SQL – SERVER

História do SQL

Nos anos 60 surgiram os **primeiros computadores, que eram simples máquinas de cálculos.** Tinham um tamanho grande, chegavam a ocupar imensas salas refrigeradas e seu poder computacional era menor que a calculadora de mesa que temos atualmente.

A funcionalidade desses computadores era simplesmente realizar contas básicas, como calcular o salário de um funcionário, por exemplo. A lista de salários e nomes de servidores **ficavam armazenados em imensas fitas magnéticas** que eram introduzidas no computador, o que demarca o início do conceito de banco de dados.

Ao final da década de 60, os dados passaram a ser armazenados em estruturas internas dos computadores, surgindo, assim, os primeiros discos rígidos, que guardavam somente alguns canais de informação suficientes para resolver alguns problemas. Essas informações internas ao computador representam banco de dados primitivos. Eles eram baseados em tabelas que expressavam apenas uma lista sequencial das informações que seriam lidas, por isso eram chamados de banco de dados sequenciais (BDs).

No início dos anos 70, os BDs já eram usados em larga escala porque se mostraram eficientes no processamento de grandes volumes de dados. Nessa época, alguns cientistas da área tecnológica passaram a imaginar outro modelo de banco de dados que pudesse se adequar melhor à realidade das empresas a fim de organizar suas transações. Ao invés de serem grandes calculadoras, passariam a ser ferramentas de auxílio para controlar a estrutura administrativa, controle de estoque, vendas, entre outros. A estrutura de banco de dados sequencial dificultava a utilização dos computadores para gerenciar os processos de uma empresa, por isso a necessidade de criar outro tipo de banco de dados.

A partir dessa necessidade, surge a ideia de modelo relacional - que ainda não se trata de banco de dados relacional. Esse modelo teórico tem como metodologia a representação das transações operacionais de uma empresa baseado no diagrama de relacionamento e utilização da teoria de conjuntos. Ele serviu como base para imaginarmos um banco de dados que implementaria, na prática, toda teoria proposta pelos cientistas ao criarem os modelos relacionais, dando início, assim, ao desenho dos primeiros bancos de dados relacionais.

Com os bancos de dados relacionais, passou-se a imaginar os dados armazenados em tabelas diferentes para cada entidade, podendo as tabelas manterem relações entre si imitando as relações das entidades - o que se encaixava muito bem no controle de processos de empresas.

Por volta da metade dos anos 70, vê-se os primeiros programas usando a modelagem de banco de dados relacional, e não mais as tabelas sequenciais

comumente utilizadas. Junto com as primeiras implementações, pensou-se em criar uma linguagem que permitisse aproveitar ao máximo as estruturas relacionais. A primeira empresa a pensar nessa questão foi a *IBM*, através do projeto **System R**, coordenado pelo cientista *Codd*, um dos grandes colaboradores do modelo teórico relacional.

Por volta do final da década de 70, a *IBM* começou a trabalhar em uma linguagem para consultar dados dos bancos de dados relacionais. Paralelamente, outras empresas também passaram a implementar seus próprios bancos relacionais e trabalhar em linguagens para consultá-los, entre elas podemos citar a *Oracle* e a *Sybase*, sendo esta última o embrião do surgimento do banco de dados *SQL Server*.

Em meio a este cenário, surge o ANSI (American National Standards Institute), um órgão governamental americano que cuida dos pesos e medidas dos Estados Unidos. Ele tinha como objetivo criar um padrão único entre todas as linguagens que estavam sendo desenvolvidas para consultar bancos de dados relacionais, a fim de facilitar a difusão do conhecimento e a implementação dos diversos bancos que estavam surgindo.

As empresas entraram em consenso e começaram a criar um padrão de linguagem para a utilização de todos. Pensando em adotar a linguagem coloquial do inglês, criou-se a *SEQUEL*, sigla para *Structure English Query Language*, traduzida para o português como *Linguagem de Consulta Estruturada em Inglês*, que posteriormente passou a ser chamada apenas de *SQL* (Structure Query Language), sendo conhecida, também como *SQL* ANSI ou padrão ANSI.

Vale ressaltar que, embora a sigla tenha perdido o termo "*English*", ainda é comum que ela seja chamada de *SEQUEL* pelos estadunidenses.

A partir dos anos 90, até o dia de hoje, as empresas voltam a se reunir periodicamente para criar novos padrões ANSI, nomeando-os com o sufixo do ano em questão, como, por exemplo *ANSI 98*, *ANSI 2002*, e assim por diante. Basicamente, os padrões são revisados para novas estruturas de dados.

Vantagens e desvantagens do padrão SQL

Como vantagens, podemos citar aprendizado, portabilidade, longevidade, comunicação e liberdade de escolha.

Aprendizado

O profissional que conhece *SQL* conseguirá transitar facilmente entre os diferentes tipos de bancos de dados relacional já que todos utilizam o padrão ANSI, embora possa haver pequenas diferenças entre as plataformas.

Portabilidade

É fácil migrar sistemas, ou seja, pegar a estrutura de um banco e adotar em outro. Quando mais fácil a implementação usando padrão ANSI, mais fácil se torna a migração entre plataformas.

Longevidade

Existe uma regra de que as implementações do padrão ANSI não deixam de funcionar mesmo que seja criada uma nova versão. Sendo assim, muitas funcionalidades que existem hoje foram criadas na primeira versão, logo, há uma garantia de que o sistema funcionará por muito tempo, independente do surgimento de novas versões do padrão ANSI.

Comunicação

Se os bancos de dados relacionais falam padrão ANSI, é fácil realizar uma comunicação entre eles. É possível, por exemplo, comunicar um banco de dados *SQL* e um banco de dados *Oracle*, pois ambos utilizam padrão ANSI.

Liberdade de escolha

Quando há um padrão de linguagem e é necessário selecionar um banco de dados relacional, a escolha torna-se mais fácil, já que os bancos falam a mesma linguagem.

Agora vamos às desvantagens, dentre as quais podemos citar a falta de criatividade e de estrutura. Vejamos a seguir.

Falta de criatividade

O *SQL* possui alguns limites que podem não atender às demandas do mercado, um exemplo são as redes sociais. Os bancos de dados relacionais estão atrelados à modelagem de processos que são, geralmente lineares. No entanto, quando se trata de redes sociais, não há um caminho único, por isso torna-se difícil de modelar uma rede social utilizando esse tipo de banco.

Inclusive, pensando nesse empecilho, foram criados outros tipos de banco de dados que são chamados de *noSQL* e atendem melhor a modelagem de sistemas de redes, mas sua estrutura é completamente diferente dos relacionais e fogem do padrão ANSI.

Falta de estrutura

Mais adiante, veremos que a linguagem *SQL* não é tão bem estruturada. Embora tenha muitos comandos, não há na linguagem de padrão ANSI, de forma nativa, comandos de repetição comuns à muitas linguagens, como **while** e **for**, por exemplo. Para suprir essa falta, os bancos relacionais criaram suas próprias linguagens internas que adicionam essa estrutura de repetição aos comandos *SQL*, mas isso foge ao padrão ANSI, já que cada banco fez sua própria implementação.

