apex ensino

# Banco de Dados

Público alvo: Desenvolvedores e estudantes que queiram compreender os conceitos básicos de banco de dados.

Pré-requisitos: Lógica e Algoritmos.

# Índice

# Copyright

## Equipe

| Conteúdos | Diagramação | Revisão |
|---|---|---|
| Felipe de Oliveira | Fernanda Pereira | Fernanda Pereira |
| Gustavo Rosauro | | |
| Ralf S. de Lima | | |

## Histórico de edições

| Edição | Idioma | Edição |
|---|---|---|
| 1ª | Português | Janeiro de 2017 |
| 2ª | Português | Julho de 2019 |
| 3ª | Português | Julho de 2020 |

# Capítulo 01 - Banco de Dados

Esse capítulo irá abordar os conceitos básicos necessários para compreendermos o uso do banco de dados em nossas aplicações. Serão abordados temas como: instalação do SQL Server, comandos SQL, interação entre tabelas, back-ups e stored procedures.

## Compreendendo o uso do Banco de Dados

A utilização de banco de dados é algo corriqueiro no desenvolvimento de sistemas, independente se é desktop, web ou mobile. A expressão banco de dados remete ao armazenamento de informações, que podem ser tanto textuais quanto de arquivos como áudios, vídeos, imagens, arquivos de texto, planilhas, entre outros formatos.

Podemos utilizar os bancos de dados com qualquer linguagem de desenvolvimento, não há exclusividade de uma linguagem com um banco de dados, porém há bancos que são mais fáceis de realizar a comunicação com determinadas linguagens, como por exemplo: C# e SQL Server ou PHP e MySQL.

## DBA

O termo DBA se refere ao Administrador de Banco de Dados ou Database Administrator, ele é responsável por toda a arquitetura do banco de dados, além de efetuar toda a configuração para a instalação e ter conhecimento para averiguar o hardware que será utilizado para que o banco de dados funcione.

Muitas empresas possuem um perito em banco de dados, no caso um DBA para poder modelar a estrutura que deverá ser desenvolvida para que as linguagens de programação tenham acesso a manipular dados.

# SGBD

O termo SGBD ou Sistema Gerenciador de Banco de Dados é uma determinada ferramenta para podermos armazenar dados. Quando mencionamos banco de dados, estamos sendo genéricos e especificando que iremos utilizar algum software para guardar informações, agora quando especificamos o tipo do banco de dados, estamos mencionando o SGBD.

Há vários SGBDs a disposição dos desenvolvedores e DBAs, abaixo você pode conferir alguns exemplos de sistemas gerenciadores de banco de dados:






# Capítulo 02 - SQL

Neste capítulo iremos abordar sobre os conceitos da linguagem SQL, que é utilizada nos principais sistemas gerenciadores de banco de dados.


## Compreendendo os comandos SQL

O SQL é uma linguagem para podermos criar estruturas e consultas através dos SGBDs, o termo SQL (Structure Query Language) ou Linguagem de Consulta Estruturada é um padrão adotado pelos principais SGBDs, para os desenvolvedores e DBAs trabalharem com a estrutura dos dados e consulta de informações.

Mesmo o sql sendo uma linguagem universal para os principais SGBDs, há mudanças em alguns comandos, mas em sua grande maioria serão mínimos.

Os comandos SQL são divididos em quatro categorias, sendo elas:

- Data Manipulation Language (DML) - Consultas
- Data Define Language (DDL) - Estrutura
- Data Control Language (DCL) - Controle de acesso
- Transactional Control Language (TCL) - Gerenciar mudanças realizadas

## Exercícios

Com base nos dois primeiros capítulos estudados, responda as questões dispostas abaixo:

a) Explique o conceito do termo Banco de Dados.

b) O que é um DBA?

c) O que é um SGBD? Cite exemplos.

d) Qual a finalidade de utilizar o SQL?

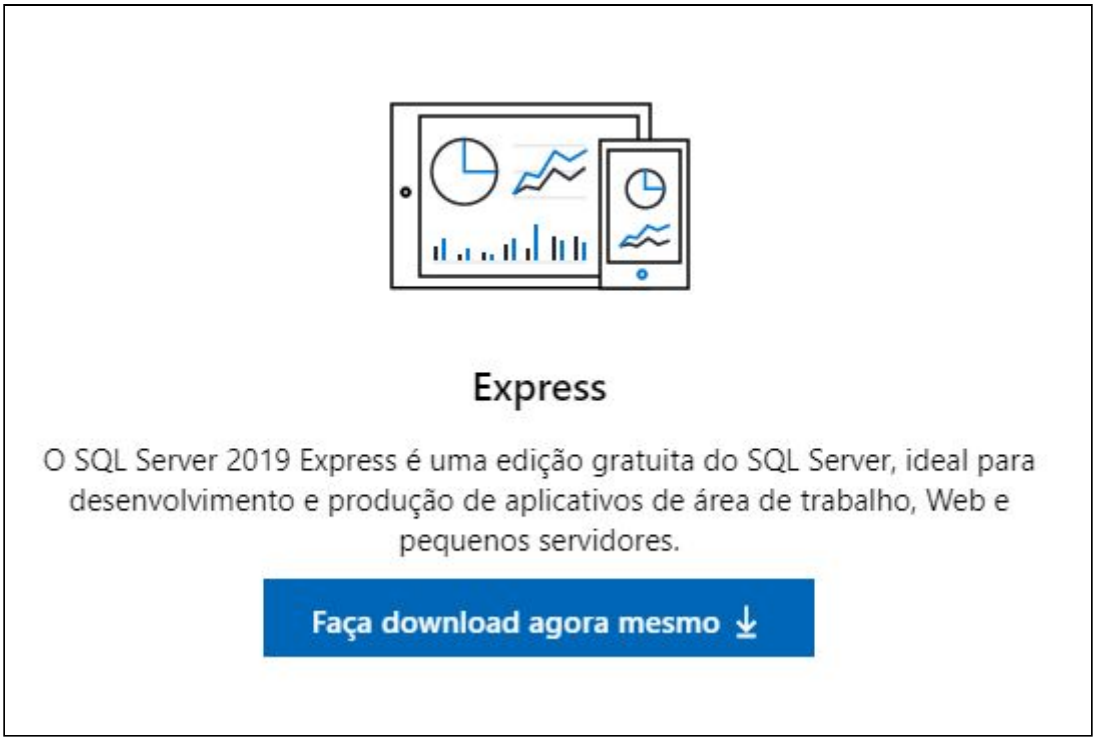
e) O que pode ser feito com os comandos SQL?

