



**Acesso a Dados**

**Bootcamp Speed Wiz Dev**

Renato Júnior

2021

## **Acesso a Dados**

### **Bootcamp Speed Wiz Dev**

Renato Júnior

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

## Sumário

---

|   |    |
|---|----|
| Capítulo 1. Preparação do Ambiente.....             | 5  |
| Instalando o SQL Server.....                        | 5  |
| Instalando o SQL Management Studio .....            | 18 |
| Capítulo 2. Introdução a Banco de Dados .....       | 24 |
| Definição de Banco de Dados.....                    | 24 |
| Modelo Entidade Relacionamento (MER) .....          | 24 |
| Diagrama Entidade Relacionamento (DER).....         | 24 |
| Entidade.....                                       | 25 |
| Tipos de Relacionamento .....                       | 26 |
| Importância de um Banco de Dados.....               | 27 |
| Tipos de Banco de Dados.....                        | 28 |
| Como gerenciar um Banco de Dados .....              | 28 |
| Por que é importante aprender SQL .....             | 29 |
| Dialeto.....  | 29 |
| Grupos de Comandos.....                             | 29 |
| <i>Como utilizar o comando CREATE DATABASE.....</i> | 31 |
| <i>Como utilizar o comando CREATE TABLE.....</i>    | 31 |
| <i>Como utilizar o comando INSERT INTO.....</i>     | 32 |
| <i>Como utilizar o comando SELECT .....</i>         | 32 |
| <i>Como utilizar o comando JOIN.....</i>            | 36 |
| <i>Como utilizar o comando UPDATE.....</i>          | 39 |
| <i>Como utilizar o comando DELETE .....</i>         | 41 |
| <i>Como utilizar o comando ALTER TABLE .....</i>    | 42 |

|  |           |
|--|-----------|
| <i>Como utilizar o comando DROP TABLE.....</i>       | <i>43</i> |
| <i>Como utilizar o comando DROP DATABASE.....</i>    | <i>44</i> |
| Capítulo 3. Introdução ao Entity Framework Core..... | 45        |
| O que é ORM.....                                     | 45        |
| Vantagens e Desvantagens de utilizar um ORM .....    | 45        |
| O que é Entity Framework Core.....                   | 46        |
| Definição de Modelo no Entity Framework Core.....    | 46        |
| Evolução .....                                       | 47        |
| Por que escolher o Entity Framework Core .....       | 48        |
| Onde podemos utilizá-lo? .....                       | 48        |
| Suporte a Banco de Dados.....                        | 48        |
| Tipos de Abordagens.....                             | 49        |
| DbContext e DbSet.....                               | 49        |
| Migrations .....                                     | 49        |
| Capítulo 4. Introdução ao Dapper.....                | 51        |
| Desempenho.....                                      | 51        |
| Vantagens e Desvantagens.....                        | 51        |
| O que ele resolve?.....                              | 52        |
| Entity Framework Core vs Dapper .....                | 52        |
| Uso do Entity Framework Core com Dapper.....         | 53        |
| Referências.....                                     | 54        |

## Capítulo 1. Preparação do Ambiente

Neste primeiro capítulo, iremos preparar o ambiente realizando a instalação das ferramentas necessárias para a criação e manipulação de um banco de dados relacional, para isso iremos instalar o SQL Server 2019 e a ferramenta SQL Management Studio que será usada para manipular o banco de dados. Caso prefira, existe uma videoaula onde mostro passo a passo o processo de instalação dessas ferramentas.

### Instalando o SQL Server

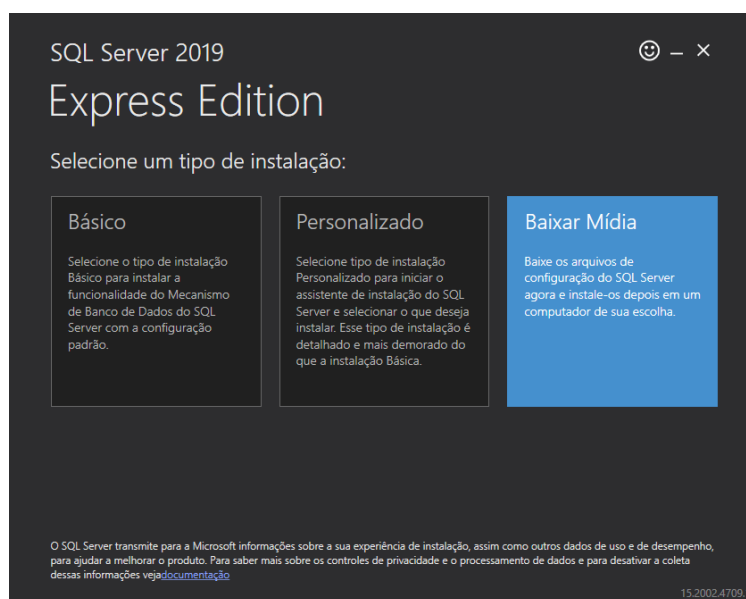
Antes de realizar a instalação, é necessário baixar o instalador disponível no link

abaixo.

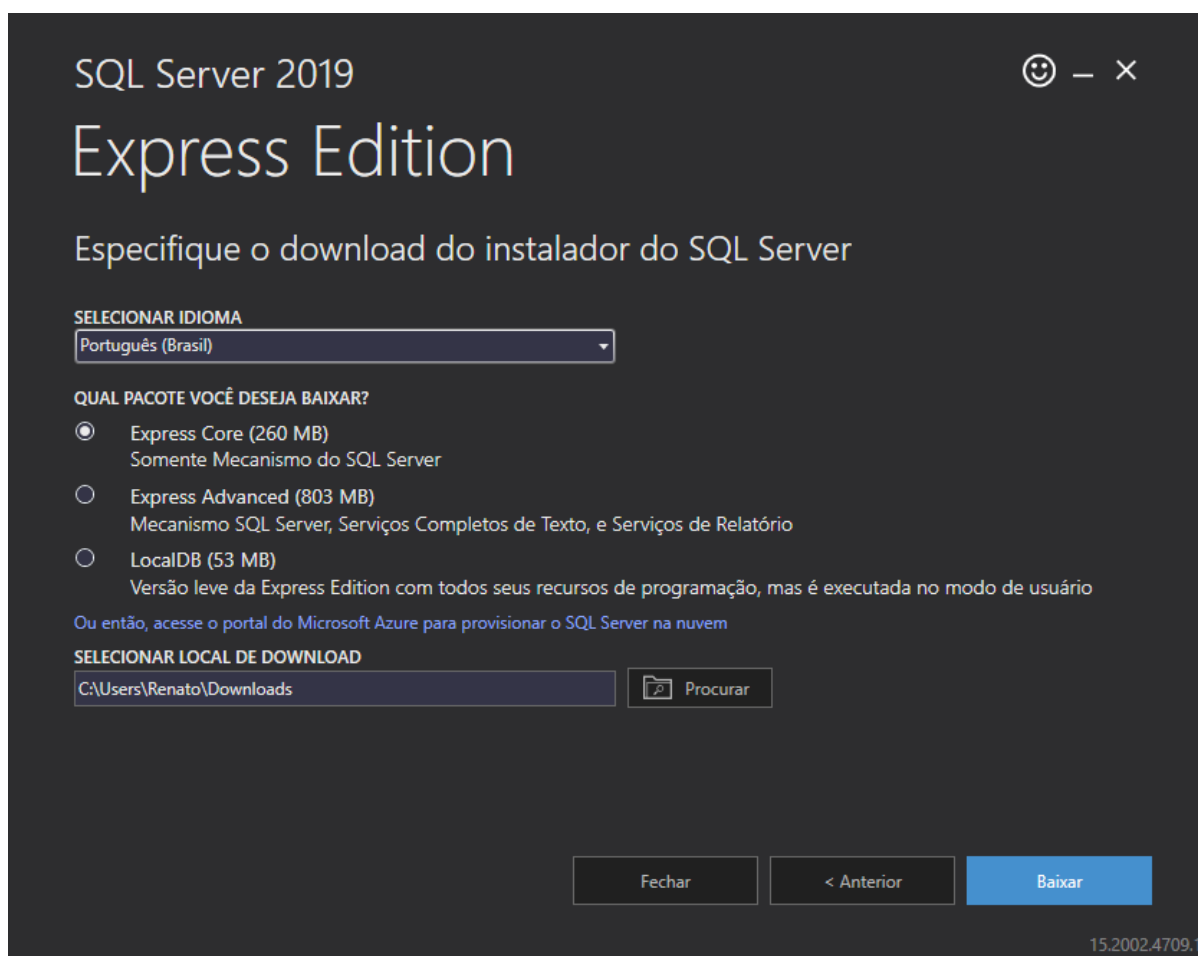
<https://www.microsoft.com/pt-br/sql-server/sql-server-downloads>

Após realizar o download no site da Microsoft e executar o instalador da Microsoft escolher a opção **Baixar Mídia**. Essa opção baixará todos os arquivos necessários para realizar a instalação sem depender de uma conexão com a internet. Após a seleção uma janela irá surgir solicitando que seja escolhido o pacote a ser feito o download. No nosso caso, iremos escolher a opção Express Core conforme **Figura 2**.