Conjunto de comandos

À medida que o padrão ANSI foi evoluindo, deixou de ser uma linguagem apenas para consulta e passou a cobrir grupos de comandos direcionados à manutenção e administração dos bancos de dados relacionais. Por isso, hoje os comandos *SQL* dividem-se em 3 grupos. Veremos quais são.

DDL (Data Definition Language)

Os comandos **DDL** criam os componentes do banco de dados, ou seja, as entidades, além de fazer a manutenção de sua estrutura. Sendo assim, a produção de tabelas e índices, bem como sua manutenção, são feitas através destes comandos. Entre eles, podemos citar os seguintes:

- create - cria tabelas, índices e outras estruturas;
- alter - altera as propriedades das entidades;
- truncate - apaga definitivamente os dados de uma tabela;
- drop - apaga não apenas os dados, mas própria tabela do banco de dados;
- rename muda o nome de alguns componentes do banco.

DML (Data Manipulation Language)

São usados para gerenciar os dados do banco, ou seja, informações que estão, por exemplo, dentro das tabelas. Alguns exemplos de comandos **DML** são:

- insert - inclui dados em uma tabela;
- update - altera dados de uma tabela;
- delete - apaga dados da tabela;
- lock - gerencia concorrência da atualização de dados de uma tabela.

DCL (Data Control Language)

É o conjunto de comandos que permite administrar o banco de dados. Essa administração não trata da estrutura do banco, mas sim de componentes como segurança, usuário e forma de armazenação dos dados. Entre estes comandos, podemos citar os seguintes:

- commit - salva o estado do banco de dados de forma definitiva;
- rollback - retorna ao estado salvo previamente;
- savepoint - salva o estado do banco de forma temporária.

Portanto, estes são os três grupos de comandos: **DDL**, **DML** e **DCL**. Falaremos mais sobre eles no decorrer do curso.

História do SQL Server

O *Microsoft SQL Server* surgiu a partir do banco de dados *Sybase*. Em 1988, a *Microsoft* resolveu adotar o *Sybase* como complemento do sistema operacional *Windows NT* - uma das primeiras versões do *Windows*, ainda baseada no antigo *DOS*, voltado para o mercado corporativo. A partir daí o banco de dados foi evoluindo, porém, em 1994, as

empresas *Microsoft* e *Sybase* sofreram uma ruptura e seguiram caminhos distintos.

Em 1995 surge a versão 6.0 do *SQL Server*, que aumentou substancialmente a performance do banco, provendo mecanismos de replicação de base e administração centralizada. No ano seguinte, em 1996, a versão 6.5 trouxe melhorias significativas para a tecnologia de banco de dados, disponibilizando novas funcionalidades.

Em 1997, a *Microsoft* lançou a versão Enterprise do *SQL* 6.5. Até então, a empresa enxergava o banco de dados de maneira departamental. Com essa nova versão, no entanto, passou-se a vê-lo como um banco de dados corporativo, voltado para o mercado empresarial. Na versão 7.0, lançada em 1998, o software foi totalmente reescrito.

Em 2000, o ano passa a compor o nome da versão do *SQL Server* em que o software é lançado, deixando de usar os números sequenciais. Na versão 2000, ele passa a ser construído sobre um novo framework. 3 anos depois, em 2003, passou-se a usar a versão 64 bits do 2000.

Em 2005, a versão ganhou o codinome Yukon e foi a primeira versão do *SQL Server* integrada com a recém-criada plataforma *.NET*. Nesse mesmo ano surgiram casos de sucesso no processamento de grandes volumes, como, por exemplo, a administração das operações de bolsas de valores adotada pela *Bovespa*. Vale ressaltar que, até esse ano, o *SQL Server* não era enxergado como um competidor direto do *Oracle*, que era consolidado no mercado como o único banco de dados relacional capaz de suportar grandes volumes e transações.

Em 2008 é lançada a versão Business Intelligence. À época, o banco de dados ia além do relacional e passou a receber novas funcionalidades, como armazenar dados geográficos e controlar cargas por usuário, por exemplo. Essa versão, no entanto, apresentou muitos bugs, e no mesmo ano foi lançada a versão 2008 R2, que aprimorava a versão de 2008 solucionando os bugs.

No ano de 2012, a versão do *SQL Server* passou a receber instâncias de *Clusters* e *Failover*, ou seja, servidores que funcionam em paralelo para o caso de falha. Com essa funcionalidade, ele finalmente passa a concorrer diretamente com o *Oracle*, que já tinha um ambiente de *Clusters* na sua versão original.

Dois anos depois, em 2014, foram introduzidas novidades como, por exemplo, suporte a ambientes virtuais e de nuvem. Em 2016, a *Microsoft* abandonou completamente a versão 32 bits e todas as versões, a partir de então, foram de 64 bits. Em 2017, a grande novidade foi uma versão do *SQL Server* para *Linux* e em 2019 passou a suportar *Big Datas*, além de melhorias pontuais, como funcionalidades de *Machine Learning*, por exemplo.

Finalmente chegamos à versão atual, de 2022. Nesta versão há uma integração completa que chamamos de SQL Server Azure. O *Azure* é a nuvem

da *Microsoft* e a empresa já permitia que houvesse instâncias de *SQL Server* nesta nuvem. Agora, com a versão 2022, essa integração torna-se quase nativa, portanto, a *Microsoft* está, cada vez mais, migrando para o *Azure* e abandonando a versão comumente instalada no servidor.

Definição das entidades

Vale ressaltar que as linhas são chamadas de **registros**, e as colunas, de **campos,** portanto, esteja atento à essa nomenclatura, pois podemos chamar cada um por ambos os nomes (linha e registro; coluna e campo).

Quando definimos o campo, a propriedade mais importante é o seu nome, que deve respeitar algumas regras. Esse nome deve ser único, portanto, não podemos ter mais de um campo com a mesma nomenclatura.

Valor vazio e a ausência de valor

Chaves primárias(primary key)

Em uma tabela, podemos classificar um ou mais campos como sendo campos de **chaves primárias**. Quando temos uma coluna assim definida, os valores não podem se repetir dentro das linhas desse campo.

No caso de chaves primárias compostas, em que temos mais de um campo como chave primária da tabela, o conjunto de valores entre esses campos não pode se repetir em outras linhas.

Vamos à um exemplo para melhor entender o conceito de chave primária.

Supondo que temos uma tabela com cadastro de cliente, sabemos que CPF é um número único para cada indivíduo. Se definimos que o campo CPF é uma chave primária, não podemos ter clientes com o mesmo CPF. Outros exemplos de seu uso, são siglas de estados e códigos de produtos.

A chave primária é usada para que busquemos determinada linha, mas seu uso não é obrigatório, portanto, uma tabela pode existir sem ter chaves primárias.

Chaves estrangeiras (foreign key) e cardinalidade

Nas aulas anteriores, falamos sobre a existência dos bancos de dados sequenciais que antecederam os relacionais. A grande diferença entre eles, é que no relacional as tabelas se relacionam, e essa relação se dá por uma entidade chamada **chave estrangeira**.

