```sql
CREATE TABLE tb_exemplo
(
      Id    INT PRIMARY KEY IDENTITY(1,1), -- cria ids iniciando pelo 1 e indo em 1 passo
      Nome VARCHAR(MAX)
);
```

VARIAVEIS NO SQL

```sql
-- declaração de apenas uma var
DECLARE <nome> <tipo> [= <valor inicial>];

DECLARE @i INT = 0;

-- declaração de varias vars
DECLARE <nome> <tipo> [= <valor inicial>],
        <nome> <tipo> [= <valor inicial>],
        <nome> <tipo> [= <valor inicial>];

DECLARE @nome VARCHAR(MAX),
        @cpf  CHAR(11),
        @tel  CHAR(11);
```

TIPOS DE VARIAVEIS

- LITERAIS
  1. `CHAR(n)` → se o literal não tiver o comprimento de n, preenche com espaços em branco
  2. `VARCHAR(n)` → não precisa ter o comprimento de n
  3. `NCHAR/NVARCHAR(n)` → o N é de unicode (ex. chars chineses)
- NUMERICOS
  1. `BIT` → para 1 bit (0 ou 1)
  2. `TINYINT` → para 8 bits (1 byte) [0, 255]
  3. `INT` → 4 bytes [-2 bilhões, +2 bilhões]
  4. `FLOAT` → Numéricos pseudorreis, precisão de 53 casas decimais (8 bytes)
  5. `REAL` → Numéricos pseudorreis, precisão de 24 casas decimais (4 bytes)
  6. `MONEY` → DECIMAL(10,2) = 8 bytes [± 9 trilhões]
  7. `SMALL MONEY` → 4 bytes [± 200 mil]
  8. `DATE` → `Ano/Mês/Dia = 3 bytes`
  9. `TIME` → `Hora:Min:seg:ms = 5 bytes`
  10. `DATETIME` → `{Ano/Mês/Dia} + {Hora:Min:seg:ms} = 8b`
  11. `DATETIME2` → `Mais preciso nos segs e ms`

APRESENTAÇÃO DE VARIAVEIS

```
PRINT (<conteudo>) -- so aceita string

SELECT <variavel> -- retorna resposta tabular
```

FUNÇÕES NATIVAS DO SQL

```
CONVERT (<tipo>, <variavel>) -- converte de um tipo para outro

DECLARE @i INT = 10;
PRINT(CONVERT(VARCHAR, @i))
```

TRATAMENTO DE ATRIBUIÇÕES EM EXPRESSÕES

```
SET <atribuicao>

DECLARE @x FLOAT;
SET @x = 3.14159265;
```

MANIPULAÇÃO DE STRINGS

1. CONCAT(<var1>, <var2>, <varN>)
2. SUBSTRING(variável, índice_inicial, num_passos) →
   a. LEFT(variável, passos)
   b. RIGHT(variável, passos)
3. LEN(variável)
4. LOWER(variável)
5. UPPER(variável)
6. NEWID() → gera um valor literal randômico ([8]-[4]-[4]-[4]-[12])
7. REPLACE(<var>, <expressão que sera substituida>, <nova expressao>) → REPLACE(@str, '-', ' ')

MANIPULAÇÃO DE STRINGS

1. GETDATE()
2. DATEPART(<parte(dia, mês, ano)>, <var de data>)
3. DATENAME(<parte>, <var de data>) → MONTH, WEEKDAY
4. DATEDIFF(<unidade>, <data1>, <data2>) → Distância entre datas

FUNÇÕES MATEMATICAS

1. `RAND()` → gera valor aleatório entre 0 e 1
2. `ROUND(<var>, <n° de casas decimais>)` → arredonda para o + próximo
3. `CEILING(<var>)` → arredonda para o próximo MAIOR inteiro
4. `FLOOR(<var>)` → arredonda para o próximo MENOR inteiro

FUNÇÕES ARITIMETICAS

1. `POWER(<var>, <expoente>)` → retorno depente da base → var^{exp}
2. `SQRT(<var>)`
3. `EXP(<var>)` → e^{var}
4. `LOG(<var>)`

FUNÇÕES TRIGONOMETRICAS

1. `RADIANS(<valor em graus>)` → retorno depende da base
2. `SIN/ASIN(<valor em radianos>)`
3. `COS/ACOS(<valor em radianos>)`
4. `TAN/ATAN(<valor em radianos>)`