# Capítulo 03 - Configurando o ambiente

Neste capítulo iremos abordar como instalar o SQL Server da Microsoft, esse poderoso SGBD é mundialmente conhecido e utilizado em várias aplicações desktop, web e mobile.

## SQL Server

O SQL Server é o nosso SGBD (Sistema Gerenciador de Banco de Dados), ele foi desenvolvido e mantido pela Microsoft. Esse SGBD é pago, porém temos uma versão gratuita que é a Express, acesse o site e efetur o download dessa versão: https://www.microsoft.com/pt-br/sql-server/sql-server-downloads



## SQL Server Management Studio

O SQL Management Studio é um software para gerenciarmos o SQL Server, com ele teremos uma interface gráfica para podermos criar usuários, trabalhar com estruturas do banco de dados como bases de dados e tabelas, manipular informações e trabalhar com aspectos relacionados à segurança.

Você pode baixá-lo totalmente gratuito através do link oficial da Microsoft: https://docs.microsoft.com/pt-br/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

# Baixar o SQL Server Management Studio (SSMS)

07/04/2020 • 4 minutos para o fim da leitura •

**APLICA-SE A:** SQL Server Banco de Dados SQL do Azure Azure Synapse Analytics (SQL DW) Parallel Data Warehouse

O SSMS (SQL Server Management Studio) é um ambiente integrado para gerenciar qualquer infraestrutura de SQL, do SQL Server para o Banco de Dados SQL do Azure. O SSMS fornece ferramentas para configurar, monitorar e administrar instâncias do SQL Server e bancos de dados. Use o SSMS para implantar, monitorar e atualizar os componentes da camada de dados usados pelos seus aplicativos, além de criar consultas e scripts.

Use o SSMS para consultar, criar e gerenciar seus bancos de dados e data warehouses, independentemente de onde estiverem – no computador local ou na nuvem.

O SSMS é gratuito!

## Baixar o SSMS

**Baixar o SSMS (SQL Server Management Studio)**

O SSMS 18.5 é a versão mais recente de GA (disponibilidade geral) do SSMS. Se você tiver uma versão de GA anterior do SSMS 18 instalada, a instalação do SSMS 18.5 atualizará o produto para a versão 18.5.

# Capítulo 04 - Manipulando estruturas

Nessa etapa é proposto a manipulação de estruturas de bases de dados e tabelas, você irá compreender os conceitos:

- O que é uma base de dados
- Criar e excluir uma base de dados
- O que é uma tabela
- Criar, alterar e excluir uma tabela

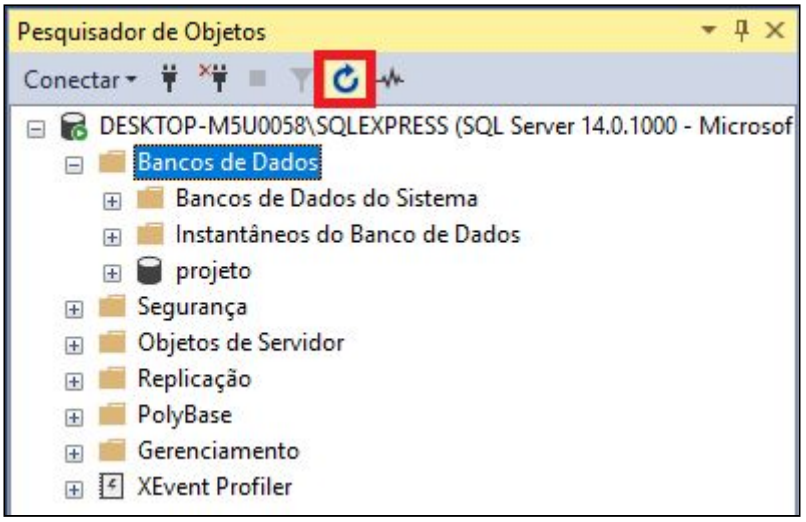
## Base de dados

Bases de dados servem para armazenarmos um conjunto de tabelas, a ideia é que cada projeto esteja em uma base de dados. Cada base de dados irá englobar estruturas que podem conter dados de clientes, produtos, serviços, fornecedores, colaboradores, entre outros tipos de dados.

```
CREATE DATABASE projeto;
```

Agora podemos executar o comando, para isso na parte superior há um botão chamado **Executar**:

Para verificar se a base de dados realmente foi criada, basta atualizarmos o Pesquisador de Objetos, que está na lateral esquerda. Há uma seta de cor azul para atualizarmos, basta clicar e visualizar a base de dados criada:




Pronto, agora temos nossa base de dados **projeto** criada, assim podemos elaborar as tabelas que irão compor a estrutura do nosso exemplo.

# Tabelas

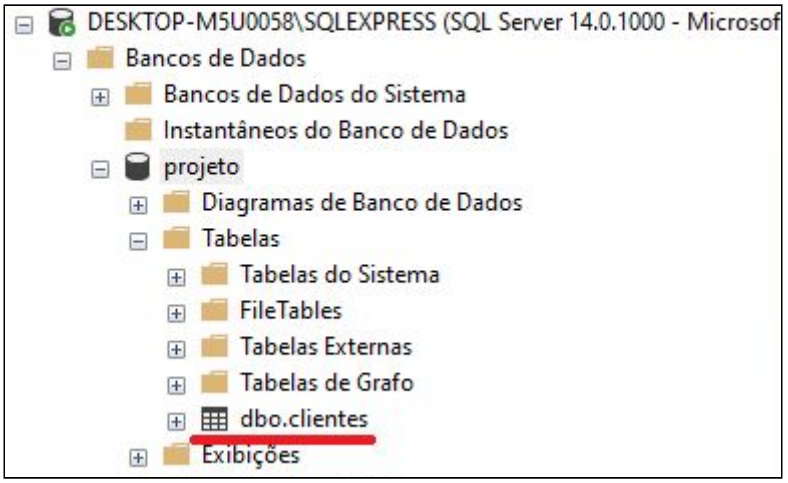
As tabelas são estruturas que ficam armazenadas nas bases de dados, cada tabela é uma pequena parte do projeto, vamos criar uma tabela para representar dados de clientes. Antes de criarmos uma tabela, precisamos selecionar a base de dados **projeto**, para isso utilize o comando **USE**, veja abaixo uma imagem ilustrando sua utilização:

```
USE projeto;
```

Agora podemos criar nossa tabela de clientes, nela teremos três informações, que são nome, cidade e idade:

```
CREATE TABLE clientes(
    nome VARCHAR(30),
    cidade VARCHAR(20),
    idade INT
);
```

Execute o código e atualize, note que agora teremos a nossa tabela dentro da base **projeto**.