Esta chave estabelece uma relação entre 2 campos de tabelas diferentes, que devem ter as mesmas propriedades. Ela liga o campo normal de uma tabela com um campo que é chave primária de outra, e a relação de **cardinalidade** entre esses campos deve ser de 1 para N **(1:N)**.

Vamos supor que temos uma tabela de estado e outra de clientes. Na tabela de clientes, temos um campo que representa o estado onde cada um mora. Nessa tabela, o identificador desse cliente é a chave primária, ou seja, o código do

cliente que não se repete. Mas o campo de estado pode se repetir, já que é possível ter mais de um cliente no mesmo estado.

Este campo de estado, na tabela de clientes, tem uma chave estrangeira com o campo estado da tabela de estados. Ou seja, na tabela de estados, esse campo é a chave primária, já que as siglas dos estados são únicas e não podem se repetir.

Ao criar a chave estrangeira, é criado um vínculo entre esses dois campos, e não podemos, portanto, ter um cliente com um estado que não esteja presente na tabela de estados. Caso isso aconteça, um erro será gerado.

Dito isso, chamamos de **cardinalidade 1:N** porque um estado da tabela de estados pode ter várias ocorrências na tabela de clientes (N), já um estado da tabela de clientes somente se associa a um componente/estado da tabela de estados (1). Já que a cardinalidade de um conjunto é uma medida do número de elementos do conjunto, podem existir diversos tipos de cardinalidade.

Índice (index)

O índice é uma estrutura que auxilia a fazer buscas mais rápidas associadas à coluna que se relaciona com o index. Uma tabela pode ter um ou mais índices associados à diferentes campos.

Vamos supor que temos uma tabela de cliente com um campo de hobbie que traz a informação do que o cliente gosta de fazer. Se desejássemos selecionar todos os clientes que gostam de futebol, mas não tivéssemos um índice associado a esse campo hobbie, o SQL Server testará linha por linha para separar as que atendem a essa condição de busca. Com o índice a busca é mais rápida, porque ele grava a posição das linhas onde cada ocorrência aparece. Logo, se temos um índice e queremos listar os que gostam de futebol, basta o SQL Server ir no índice e buscar a ocorrência futebol.

O banco de dados cria automaticamente índices para os campos que são chave primária de uma tabela. Isso porque, sempre que uma nova linha é incluída, o banco precisa testar, pela chave primária, se aquele valor já existe, então ter um índice associado facilitar essa verificação.

Outra característica importante do index, é que ele auxilia na obtenção de relatórios. Vamos supor que temos uma tabela de venda de clientes e desejamos saber a soma das vendas por estado. Neste caso, ter um índice associado ao estado, facilita a identificação das linhas de ocorrência do estado e, consequentemente, a soma dos valores.

Embora traga benefícios, em casos de tabelas com muitos índices, gera lentidão na hora de atualizar a tabela, já que a cada alteração de dados, os índices também precisarão ser atualizados.

Em suma, um banco de dados SQL Server é formado por tabelas, que possuem campos com características previamente definidas. Alguns desses campos podem ser definidos como chaves primárias, que terão índices

automaticamente criados, e o relacionamento entre essas tabelas é feito pelas chaves estrangeiras. Eventualmente, podemos criar índices também em campos que não sejam chaves primárias.

Visões (views)

As visões são como tabelas lógicas e não existem fisicamente no banco, mas podemos vê-las como se fosse uma tabela. Para visualizá-las, basta realizarmos uma consulta. Todo resultado de uma consulta é estruturado como tabela, mas esse resultado não é físico e a tabela não está no banco de dados, mas na memória. Quando falamos de consulta, estamos nos referindo à exibição de algum dado, como parte de uma tabela, utilizando filtro, por exemplo.

Podemos criar uma visão associada ao resultado de uma consulta e tratá-la como uma nova tabela do banco de dados. Quando houver alguma manipulação da visão, o SQL resolve a consulta associada à visão, coloca o resultado em memória e o transforma em uma tabela lógica, que só existirá quando estivermos usando a visão. Vale ressaltar que não podemos incluir dados através de uma visão, pois ela é utilizada apenas para auxiliar na construção de outros relatórios.

Procedimentos (procedures) e funções (functions)

Anteriormente, falamos que uma das desvantagens do SQL é não ter uma linguagem estruturada e não ter comandos de repetição, como while, for e if, por exemplo. No entanto, os bancos de dados compensaram essa falta de estrutura criando suas próprias linguagens de programação estruturada, juntando comandos de repetição e de SQL.

No SQL Server, o nome dessa linguagem é **TRANSACT SQL**, e muitas vezes precisamos fazer complexas manipulações no banco de dados possíveis somente através de um script nessa linguagem.

Trigger (gatilho)

Trigger é uma regra que se cria no banco de dados para executar alguns comandos à medida que determinados eventos ocorram. Esses comandos podem ser SQL ou Transact SQL.

Existem alguns eventos-chave do SQL Server que podem disparar os gatilhos e executar outros comandos. São eles inclusão, alteração ou exclusão, portanto, podemos criar uma trigger associada a um desses eventos ou ao conjunto deles.

Recapitulando

O banco de dados é composto de tabelas, visões, procedimentos, funções e triggers. As tabelas possuem campos, que tem característica previamente definidas. Os campos, por sua vez, podem ser chaves primárias das tabelas. Temos os índices associados aos campos e alguns campos podem ter chaves

estrangeiras que se relacionam com outros campos de outras tabelas, que serão, obrigatoriamente, chaves primárias.

Um **dicionário de dados** é o registro da criação de dados, portanto, é como se fosse um metadados dos componentes do banco.

A **chave primária** é um campo que não pode ser repetido

Não é possível definir/adicionar uma **restrição primary key** em uma coluna anulável.

Já sabemos que uma chave primária não pode ser criada em campos que aceitem valores nulos.

A diferença é que os bancos de dados sequenciais, do início da informática, armazenavam qualquer coisa sem testar as regras de negócio, isto é, sem testar as restrições. Já o banco de dados relacionais oferece a possibilidade de incluirmos restrições ao próprio banco. Por exemplo, não ter produtos com o mesmo código.

Tabelas = **Tables**

Visões = **Views**

Programação(**Programmability**) - onde temos procuderes ou functions

dbo., que é uma entidade chamada "esquema". O esquema seria como a classificação das minhas tabelas, posso ter um banco de dados com tabelas relacionadas com vendas, tabelas relacionadas com produção, tabelas de estoque.

Posso criar um esquema chamado vendas.”nome de uma tabela”

Exemplo: vendas.tabela

Quando não definimos um “esquema”, ele é associado ao padrão **dbo..**

Exemplo: dbo.tabela

Tabelas são formadas por Campos

Coluna(Columns) são os “Campos”

Registros são as “Linhas da Tabela”

Chaves(Keys) são as “Chaves estrangeiras ou primarias”

Gatilhos(Triggers)

Índices(Indexes) estrutura auxiliar que nos ajuda acha algum determinado componente dentro do banco

O SQL não e case sensitive

“SUSCOS_VENDAS_01.mdf” o **“mdf”** indica que e o arquivo de dados. Si eu colocar um dado no bancos de dados, vai ficar gravado dentro do arquivo “**.mdf**”.

