando DML.

Embora **TRUNCATE TABLE** seja semelhante a DELETE, ela é classificada como uma instrução **DDL** em vez de uma instrução **DML**.

Truncar:

- desaloca as páginas de dados em uma tabela e somente esta desalocação é armazenada no log de transações
- adquire apenas bloqueios de tabela e página para toda a tabela. como nenhum bloqueio de linha é usado, menos memória é necessária (o bloqueio é um objeto de memória puro)
- redefine a coluna de IDENTITY se houver uma
- remove TODAS as páginas. NENHUMA página vazia é deixada para trás em uma tabela
- não dispara gatilhos de exclusão

# PRIMARY KEY

- Uma **PRIMARY KEY** (chave primária) é um **campo ou uma combinação de campos que identificam um registro exclusivo**.
- A **chave primária** é uma **coluna ou conjunto de colunas que são exclusivas**.
- Todo valor é exclusivo para a **chave primária**.
- **Regras para chave primária:**
  - Cada tabela pode ter apenas uma chave primária.
  - Todos os valores são exclusivos e o valor da chave primária pode identificar exclusivamente cada linha.
  - O sistema não permitirá inserir uma linha com uma chave primária que já exista na tabela.
  - A chave primária não pode ser NULL.
- Uma **chave primária** pode ser uma **combinação de várias colunas**. Essa combinação é conhecida como **COMPOSITE PRIMARY KEY** (chave primária composta).

# PRIMARY KEY

- Uma chave primária pode ser definida na criação de uma tabela.

```
CREATE TABLE <Table_Name>
(
  Column1 datatype,
  Column2 datatype, CONSTRAINT <Name> PRIMARY KEY (Column name)
  .
);
```

- É possível definir uma coluna existente como chave primária, em uma tabela já existente.
- A instrução ALTER pode ser usada para criar uma chave primária. No entanto, a chave primária pode ser criada apenas em colunas definidas como NOT NULL. Você não pode criar uma chave primária em uma coluna que permita NULL.

```
ALTER TABLE tableName
ADD CONSTRAINT constraintName PRIMARY KEY (column_1, column_2, ... column_n);
```

# FOREIGN KEY

- Uma **FOREING KEY** (chave estrangeira) fornece uma maneira de impor a **integridade referencial no SQL Server**.
- Uma **chave estrangeira** garante que os valores em uma tabela devem estar presentes em outra tabela.
- **Regras para chave estrangeira:**
  - NULL é permitido na chave estrangeira.
  - A tabela que está sendo referenciada é chamada de tabela pai.
  - A tabela com a chave estrangeira é chamada tabela filho.
  - A chave estrangeira na tabela filha faz referência à chave primária na tabela pai.
  - Esse relacionamento pai-filho reforça a regra conhecida como "Integridade Referencial".
- Todo valor da **chave estrangeira** deve fazer parte da **chave primária de outras tabelas**.



# FOREIGN KEY

- Uma chave estrangeira é evidenciada na criação de uma tabela filho que possui algum tipo de relação com uma tabela pai.

```
CREATE TABLE childTable
(
    column_1 datatype [ NULL |NOT NULL ],
    column_2 datatype [ NULL |NOT NULL ],
    ...

    CONSTRAINT fkey_name
        FOREIGN KEY (child_column1, child_column2, ... child_column_n)
        REFERENCES parentTable (parent_column1, parent_column2, ... parent_column_n)
        [ ON DELETE { NO ACTION |CASCADE |SET NULL |SET DEFAULT } ]
        [ ON UPDATE { NO ACTION |CASCADE |SET NULL |SET DEFAULT } ]
);
```

- **fkey\_name** é o nome da restrição de **chave estrangeira** a ser criada.
- **child\_column1, child\_column2... child\_column\_n** são os nomes das colunas **childTable** para referenciar a **chave primária** em **parentTable**.
- **parentTable** é o nome da **tabela pai** cuja chave deve ser referenciada na **tabela filho**.
- **parent\_column1, parent\_column2, ... parent\_column3** são as colunas que compõem a **chave primária** da **tabela pai**.

# FOREIGN KEY

- É possível criar uma **chave estrangeira** utilizando a instrução **ALTER TABLE**.

```
ALTER TABLE childTable
ADD CONSTRAINT fkey_name
    FOREIGN KEY (child_column1, child_column2, ... child_column_n)
    REFERENCES parentTable (parent_column1, parent_column2, ... parent_column_n);
```

# DIRETO DO CONCURSO

## Questão (FGV/DPE-RJ/Técnico/2019)

No SQL Server, considere uma tabela T que contém uma coluna numérica B, que permite valores nulos, e uma coluna A, definida como primary key. Outras colunas podem existir ou não.

Nesse contexto, considere os cinco comandos SQL a seguir.

1. select \* from T where A is not null order by A
2. select \* from T where not B is null order by A
3. select \* from T where where B > ∅ order by A
4. select \* from T where where not B <= ∅ order by A
5. select \* from T order by A

Sobre a execução de cada um desses comandos, é correto afirmar que, para qualquer que seja a instância de T, os únicos comandos que sempre produzem resultados idênticos entre si são:

- a) 1 e 2;
- b) 1 e 5;
- c) 2 e 3;
- d) 3 e 4;

# GABARITO

## LETRA B.

É preciso levar em consideração que a coluna A, por ser **PRIMARY KEY**, nunca poderá ser vazio (**NULL**), mas coluna B, por não ser **PRIMARY KEY**, pode absorver qualquer valor, inclusive **NULL**.

Com isso ao realizar um **SELECT** com todas as linhas onde a coluna A não é **NULL** e todas as linhas da Tabela, o resultado será o idêntico.

# DIRETO DO CONCURSO

## Questão (CESPE/CGE-CE/AUDITOR/2019)

```
ALTER TABLE dbo.Ordens ADD CONSTRAINT  
DFT_Ordens_ordemts DEFAULT(SYSDATETIME()) FOR  
ordemts;
```

O código precedente modifica a tabela Ordens (composta por ordens de venda e ordemts – data/hora do pedido), em um banco de dados MS-SQL Server.

Após essa modificação, sempre que for inserida uma nova ordem de venda,

- a)** se não for definido um valor para o campo ordemts, o sistema invocará a função SYSDATETIME.
- b)** será obrigatório o preenchimento manual do campo ordemts.
- c)** se não for definido um valor para o campo ordemts, este ficará Null.
- d)** se for definido um valor para o campo ordemts, o banco de dados retornará erro de dado não permitido.
- e)** se não for definido um valor Null para o campo ordemts, o banco de dados retornará erro de dado não permitido.

# GABARITO

## LETRA A.

```
ALTER TABLE dbo.Ordens ADD CONSTRAINT  
DFT_Ordens_ordemts DEFAULT(SYSDATETIME()) FOR  
ordemts;
```

O comando acima está alterando a tabela **Ordens** para que a coluna **DFT\_Ordens\_ordemts** possua um valor default (padrão) caso não seja inserindo nenhum valor ao adicionar uma nova linha na tabela em questão. Esse valor default será atribuído com o resultado da chamada de função **SYSDATETIME()**.