**Figura 1 – Imagem Instalador**

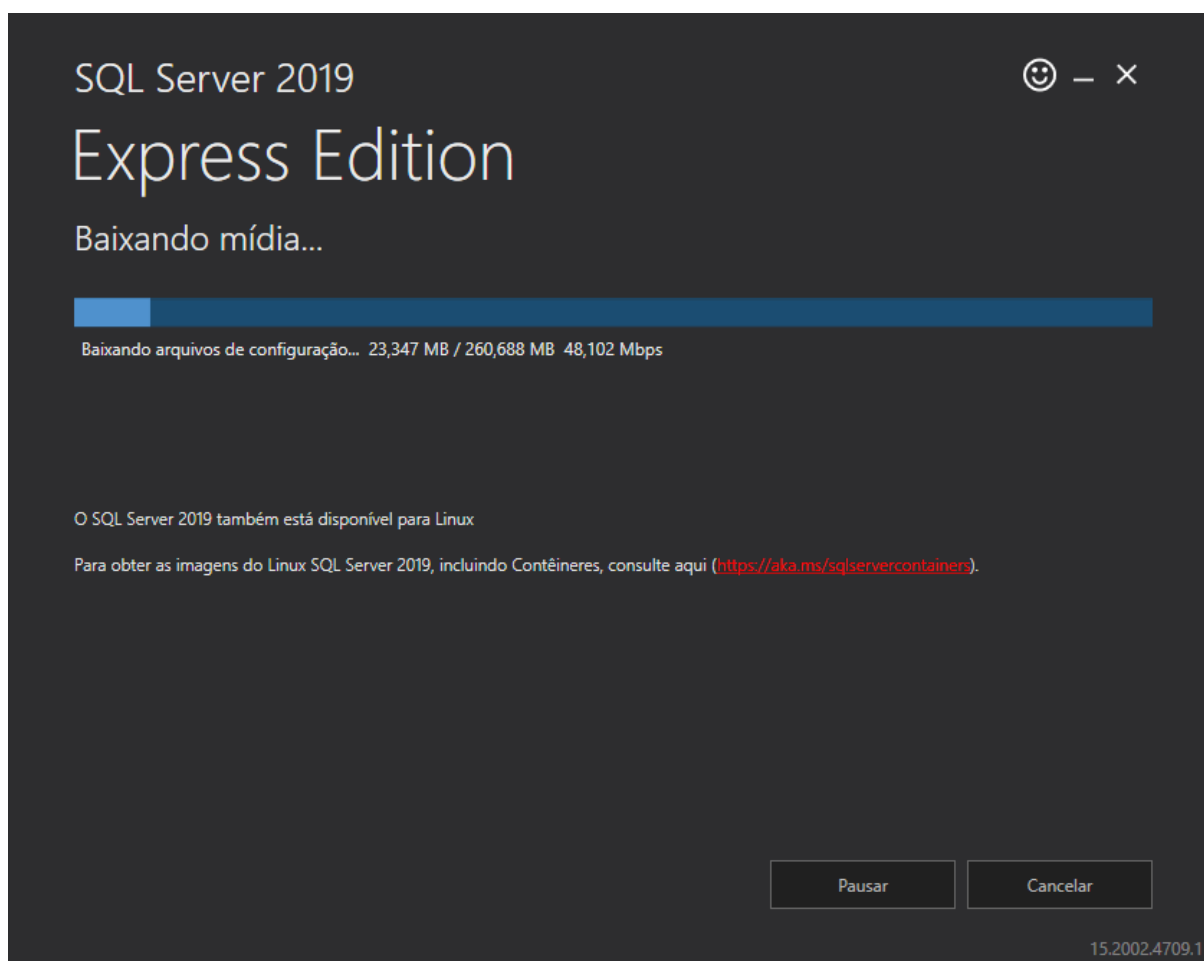


**Figura 2 – Imagem Instalador**

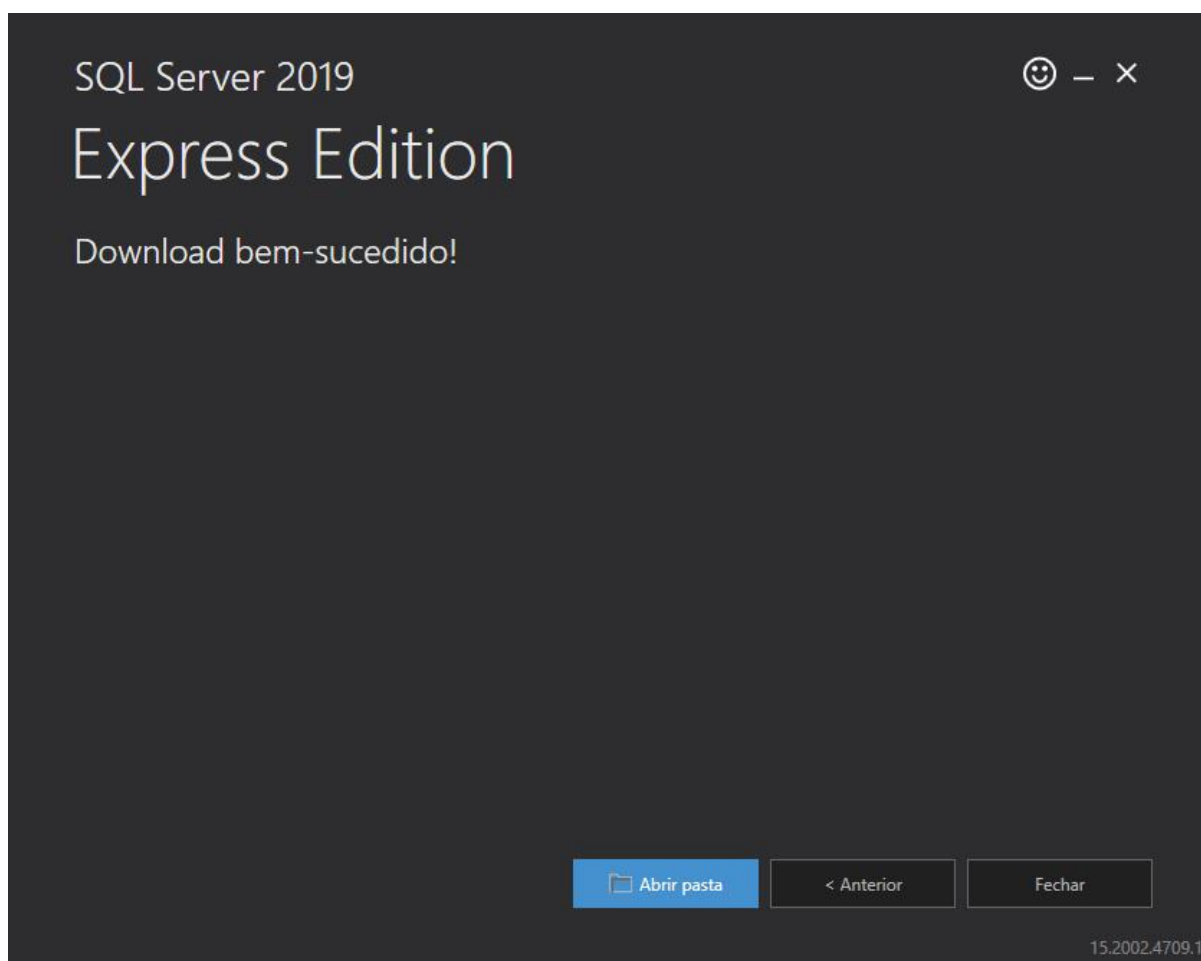


**Fonte: Imagem Própria**

Após escolher qual pacote fazer o download clique no botão **Baixar** para começar o download. Ao concluir o download do pacote deverá ser feito a extração dos arquivos para um diretório da sua escolha.

**Figura 3 – Imagem Baixando Pacote**

**Fonte: Imagem Própria**

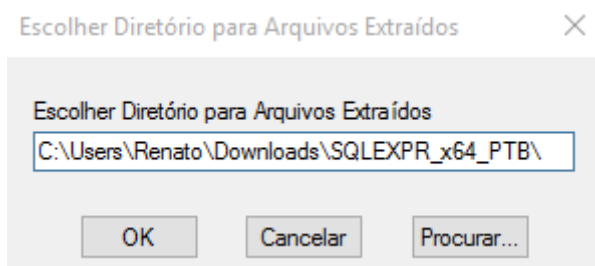
**Figura 4 – Imagem Conclusão de Download**

**Fonte: Imagem Própria**

Após clicar no botão para ***Abrir pasta*** e escolher um diretório para extração dos arquivos o instalador está disponível e pronto para ser usado. A **Figura 5** mostra a imagem que será apresentada durante o processo de extração dos arquivos para o diretório escolhido.



**Figura 5 – Imagem Arquivos Extraídos**

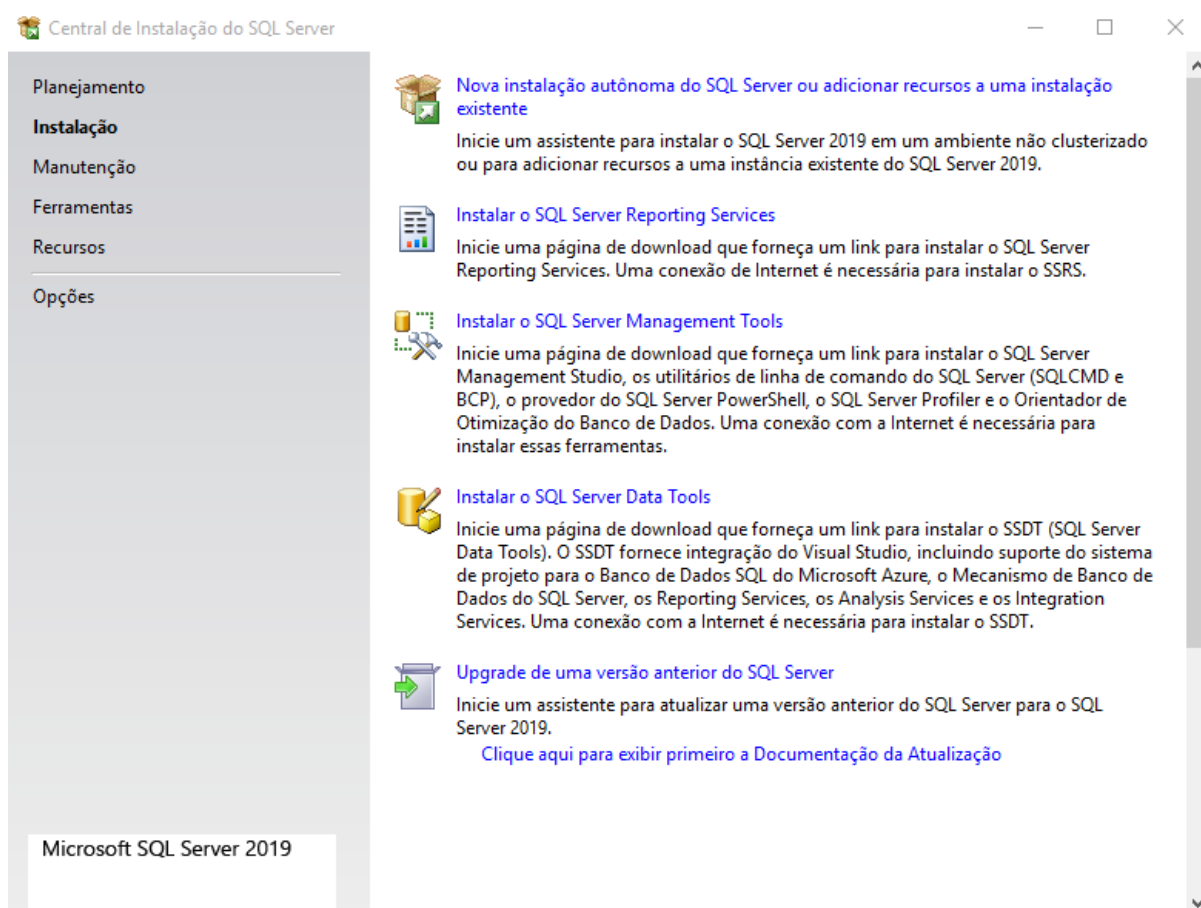


**Fonte: Imagem Própria**

Após realizar a extração dos arquivos, o instalador irá abrir conforme **Figura**

**6.**

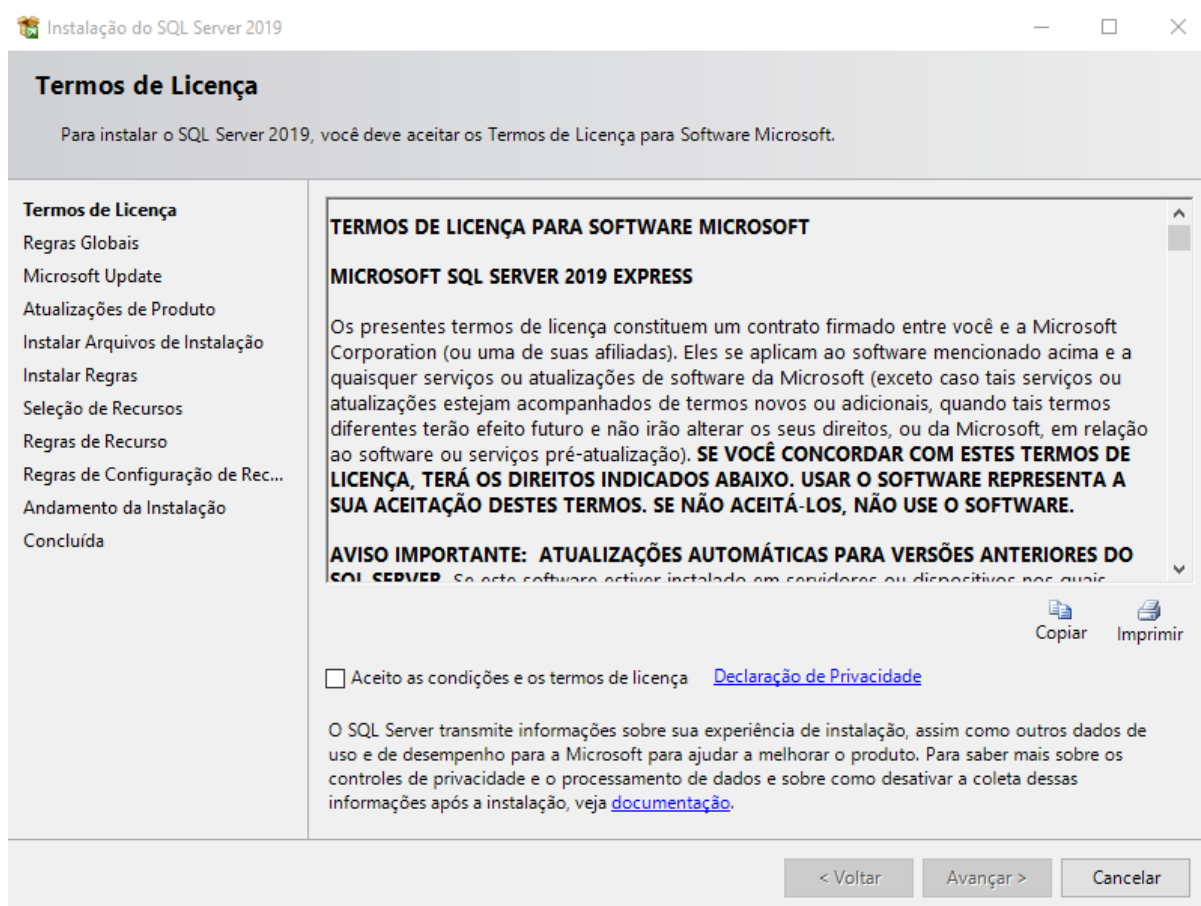
**Figura 6 – Imagem Instalador SQL Server**



**Fonte: Imagem Própria**

Escolha a primeira opção “*Nova instalação autônoma do SQL Server ou adicionar recursos a uma instalação existente*” ao executar essa ação, a tela abaixo irá surgir.

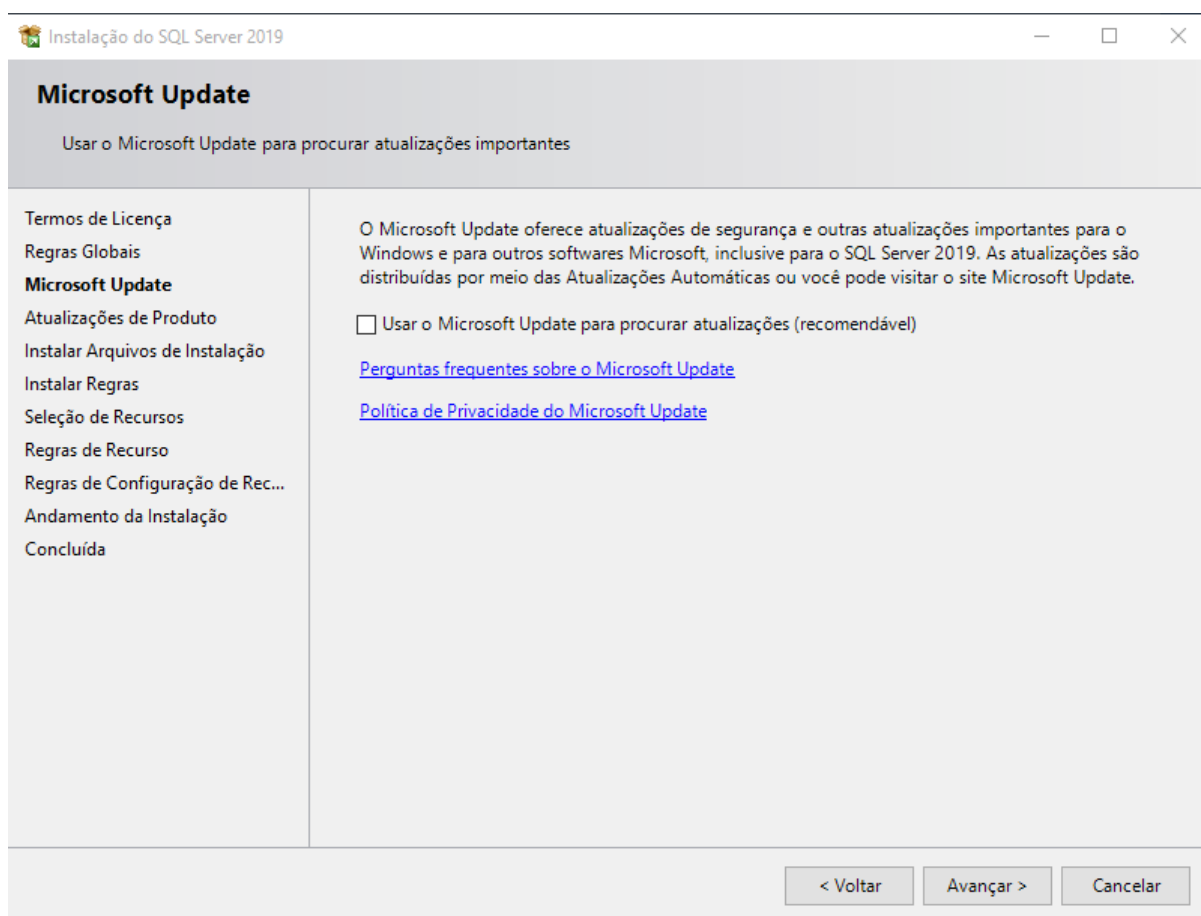
**Figura 7 – Termo de Licença SQL Server**



**Fonte: Imagem Própria**

Aceite os termos para continuar o processo de instalação e clique no botão **Avançar**, ao executar esse passo a tela abaixo irá surgir.