“SUSCOS_VENDAS_01_log.ldf” o “**ldf**” indica que e um arquivo de transações. Transação são de comandos gravados para o caso de desejarmos recuperar o estado do banco

Criando um banco de dados chamado “SUCOS_VENDAS_01”

```
CREATE DATABASE SUCOS_VENDAS_01
```

Criando um banco de dados em outro diretório:

```
CREATE DATABASE SUCOS_VENDAS_02
ON (NAME = 'SUCOS_VENDAS.DAT',
        FILENAME = 'C:\TEMP2\SUCOS_VENDAS_02.MDF',
        SIZE = 10MB,
        MAXSIZE = 50MB,
        FILEGROWTH = 5MB)
```

Nessa criação, passaremos alguns parâmetros, como:

- um nome interno, definido na variável NAME;
- o caminho da pasta na qual queremos criá-lo, passado na variável FILENAME - aqui usaremos de exemplo uma pasta aleatória do computador, chamada **TEMP2**;
- o tamanho inicial, SIZE;
- um tamanho máximo, MAXSIZE;
- a taxa de crescimento desse banco, FILEGROWTH;

A taxa de crescimento funciona da seguinte maneira: o banco inicia com 10MB. Quando passa desse valor, ganha mais 5MB, o que não significa que tenha esse valor em dados, mas sim uma área reservada. Quando essa área reservada for preenchida e chegar em 15, recebe mais 5, somando 20, e assim sucessivamente, até atingir os 50MB definidos como tamanho máximo.

```
LOG ON
(NAME = 'SUCOS_VENDAS.LOG',
        FILENAME = 'C:\TEMP2\SUCOS_VENDAS_02.LDF',
        SIZE = 10MB,
        MAXSIZE = 50MB,
        FILEGROWTH = 5MB);
```

Se quisermos passar informações sobre o log, podemos utilizar **LOG ON** e repetir os parâmetros definidos, fazendo algumas alterações, como, por exemplo, mudar **.DAT** para **.LOG**, no nome interno, e **.MDF** para **.LDF**, no caminho. É possível, ainda, alterar a taxa de crescimento, mas manteremos a mesma

O que é o arquivo LOG da base de dados do SQL Server?

Trata-se do arquivo de transações que guardam os comandos executados para futura recuperação de dados se necessário for.

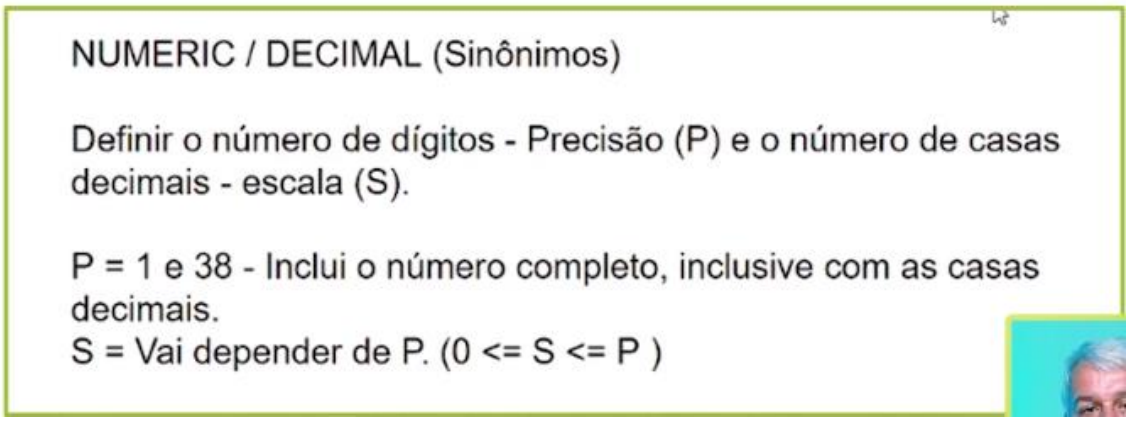
É o Log de transações que permite recuperar o estado do banco a qualquer momento.

Tipos de campos – Números

Tipos de dado	Intervalo	Armazenamento
bigint	-2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807)	8 bytes
int	-2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647)	4 bytes
smallint	-2^15 (-32.768) a 2^15-1 (32.767)	2 bytes
tinyint	0 a 255	1 bytes

Os numéricos exatos **(bigint, int, smallint e tinyint)** são números inteiros, ou seja, que não possuem casas decimais.

Numéricos exatos (com casas decimais)



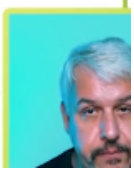
Outro tipo que se adequa à numéricos exatos são os números com casas decimais **previamente definidas**. Temos 2 tipos, que na verdade são sinônimos: **NUMERIC** e **DECIMAL**. São basicamente iguais, então não faz diferença criarmos o campo com um ou com outro.

Quando definimos um campo desse tipo, precisamos informar 2 outros valores: o da **precisão** - representada por **(P)** - e da **escala** - representada por **(S)**. A precisão é o número de dígitos que o numeral terá, incluindo o valor de casas decimais; enquanto a escala representa somente o número de casas decimais. Por essa definição, não podemos ter um valor de escala maior que o valor de precisão. Dessa forma, ao definir um campo, colocamos os valores de P e S entre parênteses, separados por vírgula (P,S).

Numéricos exatos (unidades monetárias)

MONEY / SMALLMONEY - Representamos estes números com a moeda na frente
Ex: R$30,23

Tipo de dados	Intervalo	Armazenamento
money	-922.337.203.685.477,5808 a 922.337.203.685.477,5807 (-922.337.203.685.477,58 a 922.337.203.685.477,58 para o Informatica. O Informatica dá suporte apenas a dois decimais, não quatro.)	8 bytes
smallmoney	-214.748,3648 a 214.748,3647	4 bytes



Ainda nos números exatos, temos os valores que representam dinheiro. São eles **money** e **smallmoney**. A diferença entre eles também é o menor e maior valor. Enquanto smallmoney representa números pequenos, money representa valores enormes, que chegam na casa de quatrilhões.

Em ambos os tipos, podemos representar a moeda, utilizando R$, por exemplo.