# DIRETO DO CONCURSO

## Questão (FGV/AL-RO/Analista/2018)

Tomando por referência uma instalação MS SQL Server, analise os comandos a seguir.

```
create table T (a int, b int);  
insert into T values  
        (1,1),(2,NULL),(NULL,3),(4,4),(5,5);  
select * FROM T x full join T y on x.a=y.a;
```

O número de linhas no resultado, excluída a linha de títulos, é

- a) 5
- b) 6
- c) 7
- d) 10
- e) 25

# GABARITO

## LETRA B.

SQL> select \* from T; // dado das tabela T

A	B
1	1
2	
	3
4	4
5	5

SQL> select \* from T x full join T y on x.a=y.a; // select FULL JOIN

A	B	A	B
1	1	1	1
2		2	
			3
4	4	4	4
5	5	5	5
	3		



# JOIN

- Podemos **recuperar dados de mais de uma tabela** usando a instrução **JOIN**.
- O **SQL Server** possui 4 tipos de junções:
  - **INNER JOIN/Simple JOIN**
  - **LEFT OUTER JOIN/LEFT JOIN**
  - **RIGHT OUTER JOIN/RIGHT JOIN**
  - **FULL OUTER JOIN**
- Para explicar as instruções JOIN o seguinte exemplo será usado:

**Student Table** (tabela de estudantes)

	admission	firstName	lastName	age
1	3420	Nicholas	Samuel	14
2	3380	Joel	John	15
3	3410	Japheth	Becky	16
4	3398	George	Joshua	14
5	3386	John	Lucky	15
6	3403	Simon	Dan	13
7	3400	Calton	Becham	16

**Fee table** (tabela de pagamentos)

	admission	course	amount_paid
1	3380	Electrical	20000
2	3420	ICT	15000
3	3398	Commerce	13000
4	3410	HR	12000

# JOIN - INNER JOIN

- **INNER JOIN:** esse tipo de **JOIN** retorna **linhas de todas as tabelas** nas quais a condição de associação é verdadeira.

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
INNER JOIN Fee
ON Students.admission = Fee.admission
```

- A saída será:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000

# JOIN - LEFT JOIN

- **LEFT OUTER JOIN/LEFT JOIN:** esse tipo de **JOIN** retornará **todas as linhas da tabela à esquerda**, mais os registros na tabela à direita com valores correspondentes.

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
LEFT OUTER JOIN Fee
ON Students.admission = Fee.admission
```

- A saída será:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000
5	3386	John	Lucky	NULL
6	3403	Simon	Dan	NULL
7	3400	Calton	Becham	NULL

# JOIN - RIGHT JOIN

- **RIGHT OUTER JOIN/RIGHT JOIN:** esse tipo de **JOIN** retorna **todas as linhas da tabela à direita** e somente aquelas com valores correspondentes na tabela à esquerda.

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
RIGHT OUTER JOIN Fee
ON Students.admission = Fee.admission
```

- A saída será:

	admission	firstName	lastName	amount_paid
1	3380	Joel	John	20000
2	3420	Nicholas	Samuel	15000
3	3398	George	Joshua	13000
4	3410	Japheth	Becky	12000

# JOIN - FULL JOIN

- **FULL OUTER JOIN:** esse tipo de **JOIN** retornará todas as linhas da tabela à esquerda, mais os registros na tabela à direita com valores correspondentes.

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
FULL OUTER JOIN Fee
ON Students.admission = Fee.admission
```

- A saída será:

	admission	firstName	lastName	amount_paid
1	3420	Nicholas	Samuel	15000
2	3380	Joel	John	20000
3	3410	Japheth	Becky	12000
4	3398	George	Joshua	13000
5	3386	John	Lucky	NULL
6	3403	Simon	Dan	NULL
7	3400	Calton	Becham	NULL

# DIRETO DO CONCURSO

## Questão (FGV/Prefeitura de Niterói-RJ/Analista/2018)

A QUESTÃO DEVE SER RESPONDIDA A PARTIR DAS TABELAS DE BANCO DE DADOS T1 E T2, A SEGUIR.

Analise o comando SQL a seguir.

```
select distinct A, F  
from T1 x left join T2 y on x.B = y.E
```

T1			T2		
A	B	C	D	E	F
1	2	4	1	2	NULL
2	3	5	2	5	5
4	2	4	4	2	1
6	2	NULL	7	12	1

A execução desse comando no MS SQL Server produz um resultado com várias linhas, dispostas em pares de valores. Assinale o par de valores que **não** aparece nessas linhas.

- a) 1, NULL
- b) 1, 1
- c) 2, 1
- d) 4, 1
- e) 6, NULL

# GABARITO

## LETRA C.

Ao olhar para a tabela principal (T1) onde o JOIN é executado (coluna B) para saber o que vai ter no resultado, por ser LEFT, todos os dados que tem na Coluna B que é igual a coluna E serão retornados.

Com isso, são os dados de T1 que serão retornados são: 1, 2 (como é o dado que não tem correspondente, vai vir null sempre), 4, 6.

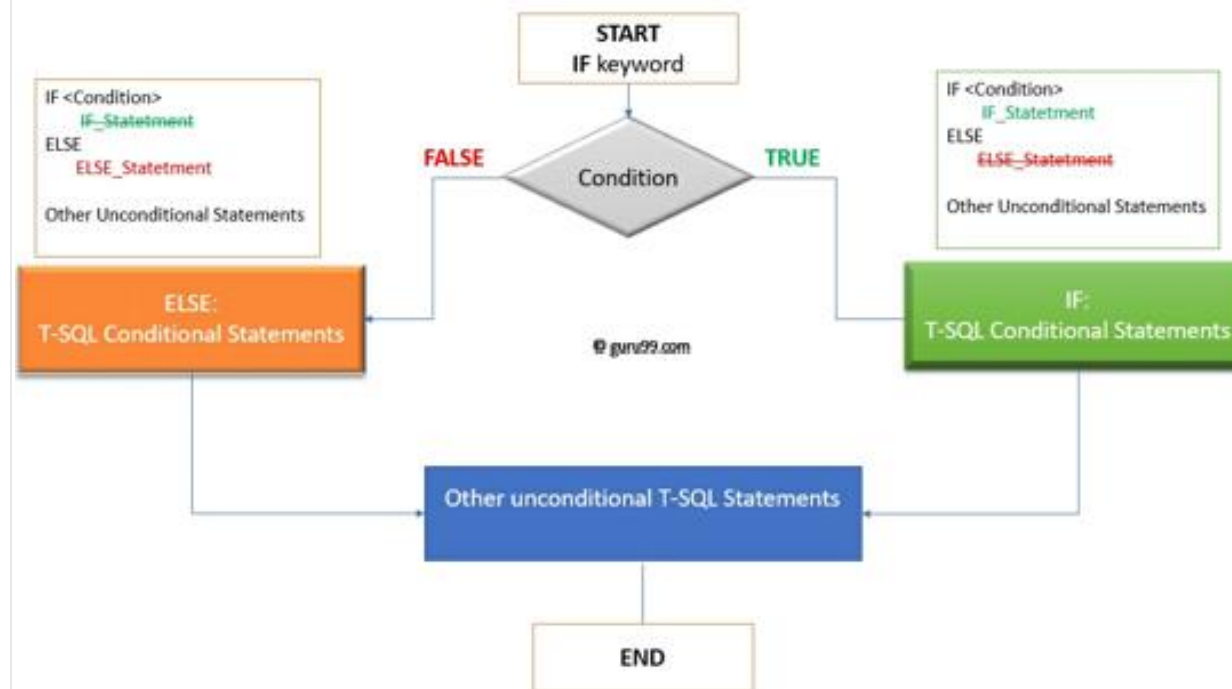
E os dados de T2 que serão retornados são: Null, 1.

Logo, as combinações são: 1 e null, 1 e 1, 2 e null ( pois não tem correspondente entre as tabelas para o valor de 3 da coluna B da tabela T1 com T2), 4 e null, 4 e 1, 6 e null, 6 e 1.

A opção 2 e 1 Nunca vai aparecer.

# CONDICIONAL - IF...ELSE

- No **SQL Server**, **IF...ELSE** é um tipo de **instrução condicional**. Qualquer instrução T-SQL pode ser executada condicionalmente usando IF...ELSE.



- Se a condição for avaliada como **True**, as instruções T-SQL seguidas pela palavra-chave **IF** serão executadas.
- Se a condição for avaliada como **False**, as instruções T-SQL seguidas pela palavra-chave **ELSE** serão executadas.
- Uma vez, as instruções **IF** T-SQL ou **ELSE** T-SQL são executadas, outras instruções T-SQL incondicionais continuam a execução.