CONTROLE DE FLUXO NO T-SQL

```
--condicional simples
IF [condicao]
  BEGIN
    <bloco de comandos>
  END
ELSE
  BEGIN
    <bloco de comandos>
  END;
```

```
--condicional
repetitivo simples
WHILE [condicao]
  BEGIN
    <bloco de comandos>
  END
```

```
--tentativa e erro
BEGIN TRY
  <código que vai tentar
fazer>
END TRY
BEGIN CATCH
  <código caso não
consiga fazer o código de
cima>
END CATCH;
```

No `TRY CATCH` existe o `RAISERROR(<msg>, 0, 0)` que força a entrada na clausula CATCH

CONFERENCIA DE EXISTENCIA

```
EXISTS <tabela>
NOT EXISTS <tabela>

IF NOT EXISTS <tabela>
BEGIN
  <criação da tabela>
END
```

FUNCAO DE EXECUCAO DE COMANDOS DINAMICOS

```
EXEC(sp_executesql)
```

CURSORES

- Estrutura de controle de fluxo por varredura (EM MALHA/REPETITIVA)
- COMPOSTA POR 4 FASES

1- DECLAREAÇÃO

```
DECLARE <nome> CURSOR
FOR
     <info tabular/tabela>
```

2- ABERTURA

```
OPEN <nome da variável do cursor>
```

3- VARREDURA

```
DECLARE <@coluna1> <tipo1>,
        <@coluna2> <tipo2>,
        <@colunaN> <tipoN>;
FETCH NEXT FROM <cursor> INTO @coluna1, coluna2, colunaN
WHILE @@FETCH_STATUS = 0
BEGIN

     FETCH NEXT FROM <cursor> INTO <var de coluna>
END;
```

4- FECHAMENTO

```
CLOSE <cursor>;
DEALLOCATE <cursor>;
```

MODULARIZAÇÃO EM T-SQL

- FUNÇÕES (estrutura simples)
  - Tem argumentos de entrada
  - Apenas um argumento de saída, podendo ser tabular
  - NÃO ALTERA TABELAS
  - O nome das functions precisam ser declaradas em um schema (ex: dbo.funcao_exemplo)

```
CREATE [OR ALTER] FUNCTION <nome> (arg1 <tipo1>,
                                   arg2 <tipo2>,
                                   argN <tipoN>)
RETURNS <tipo> -- int, float, table
AS
BEGIN
  <comandos>
  RETURN <var ou conteúdo do tipo especificado>; --precisa ser
o ultimo comando
END
```

- PROCEDIMENTOS/PROCEDURES (mais flexíveis)
  - Tem valores de entrada
  - Condição de rotinas de comando
  - Normalmente sem retorno, mas é possível conduzir a simulação por referência
  - ALTERAM TABELAS

```
CREATE [OR ALTER] PROCEDURE <nome>
  arg1 <tipo1>,
  arg2 <tipo2>,
  argN <tipoN>
BEGIN
  <comandos>
END;

EXEC <procedure> arg1, arg2, argN; -- args podem ser valores
```

- GATILHOS/TRIGGERS (mais rígidos)
    - Estruturas auditoras de Eventos (inserção, remoção, atualização, etc.)
    - Executa eventos de escrita sobre a tabelas
    - Não possuem argumentos
    - Associados a estruturas existentes
    - Dispara um conjunto de comandos a partir do tipo de evento que a estrutura acompanha
    - TIPO DE DISPARO: Após o evento (conclui o evento antes do disparo) & Em vez do evento (não conclui o evento).
    - TABELAS DAS TRIGGERS: INSERTED (tabela com a mesma estrutura da tabela na qual a trigger está associada e seus registros são os registros os quais se tem a intenção de inserir na tabela), DELETED (estrutura análoga à tabela inserted com a ressalva de serem registros nos quais se tem a intenção de conduzir remoção)
    - FATOS: se INSERTED = 0 & DELETED ≠ 0 → REMOÇÃO
      se INSERTED ≠ 0 & DELETED = 0 → INSERÇÃO
      se INSERTED ≠ 0 & DELETED ≠ 0 → ATUALIZAÇÃO

```
CREATE [OR ALTER] TRIGGER <nome>
ON <tabela> --vai executar os eventos associados a esta tabela
<tipo do evento> <evento1>[, <evento2>, <evento3>]
AS
BEGIN
  <comandos>
END;
```

Os eventos podem ser INSERT, DELETE, UPDATE

PEGAR RESTO DOS PDFS E EXCS DO PEN DRIVE