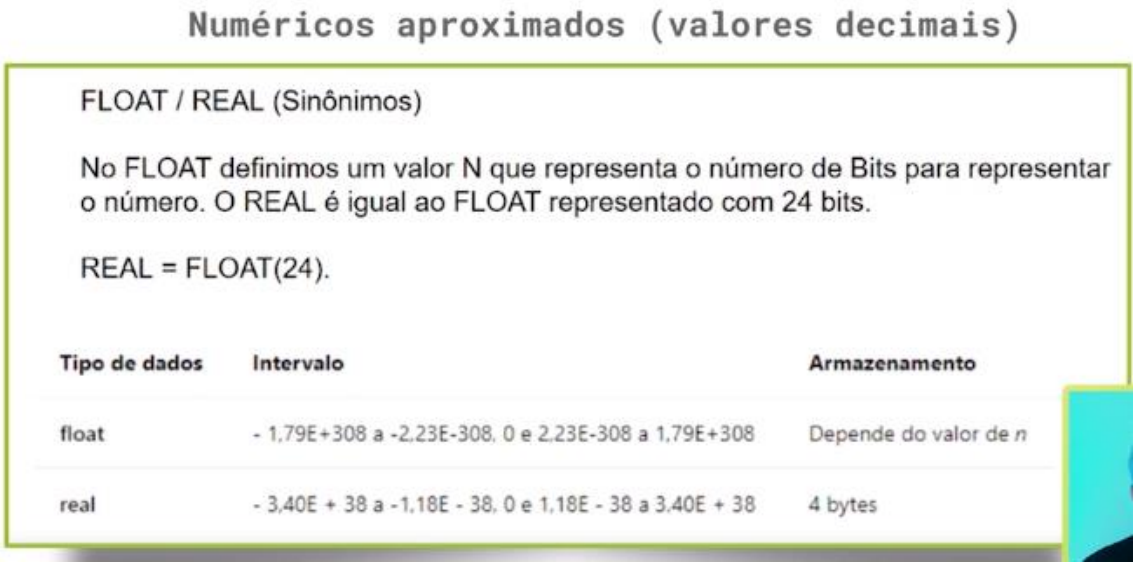
Numéricos exatos (valor lógico)

Não existe um tipo de dados lógico (VERDADEIRO ou FALSO).
O que temos é o tipo BIT que representa o valor 1 ou 0.

0 - FALSO
1 - VERDADEIRO

Podemos representá-lo com as palavras em inglês: TRUE ou FALSE.

O valor lógico também faz parte dos números exatos. Na verdade, no SQL Server não há necessariamente um valor lógico do tipo booleano (verdadeiro ou falso), mas sim um tipo chamado **BIT**. Ou seja, um número de um byte que receberá o valor 1 ou 0, sendo 0 correspondente à falso, e 1, à verdadeiro. Dessa maneira, podemos passar para o SQL Server os valores 1 ou 0, ou as palavras **TRUE** ou **FALSE**. Ao passar TRUE, ele gravará 1; ao passar FALSE, gravará 0.



Os números aproximados também são números com casas decimais. Mas, neste caso, não precisamos definir a quantidade de casas decimais, pois a precisão é definida internamente pelo banco de dados.

Temos 2 tipos, **FLOAT** e **REAL**, que possuem basicamente as mesmas características. No entanto, REAL é um campo que suporta um limite menor de números; é, na verdade, um FLOAT representado com 24 bits. Se colocamos um número maior do que REAL pode suportar, devemos defini-lo como FLOAT e, entre parênteses, colocar seu tamanho em bits.

FLOAT/REAL (sinônimos) No FLOAT, definimos um valor N que representa o número de Bits para representar o número. O REAL é igual ao FLOAT representado com 24 bits.

É importante salientar que não podemos colocar qualquer valor em bits, pois até o número em FLOAT possui limites de mínimo e máximo, tanto para negativo, quanto para positivo. Se representamos números negativos, o limite é menor do que se representássemos somente positivos.

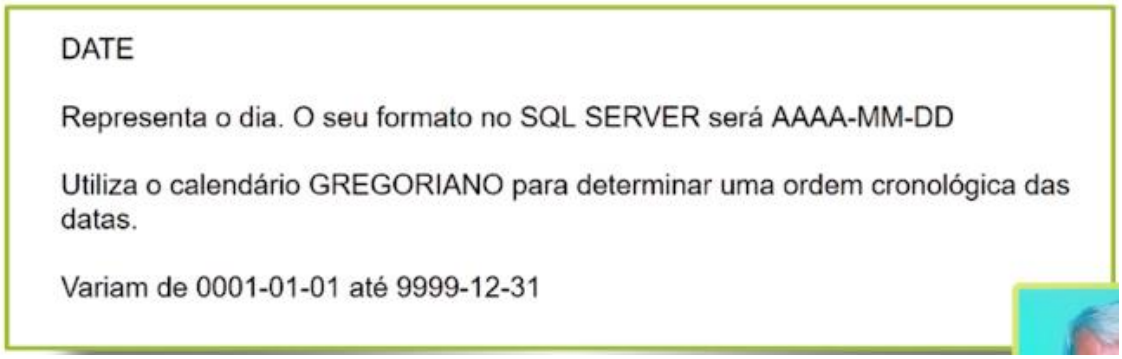
Se, no FLOAT, representamos números positivos e negativos, a variação será de -1,79 * 0^308 a 1,79 * 0^308. Mas se forem somente positivos, o valor aumenta para 2,23 * 0^308.

Em REAL, por ser um FLOAT de somente 24 bits, seu limite negativo e positivo será bem menor que FLOAT, sendo a potência ^38. Se representamos somente positivos, teremos um valor de 0 a 3,40 * 0^38. Mas, se levarmos em

consideração, números negativos e positivos, a potência cai para -1,18 * 0^38 a 1,18 * 0^38.

FLOAT é um número de ponto flutuante de precisão simples.

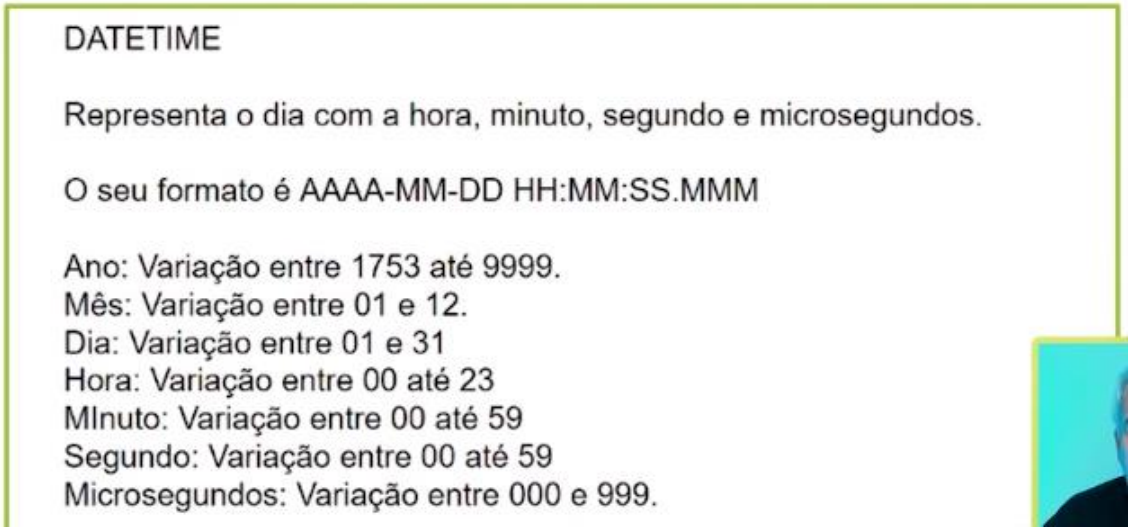
Data e hora - Valor do dia (dia)



O primeiro tipo desse grupo é o **DATE**, que representa uma data simples. No SQL Server, o formato é **AAAA-MM-DD** - 4 dígitos para o ano, 2 para o mês e 2 para o dia. É importante lembrar que utiliza-se o calendário **GREGORIANO** para determinar uma ordem cronológica das datas.