# CONDICIONAL - IF...ELSE

- **Regras para IF...ELSE:**

- A condição deve ser uma expressão booleana, ou seja, a condição resulta em valor booleano quando avaliada (Verdadeiro ou Falso).
- **IF... ELSE** pode manipular condicionalmente uma única instrução **T-SQL** ou bloco de instruções **T-SQL**.
- O bloco de instruções deve começar com a palavra-chave **BEGIN** e fechar com a palavra-chave **END**.
- O uso de **BEGIN** e **END** ajuda o **SQL Server** a identificar o bloco de instruções que precisa ser executado e separá-lo do restante das instruções **T-SQL** que não fazem parte do bloco **IF... ELSE T-SQL**.
- **ELSE** é opcional.

```
IF <Condition>
    {Statement | Block_of_statement}
[ ELSE
    {Statement | Block_of_statement}]
```

# CONDICIONAL - CASE

- **CASE** é a **extensão da instrução IF...ELSE**. Ao contrário do **IF...ELSE**, onde apenas o máximo de uma condição é permitido, o **CASE** permite que seja **aplicado várias condições** para executar diferentes conjuntos de ações no **SQL Server**. No **SQL Server**, existem dois tipos de **CASE**.

- **Simple CASE** (caso simples)

```
CASE <Case_Expression>
  WHEN Value_1 THEN Statement_1
  WHEN Value_2 THEN Statement_2
  .
  .
  WHEN Value_N THEN Statement_N
  [ELSE Statement_Else]
END AS [ALIAS_NAME]
```

- **Regras para Simple CASEs:**

- Um caso simples permite apenas a verificação de igualdade de **Case\_Expression** com **Value\_1** para **Value\_N**.
- Uma **Case\_Expression** é comparada com **Value**, na ordem que começa no primeiro valor. Abordagem de execução:
  - Se **Case\_Expression** for equivalente a **Value\_1**, as instruções **WHEN...THEN** adicionais serão ignoradas e a execução do **CASE** será encerrada imediatamente.
  - Se **Case\_Expression** não corresponder a **Value\_1**, **Case\_Expression** será comparado com **Value\_2** para equivalência. Esse processo de comparação de **Case\_Expression** com **Value** continuará até que **Case\_Expression** encontre um valor equivalente correspondente do conjunto **Value\_1, Value\_2, ...**
  - Se nada corresponder, o controle será direcionado para a instrução **ELSE** e o **Statement\_Else** será executado.
- **ELSE** é opcional.
- **Searched CASE** (caso pesquisado).

# CONDICIONAL - CASE

- **Searched CASE** (caso pesquisado).

```
CASE
  WHEN <Boolean_Expression_1> THEN Statement_1
  WHEN <Boolean_Expression_2> THEN Statement_2
  .
  .
  WHEN <Boolean_Expression_N> THEN Statement_N
  [ELSE Statement_Else]
END AS [ALIAS_NAME]
```

- **Regras para Searched CASEs:**

- Ao contrário do caso simples, o caso pesquisado **não se restringe apenas à verificação de igualdade**, permite a expressão booleana.
- A expressão booleana é avaliada, em ordem a partir da primeira expressão booleana, ou seja, Boolean\_expression\_1. Abordagem de execução:
  - Se **Boolean\_expression\_1** for **TRUE**, as instruções **WHEN...THEN** adicionais serão ignoradas e a execução do **CASE** será encerrada imediatamente.
  - Se **Boolean\_expression\_1** for **FALSE**, **Boolean\_expression\_2** será avaliado quanto à condição **TRUE**. Esse processo de avaliação da expressão booleana continuará até que uma das expressões booleanas retorne **TRUE**.
  - Se nada corresponder, o controle será direcionado para a instrução **ELSE** e o **Statement\_Else** será executado.
- Como o **Simple Case**, o **ELSE** também é opcional no caso de pesquisa.
- Se **ELSE** não estiver presente e nenhuma das **Boolean\_expression** retornar **TRUE**. **NULL** será exibido.

# CONDICIONAL

## - CASE

- Diferenças entre **Simple CASE** e **Searched CASE**.

Simple Case	Searched Case
<p>CASE keyword is immediately followed by CASE_Expression and before WHEN statement.</p> <p>E.g.:</p> <p>CASE &lt;Case_Expression&gt;</p> <p>WHEN Value_1 THEN Statement_1...</p>	<p>Case keyword is followed by the WHEN statement, and there is no expression between CASE and WHEN.</p> <p>E.g.:</p> <p>CASE WHEN &lt;Boolean_Expression_1&gt; THEN Statement_1...</p>
<p>In Simple Case, VALUE exists for each WHEN statement. This Values: Value_1, Value_2... Are compared with single CASE_Expression sequentially. The result gets evaluate for the TRUE/FALSE condition for each WHEN Statement.</p> <p>E.g.:</p> <p>CASE &lt;Case_Expression&gt;</p> <p>WHEN Value_1 THEN Statement_1...</p> <p>WHEN Value_2 THEN Statement_2...</p>	<p>In Searched Case, Boolean_Expression exists for each WHEN statement. This Boolean_Expressions: Boolean_Expression_1, Boolean_Expression_2,... evaluates the TRUE/FALSE condition for each WHEN Statement.</p> <p>E.g.:</p> <p>CASE</p> <p>WHEN &lt;Boolean_Expression_1&gt; THEN Statement_1...</p> <p>WHEN &lt;Boolean_Expression_2&gt; THEN Statement_2...</p>
<p>Simple Case support only equality check. I.e. whether CASE_Expression = VALUE_1, VALUE_2...</p> <p>E.g.:</p> <p>CASE &lt;Case_Expression&gt; WHEN Value_1 THEN Statement_1... In the above example, the only operation performed by the system is checking if Case_Expression = Value_1</p>	<p>With Boolean_Expression_N, Search Case support any operation which results in a Boolean value. It includes equal and not equal to operator.</p> <p>E.g.:</p> <p>CASE WHEN &lt;Boolean_Expression_1&gt; THEN Statement_1... In above example, Boolean_Expression_1 can contain both 'equal to' and 'not equal to' operator like A = B, A != B.</p>

# USUÁRIOS e PERMISSÕES

- É possível criar um novo USUÁRIO usando o comando T-SQL CREATE USER.

```
create user <user-name> for login <login-name>
```

- Atribuir permissão a um usuário refere-se às regras que **governam os níveis de acesso** que os usuários têm nos recursos protegidos do **SQL Server**.
- O **SQL Server** permite **conceder, revogar e negar** essas permissões.
- Para conceder permissão a um usuário usando **T-SQL**, primeiro selecione o banco de dados usando a instrução **USE**, então é possível atribuir a permissão a um usuário usando a instrução **GRANT**.

```
use <database-name>  
grant <permission-name> on <object-name> to <username\principle>
```

# USUÁRIO e PERMISSÃO

- Para remover permissão de um usuário usando **T-SQL**, primeiro selecione o banco de dados usando a instrução **USE**, então é possível remover a permissão de um usuário usando a instrução **REVOKE**.

```
REVOKE [ GRANT OPTION FOR ] <permission> [ ,...n ] ON  
    [ OBJECT :: ] [ schema_name ]. object_name [ ( column [ ,...n ] ) ]  
    { FROM | TO } <database_principal> [ ,...n ]  
    [ CASCADE ]  
    [ AS <database_principal> ]
```

- **permission** - especifica uma permissão que pode ser revogada em um objeto contido em esquema.
- **ALL - REVOKE ALL** revoga todas as possíveis permissões. **REVOKE ALL** é equivalente a revogar todas as 92 permissões ANSI aplicáveis ao objeto especificado (SELECT, INSERT, DELETE, UPDATE e etc).
- **GRANT OPTION** - indica que o direito de conceder a permissão especificada a outros principais será revogado. A permissão em si não será revogada.
- **CASCADE** - indica que a permissão que está sendo revogada também é revogada de outros principais aos quais ela foi concedida ou negada por esse principal.