## Comandos extras

Abaixo iremos exemplificar alguns comandos que podemos utilizar para trabalhar com tabelas e bases de dados, haverá um comentário para especificar cada funcionalidade.

```
--Adicionar uma nova coluna
ALTER TABLE clientes ADD email VARCHAR(30);

--Remover coluna
ALTER TABLE clientes DROP COLUMN email;

--Renomear tabela
exec sp_rename 'clientes', 'usuarios';

--Excluir base de dados
DROP DATABASE projeto;
```

# Capítulo 05 - Manipulando dados

Há diversas maneiras de manipular dados através de tabelas, para isso utilizamos os comandos DML (Data Manipulation Language) que em tradução livre significa Linguagem de manipulação de dados. Esses comandos servem para realizarmos ações como: cadastros, seleções, alterações e exclusões em tabelas, e serão muito úteis para fornecermos um conjunto de funcionalidades para o nosso usuário, juntamente com linguagens como Java, C#, PHP, Python, Node.js, entre outras.

## Cadastrar dados

Para realizar cadastros utilizamos o comando *insert*, há três maneira de utilizarmos esse comando, veja a imagem abaixo para compreender o funcionamento dessa instrução sql:

```sql
--Cadastrar dados
INSERT INTO clientes VALUES ('Ralf', 'Blumenau', 30);

--Cadastrar dados específicos
INSERT INTO clientes (nome, idade) VALUES ('Tatiana', 29);

--Cadastrar vários usuários em um único comando
INSERT INTO clientes VALUES
('Mayra', 'Joinville', 31),
('Henrique', 'Blumenau', 30),
('Paloma', 'Florianópolis', 32);
```

## Selecionar dados

Para selecionar dados, precisamos utilizar o comando *select*, há várias opções para podermos extrair dados de uma tabela. Inicialmente vamos pelo mais básico:

```
--Selecionar todos os dados
SELECT * FROM clientes;
```

O resultado do comando acima pode ser visualizado na imagem abaixo:

| | nome | cidade | idade |
|---|---|---|---|
| 1 | Ralf | Blumenau | 30 |
| 2 | Tatiana | NULL | 29 |
| 3 | Mayra | Joinville | 31 |
| 4 | Henrique | Blumenau | 30 |
| 5 | Paloma | Florianópolis | 32 |

O comando **SELECT * FROM clientes** retorna todos os dados da tabela clientes, porém podemos restringir a exibição dos dados especificando as colunas que queremos exibir, isso porque o asterisco significa que todas as colunas serão exibidas:

```
--Selecionar os nomes e idades
SELECT nome, idade FROM clientes;
```

## Alterar dados

Para alterar dados, é necessário utilizar o comando *update*, através desse comando, podemos especificar o que desejamos alterar e qual linha será executada determinada alteração. Tome cuidado ao utilizar esse comando, pois se você simplesmente executar o comando, os dados são alterados e não conseguiremos desfazer a ação.

```
--Alterar a cidade de Paloma para Joinville
UPDATE clientes SET cidade = 'Joinville' WHERE nome = 'Paloma';
```

Na imagem acima, você pode notar que utilizamos o comando **SET** para especificar o que iremos alterar, em seguida utilizamos o comando **WHERE** para informar a referência das alterações.

## Excluir dados

A remoção de informações de uma tabela é algo bem simples, utilizamos o comando *delete* para realizar determinada ação. Veja a imagem abaixo para ilustrar o funcionamento dessa função:

```
--Excluir o cliente de nome Paloma
DELETE FROM clientes WHERE nome = 'Paloma';
```

A imagem acima é um exemplo do comando de exclusão *delete*, note que foi utilizado o comando **WHERE** para especificar o que será excluído, vale ressaltar que como no comando *update*, o comando *delete* não há como desfazer a ação.

# Capítulo 06 - Filtrando dados

Agora que você já tem uma base dos comandos para manipular dados, podemos implementar ainda mais nossas consultas. Esse capítulo dará ênfase na seleção de informações.

## Where

O comando **Where** é um dos principais para realizar filtragens simples, o significado dessa função é onde. Sendo assim podemos selecionar dados como por exemplo: selecione os clientes com idade superior a 30 anos, para isso observe o código abaixo:

```
--Exibir os clientes com idade superior a 30
SELECT * FROM clientes WHERE idade > 30;
```

No curso de lógica e algoritmos, o uso de operadores relacionais é algo normal, e aqui no banco de dados também são muito úteis, vamos relembrar quais operadores relacionais temos a disposição, e como são suas estruturas:

- > Maior
- < Menor
- >= Maior ou igual
- <= Menor ou igual
- = Igual
- <> Diferente

## Operadores lógicos

Os operadores lógicos têm a finalidade de verificarmos duas ou mais situações, na programação há dois muito conhecidos que são o operador **E** e o operador **OU**. Para compreender o seu uso, vamos supor que queremos filtrar duas informações, por exemplo a idade precisa ser maior que 30 e a cidade precisa ser diferente de Blumenau:

```
--Exibir os clientes com idade
--superior a 30 e que não sejam de Blumenau
SELECT * FROM clientes WHERE idade > 30 AND cidade <> 'Blumenau';
```

## Min e Max

Os comandos **Min** e **Max** são úteis para podermos retornar o maior e o menor valor de uma tabela, vamos exemplificar utilizando a característica idade da nossa tabela de testes:

```
--Maior idade
SELECT MAX(idade) FROM clientes;

--Menor idade
SELECT MIN(idade) FROM clientes;
```

## Count, Avg e Sum

Os comandos **Count**, **Avg** e **Sum**, agrupam informações, no caso o **Count** retorna a quantidade de registros, o **Avg** a média de um determinado tipo de informação, já o **Sum** retorna a soma dos valores de uma coluna.

```
--Contar os clientes cadastrados
SELECT COUNT(*) FROM clientes;

--Média das idades
SELECT AVG(idade) FROM clientes;

--Soma das idades
SELECT SUM(idade) FROM clientes;
```