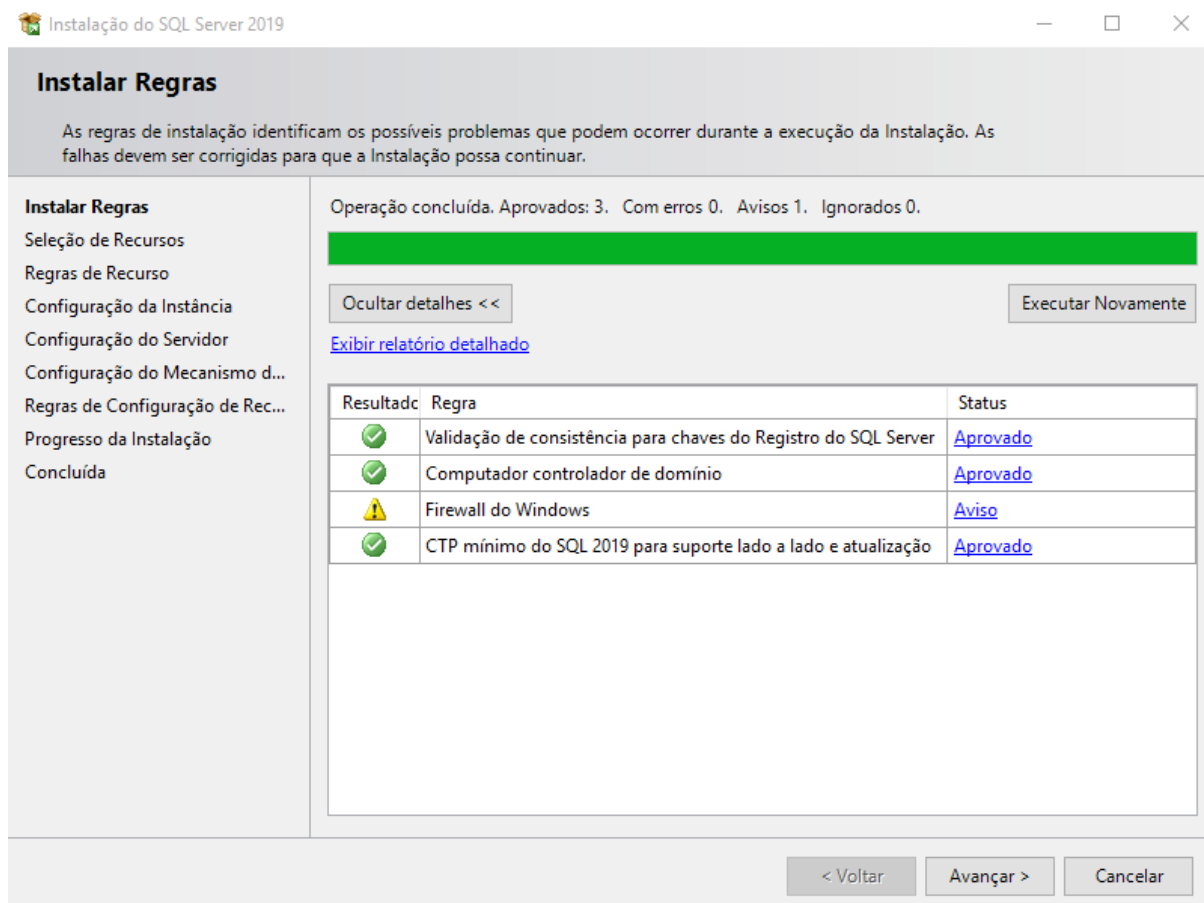
**Figura 8 – Imagem Microsoft Update**



**Fonte: Imagem Própria**

É recomendado marcar o a opção “Usar o Microsoft Update para procurar atualizações”. Marcando essa opção, sempre que houver uma nova atualização do SQL Server ela estará disponível para ser instalado através do Microsoft Update. Clique em **Avançar** para dar continuidade a instalação. Será feita uma verificação de possíveis problemas que podem ocorrer durante o processo de instalação e a tela abaixo será mostrada informando ao usuário o resultado da verificação.

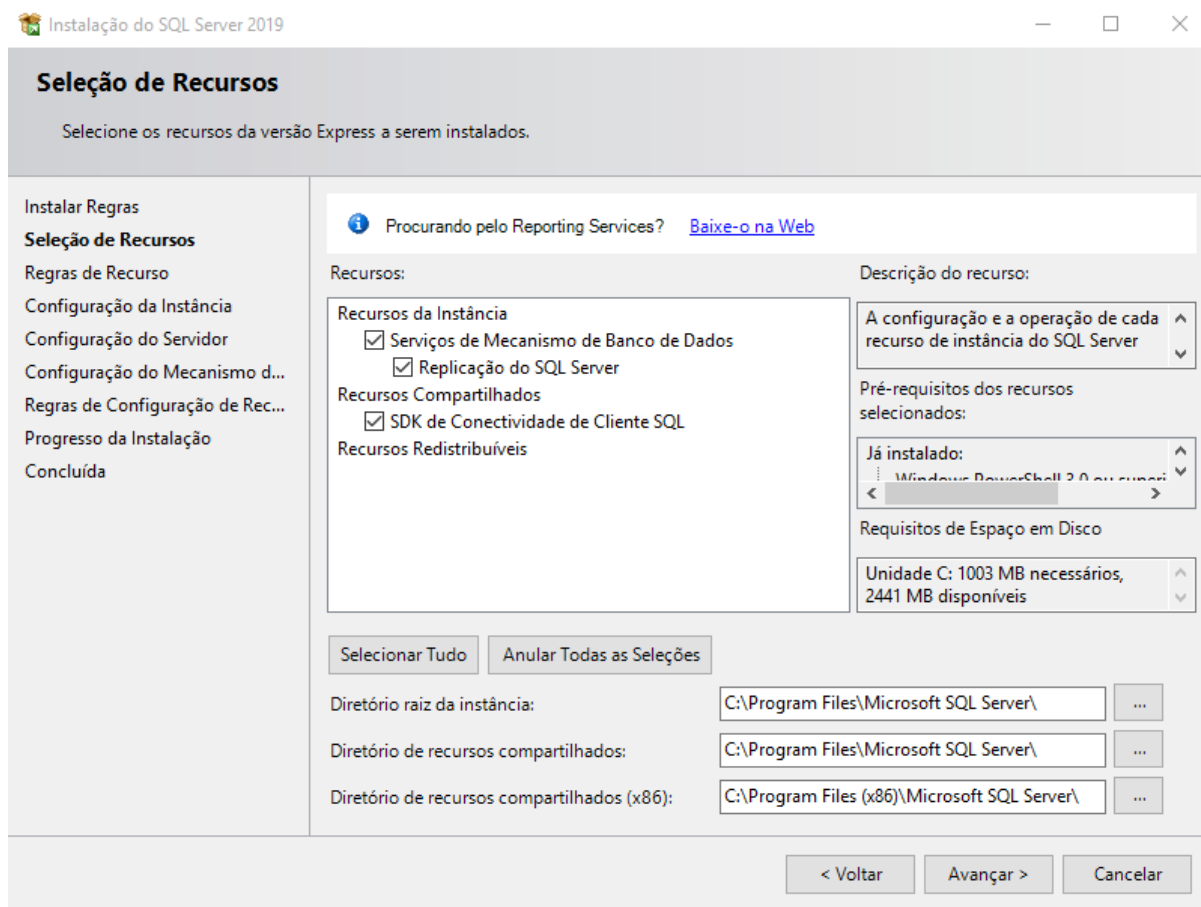
**Figura 9 – Imagem de Verificação de Regras de Instalação**



**Fonte: Imagem Própria**

Note que após a verificação das regras apenas o Firewall do Windows apresentou um aviso, isso ocorreu devido ao Firewall do computador estar habilitado. Clique em **Avançar**, o próximo passo é selecionar os recursos que serão instalados, conforme **Figura 10**.

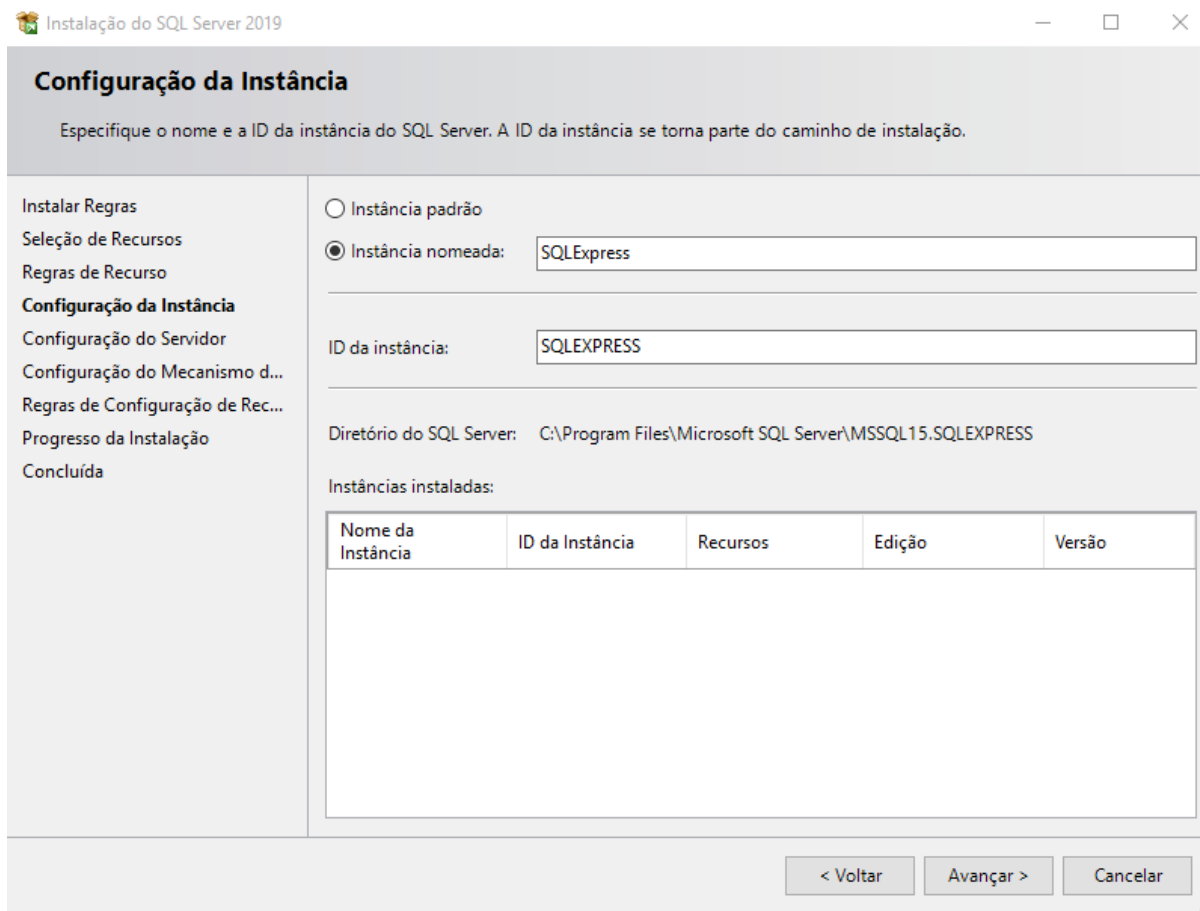
**Figura 10 – Imagem de Seleção de Recursos**



**Fonte: Imagem Própria**

Selecione todas as opções conforme a imagem acima e clique em **Avançar**. Será aberta a imagem abaixo, conforme **Figura 11**, para configurarmos a instância do SQL Server.

**Figura 11 – Imagem de Configuração da Instância**



**Configuração da Instância**

Especifique o nome e a ID da instância do SQL Server. A ID da instância se torna parte do caminho de instalação.