# VIEW

- Definida como uma **tabela virtual composta por linhas e colunas de dados vindos de tabelas relacionadas** em uma **QUERY** (geralmente uma instrução **SELECT**).
- As linhas e colunas da **VIEW** são geradas dinamicamente no momento em que é feita uma referência a ela.
- **VIEW** é um **objeto de caráter permanente**, podem ser lidas por vários usuários simultaneamente.
- **VIEW** permite que seja ocultado determinadas colunas de uma tabela relacional.
- **VIEW** permite criar um código de programação muito mais limpo.
- Parar explicar uma VIEW o exemplo a seguir será utilizado:

Tabela Produtos

Id	Nome	Fabricante	Quantidade	VUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

# VIEW - Criando uma VIEW

- Criando uma VIEW.

```
CREATE VIEW vwProdutos AS
SELECT IdProduto AS Código,
       Nome AS Produto,
       Fabricante,
       Quantidade,
       VlUnitario AS [ValorUnitario],
       Tipo
FROM Produtos
```

- A saída será:

Código	Produto	Fabricante	Quantidade	Valor Unitário	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

- **VIEW** tem algumas diferenças em relação a tabela de Produtos.



# VIEW - Alterando uma View

- É possível atualizar uma **VIEW** mesmo após ela já ter sido criada e necessitar de alterações, utilizando a instrução **ALTER VIEW**.

```
ALTER VIEW vwProdutos AS
SELECT IdProduto AS Código,
       Nome AS Produto,
       Fabricante,
       Quantidade,
       VlUnitario AS [ValorUnitario],
       Tipo
FROM Produtos
WHERE VlUnitario > 499.00
```

- Após a alteração, a nova saída será:

Código	Produto	Fabricante	Quantidade	Valor Unitário	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

# VIEW - Removendo uma VIEW

- Parar excluir uma **VIEW**, use a instrução **DROP VIEW**.

```
DROP VIEW vwProdutos
```

- A exclusão de uma **VIEW** implica na **exclusão de todas as permissões que tenham sido dadas** sobre ela.

# STORED PROCEDURES

- **STORED PROCEDURE** (procedimento armazenado), é um **conjunto de comandos em SQL que podem ser executados de uma só vez**, como em uma função.
- **Armazena tarefas repetitivas e aceita parâmetros de entrada** para que a tarefa seja efetuada de acordo com uma necessidade específica.
- **STORED PROCEDURE** pode **reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento**, etc.
- **Procedimentos Locais**: criados a partir de um banco de dados do próprio usuário.
- **Procedimentos Temporários**: existem dois tipos de procedimentos temporários. **Locais**, que devem começar com **#** e **Globais**, que devem começar com **##**.
- **Procedimentos de Sistema**: armazenados no banco de dados padrão do SQL Server (Master). São identificados com as siglas **sp**, que se origina de **stored procedure**.
- **Procedimentos Remotos**: utilizadas apenas para compatibilidade.
- **Procedimentos Estendidos**: são procedimentos executadas fora do **SQL Server**.

# STORED PROCEDURES

- Para criar uma **STORED PROCEDURE** será utilizado a instrução **CREATE PROCEDURE**.

```
USE BancoDados
GO
CREATE PROCEDURE Busca --- Declarando o nome da procedure
@CampoBusca VARCHAR (20) --- Declarando variável (note que utilizamos o @ antes do nome da variável)
AS
SELECT Codigo, Descrição --- Consulta
FROM NomeTabela
WHERE Descricao = @CampoBusca --- Utilizando variável como filtro para a consulta
```

- Para executar uma **STORED PROCEDURE** é utilizado a instrução **EXECUTE** seguido pelo nome da procedure.
- Para excluir uma **STORED PROCEDURE** é necessário utilizar a instrução **DROP PROCEDURE**.

```
DROP PROCEDURE Busca.
```

# FUNÇÕES

- Funções em **T-SQL** são rotinas que retornam valores ou tabelas.
- É possível criar uma **função para agrupar um conjunto de instruções SQL**.
- **SQL Server** adiciona algumas funções internas a todos os bancos de dados. Um exemplo de função que já está embutida no **SQL Server** é DATENAME.

```
USE schooldb

SELECT name, DATENAME(YEAR, DOB) AS BIRTH_YEAR
FROM student
```

- As **funções internas** nem sempre oferecem a funcionalidade desejada.
- **SQL Server** permite que os usuários criem **funções personalizadas** de acordo com seus requisitos exatos.
- Existem **três tipos de funções definidas pelo usuário** no **SQL Server**:
  - **Funções escalares** (retorna um único valor).
  - **Funções com valor de tabela embutido** (contém uma única instrução T-SQL e retorna um conjunto de tabelas).
  - **Funções com valor de tabela de várias instruções** (contém várias instruções T-SQL e retorna o conjunto de tabelas).

# FUNÇÕES

- Para criar uma **função** no **SQL Server** é utilizado a instrução **T-SQL CREATE FUNCTION**.
- Seguindo um exemplo para explicar **funções**, será criado uma função para retornar uma formato de data específico.

```
USE schooldb
GO

CREATE FUNCTION getFormattedDate
(
    @DateValue AS DATETIME
)
RETURNS VARCHAR(MAX)
AS
BEGIN
    RETURN
        DATENAME(DW, @DateValue) + ', ' +
        DATENAME(DAY, @DateValue) + ' ' +
        DATENAME(MONTH, @DateValue) + ', ' +
        DATENAME(YEAR, @DateValue)

END
```

- **GO** - para criar uma nova instrução em lote.

# FUNÇÕES

- Parar verificar se a **função** está funcionando pode ser usado a instrução **SELECT**.

```
USE schooldb

SELECT
  name,
  [dbo].[getFormattedDate] (DOB)
FROM student
```

- A saída deve ser:

Nome	(Sem nome da coluna)
Alegre	Segunda-feira, 12 de junho de 1989
Jon	Sábado, 2 de fevereiro de 1974
Sara	Segunda-feira, 7 de março de 1988
Laura	Terça-feira, 22 de dezembro de 1981
Alan	Quinta-feira, 29 de julho de 1993

# SUBSTRING

- **SUBSTRING** é uma **função** no **T-SQL** que permite ao usuário derivar **substring** de qualquer **string** definida conforme a necessidade do usuário.