## Like

O comando Like possui três maneiras de filtrar dados, podendo ser: termos que iniciem, que finalizam ou que contenham. Veja a imagem abaixo para compreender melhor o seu funcionamento.

```
--Exibir os clientes que iniciam com a letra H
SELECT * FROM clientes WHERE nome LIKE 'H%';

--Exibir os clientes que finalizam com a letra A
SELECT * FROM clientes WHERE nome LIKE '%A';

--Exibir os clientes que contenham a letra R
SELECT * FROM clientes WHERE nome LIKE '%R%';
```

## In

O comando **In** tem a finalidade de exibir a linha da tabela que contenha um dos possíveis termos informados pelo desenvolvedor. Para compreender melhor seu uso, imagine que queremos exibir todos os clientes nas cidades de Blumenau ou Joinville, veja como ficará a estrutura:

```
--Exibir os clientes nas cidades de Blumenau e Joinville
SELECT * FROM clientes WHERE cidade IN ('Blumenau', 'Joinville');
```

## Between

O comando **Between** tem a tradução de entre, onde podemos verificar se uma determinada informação está entre dois valores. Vamos supor que queres exibir os clientes com idade entre 20 e 30 anos, para isso a seguinte estrutura será utilizada:

```
--Exibir os clientes com idade entre 20 e 30
SELECT * FROM clientes WHERE idade BETWEEN 20 AND 30;
```

## Order By

O comando **Order By** tem a função de ordenar informações numéricas ou textuais de maneira crescente ou decrescente. Utilizando o comando **ASC**, os dados serão exibidos de maneira crescente, sendo assim para letras serão exibidos em ordem alfabética de A a Z e numérica, no menor para o maior, já o **DESC** fará o contrário, em ordem alfabética de Z a A e numérica do maior para o menor, veja abaixo alguns exemplos:

```
--Ordenar por nome (A - Z)
SELECT * FROM clientes ORDER BY nome ASC;

--Ordenar por idade (Maior para menor)
SELECT * FROM clientes ORDER BY idade DESC;
```

## Group By

O comando Group By tem a finalidade de agrupar os dados. Essa funcionalidade é muito boa para podermos dispor no sistema informações referentes a uma determinada tabela, veja abaixo um exemplo:

```
--Exibir o nome da cidade e a quantidade
SELECT cidade, count(*) FROM clientes GROUP BY cidade;
```

| | cidade | (Nenhum nome de coluna) |
|---|---|---|
| 1 | NULL | 1 |
| 2 | Blumenau | 2 |
| 3 | Joinville | 1 |

## Exercícios

1. Criar uma base de dados chamada: **exercicio**

2. Selecionar a base criada anteriormente

3. Criar uma tabela chamada **uzuarios**, com a seguinte estrutura:
    a. nome varchar de 20 caracteres
    b. sobrenome de 40 caracteres
    c. idade do tipo inteiro

4. Renomeie a tabela **uzuarios** para **usuarios**.

5. Adicione uma coluna e-mail com suporte até 40 caracteres

6. Exclua a coluna sobrenome

7. Cadastre os seguintes dados:

| | | |
|---|---|---|
| Vanessa | 16 | vanessa.rosa@gmail.com |
| Adailton | 22 | adailton.mas@yahoo.com |
| Andressa | 36 | andressa.simas@uol.com.br |
| Mayra | 24 | mayra_antunes@gmail.com |
| Cristiane | 14 | cris.maya@gmail.com |
| Carina | 27 | carina.almeida@gmail.com |
| Clóvis | 29 | clovis.simao@hotmail.com |
| Gabriela | 23 | gabriela.bragantino@live.com |
| Cibele | null | cibele_lins@uol.com.br |

8. Exiba quantos registros existem na tabela

9. Exibir a quantidade de usuários com idade até 17 anos

10. Retornar a quantidade de usuários com o e-mail **gmail**

11. Retornar o nome e a idade da pessoa mais velha

12. Retornar os dados do usuário com idade igual a nulo

13. Alterar para 27 todas as idades nulas

14. Deletar todos os usuários com idade inferior a 18 anos

15. Excluir todos os dados da tabela e reiniciar a contabilização

16. Excluir a base de dados

# Capítulo 07 - Relacionamento entre tabelas

Os relacionamentos entre tabelas são muito utilizados, pois nem todos os dados podem ficar numa só tabela, pensando na reutilização e também na sobrecarga de informações. Uma alternativa muito interessante é dividirmos nossos projetos em pequenas tabelas, assim conseguimos organizar melhor os dados e também não ter estruturas muito complexas, dificultando a manutenção e compreensão do projeto.

## Chave primária

A utilização de uma chave primária é muito importante, pois nos garante que a informação não poderá ser repetida, assim teremos a integridade de um dado, além disso o uso de uma chave primária nos possibilita trabalharmos com chaves estrangeiras, que será visto no próximo tópico.

Para trabalhar com uma chave primária você precisa ter a absoluta certeza de que o dado não poderá ser repetido, então é interessante aplicar em códigos incrementadores de maneira crescente ou através de um CPF por exemplo.

## Auto incremento

Juntamente com a chave primária, podemos utilizar o comando **identity()**, que significa que haverá um auto incremento deste código. Essa função trabalha com dois parâmetros, sendo o primeiro o ponto inicial do incremento, em seguida o tipo de incremento, neste caso será de um em um, porém se fosse utilizado o comando **identity(1, 2)**, seria incrementado de dois em dois, exemplo: 1, 3, 5, 7, 9…

Para compreender o uso de uma chave primária e do auto incremento, veja abaixo um exemplo utilizando o banco SQL Server:

```
CREATE TABLE usuarios (
    codigo INT IDENTITY(1,1),
    nome VARCHAR(30),
    idade INT,
    email VARCHAR(40),
    PRIMARY KEY(codigo)
);
```

Importante, o **identity()** é um comando específico para o SQL Server, porém bancos como o MySQL e o PostgreSQL também possuem esse tipo de função, basta utilizar o comando **auto_increment** ao invés de **identity**, que o resultado será o mesmo.

## Chave estrangeira

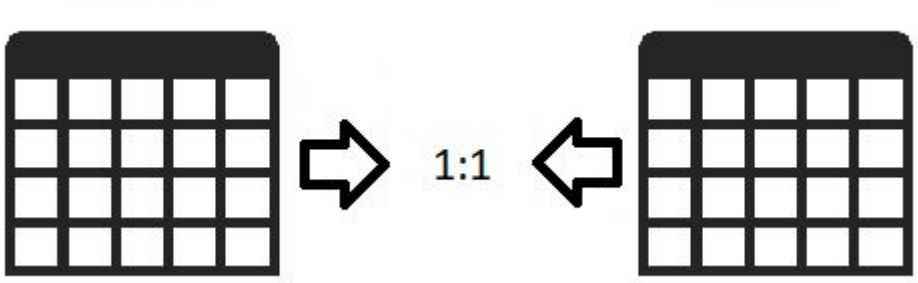
A chave estrangeira tem como finalidade utilizar chaves primárias de outras tabelas, isso garante que a informação irá existir. Apenas será possível cadastrar uma informação em determinada tabela, se houver uma chave primária com um determinado código na tabela de referência.