Esse tipo de data possui um limite mínimo e máximo, variando de 0001-01-01 a 9999-12-31, ou seja, 1º de Janeiro do ano 1 a 31 de Dezembro do ano 9999.

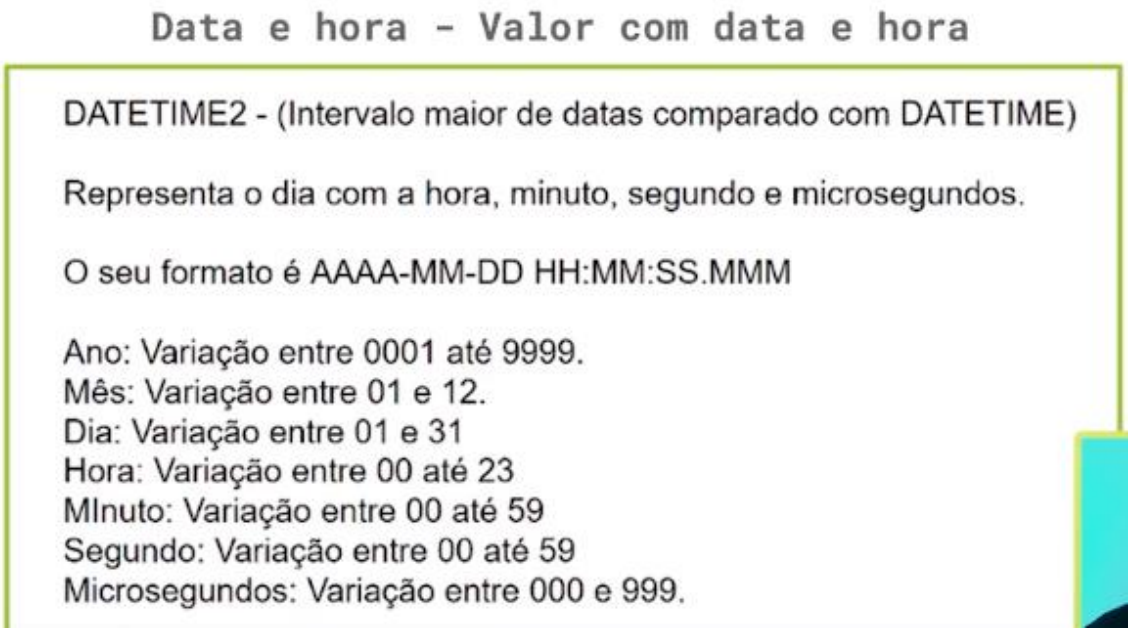
Data e hora - Valor com data e hora



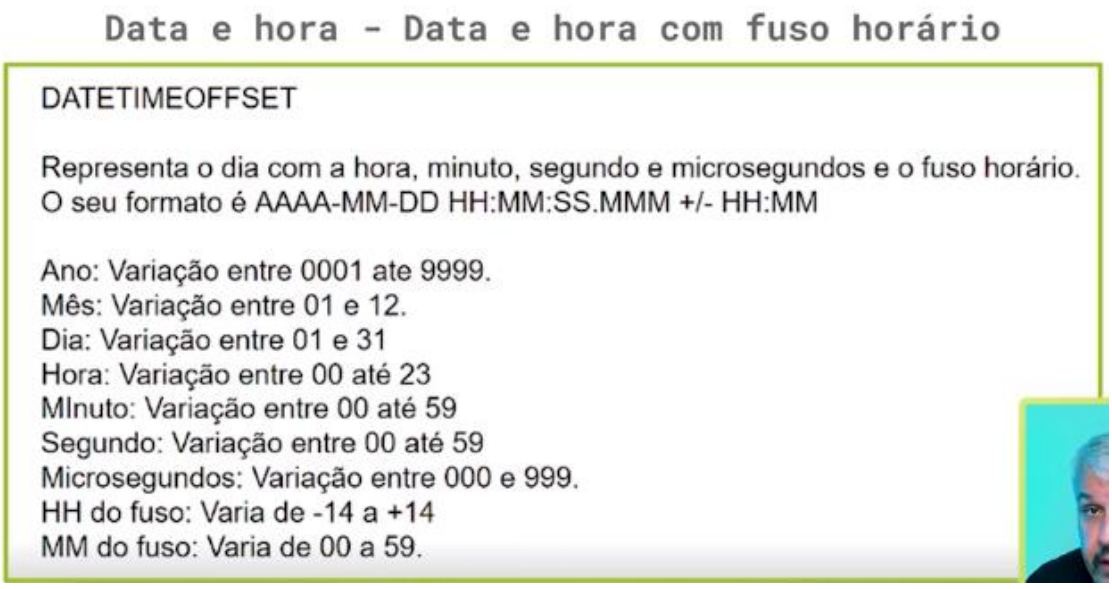
Além de armazenar o ano, mês e dia, esse tipo armazena a hora, minuto, segundo e microsegundo. O formato é **AAAA-MM-DD HH:MM:SS.MMM**. A primeira parte refere-se ao ano, mês e dia, assim como no tipo date. Já a segunda parte elenca 2 dígitos para hora (HH), 2 dígitos para minuto (MM), 2

dígitos para segundo (SS) e 3 dígitos para microssegundos (MMM). Há, ainda, as seguintes variações:

- Ano: entre 1753 até 9999;
- Mês: entre 01 e 12;
- Dia: entre 01 e 31;
- Hora: entre 00 até 23;
- Minuto: entre 00 até 59;
- Segundo: entre 00 até 59;
- Microssegundos: entre 000 e 999.



Semelhante ao anterior, o **DATETIME2** comporta um intervalo maior de datas, já que seu limite inferior é 1º de Janeiro do ano 1 (0001-01-01). De resto, a variação dos outros campos e seu valor máximo, coincide com a representação de DATETIME, portanto, seu formato também é **AAAA-MM-DD HH:MM:SS.MMM**.