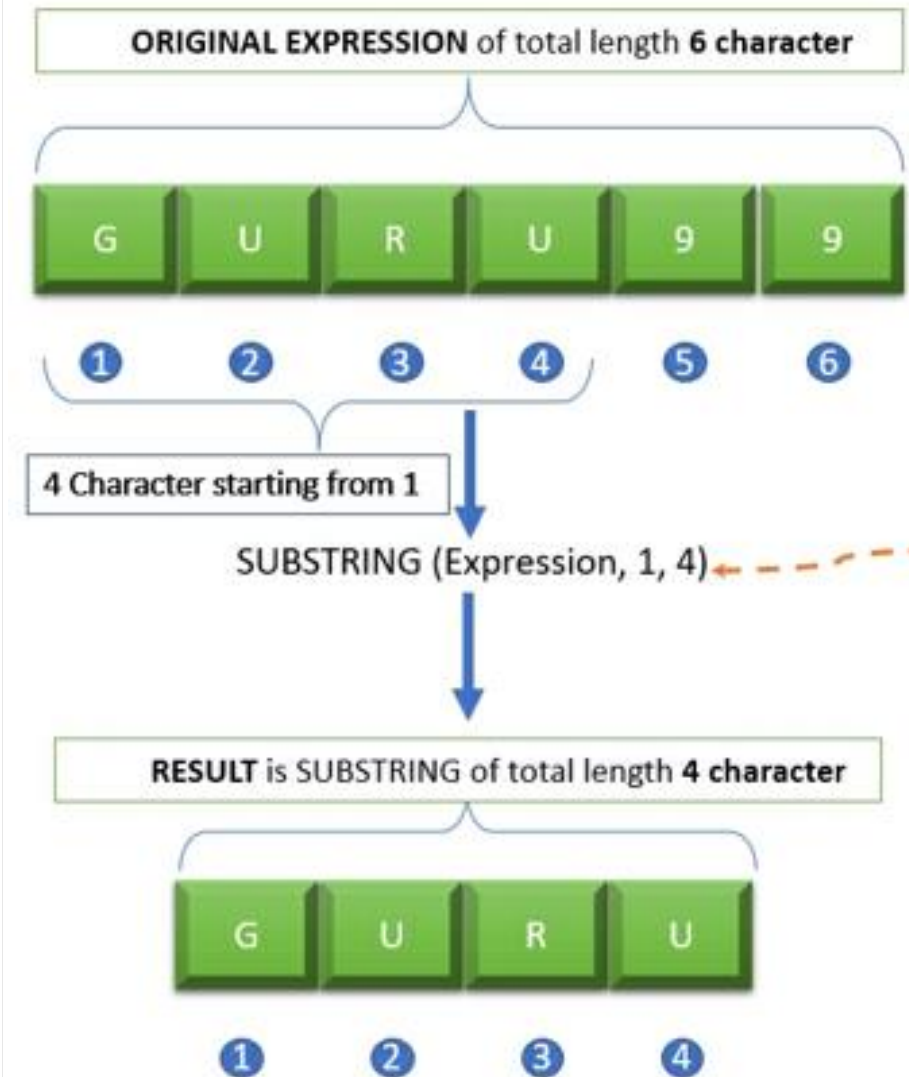
```
SUBSTRING (Expression, Starting Position, Total Length)
```

- A expressão pode ser qualquer **caractere, binário, texto ou imagem**.
- Expressão é a **string** de origem da qual buscaremos a substring conforme nossa necessidade.
- **Posição inicial** determina a posição na expressão de onde a nova substring deve começar.
- Comprimento total é o comprimento total esperado da substring de resultado da expressão, iniciando na Posição inicial.
- **Regras para usar SUBSTRING:**
  - Todos os três argumentos são obrigatórios.
  - Se a posição inicial for maior que o número máximo de caracteres em uma expressão, nada será retornado.
  - O comprimento total pode exceder o comprimento máximo de caracteres da sequência original. Nesse caso, a **substring** resultante será a sequência inteira, começando da Posição inicial na expressão até o caractere final de Expressão.



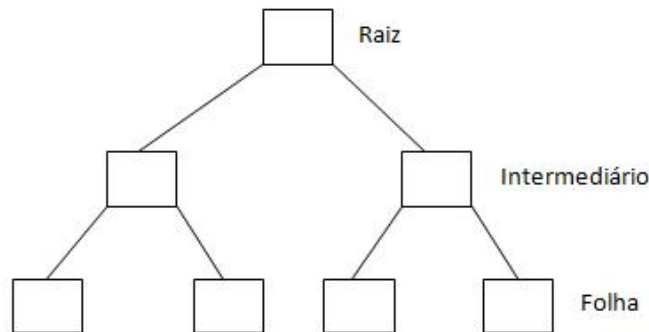
# SUBSTRING

- Representação de uma **SUBSTRING**.



# ÍNDICES

- Índices nos bancos de dados são **utilizados para facilitar a busca de informações em uma tabela com o menor número possível de operações de leituras**, tornando assim a busca mais rápida e eficiente.
- O SQL Server utiliza o mesmo princípio da lista telefônica **gravando as informações dos índices em uma estrutura chamada de B-Tree**.
- **B-Tree possui um nó-raiz que contém uma única página de dados**, uma ou mais páginas de níveis intermediários e uma ou mais páginas de níveis folhas.



- **B-Tree sempre é simétrica**, ou seja, possui o mesmo número de páginas à esquerda e à direita de cada nível.

# ÍNDICES

- No **SQL Server** é possível criar **índices clusterizados** (clustered) e **não clusterizados** (nonclustered).
- **Índices clusterizados** são **ordenados conforme a chave do cluster** fornecendo assim uma ordem de classificação para o armazenamento da tabela.
- Esta ordem de classificação não é a ordem física dos dados e sim a classificação lógica das páginas do índice.
- É possível definir somente um **índice clusterizado** por tabela, pois a mesma só pode ser ordenada de uma única maneira.
- **Índices não clusterizados não classificam ordens** e portando é possível criar até **1000 índices nonclustered** por tabela tendo cada um no **máximo 900 bytes** na chave de índice e no **máximo 16 colunas**.

# ÍNDICES

- **Campos para serem indexados a fim de ganhar desempenho:**
  - Chaves Primárias;
  - Chaves Estrangeiras;
  - Colunas acessadas por ranges (between);
  - Campos utilizados em group by ou order by;
- **Campos que não devem ser indexados:**
  - Campos dos tipos: text, image, decimais;
  - Campos calculados;
  - Campos com alta cardinalidade (Masculino ou Feminino);

# ÍNDICES

- Para criar um índice em uma tabela será utilizado a instrução **CREATE INDEX**.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH ( [ ,...n ] ) ]
[ ON { partition_scheme_name ( column_name )
      | filegroup_name
      | default
    }
]
```

- É possível criar **índice clusterizado** para campos de chave primária com a propriedade identity.

```
CREATE TABLE clientes
(
    Cod INT IDENTITY(1,1) PRIMARY KEY,
    Nome VARCHAR(100),
    UF VARCHAR(2),
    CEP VARCHAR(8)
)
```

# ÍNDICES

- Tabelas que sofrem muitas alterações (INSERT, UPDATE E DELETE) refletem essas modificações nos índices, pois acabam **deixando espaços em brancos nas páginas** dos mesmos.
- Estes espaços não utilizados refletem em maior espaço em disco e como consequência um **desperdício de tempo ao percorrer a estrutura do índice**.
- Para resolver esses problemas é necessário **manter a integridade dos índices**.