Vamos supor que você tem uma tabela com três tipos de cursos com as seguintes chaves primárias: 1 - Java, 2 - C# e 3 - Python. Podemos ter uma tabela chamada turmas, onde essa tabela pode ter uma coluna chamada curso, onde poderá armazenar qualquer um dos três códigos. Isso significa que se porventura alguém tentar adicionar um número diferente desses três, o banco de dados irá barrar o cadastro, pois obrigatoriamente precisa ser utilizado um desses códigos.

Outra característica importante do uso da chave estrangeira é que não podemos excluir ou alterar a informação, pois se pudermos alterar ou excluir, a tabela que possui o vínculo com a outra tabela não terá a garantia que exista determinado código. Em outras palavras, se na tabela de turmas houver uma turma com o código do curso 1, não podemos alterar ou excluir o curso Java, pois a tabela de turmas está utilizando essa informação, e o banco garante que enquanto houver a chave estrangeira, a informação não sofrerá mudanças.

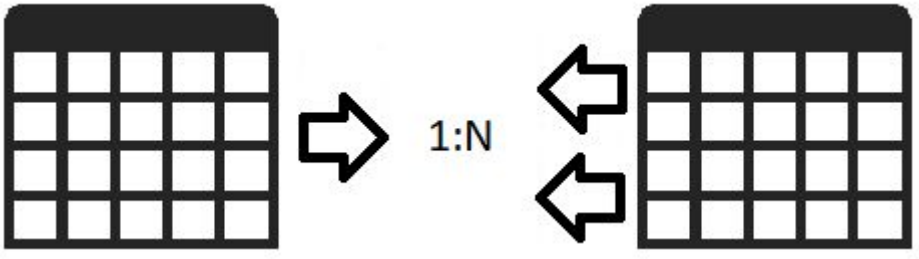
## Relacionamento 1:1

O entre tabelas relacionamento 1:1 (um para um), é quando uma tabela possui apenas uma referência com outra tabela, podemos exemplificar isso com uma tabela de CPFs, onde há apenas um CPF por habitante.



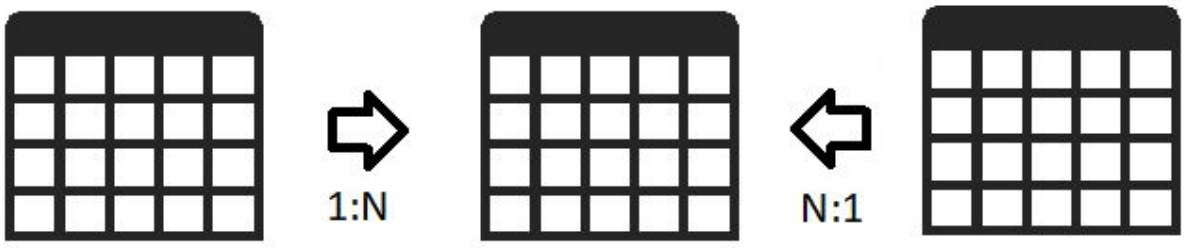

## Relacionamento 1:N

O uso do relacionamento 1:N (um para muitos), é quando uma tabela possui muitos vínculos com outra tabela. Suponha que tenhamos uma tabela com todos os estados e outra com todas as cidades de cada estado, neste caso um estado pode ter muitas cidades.



## Relacionamento N:N

O uso do relacionamento N:N (muitos para muitos), é quando uma tabela se relaciona com vários registros de outras tabelas. Vamos supor que tenhamos uma tabela de filmes e outra de atores, precisamos relacionar um filme com um ator, lembrando que um filme poderá ter vários atores e um ator pode ter participado de vários filmes. Essa tabela por padrão possui duas chaves estrangeiras, veja abaixo um exemplo dessa estrutura na imagem abaixo:




## Exercícios

Para compreender melhor os conceitos de chave primária, chave estrangeira e relacionamentos, desenvolva as seguintes questões propostas:

1. O que é uma chave primária? Quais são as suas funcionalidades?

2. O que é uma chave estrangeira? Qual a importância em utilizar em algum projeto?

3. Explique e exemplifique o uso de um relacionamento 1:1.

4. Explique e exemplifique o uso de um relacionamento 1:N.

5. Explique e exemplifique o uso de um relacionamento N:N.

# Capítulo 08 - Joins

O uso de Joins é muito útil para podermos trabalhar com dados de múltiplas tabelas, assim não fica sob responsabilidade de uma única tabela armazenar os dados, além disso podemos deixar as tabelas mais compactas e reutilizar seus dados em diversas ocasiões em nossos projetos. Neste capítulo iremos abordar sobre o uso dos Joins, que são separados em três: Inner Join, Left Join e Right Join.

Antes de trabalhar com essas três funcionalidades, iremos criar duas tabelas para podermos compreender melhor seu uso, veja a estrutura das tabelas na imagem abaixo:

```
CREATE TABLE cidades (
    codigo INT IDENTITY(1,1),
    nome VARCHAR(30),
    PRIMARY KEY(codigo)
);

CREATE TABLE clientes(
    codigo INT IDENTITY(1,1),
    nome VARCHAR(15),
    cidade INT,
    PRIMARY KEY(codigo),
    FOREIGN KEY(cidade) REFERENCES cidades(codigo)
);
```

Aproveitando a estrutura de tabelas criada, podemos cadastrar alguns dados, para que posteriormente testes sejam realizadas utilizando as tabelas cidades e clientes:

```
INSERT INTO cidades (nome) VALUES
('Blumenau'),
('Camboriú'),
('Joinville'),
('Indaial');

INSERT INTO clientes (nome, cidade) VALUES
('Ana', 1),
('Júlio', 3),
('Larissa', 1),
('Christian', 2);
```

Com esses dados devidamente cadastrados, poderemos efetuar pesquisas unindo duas tabelas ou mais através de Joins. Veremos no próximo capítulo essas funcionalidades extremamente utilizadas pelos desenvolvedores.