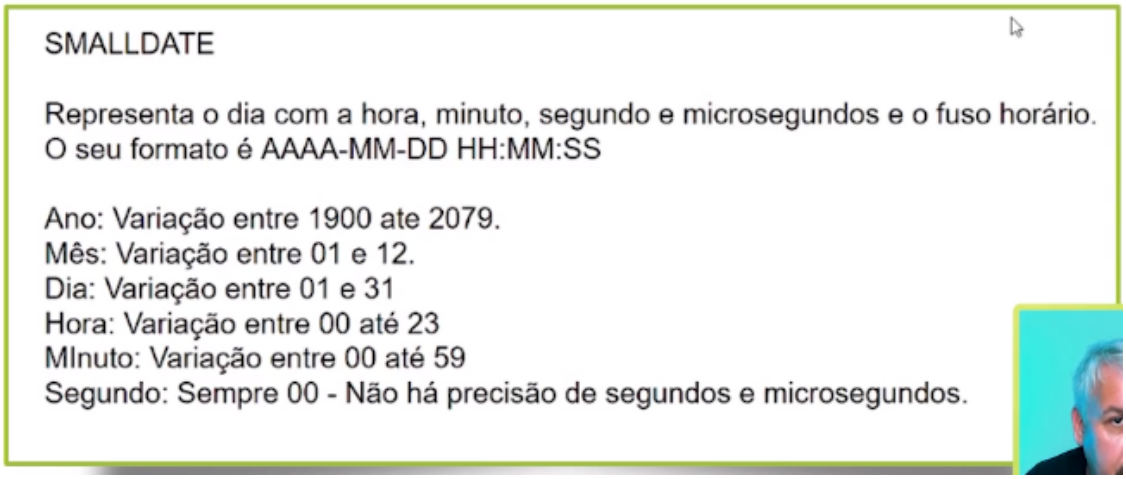


Este tipo é o datetime que leva em consideração o fuso-horário. Portanto, seu formato é **AAAA-MM-DD HH:MM:SS.MMM +/-HH:MM** - ano, dia e mês; hora, minuto, segundo e microssegundo; seguido do sinal + ou - com o número de horas e minutos que deve-se atrasar ou adiantar a hora com relação ao fuso 00, do meridiano Greenwich. Suas variações são as seguintes:

- Ano: entre 0001 até 9999;
- Mês: entre 01 e 12;
- Dia: entre 01 e 31;
- Hora: entre 00 até 23;
- Minuto: entre 00 até 59;
- Segundo: entre 00 até 59;
- Microssegundos: entre 000 e 999;
- HH do fuso: varia de -14 a +14;
- MM do fuso: Varia de 00 a 59.

O Meridiano de Greenwich é o principal meridiano do planeta. Trata-se de uma linha imaginária que corta a Terra, do Polo Norte ao Sul, passando no meio da Inglaterra, e representando o fuso 00. Ao avançar para Leste, adicionamos horas; ao avançar para Oeste, atrasamos horas.

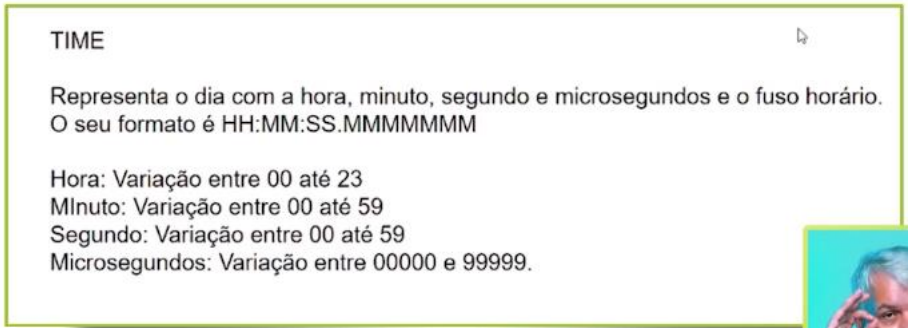
Data e hora - Data e hora sem os segundos



É uma data menor, que vai do ano 1900 (menor ano) a 2079 (maior ano). Trata-se de um datetime que só representa hora e minuto, não utilizando segundo e microssegundo, portanto, seu formato é **AAAA-MM-DD HH:MM:SS**. Sendo assim, é usado em situações que não necessitam de muita precisão. Suas variações são:

- Ano: entre 1900 até 2079;
- Mês: entre 01 e 12;
- Dia: entre 01 e 31;
- Hora: entre 00 até 23;
- Minuto: entre 00 até 59;
- Segundo: Sempre 00 - Não há precisão de segundos e microssegundos.

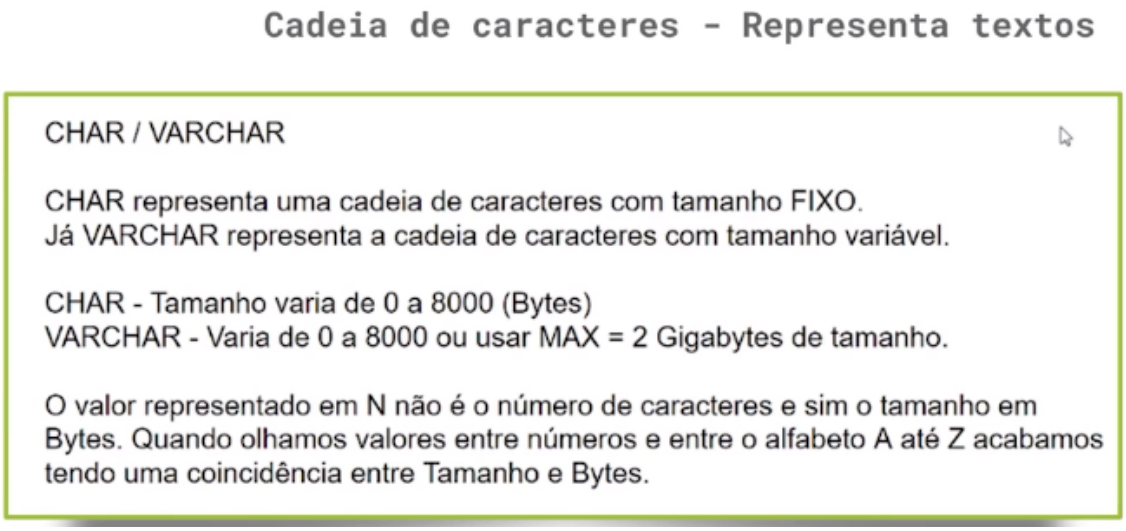
Data e hora - Representa apenas as horas



Este dado representa somente a hora, portanto, a precisão de microssegundo aumenta, sendo representado com 5 dígitos. Seu formato é **HH:MM:SS.MMMMM** e suas variações são:

- Hora: entre 00 até 23;
- Minuto: entre 00 até 59;

- Segundo: entre 00 até 59;
- Microssegundo: entre 00000 e 99999.

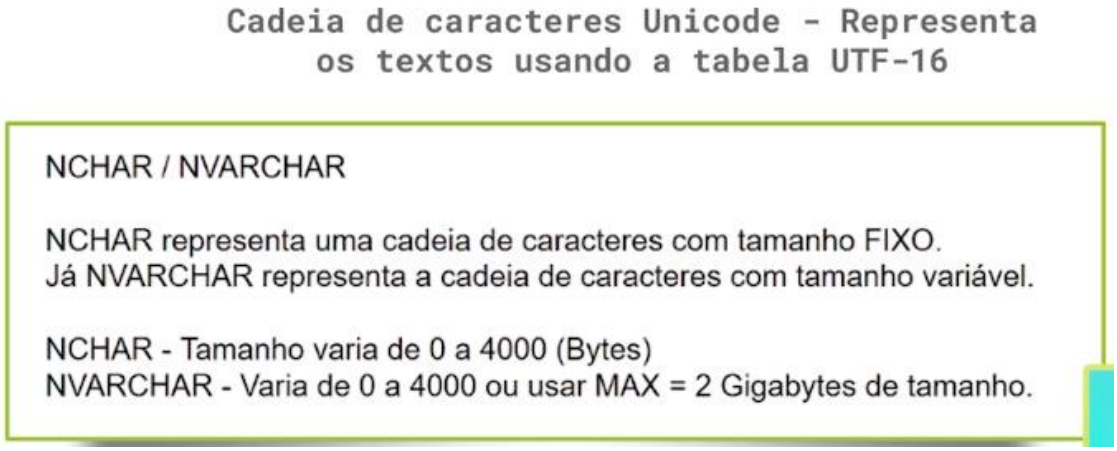


A diferença entre **CHAR** e **VACHAR**, é que o primeiro possui um tamanho fixo, enquanto o segundo possui uma cadeia de caracteres com tamanho variável. Dessa maneira, se definimos que um campo é CHAR de tamanho 10, mas gravamos somente uma letra, o SQL Server preencherá a letra em questão e mais 9 espaços em branco. Ou seja, sempre ocupará o tamanho de caracteres definidos.