```
ALTER INDEX {nome_indice} ON REBUILD  
ALTER INDEX {nome_indice} ON REORGANIZE
```

- A opção **REORGANIZE** remove somente a fragmentação no nível folha e a opção **REBUILD** reconstrói todos os níveis do índice.

# ÍNDICES

- Os acessos aos dados das tabelas e índices podem ser de duas formas: **SEEK** ou **SCAN**.
- **SCAN**: busca em **TODOS os elementos da estrutura** (que pode ser uma tabela ou um índice). É **usado quando não possui índices** que atendam a instrução de **SELECT** ou quando a quantidade de registros que a **QUERY** retorna (em percentual) é grande.
- **SEEK**: busca **binária nos elementos de um índice**. É usado quando existe um índice que é adequado e a quantidade de registros (em percentual) retornados é pequena.
- Sendo assim, é possível executar as seguintes operações para acesso nas tabelas/índices:
  - **TABLE SCAN**: busca em todos os elementos da tabela, de forma sequencial;
  - **INDEX SCAN**: busca em todos os elementos de um índice nonclustered, de forma sequencial;
  - **INDEX SEEK**: busca binária num índice nonclustered;
  - **CLUSTERED INDEX SCAN**: busca em todos os elementos de um índice clustered, de forma sequencial;
  - **CLUSTERED INDEX SEEK**: busca binária num índice clustered.

# PROPRIEDADE E SEPARAÇÃO DO ESQUEMA DO USUÁRIO

- Um conceito central de segurança do SQL Server é que os proprietários de objetos têm permissões irrevogáveis para administrá-los.
- Você não pode remover os privilégios de um proprietário de objeto e não pode eliminar usuários de um banco de dados se eles possuírem objetos nele.



# SEPARAÇÃO DO ESQUEMA DO USUÁRIO

- A separação do esquema do usuário permite maior flexibilidade em gerenciar permissões do objeto de banco de dados.
- Um **esquema** é um contêiner nomeado para objetos de banco de dados, que permite que você agrupe objetos em **namespaces** separados.
- Por exemplo, o banco de dados de exemplo **AdventureWorks** contém esquemas para **Production**, **Sales** e **HumanResources**.
- A sintaxe de nomeação de quatro partes para referir-se a objetos especifica o nome do esquema.
  - *Server.Database.DatabaseSchema.DatabaseObject*

# ESQUEMAS INTERNOS

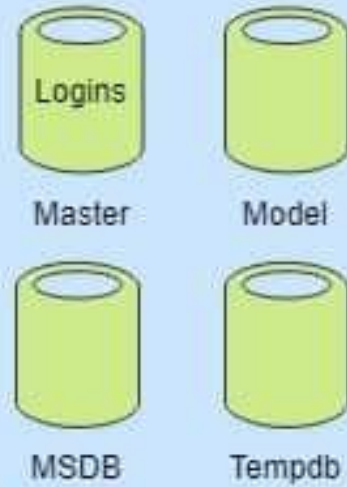
- O SQL Server vem com dez esquemas predefinidos que têm os mesmos nomes que os usuários e as funções internas do banco de dados.
- Eles existem principalmente para compatibilidade com versões anteriores. Você pode remover os esquemas que têm os mesmos nomes que as funções de banco de dados fixas se não precisar deles.
- Você não pode remover os esquemas a seguir:
  - dbo
  - guest
  - sys
  - INFORMATION\_SCHEMA

## 📌 Observação

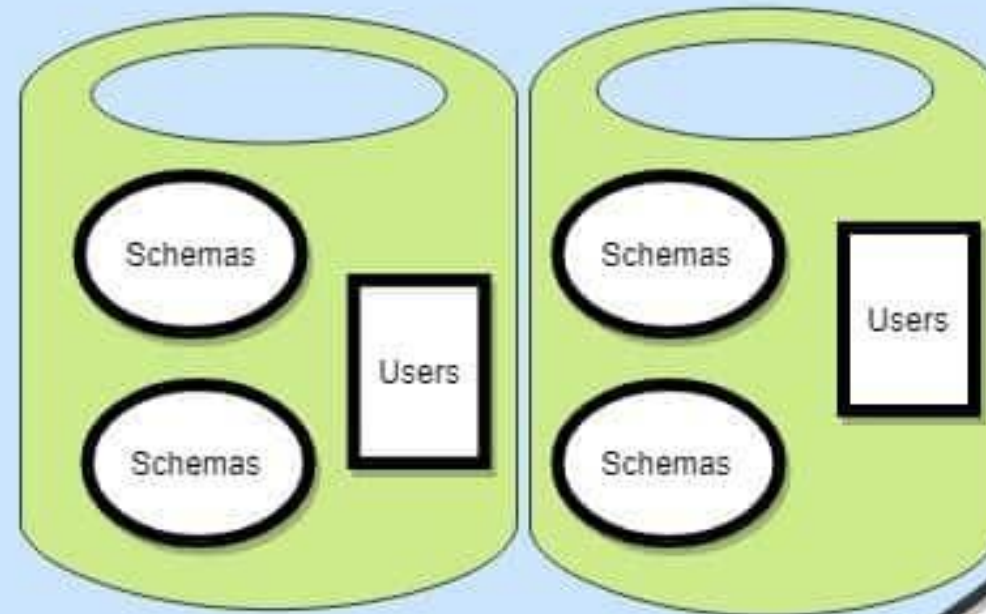
Os esquemas `sys` e `INFORMATION_SCHEMA` são reservados para objetos do sistema. Você não pode criar objetos nesses esquemas e não pode removê-los.

## Instance of SQL Server

### System Databases



### User Databases



# TRANSAÇÕES

- O SQL Server opera nos seguintes modos de transação:
  - **Transações de confirmação automática**  
Cada instrução individual é uma transação.
  - **Transações explícitas**  
Cada transação é iniciada explicitamente com a instrução **BEGIN TRANSACTION** e finalizada explicitamente com uma instrução **COMMIT** ou **ROLLBACK**.
  - **Transações implícitas**  
Uma transação nova é iniciada implicitamente quando a transação anterior é concluída, mas cada transação é explicitamente concluída com uma instrução **COMMIT** ou **ROLLBACK**.

# COMMIT WORK (TRANSACTION-SQL)

- Essa instrução funciona identicamente a **COMMIT TRANSACTION**, com exceção de que **COMMIT TRANSACTION** aceita um nome de transação definido pelo usuário.
- Esta sintaxe de **COMMIT**, com ou sem especificar a palavra-chave opcional **WORK**, é compatível com SQL-92.
- Exemplo:
  - COMMIT [ WORK ]
  - [ ; ]

# ROLLBACK WORK (TRANSACTION-SQL)

- Essa instrução funciona identicamente a **ROLLBACK TRANSACTION** exceto que **ROLLBACK TRANSACTION** aceita um nome de transação definido pelo usuário.
- Com ou sem especificar a palavra-chave opcional **WORK**, essa sintaxe **ROLLBACK** é compatível com o ISO.
- Exemplo:
  - ROLLBACK [ WORK ]
  - [ ; ]

# TRIGGER

- Embora a instrução **TRUNCATE TABLE** seja de fato uma instrução **DELETE**, ela não ativa um gatilho porque a operação não registra exclusões de linha individuais.
- Entretanto, somente usuários com permissões para executar uma instrução **TRUNCATE TABLE** precisam se preocupar em evitar inadvertidamente um gatilho **DELETE** dessa maneira.

# TRIGGER

- As seguintes instruções Transact-SQL não são permitidas em um gatilho DML:

ALTER DATABASE	CREATE DATABASE	DROP DATABASE
RESTORE DATABASE	RESTORE LOG	RECONFIGURE



# EXEMPLO

```
-- =====  
SET ANSI_NULLS ON  
GO  
SET QUOTED_IDENTIFIER ON  
GO  
-- =====  
-- Author:      <Author,,Name>  
-- Create date: <Create Date,,>  
-- Description: <Description,,>  
-- =====  
CREATE TRIGGER reminder1  
ON dbo.funcionario  
AFTER INSERT, UPDATE  
AS RAISERROR ('Funcionário Inserido', 16, 10);  
GO
```

# Severity

- É o nível de severidade definido pelo usuário associado a essa mensagem.
- Níveis de severidade de 0 a 18 podem ser especificados por qualquer usuário. Níveis de severidade de 19 a 25 podem ser especificados apenas por membros da função de servidor fixa **sysadmin**.

## ⊗ Cuidado

Níveis de severidade de 20 a 25 são considerados fatais. Se um nível de severidade fatal for encontrado, a conexão de cliente é encerrada depois de receber a mensagem, e o erro é registrado nos logs de erro e de aplicativo.

# State

- *state*  
É um número inteiro de 0 a 255. Os valores negativos usam 1 como padrão. Valores maiores que 255 não devem ser usados.
- Se o mesmo erro definido pelo usuário for gerado em vários locais, o uso de um número de estado exclusivo para cada local pode ajudar a encontrar a seção de código que está gerando os erros.

# TRIGGER


## Exemplos

Os exemplos a seguir são descritos no banco de dados AdventureWorks2012.

### a. Desabilitando um gatilho DML em uma tabela

O exemplo a seguir desabilita o gatilho `uAddress` que foi criado na tabela `Person`.

SQL


 Copiar

```
DISABLE TRIGGER Person.uAddress ON Person.Address;  
GO
```

### B. Desabilitando um gatilho DDL

O exemplo a seguir cria um gatilho DDL `safety` com escopo definido no banco de dados e depois desabilita o mesmo.

SQL

 Copiar