## Inner Join

O comando Inner Join terá como finalidade exibir dados de duas tabelas ou mais, desde que uma tabela tenha uma chave primária e outra uma chave estrangeira. Vamos supor que queremos exibir o nome e a cidade de nossos clientes, para isso veja o exemplo abaixo:

```
SELECT
    clientes.nome,
    cidades.nome
FROM clientes
INNER JOIN cidades
ON clientes.cidade = cidades.codigo;
```

O resultado do uso da utilização do comando Inner Join será esse:

| | nome | nome |
|---|---|---|
| 1 | Ana | Blumenau |
| 2 | Júlio | Joinville |
| 3 | Larissa | Blumenau |
| 4 | Christian | Camboriú |

Note que a estrutura do comando Inner Join é necessário especificar as tabelas, em seguida as colunas, além disso após o comando **ON** precisamos informar o que as duas tabelas têm em comum, neste caso ambas possuem um código de referência a cidade.

## Left Join

Agora imagine que você queira contar quantas pessoas estão cadastradas em determinadas cidades, porém você obrigatoriamente quer exibir todas as cidades dispostas na tabela de cidades.

Há cidades cadastradas que não estão sendo utilizadas, então como obrigar a exibir essas tabelas, se não há clientes cadastrados? Simples, o Inner Join irá forçar a exibir o nome de todas as cidades, e caso não existam clientes cadastrados em determinadas cidades, o valor exibido será zero, veja a imagem abaixo para compreender a implementação dessa funcionalidade:

```
SELECT
    cidades.nome,
    COUNT(clientes.nome)
FROM clientes
LEFT JOIN cidades
ON clientes.cidade = cidades.codigo
GROUP BY cidades.nome;
```

Esse comando irá retornar a quantidade de clientes em cada cidade, note que utilizamos o comando Left Join e um Order By com ênfase em exibir os nomes das cidades, independente se a cidade possui ou não algum cliente cadastrado, o resultado desse comando será esse:

| | nome | (Nenhum nome de coluna) |
|---|---|---|
| 1 | Blumenau | 2 |
| 2 | Camboriú | 1 |
| 3 | Joinville | 1 |

## Right Join

Vamos para o nosso último exemplo de Joins, o comando Right Join é muito parecido com o Left Join, porém vamos obrigar a aparecer uma lista a direita, já o Left Join obriga a listar os dados a esquerda. Vamos listar todos os nomes dos clientes e suas respectivas cidades, independente se o cliente possua ou não uma cidade cadastrada:

```
SELECT
    cidades.nome,
    clientes.nome
FROM clientes
RIGHT JOIN cidades
ON clientes.cidade = cidades.codigo;
```

A ideia é exibir todas as cidades, mesmo que não haja clientes cadastrados em determinadas cidades. Em nosso exemplo a cidade de Indaial não possui nenhum cliente, sendo assim irá retornar o termo null, veja abaixo a listagem dos dados:

| | nome | nome |
|---|---|---|
| 1 | Blumenau | Ana |
| 2 | Blumenau | Larissa |
| 3 | Camboriú | Christian |
| 4 | Joinville | Júlio |
| 5 | Indaial | NULL |

# Exercícios

Com base neste capítulo, desenvolva as questões abaixo utilizando as funcionalidades dos Joins:

1. Criar uma base de dados chamada **exercicio_joins**

2. Implemente a tabela **cursos**, com os seguintes campos:

| | |
|---|---|
| Código | INT AI e PK |
| Curso | VARCHAR(20) |

3. Desenvolva a tabela clientes, com os seguintes campos:

| | |
|---|---|
| Código | INT AI e PK |
| Cliente | VARCHAR(30) |
| Codigo_Curso | INT FK |

4. Cadastre os cursos:

| | |
|---|---|
| 1 | Java |
| 2 | C# |
| 3 | Python |
| 4 | PHP |
| 5 | Node.js |

5. Cadastre os seguintes clientes:

| | | |
|---|---|---|
| 1 | Larissa | 3 |
| 2 | Gabriel | 1 |
| 3 | Jean | 1 |
| 4 | Gabriella | 2 |
| 5 | Robson | 3 |
| 6 | Isabella | 3 |
| 7 | Eduardo | 2 |
| 8 | Juliana | 3 |
| 9 | Carlos | 2 |
| 10 | Lorena | 1 |

6. Liste o nome dos clientes e o nome dos cursos que cada um está participando.

7. Contar a quantidade de cursos adquiridos pelos clientes. Exiba o nome e a quantidade desses cursos. Será obrigatório a exibição de todos os cursos, independente se há clientes realizando esse curso.

# Capítulo 09 - Views

A utilização de Views é algo muito interessante e prático para os desenvolvedores. Imagine que exista um comando que esteja vinculado cinco tabelas, imagine a complexidade do Inner Join criado, provavelmente será um comando extenso e não queremos utilizar determinado comando em uma linguagem de programação, pois ficaria confuso para o desenvolvedor compreender a funcionalidade, além de não ter motivos para o desenvolver saber quais tabelas estão sendo manipuladas, para isso existem as Views, que são atalhos de visualização de dados.

## Exemplo prático

Uma View apenas pode ser utilizada para selecionar dados, sendo assim o comando Select será implementado em todas as Views. Vamos supor que você queira criar uma View para listar o nome do cliente e o código da cidade, para isso veja o exemplo abaixo de como podemos criar, executar e remover uma View:

```
--Criar view
CREATE VIEW visao AS
SELECT nome, cidade FROM clientes;

--Executar view
SELECT * FROM visao;

--Excluir view
DROP VIEW visao;
```

## Exercícios

Implementando o exercício anterior de Joins, crie uma View contendo as seguintes funcionalidades:

1. Listar o nome do cliente e o nome do curso.

2. Listar todos os cursos e o nome dos clientes que fazem determinado curso, caso não exista cliente para determinado curso, deverá aparecer **null** no nome do cliente.

3. Exibir o nome dos cursos e a quantidade de clientes cadastrados em cada curso.

4. Exibir em ordem alfabética o nome dos clientes.

5. Excluir todas as Views criadas.

# Capítulo 10 - Stored Procedure

Stored Procedure ou simplesmente Procedure, é uma técnica utilizada para criar funções. Nessas funções podemos trabalhar com todos os comandos SQL normalmente, além de implementar alguns aspectos já vistos na programação como por exemplo: variáveis, condicionais, laços, vetores, entre outras funcionalidades.