Mas, se definimos um campo VARCHAR com 10 posições e gravamos apenas uma letra, os demais caracteres não serão preenchidos, ficando somente a letra gravada.

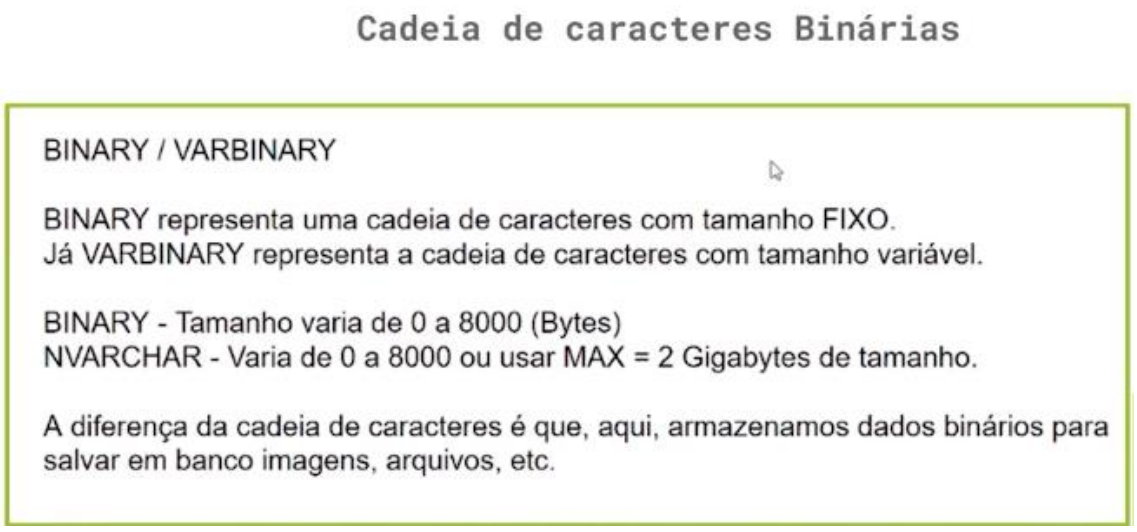
O CHAR possui uma variação limitada de 0 a 8000 bytes. O VARCHAR também possui essa variação de 0 a 8000 bytes, porém, é possível defini-lo como **MAX**, o que significa que podemos gravar um texto de até 2 gigabytes de tamanho. Sendo assim, utilizamos essa definição em VARCHAR para armazenar um grande número de caracteres.

Quando olhamos valores numéricos de 1 a 9, e letras de A a Z no alfabeto, há uma coincidência entre tamanho e bytes. No entanto, é importante ressaltar que, quando definimos o valor do campo (N), não estamos representando o número de caracteres, mas sim o número de bytes.



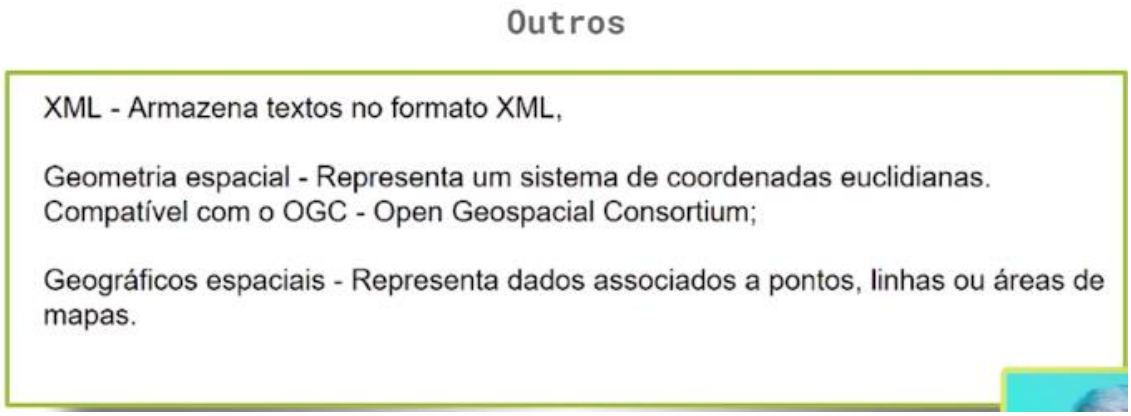
A cadeia de caracteres unicode utiliza um grupo de símbolos maior do que a cadeia de caracteres usada em CHAR e VARCHAR, como letras não pertencentes ao alfabeto ocidental, por exemplo. O unicode faz uso dos códigos da tabela UTF-16, que comporta mais símbolos que a tabela UTF-8, utilizada no CHAR e VARCHAR.

Por usar caracteres unicode, o limite de bytes mínimo e máximo é menor, variando somente de 0 a 4000 tanto em **NCHAR** quanto em **NVARCHAR**. No entanto, assim como em VARCHAR, o NVACHAR também pode ser definido como **MAX** para que possamos armazenar caracteres de até 2 gigabytes de tamanho.



O conceito de **BINARY** e **VARBINARY** assemelha-se à CHAR e VARCHAR. Dessa forma, BINARY possui um tamanho fixo e preenche os caracteres que sobram com espaços em branco, enquanto VARBINARY representa uma cadeia de caracteres com tamanho variável.

Nestes campos, não gravamos textos, e sim **dados binários**, ou seja, **bytes de arquivos**, como, por exemplo, imagens, documentos *PDF* ou *Word*, planilhas de *Excel*, entre outros. Ambos variam de 0 a 8000 bytes e VARBINARY pode ser definido como **MAX** para até 2 gigabytes de tamanho.



XML

Esse formato armazena textos em XML, mas não o arquivo em si, e sim seu conteúdo.

Geometria Espacial

Representa um sistema de coordenadas euclidianas e relaciona-se com símbolos geométricos. Além do mais, é compatível com o padrão **OGC - Open Geospatial Consortium**, uma organização voluntária internacional de consenso aberto que colabora com o desenvolvimento e a implementação de padronização de conteúdos nas áreas geoespacial e de serviços baseados em localização.

Geográficos espaciais

Os dados geográficos representam pontos, linhas ou áreas associadas a mapas. Desta forma, pode-se gravar, rotas, por exemplo.

Existem outros tipos de dados no SQL Server. Mas vamos nos limitar aos citados até aqui, uma vez que não faremos uso de todos eles. Em geral, utilizamos, principalmente, números, caracteres, valores lógicos ou datas.