```
CREATE TRIGGER safety  
ON DATABASE  
FOR DROP_TABLE, ALTER_TABLE  
AS  
    PRINT 'You must disable Trigger "safety" to drop or alter tables!'  
    ROLLBACK;  
GO  
DISABLE TRIGGER safety ON DATABASE;  
GO
```

# BULK INSERT

- Importa um arquivo de dados para uma tabela ou exibição de banco de dados em um formato especificado pelo usuário no SQL Server.

SQL

 Copiar

```
BULK INSERT Sales.Orders  
FROM '\\SystemX\DiskZ\Sales\data\orders.csv'  
WITH ( FORMAT='CSV');
```



STUDY **HARD**

# GABARITO FINAL

QUESTÃO	RESPOSTA
01	LETRA B
02	LETRA B
03	ERRADO
04	LETRA A
05	LETRA B
06	LETRA B
07	LETRA C

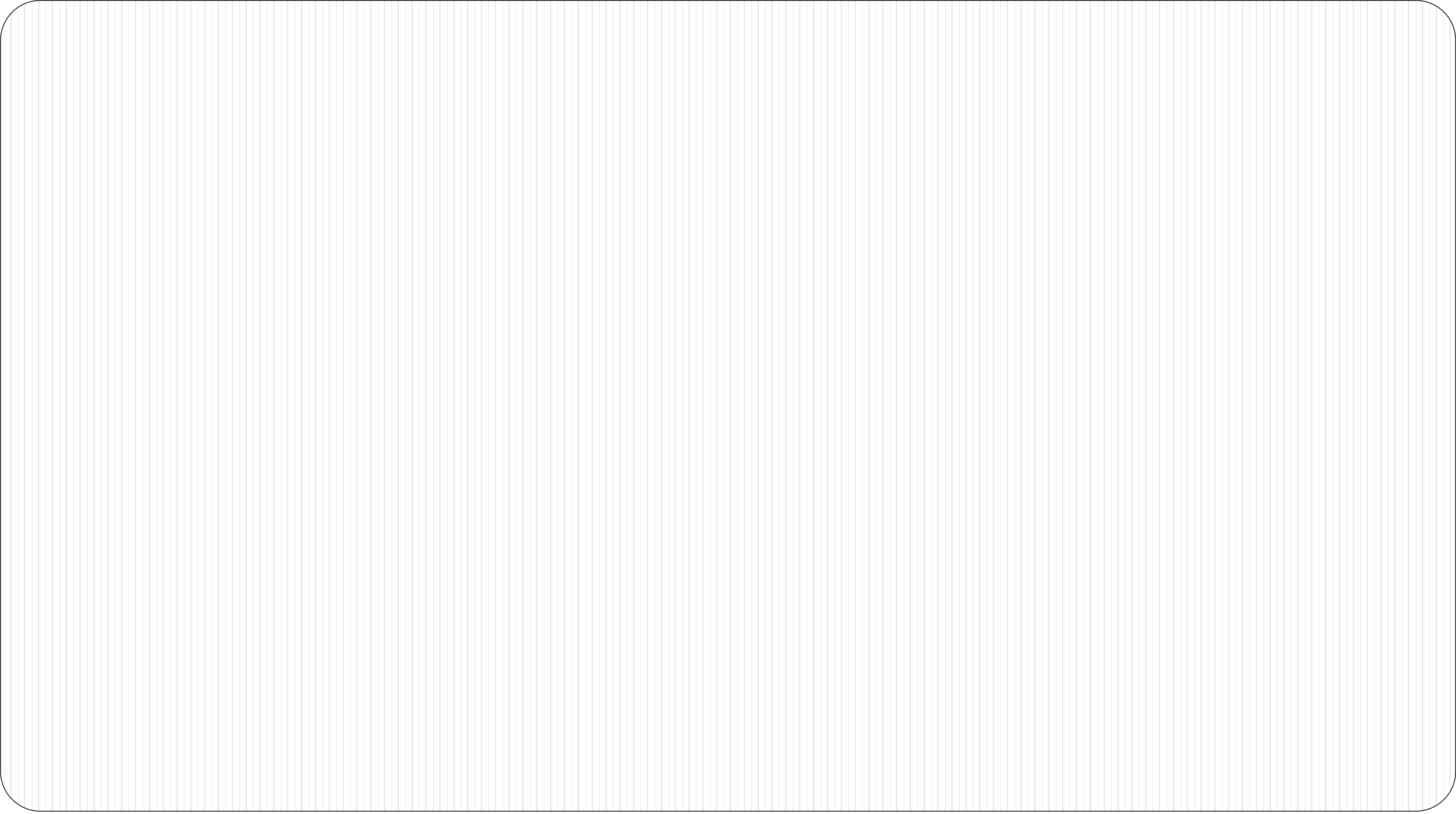
# Referências

- <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15>
- <https://www.devmedia.com.br/indices-no-sql-server/18353>
- <https://www.sqlshack.com/use-sql-server-built-functions-create-user-defined-scalar-functions/>
- <https://www.devmedia.com.br/conceitos-e-criacao-de-views-no-sql-server/22390>
- <https://www.devmedia.com.br/introducao-aos-stored-procedures-no-sql-server/7904>
- <http://www.bosontreinamentos.com.br/sql-com-sql-server/como-realizar-backup-e-restore-no-microsoft-sql-server/>



# Dúvidas?

Prof. Washington Almeida  
@profwashington.almeida - Instagram



# Banco de Dados – MS SQL SERVER

## EXERCÍCIOS

Prof. Washington Almeida, MSC, ISF 27002

# Questão 1

**Ano:** 2017 **Banca:** FGV **Órgão:** IBGE **Prova:** FGV - 2017 - IBGE - Analista Censitário - Análise de Sistemas - Desenvolvimento de Aplicações

O SQL Server fornece uma série de funções internas disponibilizadas pelo próprio sistema e também permite criar funções definidas pelo usuário. As funções internas são organizadas em categorias como, por exemplo, as categorias de funções: lógicas, de agregação, de replicação, matemáticas, criptográficas, escalares, de segurança, de classificação, etc.

É uma função de agregação do SQL Server:

- a) IIF;
- b) STDEV;
- c) RAND;
- d) ROW\_NUMBER;
- e) USER\_ID.

# Questão 1

**Ano:** 2017 **Banca:** FGV **Órgão:** IBGE **Prova:** FGV - 2017 - IBGE - Analista Censitário - Análise de Sistemas - Desenvolvimento de Aplicações

O SQL Server fornece uma série de funções internas disponibilizadas pelo próprio sistema e também permite criar funções definidas pelo usuário. As funções internas são organizadas em categorias como, por exemplo, as categorias de funções: lógicas, de agregação, de replicação, matemáticas, criptográficas, escalares, de segurança, de classificação, etc.

É uma função de agregação do SQL Server:

- a) IIF;
- b) STDEV;
- c) RAND;
- d) ROW\_NUMBER;
- e) USER\_ID.

LETRA B

## Justificativa:

- a) IFF: função lógica que Retorna um de dois valores, dependendo de a expressão booliana ser avaliada como true ou false no SQL Server.
- b) STDEV: função de agregação que retorna o desvio padrão estatístico de todos os valores da expressão especificada.
- c) RAND: retorna um valor float pseudoaleatório de 0 a 1, exclusivo
- d) ROW\_NUMBER: Retorna o número sequencial de uma linha em uma partição de um conjunto de resultados, iniciando em 1 para a primeira linha de cada partição
- e) USER\_ID: Retorna o número de identificação para um usuário de banco de dados. Essa função foi substituída pela DATABASE\_PRINCIPAL\_ID;

## Questão 2

**Ano:** 2019 **Banca:** VUNESP **Órgão:** Prefeitura de Birigui - SP **Prova:** VUNESP - 2019 - Prefeitura de Birigui - SP - Técnico em Informática

No sistema gerenciador de bancos de dados Microsoft SQL Server 2016, o comando para se declarar a exclusão condicional de uma tabela denominada T1 é:

- a) DROP TABLE IN CASE T1;
- b) DROP TABLE IF EXISTS T1;
- c) DROP TABLE FOR T1;
- d) DELETE TABLE IN CASE T1;
- e) DELETE TABLE IF THERE IS T1;

## Questão 2

**Ano:** 2019 **Banca:** VUNESP **Órgão:** Prefeitura de Birigui - SP **Prova:** VUNESP - 2019 - Prefeitura de Birigui - SP - Técnico em Informática

No sistema gerenciador de bancos de dados Microsoft SQL Server 2016, o comando para se declarar a exclusão condicional de uma tabela denominada T1 é:

- a) DROP TABLE IN CASE T1;
- b) DROP TABLE IF EXISTS T1;
- c) DROP TABLE FOR T1;
- d) DELETE TABLE IN CASE T1;
- e) DELETE TABLE IF THERE IS T1;

LETRA B

### Justificativa:

Esse comando está marcado para ser descontinuado mas ainda funciona no SQL SERVER 2016.

Exemplo do Novo comando:

IF EXISTS (SELECT \* FROM sys.triggers WHERE name = 'trProductInsert') DROP TRIGGER trProductInsert

[docs.microsoft.com/pt-br/archive/blogs/sqlserverstorageengine/drop-if-exists-new-thing-in-sql-server-2016](https://docs.microsoft.com/pt-br/archive/blogs/sqlserverstorageengine/drop-if-exists-new-thing-in-sql-server-2016)

# Questão 3

**Ano:** 2018 **Banca:** FGV **Órgão:** Câmara de Salvador - BA **Prova:** FGV - 2018 -  
Câmara de Salvador - BA - Analista de Tecnologia da Informação

No MS SQL Server, a especificação de chaves estrangeiras (*foreign keys*) admite o uso das cláusulas ON UPDATE e ON DELETE.

As opções que complementam essas cláusulas são:

- a) ERROR, CASCADE, SET NULL, SET TIME
- b) RESUME, UPDATE, SET NULL, SET EMPTY
- c) STOP, DELETE, SET ZERO, SET NULL
- d) ERROR, SET NULL, SET DEFAULT
- e) NO ACTION, CASCADE, SET NULL, SET DEFAULT



# Questão 3

**Ano:** 2018 **Banca:** FGV **Órgão:** Câmara de Salvador - BA **Prova:** FGV - 2018 -  
Câmara de Salvador - BA - Analista de Tecnologia da Informação

No MS SQL Server, a especificação de chaves estrangeiras (*foreign keys*) admite o uso das cláusulas ON UPDATE e ON DELETE.

As opções que complementam essas cláusulas são:

- a) ERROR, CASCADE, SET NULL, SET TIME
- b) RESUME, UPDATE, SET NULL, SET EMPTY
- c) STOP, DELETE, SET ZERO, SET NULL
- d) ERROR, SET NULL, SET DEFAULT
- e) NO ACTION, CASCADE, SET NULL, SET DEFAULT

LETRA E

**Justificativa:**