## Exemplo prático

A utilização de procedures é algo muito interessante, pois muitas vezes o DBA pode criar as procedures e passar para o desenvolvedor apenas o nome dessas procedure e ela realizar alguma ação. As procedures também podem possuir parâmetros, abaixo veja como criar, executar e excluir uma procedure:

```
--Criar procedure
CREATE PROCEDURE inserirCidade @nome nvarchar(15)
AS INSERT INTO cidades (nome) VALUES (@nome);

--Executar procedure
EXEC inserirCidade @nome = 'Florianópolis';

--Excluir procedure
DROP PROCEDURE inserirCidade;
```

# Exercícios

Com base no capítulo 10 sobre Stored Procedure, elabore as questões propostas abaixo:

1. Criar uma base de dados chamada exercicios_procedure.

2. Desenvolva uma tabela com os campos código (int, ai e pk), nome (varchar 30) e idade (int).

3. Crie uma procedure para cadastrar novos usuários, essa procedure deverá ter dois parâmetros, sendo eles o nome e a idade.

4. Implementar uma procedure para alterar dados, para isso peça o código, nome e idade.

5. Desenvolver uma procedure para remover um usuário através do nome.

6. Excluir todas as procedures.

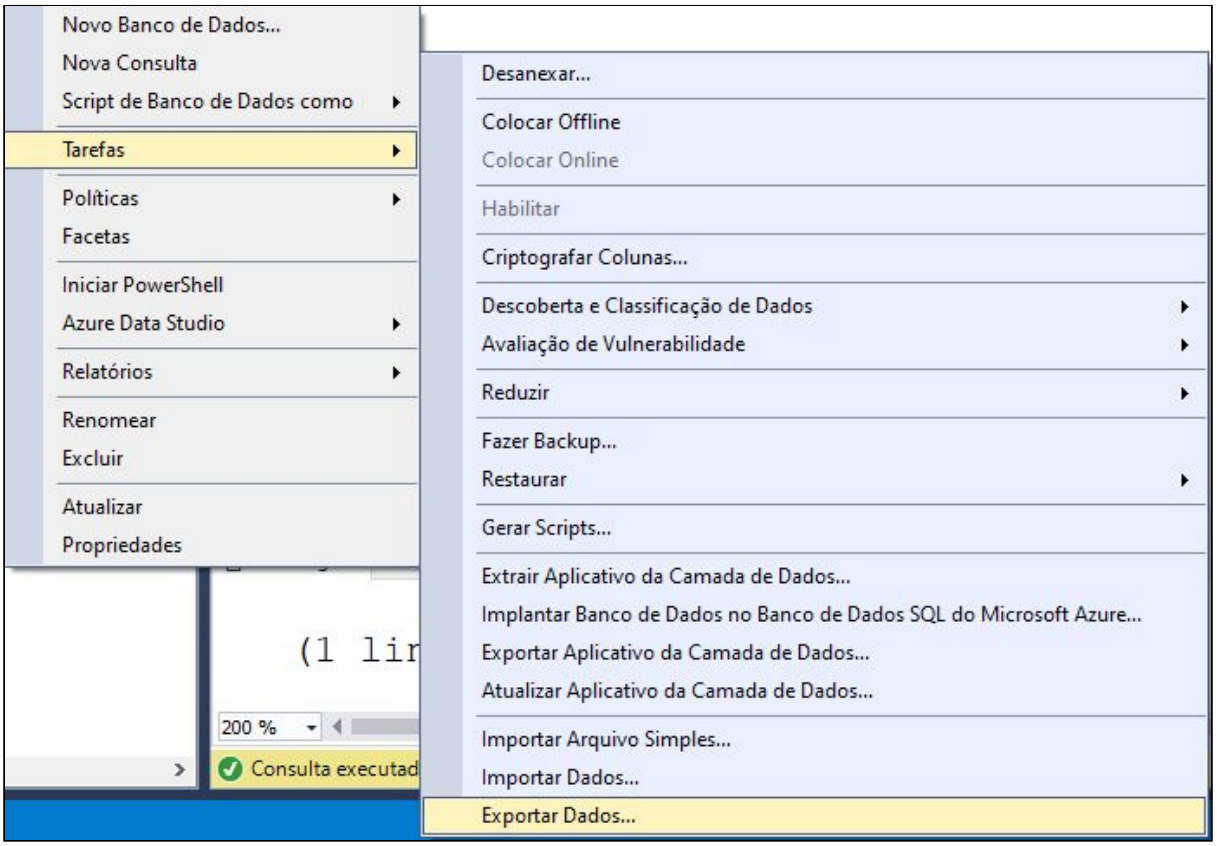
7. Excluir a base de dados.

# Capítulo 11 - Importando e exportando projetos

Um ponto muito importante quando se trabalha com banco de dados, é a segurança da informação. Quem garante que o computador não irá dar algum problema, ou algum desenvolvedor não irá danificar alguma tabela ou base de dados? Pensando nisso podemos exportar e importar estruturas completas de maneira muito simples.