☐ Instância padrão  
☒ Instância nomeada:

ID da instância:

Diretório do SQL Server: C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS

Instâncias instaladas:

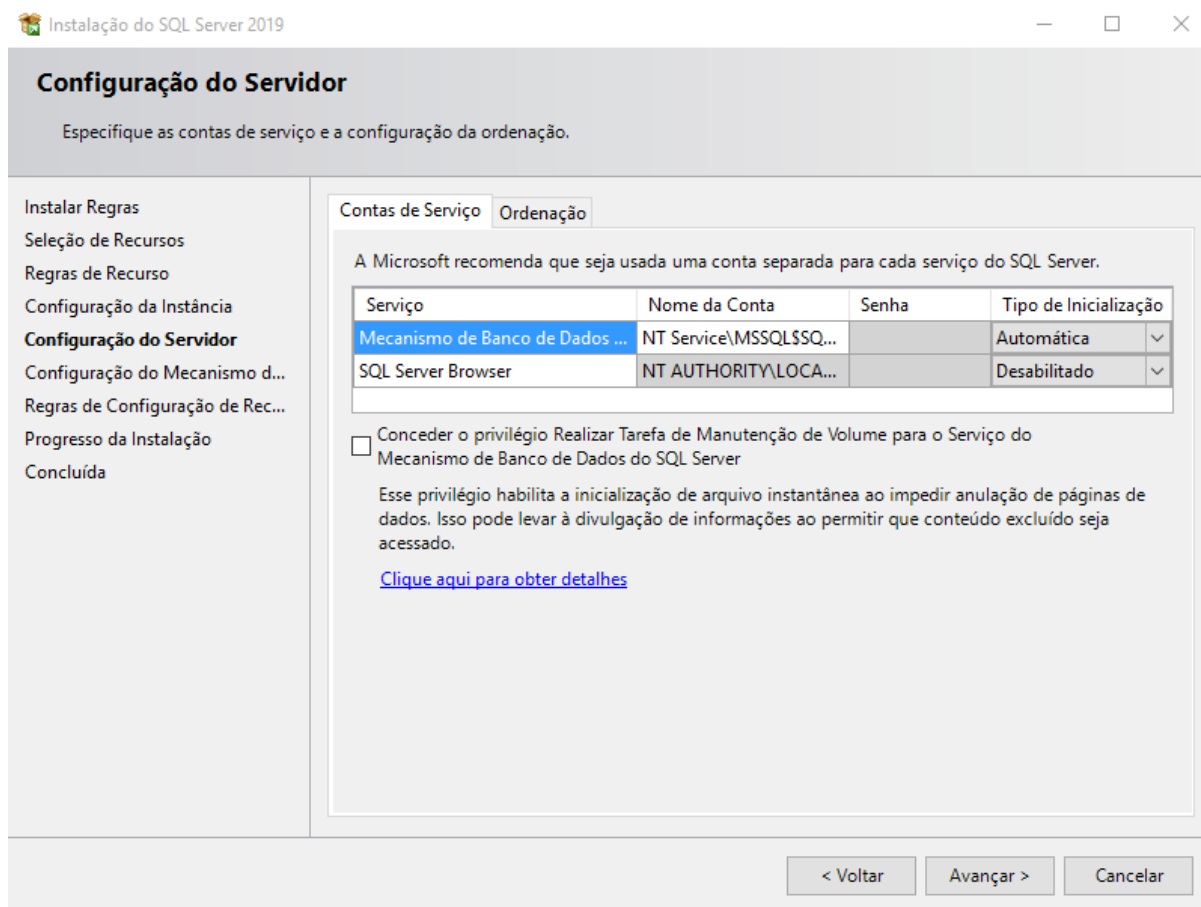
| Nome da Instância | ID da Instância | Recursos | Edição | Versão |
|-------------------|-----------------|----------|--------|--------|
|-------------------|-----------------|----------|--------|--------|

< Voltar   Avançar >   Cancelar

**Fonte: Imagem Própria**

Escolha a opção *Instância Nomeada* e mantenha o nome padrão SQLExpress para a instancia SQL. Clique em **Avançar**.

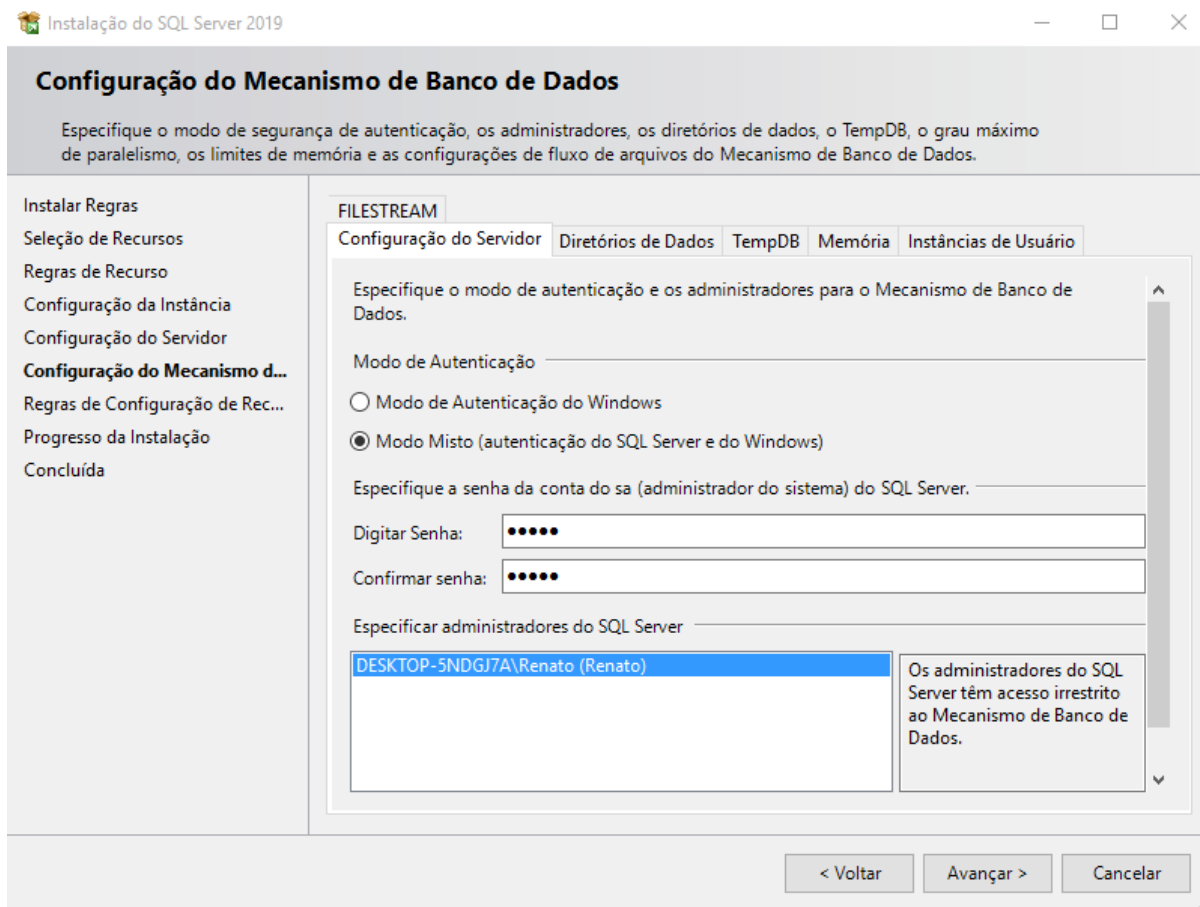
**Figura 12 – Imagem de Configuração do Servidor**



**Fonte: Imagem Própria**

Deixe as configurações de acordo com a sugeridas e clique em **Avançar**. Na tela que irá surgir, iremos configurar o mecanismo de banco de dados definindo a forma de autenticação conforme **Figura 13**.

**Figura 13 – Imagem de Configuração do Mecanismo de Banco de Dados**

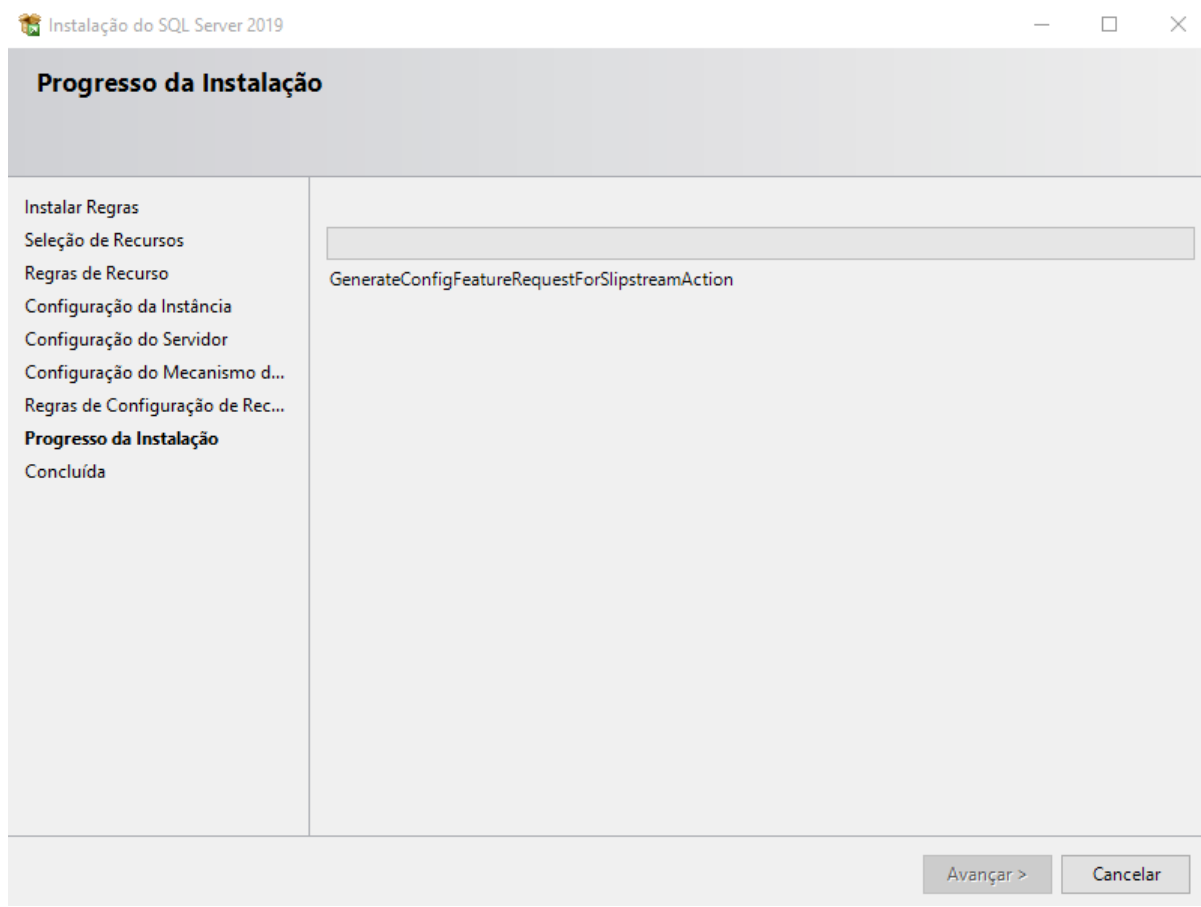


**Fonte: Imagem Própria**

Escolha a opção “Modo Misto” e especifique a senha para o usuário administrador do sistema “sa”. Feito isso clique no botão **Avançar**. O processo de instalação irá iniciar conforme **Figura 14**.



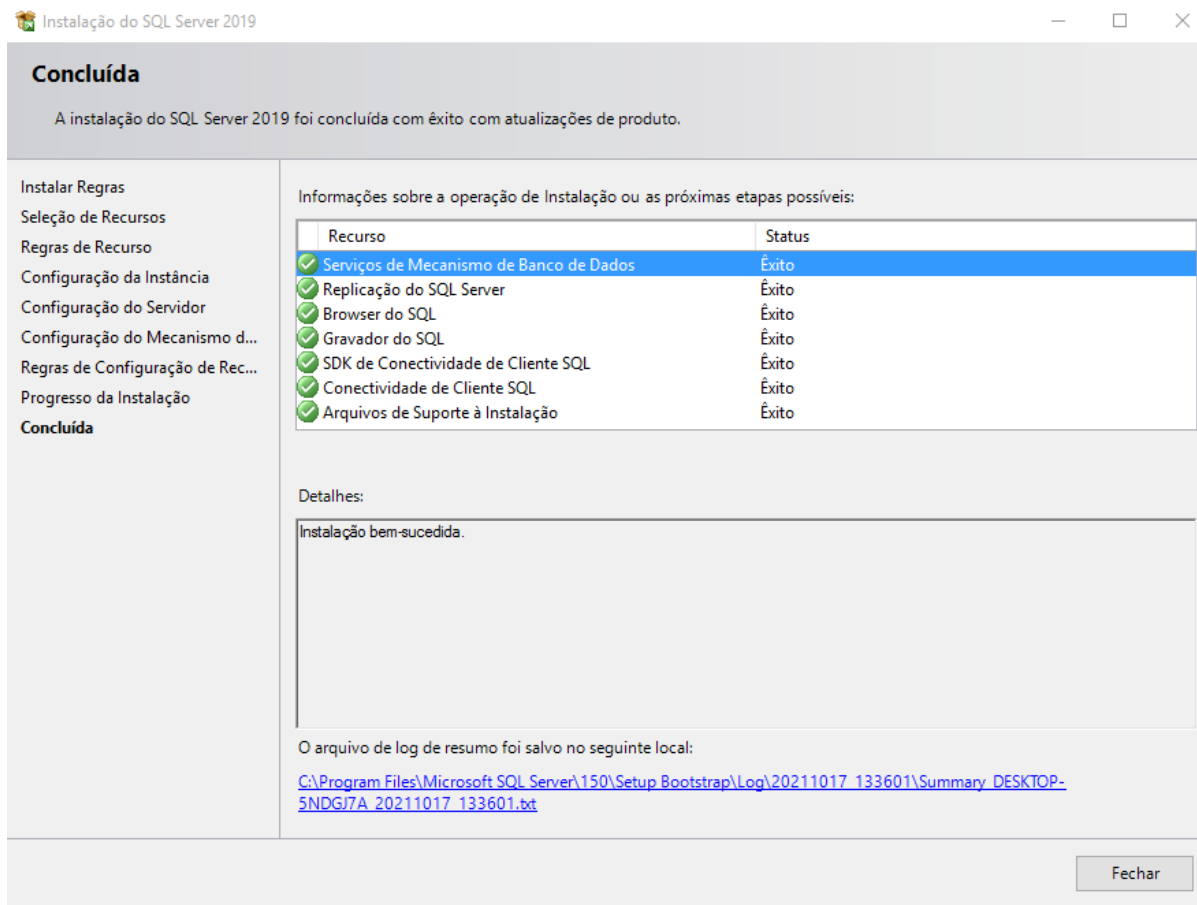
**Figura 14 – Imagem de Processo de Instalação**