```
| FOREIGN KEY ( column [ ,...n ] ) REFERENCES referenced_table_name [ ( ref_column [ ,...n ] ) ] [ ON DELETE { NO  
ACTION | CASCADE | SET NULL | SET DEFAULT } ] [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]  
[ NOT FOR REPLICATION ]
```

# Questão 4

**Ano:** 2018 **Banca:** VUNESP **Órgão:** SAAE de Barretos - SP **Prova:** VUNESP - 2018 - SAAE de Barretos - SP - Assistente de Tecnologia da Informação

No sistema gerenciador de bancos de dados Microsoft SQL Server 2016, quando um novo banco de dados é criado, são gerados, pelo próprio sistema gerenciador, alguns bancos de dados do sistema, sendo que dois desses bancos de dados são:

- a) initial e final.
- b) master e msdb.
- c) small e principal.
- d) preliminar e accdb.
- e) primary e secondary.

# Questão 4

**Ano:** 2018 **Banca:** VUNESP **Órgão:** SAAE de Barretos - SP **Prova:** VUNESP - 2018 - SAAE de Barretos - SP - Assistente de Tecnologia da Informação

No sistema gerenciador de bancos de dados Microsoft SQL Server 2016, quando um novo banco de dados é criado, são gerados, pelo próprio sistema gerenciador, alguns bancos de dados do sistema, sendo que dois desses bancos de dados são:

- a) initial e final.
- b) master e msdb.
- c) small e principal.
- d) preliminar e accdb.
- e) primary e secondary.

LETRA B

**Justificativa:**

Sql SERVER inclui os seguintes system databases:

**master, msdb, model, Resource e tempdb;**

<https://docs.microsoft.com/en-us/sql/relational-databases/databases/system-databases?view=sql-server-ver15>

# Questão 5

**Ano:** 2009 **Banca:** CESPE **Órgão:** CEHAP-PB **Prova:** CESPE - 2009 - CEHAP-PB - Programador de computador

O SQL Server 2008 é um banco de dados com diversas características que o destacam no mercado. Julgue os itens a seguir, com base nas características desse programa.

I - O SQL Server 2008 permite a criptografia de um banco de dados inteiro.

II - O SQL Server 2008 proíbe que os usuários criem e gerenciem auditorias DDL.

III - No SQL Server 2008, os clientes podem adicionar database mirroring sem mudança nas aplicações.

IV - O SQL Server 2008 incapaz de adicionar recursos de memória online.

Estão certos apenas os itens

- a) I e II.
- b) I e III.
- c) II e IV.
- d) III e IV.

# Questão 5

**Ano:** 2009 **Banca:** CESPE **Órgão:** CEHAP-PB **Prova:** CESPE - 2009 - CEHAP-PB - Programador de computador

O SQL Server 2008 é um banco de dados com diversas características que o destacam no mercado. Julgue os itens a seguir, com base nas características desse programa.

I - O SQL Server 2008 permite a criptografia de um banco de dados inteiro.

II - O SQL Server 2008 proíbe que os usuários criem e gerenciem auditorias DDL.

III - No SQL Server 2008, os clientes podem adicionar database mirroring sem mudança nas aplicações.

IV - O SQL Server 2008 incapaz de adicionar recursos de memória online.

Estão certos apenas os itens

- a) I e II.
- b) I e III.
- c) II e IV.
- d) III e IV.

**LETRA B**

## **Justificativa:**

*Espelhamento de banco de dados* é uma solução para aumentar a disponibilidade de um banco de dados do SQL Server . O espelhamento é implementado por base de banco de dados e só funciona com bancos de dados que usam o modelo de recuperação completa. <https://www.microsoft.com/brasil/servidores/sql/techinfo/whitepapers/sql2008Overview.msp>

## Questão 6

**Ano:** 2016 **Banca:** CESPE **Órgão:** TCE-PA **Prova:** CESPE - 2016 - TCE-PA - Auditor de Controle Externo - Área Informática - Analista de Suporte

Acerca da configuração e administração dos bancos de dados SQL Server 2008 R2 e MySQL 5.7, julgue o item subsequente.

Caso a senha de uma conta do SQL Server 2008 R2 seja alterada, a nova senha entrará em vigor imediatamente, sem a necessidade de reinicialização do SQL Server.

Certo

Errado

## Questão 6

**Ano:** 2016 **Banca:** CESPE **Órgão:** TCE-PA **Prova:** CESPE - 2016 - TCE-PA - Auditor de Controle Externo - Área Informática - Analista de Suporte

Acerca da configuração e administração dos bancos de dados SQL Server 2008 R2 e MySQL 5.7, julgue o item subsequente.

Caso a senha de uma conta do SQL Server 2008 R2 seja alterada, a nova senha entrará em vigor imediatamente, sem a necessidade de reinicialização do SQL Server.

Certo

Errado

**CERTO**

# Gabarito

Questão	Resposta
1	<i>B</i>
2	<i>B</i>
3	<i>E</i>
4	<i>B</i>
5	<i>B</i>
6	<i>CERTO</i>





STUDY **HARD**