**Fonte: Imagem Própria**

Após o processo de instalação será apresentado o resumo conforme **Figura 15.**

**Figura 15 – Imagem Conclusão**



**Fonte: Imagem Própria**

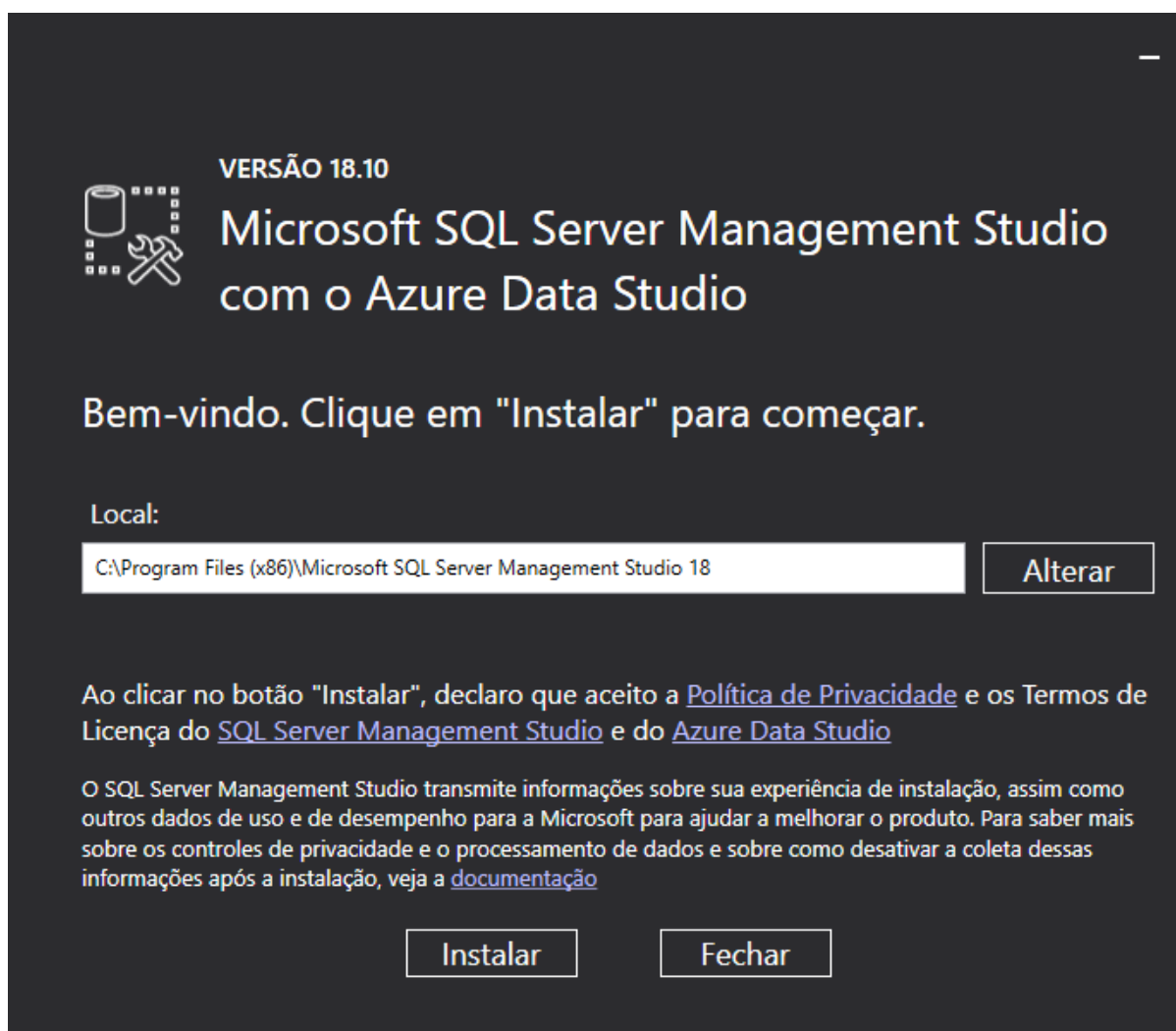
## Instalando o SQL Management Studio

Antes de realizar a instalação, é necessário baixar o instalador disponível no link abaixo.

<https://docs.microsoft.com/pt-br/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>

Após realizar o download no site da Microsoft e executar o instalador, será apresentado a **Figura 16**, clique no botão **Instalar**.

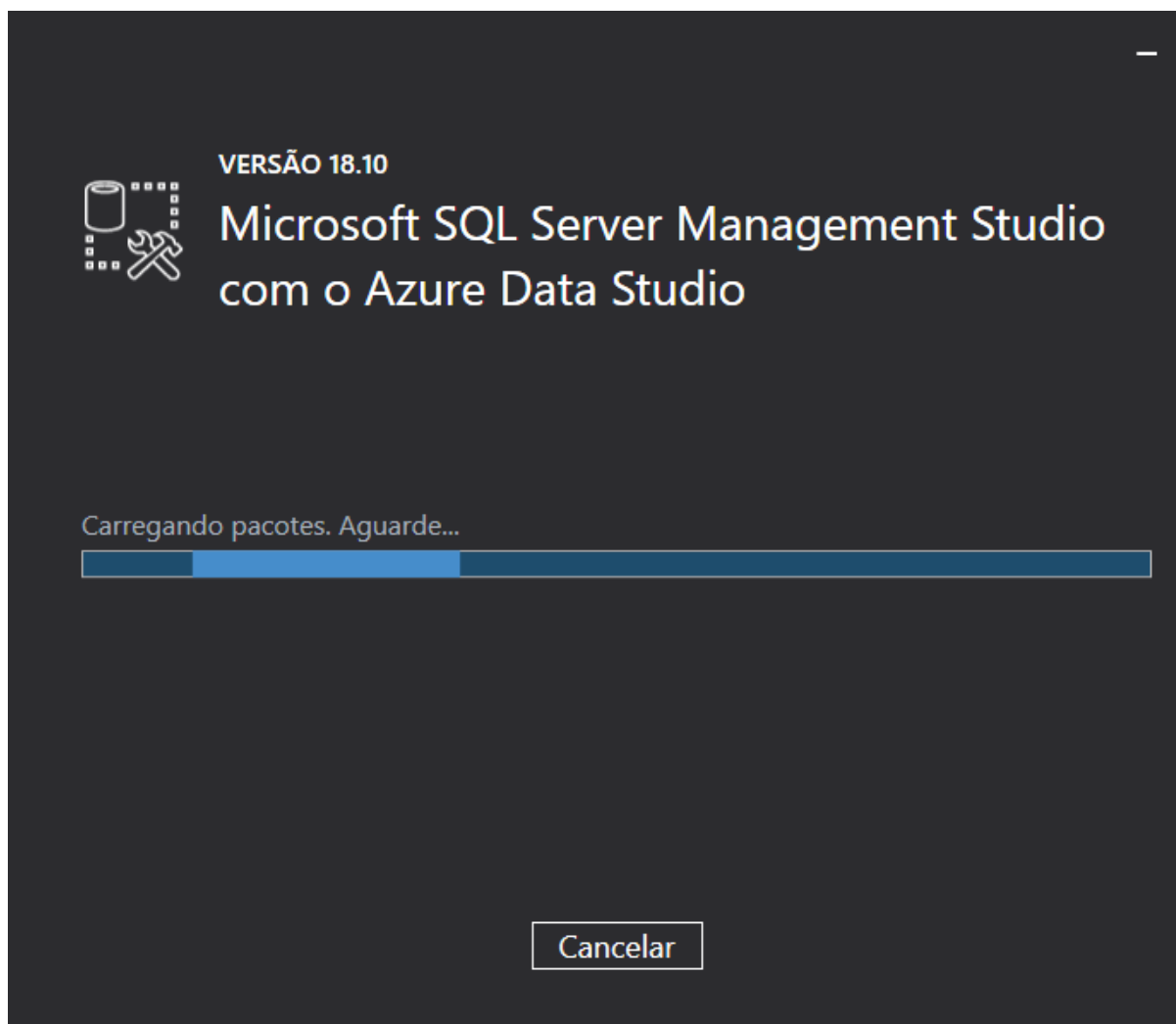
**Figura 16 – Imagem Instalador SQL Management Studio**



**Fonte: Imagem Própria**

A instalação será iniciada conforme a **Figura 17**.

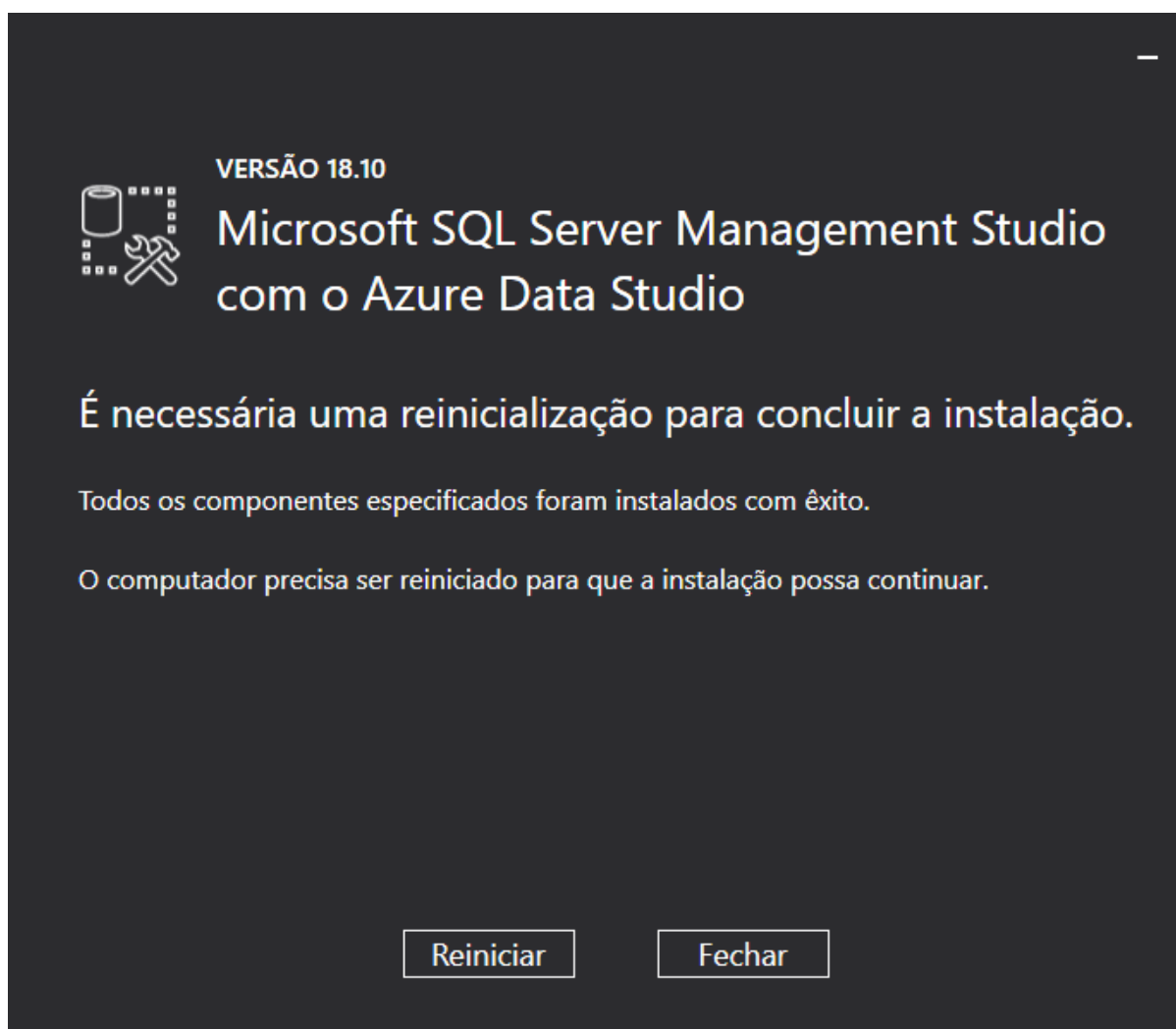
**Figura 17 – Imagem Processo de Instalação**



**Fonte: Imagem Própria**

Após concluir o processo de instalação a **Figura 18** deve ser exibida.

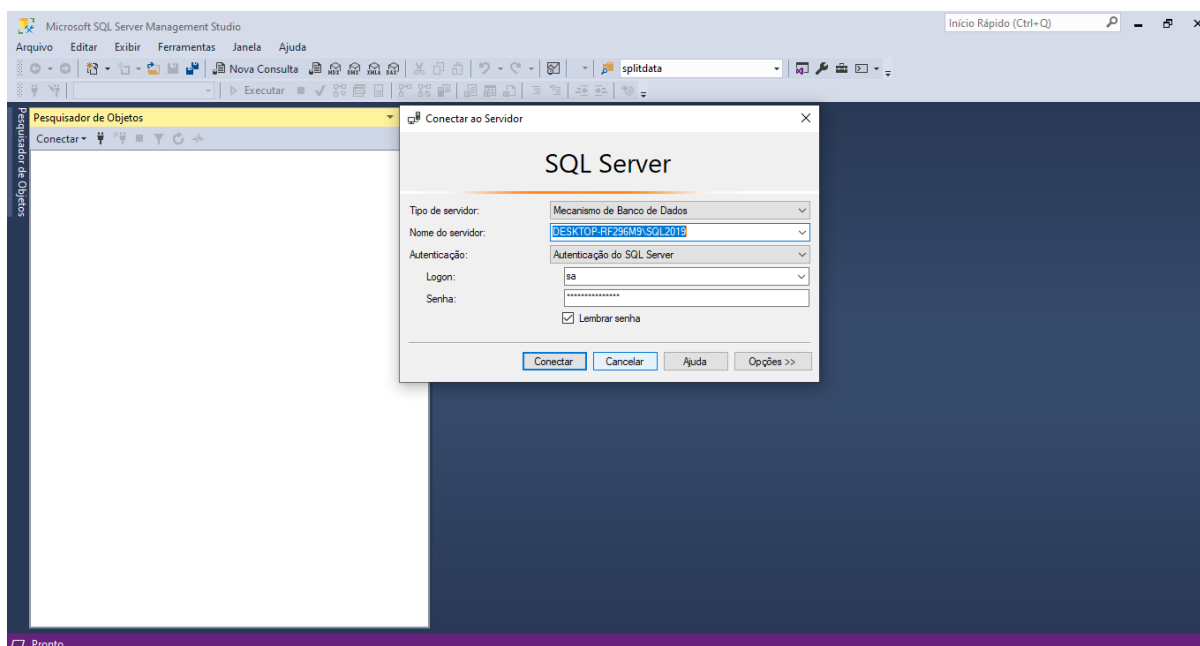
**Figura 18 – Imagem Conclusão Instalação**



**Fonte: Imagem Própria**

É necessário reiniciar o computador para que a instalação seja concluída. Após reiniciar o computador basta abrir pesquisar por Microsoft SQL Management Studio e abri-lo. A **Figura 19** mostra a interface do gerenciador de banco de dados da Microsoft.

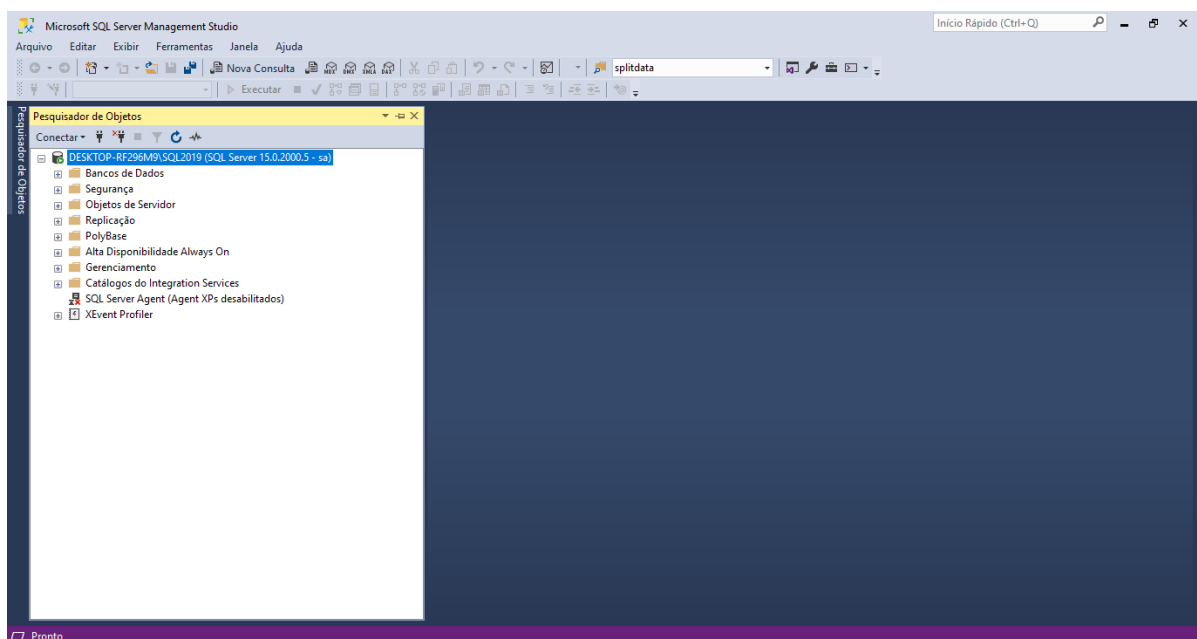
**Figura 19 – Imagem Microsoft SQL Management Studio**



**Fonte: Imagem Própria**

Informe no campo nome do servidor o nome do computador que o SQL foi instalado nesse caso está o nome do meu computador seguido do nome da instancia SQL que foi definida no momento da configuração da instancia durante o processo de instalação do SQL Server, caso tenha seguido o passo a passo o nome é SQLEXPRESS. Após informar o nome do servidor, é hora de informar o logon como configuramos para o modo misto, o usuário será o administrador do sistema representado por **sa** e a senha a definida durante a etapa de configuração do mecanismo de banco de dados. Após clicar em **Conectar**, a tela abaixo irá surgir conforme a **Figura 20**.

**Figura 20 – Imagem Microsoft SQL Management Studio**



**Fonte: Imagem Própria**

Pronto. Estamos aptos a criar banco de dados ou dar manutenções em bancos de dados existentes através de comandos SQL.

## Capítulo 2. Introdução a Banco de Dados

---

Nesse segundo capítulo, iremos entender os conceitos de banco de dados suas características e como fazer para utilizá-lo.

### Definição de Banco de Dados

---

Um banco de dados é uma coleção de informações onde armazenamos dados de forma estruturada e que podemos consultar a qualquer momento para realizar análises e extração de informações. Ele permite o registro de informações a fim de armazená-las de forma estruturada. A Linguagem SQL é a padrão na maioria dos bancos de dados, mas existem dialetos que foram definidos por cada fabricante de banco.

### Modelo Entidade Relacionamento (MER)

---

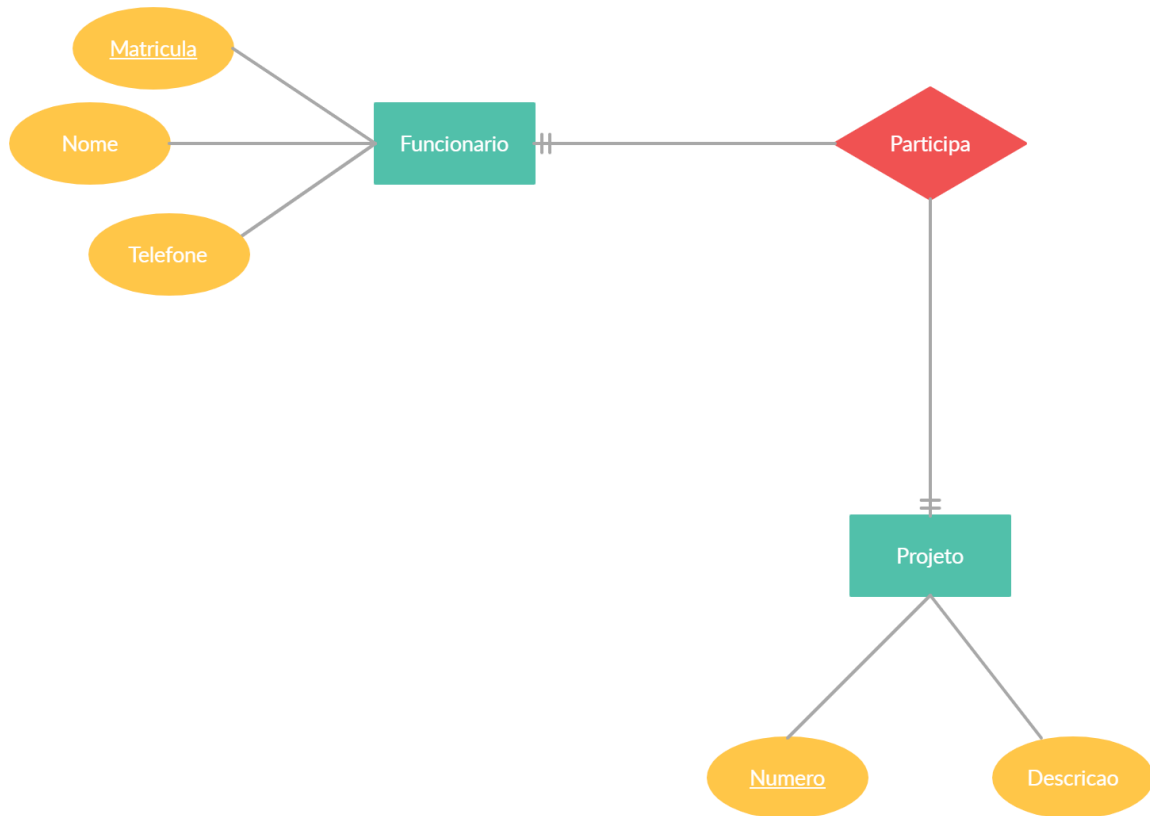
Usado para descrever objetos do mundo real fazendo uso das entidades, permite representar os tipos de relacionamentos existentes. Tem como objetivo principal definir o modelo de alto nível.

### Diagrama Entidade Relacionamento (DER)

---

É o fluxograma que demonstra de forma gráfica como as entidades se relacionam entre si dentro de um sistema, são usados para modelagem e criação de banco de dados relacionais. Utilizados também para descobrir possíveis erros de logica na criação das entidades do banco de dados.





Fonte: Imagem Própria

## Entidade

É representado por um conjunto de informações baseado em um objeto do mundo real. Toda entidade possui atributos.

- **Atributos:** *Descrevem as propriedades das entidades. São divididos em:*
  - **Chave:** *São atributos únicos em uma entidade, usado para distinguir os registros, podendo ter uma combinação de atributos que geram uma chave para a entidade. Exemplo o CPF de uma pessoa é único.*

- **Simples:** Normalmente são atributos que não possuem divisão. Um exemplo seria o CPF de uma pessoa.
- **Compostos:** São atributos que são divisíveis em mais de uma parte. Um exemplo seria nosso endereço, que pode ser dividido em rua, bairro e número.
- **Multivalorados:** Representam mais de um valor como por exemplo o Telefone, podemos ter um número de telefone fixo e um número para um ou mais celulares.

| Results |    | Messages  |               |                            |                         |             |
|---------|----|-----------|---------------|----------------------------|-------------------------|-------------|
|         | Id | Matricula | Nome          | Email                      | DataNascimento          | Telefone    |
| 1       | 1  | al001     | Renato Júnior | renatojunior2009@gmail.com | 2021-10-10 08:27:22.677 | 31 12345685 |
| 2       | 2  | al002     | Joao          | joao123@gmail.com          | 2021-10-10 08:27:55.067 | 31 1111111  |
| 3       | 3  | al003     | Maria         | maria123@gmail.com         | 2021-10-10 08:28:14.430 | 31 2222222  |
| 4       | 4  | al004     | Pedro         | pedro123@uol.com           | 2021-10-10 08:29:00.567 | 31 3333333  |
| 5       | 5  | al005     | Paulo         | paulo123@uol.com           | 2021-10-10 08:29:22.217 | 31 4444444  |
| 6       | 6  | al006     | Marcelo       | marcelo123@ig.com.br       | 2021-10-10 08:31:12.703 | 31 5555555  |
| 7       | 7  | al007     | Matheus       | matheus123@terra.com.br    | 2021-10-10 08:31:35.133 | 31 6666666  |
| 8       | 8  | al008     | Ana           | ana456@terra.com.br        | 2021-10-10 08:32:02.660 | 31 7777777  |

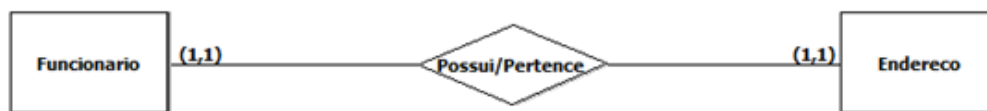
Query executed successfully.

Fonte: Imagem Própria

## Tipos de Relacionamento

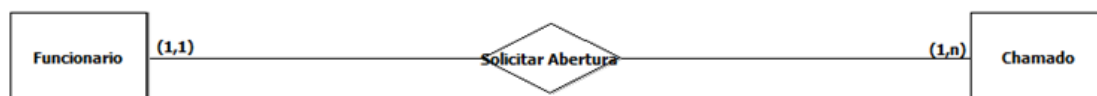
Quando temos duas ou mais tabelas associadas, dizemos que elas estão relacionadas, e esse relacionamento pode ser de 3 tipos.

- **Um para Um (1:1):** Relacionamento entre duas tabelas, quando a chave primaria do registro da tabela só pode ser utilizado uma única vez em um registro da outra tabela.



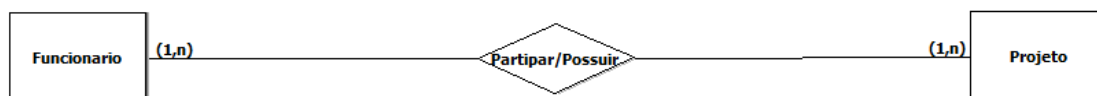
Fonte: Imagem Própria

- **Um para muitos (1:N):** Quando o registro da tabela 1 pode se relacionar com vários registros em outra tabela.



Fonte: Imagem Própria

- **Muitos para Muitos (N:N):** Quando o registro de uma tabela pode se relacionar com vários registros de outra tabela e vice versa.



Fonte: Imagem Própria

## Importância de um Banco de Dados

**Garantir a segurança dos dados:** Proteção das informações, através de mecanismos de segurança como senhas, criptografia das informações, verificações ou até mesmo leis como LGPD.

**Análise para a tomada de decisões:** Um caso bastante conhecido é o Walmart, em que por meio da análise, se descobriu que sempre quando era comprado fraldas se comprava cerveja, ou seja, quem geralmente efetuava as compras eram os pais.

## Tipos de Banco de Dados

---

**Relacionais:** Surgiu por volta na década de 70. Trata-se de um banco baseado no modelo relacional, em que o foco é a entidade conhecida também por “tabela” e o relacionamento entre as entidades.

- SQL Server
- My SQL
- Oracle
- PostgreSQL
- SQLite

**Não Relacionais:** Um banco de dados não relacional é um banco de dados que não usa o esquema de tabela de linhas e colunas, encontrado na maioria dos sistemas de banco de dados tradicionais.

- MongoDB
- Redis
- Azure DB
- Cassandra
- DynamoDB

## Como gerenciar um Banco de Dados

---

Normalmente, é usado softwares que nos permitem realizar a manipulação do banco de dados. Esses softwares são os Sistemas Gerenciadores de Banco de Dados conhecidos também como SGBD. Para nosso curso, utilizaremos o Microsoft SQL Server Management Studio.

**SGBD:** É o sistema de software responsável pelo gerenciamento de um ou mais bancos de dados, seu principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a persistência, a manipulação e a organização dos dados.

## Por que é importante aprender SQL

---

É a linguagem padrão utilizada para o gerenciamento de banco de dados relacionais, está presente nas mais diversas aplicações. As gigantes da tecnologia como Google, Amazon, Facebook, Uber utilizam o SQL para realizar consultas e analisar os dados, é muito intuitiva e de fácil aprendizagem. O conhecimento em SQL permite que o profissional consiga organizar e estruturar os bancos de dados, extraindo informações úteis.

## Dialetos

---

É a linguagem utilizada pelo banco de dados, cada fabricante possui diferenças na sintaxe ao escrever códigos SQL. Exemplo são os tipos de dados como pode ser visto abaixo.

**Transact-SQL:** Personalização da Linguagem SQL usada pelo SQL Server da Microsoft. Aqui temos o tipo INTEGER.

**PL-SQL:** Personalização da Linguagem SQL usada pela Oracle. Aqui temos o tipo NUMBER.

- Ambos atendem ao mesmo propósito.

## Grupos de Comandos

---

Os comandos SQL são divididos em 3 principais grupos.

**DDL:** Comandos responsáveis por definir e administrar objetos em um banco de dados.

- **CREATE:** Comando utilizado para criar objetos em um banco de dados.
- **ALTER:** Comando utilizado para alterar os objetos em um banco de dados.

- **DROP:** Comando utilizado para excluir os objetos em um banco de dados.

**DCL:** Comandos responsáveis por controlar o acesso aos dados.

- **GRANT:** Comando utilizado para conceder permissões de acesso.

- **Exemplo:**

```
GRANT SELECT ON nome_tabela TO nome_usuario_banco
```

- **REVOKE:** Comando utilizado para reverter uma permissão concedida.

- **Exemplo:**

```
REVOKE SELECT ON nome_tabela TO nome_usuario_banco
```

- **DENY:** Comando utilizado para não permitir o acesso.

- **Exemplo:**

```
DENY SELECT ON nome_tabela TO nome_usuario_banco
```

**DML:** Comandos responsáveis por realizar a manipulação em um banco de dados.

- **SELECT:** Comando utilizado para pesquisar informações em um banco de dados.
- **INSERT:** Comando utilizado para realizar a inserção de informação em uma tabela do banco de dados.
- **UPDATE:** Comando utilizado para atualizar as informações em um banco de dados.
- **DELETE:** Comando utilizado para deletar um mais registro(s) em uma tabela no banco de dados.

## Como utilizar o comando *CREATE DATABASE*

---

Comando utilizado para criar um novo banco de dados.

**Sintaxe:** **CREATE DATABASE** *nome\_banco*

**Exemplo:** **CREATE DATABASE** biblioteca

Ao executar esse comando no Gerenciador de Banco de Dados (SGBD), um novo banco será criado. Veremos um exemplo prático durante as aulas.

## Como utilizar o comando *CREATE TABLE*

---

Comando utilizado para criar uma nova tabela.

**Sintaxe:** **CREATE TABLE** *nome\_tabela*

```
(  
    nome_campo1 tipo_dado argumento,  
    nome_campo2 tipo_dado argumento,  
    nome_campo3 tipo_dado argumento,  
    nome_campo tipo_dado argumento,  
    CONSTRAINT nome_constraint PRIMARY KEY(nome_campo)  
)
```

***nome\_tabela*** é o nome da tabela a ser criada.

***nome\_campo*** é o nome do campo a ser criado.

***tipo\_dado*** são os tipos dos campos que serão criados.

**Exemplo de Tipos mais usados:** int, bigint, varchar, char, bit e date, datetime.

**Constraint** – Usado para criar restrições nas tabelas, as constraints mais utilizadas são:

- **NOT NULL:** Usado para definir que o campo da tabela que possua esse argumento seja obrigatório informar um valor.

- **PRIMARY KEY:** Usado para definir qual será o campo chave da tabela.
- **FOREIGN KEY:** Usado para ligar o campo de uma tabela em um campo de outra tabela, ou seja, usamos para criar relação entre as tabelas.

**Exemplo:**

```
CREATE TABLE livros
(
    Codigo INT IDENTITY(1,1) NOT NULL,
    Descricao VARCHAR(60) NOT NULL,
    ISBN INT NOT NULL,
    CONSTRAINT PK_Codigo PRIMARY KEY(Codigo)
)
```

Veremos exemplos práticos durante as aulas.

### Como utilizar o comando INSERT INTO

---

Comando utilizado para inserir informações nas tabelas.

**Sintaxe:** **INSERT INTO** nome\_tabela (nome\_coluna1, nome\_coluna2, nome\_coluna3...) **VALUES** (valor1, valor2, valor3...)

**Exemplo:**

```
INSERT INTO clientes (Nome, CPF, Email, Especial)
VALUES ('Renato', '123456789', 'renatojunior2009@gmail.com', 0)
```

A ordem em que as colunas forem definidas deve ser e a mesma que para a passagem dos valores. Veremos exemplos práticos durante as aulas.

### Como utilizar o comando SELECT

---

Comando utilizado para pesquisar informações em um banco de dados.



**Sintaxe:** **SELECT** nome coluna **FROM** nome\_tabela

**Exemplo:**

```
SELECT
    Nome
FROM clientes
```

*Nesse exemplo acima, ao executar esse comando no SGBD o retorno da consulta irá trazer como resultado o nome de todos os clientes existentes na tabela clientes.*

*Variações do Comando **SELECT***

### **SELECT COM MAIS DE UMA COLUNA**

**Sintaxe:** **SELECT** nome\_coluna1, nome\_coluna2 **FROM** nome\_tabela

**Exemplo:**

```
SELECT
    Nome,
    Email
FROM clientes
```

*No exemplo acima, ao executar esse comando no SGBD o retorno da consulta irá trazer como resultado o nome e o e-mail de todos os clientes existentes na tabela clientes.*

### **SELECT COM A CLÁUSULA ORDER BY**

A cláusula ORDER BY é usada para ordenar o resultado em uma consulta.

- O resultado pode ser ordenado de 2 formas.

- **ASC** – Ascendente (*Ordem Crescente*)
- **DESC** – Descendente (*Ordem Decrescente*)

**Sintaxe:** **SELECT** nome\_coluna1, nome\_coluna2 **FROM** nome\_tabela  
**ORDER BY** nome\_coluna\_ordenacao

**Exemplo:**

```
SELECT
    Nome,
    Email
FROM clientes
ORDER BY Nome
```

No exemplo acima, ao executar esse comando no SGBD o retorno da consulta irá trazer como resultado o nome e o e-mail de todos os clientes existentes na tabela clientes ordenados pelo nome do cliente.

### **SELECT COM A CLÁUSULA DISTINCT**

A cláusula DISTINCT é usada para eliminar as repetições nos registros.

Imagine o cenário onde precisamos a partir da tabela de clientes gerar uma lista de e-mail para que seja enviado um e-mail de promoção de final de ano para os clientes e dentre o cadastro existem 2 clientes com o mesmo e-mail, para evitar que o cliente receba 2 e-mails promocionais podemos utilizar a cláusula *DISTINCT* para eliminar os registros duplicados retornando o e-mail apenas uma vez na consulta. Veja o exemplo abaixo.

**Sintaxe:** **SELECT DISTINCT** nome\_coluna **FROM** nome\_tabela

**Exemplo:**

```
SELECT  
    DISTINCT  
    Email  
FROM clientes
```

### **SELECT COM A CLÁUSULA WHERE**

A cláusula WHERE permite filtrar registros em uma tabela.

Imagine o cenário onde precisamos filtrar a partir da tabela de clientes todos os clientes que são considerados clientes especiais, para isso podemos utilizar a cláusula *WHERE* que nos permitirá realizar o filtro através do campo desejado. Veja o exemplo abaixo.

**Sintaxe:** **SELECT \* FROM** *nome\_tabela* **WHERE** *nome\_coluna* = valor

**Exemplo:**

```
SELECT  
    *  
FROM clientes  
WHERE Especial = 1
```

### **SELECT + WHERE + OPERADOR AND / OR**

O operador **AND** e **OR** permite filtrar registros usando mais de uma coluna da tabela. O operador **AND** mostrará o registro se ambas as condições forem verdadeiras, já o operador **OR** mostrará o registro se ao menos uma das condições forem verdadeiras.

Uso do Operador **AND**

Imagine o cenário onde precisamos filtrar a partir da tabela de clientes todos os clientes que são considerados clientes especiais e que o nome seja Maria, para

isso podemos utilizar a cláusula *WHERE* que nos permitirá realizar o filtro pela primeira coluna e o operador *AND* para a segunda coluna. Veja o exemplo abaixo.

**Sintaxe:** **SELECT \* FROM** *nome\_tabela* **WHERE** nome\_coluna = valor **AND**  
nome\_coluna = 'valor'

**Exemplo:**

```
SELECT
    *
FROM clientes
WHERE Especial = 1 AND
Nome = 'Maria'
```

Uso do Operador **OR**

Imagine o cenário onde precisamos filtrar a partir da tabela de clientes todos os clientes que o nome seja Maria ou Renato, para isso podemos utilizar a cláusula *WHERE* que nos permitirá realizar o filtro pela primeira coluna e o operador *OR* para a segunda coluna. Veja o exemplo abaixo.

**Sintaxe:** **SELECT \* FROM** *nome\_tabela* **WHERE** nome\_coluna = valor **OR**  
nome\_coluna = 'valor'

**Exemplo:**

```
SELECT
    *
FROM clientes
WHERE Nome = 'Renato' OR
Nome = 'Maria'
```

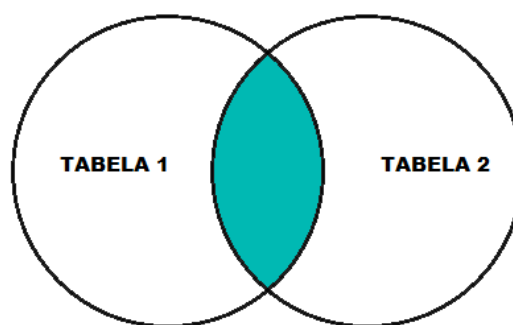
### *Como utilizar o comando JOIN*

---

Comando usado para obter registros de duas ou mais tabelas, baseado no relacionamento entre as colunas das tabelas.

- Tipos de **JOIN**
  - **INNER JOIN**
  - **LEFT JOIN**
  - **RIGHT JOIN**

**INNER JOIN:** Retorna os registros da tabela quando existir ao menos uma associação entre as tabelas analisadas no comando.



Fonte: Imagem Própria

**Sintaxe:** **SELECT** nome\_coluna1, nome\_coluna2  
**FROM** nome\_tabela1  
**INNER JOIN** nome\_tabela2  
**ON** nome\_tabela1.nome\_coluna =  
nome\_tabela2.nome\_coluna

**Onde:**

*nome\_tabela1* é a tabela da esquerda.

*nome\_tabela2* é a tabela da direita.

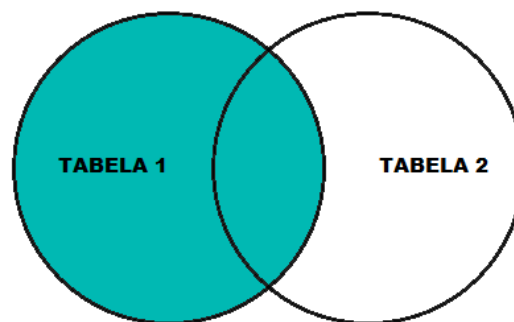
Imagine o cenário onde temos as tabelas “livros” e “autores”, onde na tabela “livros” temos os campos **Codigo**, **Descricao**, **ISBN**, **CodigoAutor** e na tabela “autores” temos os campos **Codigo**, **Nome**. Na tabela de livros temos 4 registros, onde apenas 1 deles não foi informado o autor, e na tabela autores temos 2 registros, se quiséssemos saber todos os livros que possuem autores devemos montar a

consulta abaixo. Onde seria mostrado como resultado apenas 3 registros, uma vez que em nosso exemplo 1 livro não possui autor.

**Exemplo:**

```
SELECT
    livros.Descricao,
    autores.Nome
FROM livros
INNER JOIN autores
    ON livros.CodigoAutor = autores.Codigo
```

**LEFT JOIN:** Retorna os registros da tabela da esquerda, mesmo se não existir associação na tabela da direita.



Fonte: Imagem Própria

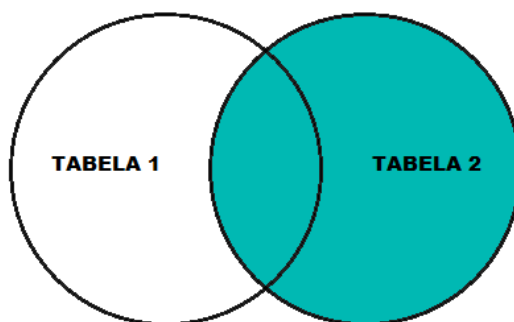
**Sintaxe:** **SELECT** nome\_coluna1, nome\_coluna2  
**FROM** nome\_tabela1  
**LEFT JOIN** nome\_tabela2  
**ON** nome\_tabela1.nome\_coluna =  
nome\_tabela2.nome\_coluna

**Onde:**

*nome\_tabela1* é a tabela da esquerda.  
*nome\_tabela2* é a tabela da direita.

Com base no cenário anterior, o resultado da consulta trará todos os livros que possuem autores e também os que não possuem, uma vez que o *LEFT JOIN* considera todos os registros tabela da esquerda e os registros que possuem associação na tabela da direita.

**RIGHT JOIN:** Retorna os registros da tabela da direita, mesmo se não existir associação na tabela da esquerda.



Fonte: Imagem Própria

**Sintaxe:** `SELECT nome_coluna1, nome_coluna2`  
`FROM nome_tabela1`  
`RIGHT JOIN nome_tabela2`  
`ON nome_tabela1.nome_coluna =`  
`nome_tabela2.nome_coluna`

**Onde:**

*nome\_tabela1* é a tabela da esquerda.

*nome\_tabela2* é a tabela da direita.

Com base no cenário anterior, o resultado da consulta trará todos os livros que possuem autores e também os que não possuem, uma vez que o *RIGHT JOIN* considera todos os registros tabela da direita e os registros que possuem associação na tabela da esquerda.

### *Como utilizar o comando UPDATE*

---

Comando utilizado para atualizar as informações em um banco de dados.

**Sintaxe:** `UPDATE nome_tabela SET nome_campo = valor`

**Exemplo:**

```
UPDATE livros SET Descricao = 'Robert Cecil Martin'
```

*O comando acima irá atualizar a descrição de todos os livros cadastrados na tabela. Evite usar o comando UPDATE sem a cláusula WHERE, o resultado pode ser catastrófico a não ser que realmente seja necessário.*

*Variações do Comando **UPDATE***

### **UPDATE + WHERE**

**Sintaxe:** `UPDATE nome_tabela SET nome_campo = novo_valor WHERE  
nome_campo = valor`

**Exemplo:**

```
UPDATE livros SET Descricao = 'Robert Cecil Martin'  
WHERE Codigo = 1
```

*Fazendo uso do WHERE junto ao UPDATE garantimos maior segurança na atualização das informações das tabelas, como pode ser visto no exemplo acima, apenas o livro com código 1 terá sua descrição atualizada.*

### **UPDATE + WHERE + OPERADOR AND**

**Sintaxe:** `UPDATE nome_tabela SET nome_campo = novo_valor WHERE  
nome_campo = valor AND nome_campo = valor`

**Exemplo:**



```
UPDATE livros SET Descricao = 'Robert Cecil Martin',  
ISBN = 858585  
WHERE Descricao = 'teste' AND  
AnoLancamento = 2021
```

*Fazendo uso do WHERE + OPERADOR AND junto ao UPDATE garantimos maior segurança na atualização das informações das tabelas, como pode ser visto no exemplo acima, apenas o livro com descrição teste e ano lançamento igual a 2021 terá sua descrição e o ISBN atualizados.*

### **UPDATE EM MAIS DE UM CAMPO DA TABELA**

Para atualizar mais de um campo da tabela, basta separar os campos por virgula conforme a sintaxe abaixo.

**Sintaxe:** UPDATE nome\_tabela SET nome\_campo1 = novo\_valor,  
nome\_campo2 = novo\_valor WHERE nome\_campo = valor

**Exemplo:**

```
UPDATE livros SET Descricao = 'Robert Cecil Martin',  
ISBN = 858585  
WHERECodigo = 1
```

*No exemplo acima além de atualizar a descrição do livro estamos atualizando também o ISBN do livro onde o código é igual a 1.*

### *Como utilizar o comando DELETE*

---

Comando utilizado para deletar as informações em um banco de dados.

**Sintaxe:** DELETE FROM nome\_tabela

**Exemplo:**

```
DELETE FROM livros
```

No exemplo acima irá deletar todos os registros da tabela livros, assim como mencionado no comando UPDATE, evite o uso do DELETE sem a cláusula WHERE, a não ser que realmente seja necessário.

### Variações do Comando **DELETE**

#### **DELETE + WHERE**

**Sintaxe:** **DELETE FROM** nome\_tabela **WHERE** nome\_campo = valor

**Exemplo:**

```
DELETE FROM livros WHERECodigo = 9
```

Fazendo uso do WHERE junto ao DELETE garantimos maior segurança, como pode ser visto no exemplo acima, apenas o livro com código 1 será deletado da tabela livros.

#### **DELETE + WHERE + OPERADOR AND**

**Sintaxe:** **DELETE FROM** nome\_tabela **WHERE** nome\_campo = valor **AND**  
nome\_campo = valor

**Exemplo:**

```
DELETE FROM livros WHERECodigo = 9  
ANDDescricao = 'teste'
```

Fazendo uso do WHERE + OPERADOR AND junto ao DELETE garantimos maior segurança, como pode ser visto no exemplo acima, apenas o livro com código 1 e descrição igual a teste será deletado da tabela livros.

### Como utilizar o comando **ALTER TABLE**

---

Comando utilizado para alterar a estrutura da tabela em um banco de dados.

Para **CRIAR** um novo campo na tabela

**Sintaxe:** **ALTER TABLE** *nome\_tabela* **ADD** *nome\_coluna* | *tipo\_dados* | argumentos

**Exemplo:**

```
ALTER TABLE livros ADD AnoLancamento int NOT NULL
```

No exemplo acima será criado na tabela livros uma coluna com nome AnoLancamento do tipo **int**, ou seja, que recebe apenas números e que será obrigatório seu preenchimento devido a constraint **NOT NULL** passado como argumento para o comando.

Para **REMOVER** uma coluna da tabela

**Sintaxe:** **ALTER TABLE** *nome\_tabela* **DROP COLUMN** *nome\_coluna*

**Exemplo:**

```
ALTER TABLE livros DROP COLUMN AnoLancamento
```

No exemplo acima o comando ao ser executado remove da tabela livros a coluna com nome AnoLancamento.

*Como utilizar o comando DROP TABLE*

---

Comando utilizado para excluir a tabela em um banco de dados.

**Sintaxe:** **DROP TABLE** *nome\_tabela*

**Exemplo:**

```
DROP TABLE categoria_livro
```

No exemplo acima o comando ao ser executado apaga a tabela categoria\_livros do banco de dados.

### *Como utilizar o comando DROP DATABASE*

Comando utilizado para excluir o banco de dados.

**Sintaxe:** **DROP DATABASE** *nome\_banco*

**Exemplo:**

**DROP DATABASE** locadora

*No exemplo acima o comando ao ser executado deleta o banco de dados com nome locadora.*

## Capítulo 3. Introdução ao Entity Framework Core

---

Neste terceiro capítulo, iremos dar uma introdução sobre esse poderoso ORM e uma dos mais utilizados em aplicações na atualizada.

### O que é ORM

---

Object Relational Mapping (ORM) ou Mapeamento Objeto Relacional tem a função de abstrair as tabelas e os relacionamentos de um banco de dados relacional, realizar o mapeamento dos dados e entregá-los ao desenvolvedor de uma forma que esses dados sejam convertidos em objetos na linguagem orientada a objetos a qual ele estiver trabalhando.

### Vantagens e Desvantagens de utilizar um ORM

---

- **Vantagens:**
  - Facilidade para escrever consultas simples, realizar inserção ou atualização de dados e exclusão de registros no banco de dados.
  - Redução no Tempo de Desenvolvimento.
  - Redução de Custo de Desenvolvimento.
- **Desvantagens:**
  - Demora para aprender a utilizar o ORM da forma correta.
  - ORM tende a ser mais lento.
  - Para montagens de consultas mais complexas pode não ser tão interessante devido a alta complexidade para a criação das expressões utilizando o LINQ, além dessa consulta gerada pelo próprio EF não ser tão performática.

## O que é Entity Framework Core

---

O EF (Entity Framework) Core é uma versão leve, extensível de software livre e multiplataforma da popular tecnologia de acesso a dados do Entity Framework. EF Core pode servir como um mapeamento relacional de objeto (O/RM), que:

- Permite que os desenvolvedores do .NET trabalhem com um banco de dados usando objetos .NET.
- Elimina a necessidade de maior parte do código de acesso a dados que normalmente precisa ser gravado.

## Definição de Modelo no Entity Framework Core

---

É utilizado para realizar o acesso a dados sendo representado por uma classe. Há duas formas de criar um modelo no Entity Framework Core.

- Através de um banco de dados já existente, ou seja, quando já temos um banco de dados criado.
- Fazendo uso das migrações, onde criamos o banco de dados através dos modelos definidos.

```

0 references
public class Pedido
{
    0 references
    public int Numero { get; set; }
    0 references
    public string Nome { get; set; }
    0 references
    public decimal Total { get; set; }
    0 references
    public List<Produto> Produtos { get; set; }
}

1 reference
public class Produto
{
    0 references
    public int Codigo { get; set; }
    0 references
    public string Descricao { get; set; }
    0 references
    public decimal Valor { get; set; }
}

```

Fonte: Imagem Própria

## Evolução

| Evolução  |   |
|---|---|
| Entity Framework 6  | Entity Framework Core                                   |
| Primeiro Release Lançado em 2008 com o .NET Framework 3.5 SP1 | Primeiro Release em Junho de 2016 com o .NET Core 1.0   |
| Roda apenas em Windows  | Multiplataforma   |
| Funciona no .NET Framework 3.5                                | Funciona no .NET Framework 4.5 ou Superior ou .NET Core |
| Open Source   | Open Source   |

Fonte: Imagem Própria

## Por que escolher o Entity Framework Core

### Produtividade

- Você não precisa saber SQL, pois o Entity Framework faz toda abstração através do uso do LINQ.
- Todas as consultas são realizadas com foco no objeto e não no banco de dados.

```

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        using(var db = new MeuContexto())
        {
            //retorna o produto cujo o codigo seja igual a 1.
            var produto = db.Produtos.Where(x => x.Codigo == 1).FirstOrDefault();
        }
    }
}

```

Fonte: Imagem Própria

### Onde podemos utilizá-lo?

Uma das motivações que podemos considerar para utilizá-lo, seria em cenários onde o banco de dados geralmente é criado no final da criação dos modelos. Essa criação fica a cargo de ferramentas que usamos para realizar as migrações.

### Suporte a Banco de Dados

O Entity Framework Core é compatível com diversos banco de dados. O acesso a esses bancos é por meio de provedores de bancos de dados.

- SQL Server
- SQLite



- PostgreSQL
- MySQL
- Azure Cosmo DB
- Oracle

## Tipos de Abordagens

---

**Database First:** Abordagem utilizada geralmente quando existe um banco de dados criado. Imagine uma empresa que já possui uma aplicação legada e deseja migrar essa aplicação para o .NET Core. Obviamente, o banco não será criado do zero, então a estratégia seria utilizar-se desse banco e criar todos os modelos de dados de forma manual ou fazer uso do scaffold para que os modelos sejam criados de forma automática.

**Code First:** A ideia dessa abordagem é o desenvolvedor escrever classes que descrevam o modelo conceitual, para que o Entity gere o banco de dados com base nesse modelo de forma automática. Usando essa abordagem, temos um maior controle do banco de dados a ser gerado.

## DbContext e DbSet

---

**DbContext:** Classe de contexto, sua principal responsabilidade é coordenar as funcionalidades do EF core para um ou mais modelos de dados, é o contexto que delega quais serão as entidades que pertencerão ao modelo de dados, tem a responsabilidade de fazer a ligação da aplicação com o banco de dados.

**DbSet:** Responsável por criar uma coleção de entidades, são especificadas no contexto. Normalmente para cada tabela a ser gerada em um banco de dados cria-se uma propriedade do tipo DbSet.

## Migrations

---

Responsável por realizar a criação da base de dados e as suas respectivas tabelas de acordo com os modelos de dados criados. Caso seja necessário incluir

mais um campo em uma tabela, por exemplo, basta incluir o novo campo no modelo de dados e gerar uma nova migração para que no esquema de banco seja sincronizado com o modelo de dados da aplicação.

O comando abaixo é usado para criar uma nova migração.

➤ *add-migration nome\_migracao*

Será criada uma pasta chamada Migrations e para cada migração gerada, serão gerados 3 arquivos.

Para que o Entity crie ou atualize o banco de dados e as tabelas a partir da migração criada, é necessário executar o comando abaixo.

➤ *update-database*

## Capítulo 4. Introdução ao Dapper

Neste quarto capítulo, teremos uma introdução sobre o micro ORM Dapper, capaz de simplificar o trabalho para obter as informações vindas de um banco de dados através da simplicidade de fazer o mapeamento dos objetos, além da alta performance.

### Desempenho

Desempenho do mapeamento SELECT em mais de 500 interações.

| Método                | Duração |
|-----------------------|---------|
| Hand Coded            | 47 ms   |
| Dapper                | 49 ms   |
| ServiceStack. OrmLite | 50 ms   |
| PetaPoco              | 52 ms   |
| BLToolkit             | 80 ms   |
| SubSonic CodingHorror | 107 ms  |
| Nhibernate SQL        | 104 ms  |
| Linq 2 SQL            | 181 ms  |
| Entity Framework      | 631 ms  |

Fonte: Imagem Própria

### Vantagens e Desvantagens

- **Vantagens:**
  - Consultas complexas possuem uma excelente performance, uma vez que as consultas são feitas de forma manual.

- Recebe o SQL bruto para realizar as operações no banco de dados.
- **Desvantagens:**
  - Dificuldade na escrita de queries complexas caso o desenvolvedor não tenha um bom conhecimento na linguagem SQL.
  - Não possui Lazy Load “Carregamento sob Demanda”, os dados são obtidos de uma só vez.
  - Não possui Tracking como no Entity Framework, que fica observando as mudanças dos objetos para realizar a persistência no banco.

### O que ele resolve?

---

Caso você esteja passando por problemas de performance em seu projeto, principalmente quando se trata de consultas complexas na aplicação o Dapper, é altamente recomendável devido seu alto desempenho nesses cenários onde o tempo é primordial para o negócio, seja na geração de um relatório, seja na gravação de um grande volume de dados ele irá contribuir muito para que a aplicação se torne performática.

### Entity Framework Core vs Dapper

---

Diferença entre Entity Framework e Dapper.

| Comparativo                | Entity Framework   | Dapper  |
|----------------------------|--|---|
| Criação de Tabelas         | Possibilidade de escrever classes que mais tarde serão transformadas em tabelas no banco de dados.             | Não é possível gerar as tabelas na base de dados. As tabelas devem ser criadas anteriormente antes de serem mapeadas. |
| Rastreabilidade            | Capacidade de rastrear alterações nos objetos afim de simplificar o processo de atualização do banco de dados. | Não possuem rastreio de objetos com isso não observa as mudanças.   |
| Sintaxe                    | Utiliza a Sintaxe do LINQ para todas as consultas de objetos.  | Facilita a parametrização das consultas por utilizar SQL bruto.   |
| Esforço de Desenvolvimento | Reduz o Custo e Tempo de Desenvolvimento.  | Pode facilmente executar consultas.   |
| Alterações em Banco        | Se houver alguma mudança no banco será necessário atualizar o esquema em seu projeto.                          | Basta ajustar a consulta na aplicação para que a mudança seja identificada pelo Dapper.                               |

**Fonte: Imagem Própria**

## Uso do Entity Framework Core com Dapper

É possível sim, uma vez que na maioria das aplicações as funcionalidades de consulta superam as operações de gravação. Os ORM's são tradições que facilitam bastante a vida do desenvolvedor, pois abstraem a complexidade de conexão com o banco e facilita a escrita das consultas, mas isso pode trazer problemas de performance, uma vez que a query gerada automaticamente pelo ORM pode não ser das mais performáticas, aí que entra o uso do micro ORM Dapper para realizar o trabalho dessas consultas trazendo desempenho para a aplicação.

## Referências

---

DAPPER Tutorial. **Dapper**. Disponível em: <<https://dapper-tutorial.net/dapper>>. Acesso em: 1 nov. 2021

DOCUMENTAÇÃO Técnica do SQL Server. **Docs Microsoft**, c2021. Disponível em: <<https://docs.microsoft.com/pt-br/sql/sql-server/?view=sql-server-ver15>>. Acesso em: 1 nov. 2021.

VICKERS, Arthur; OKUMS, Fernando. Entity Framework Core. **Docs Microsoft**, 25 mai. 2021. Disponível em: <<https://docs.microsoft.com/pt-br/ef/core/>>. Acesso em: 1 nov. 2021.