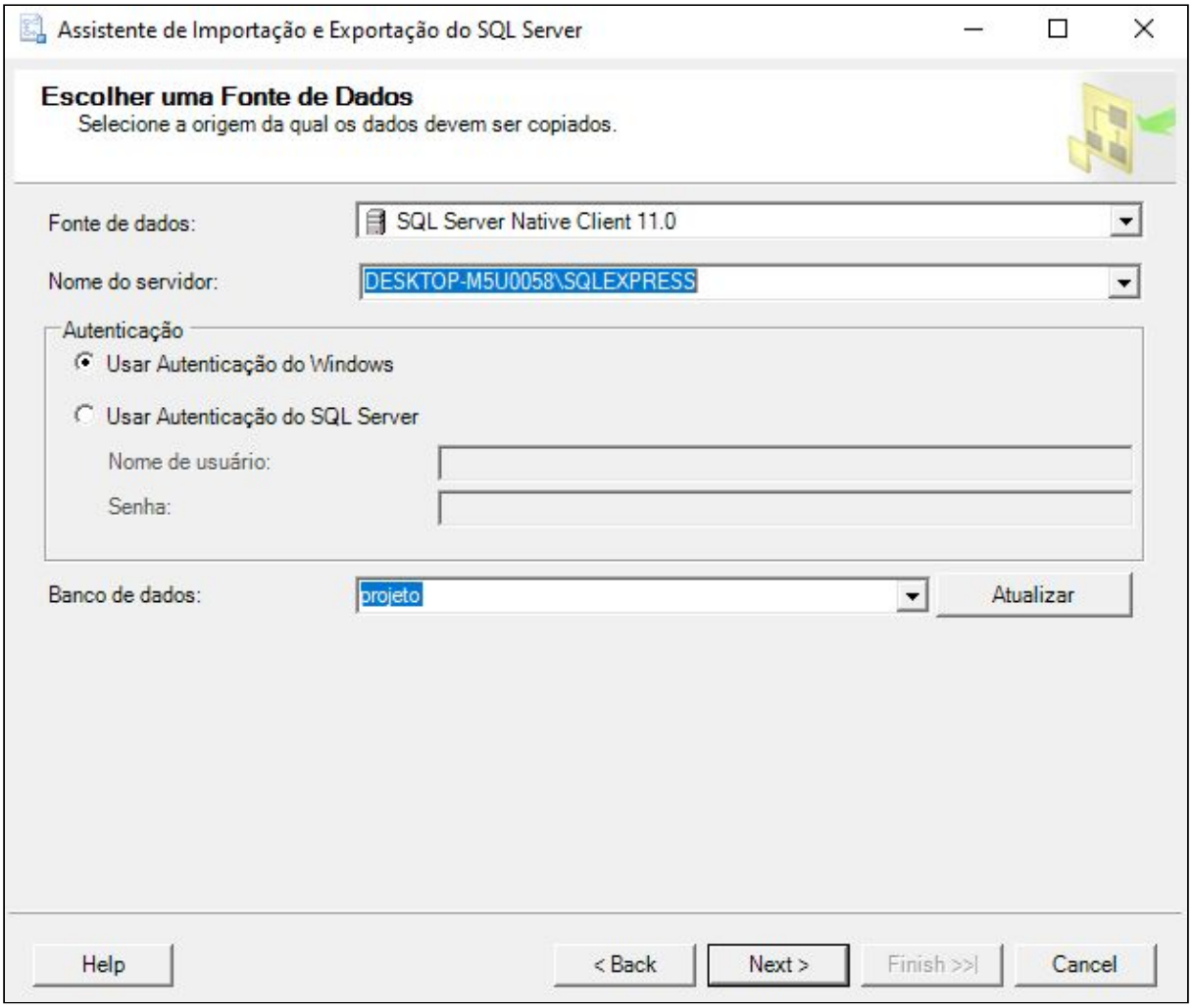
## Exportando base de dados

Clique com o botão direito na base de dados, procure pela opção **Tarefas**, em seguida **Exportar Dados...**

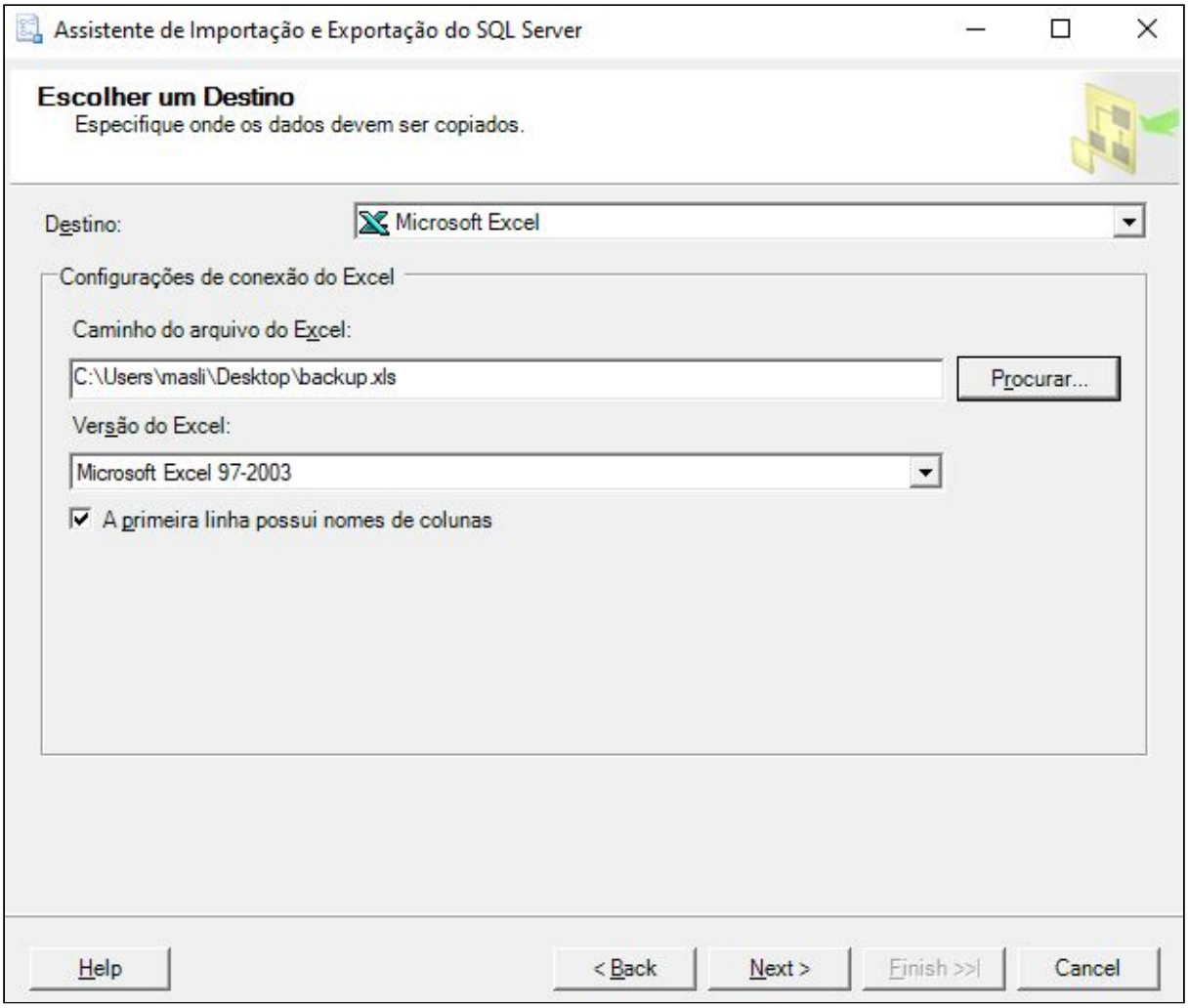


Será aberta uma janela de boas vindas, apenas dê um **Next >**

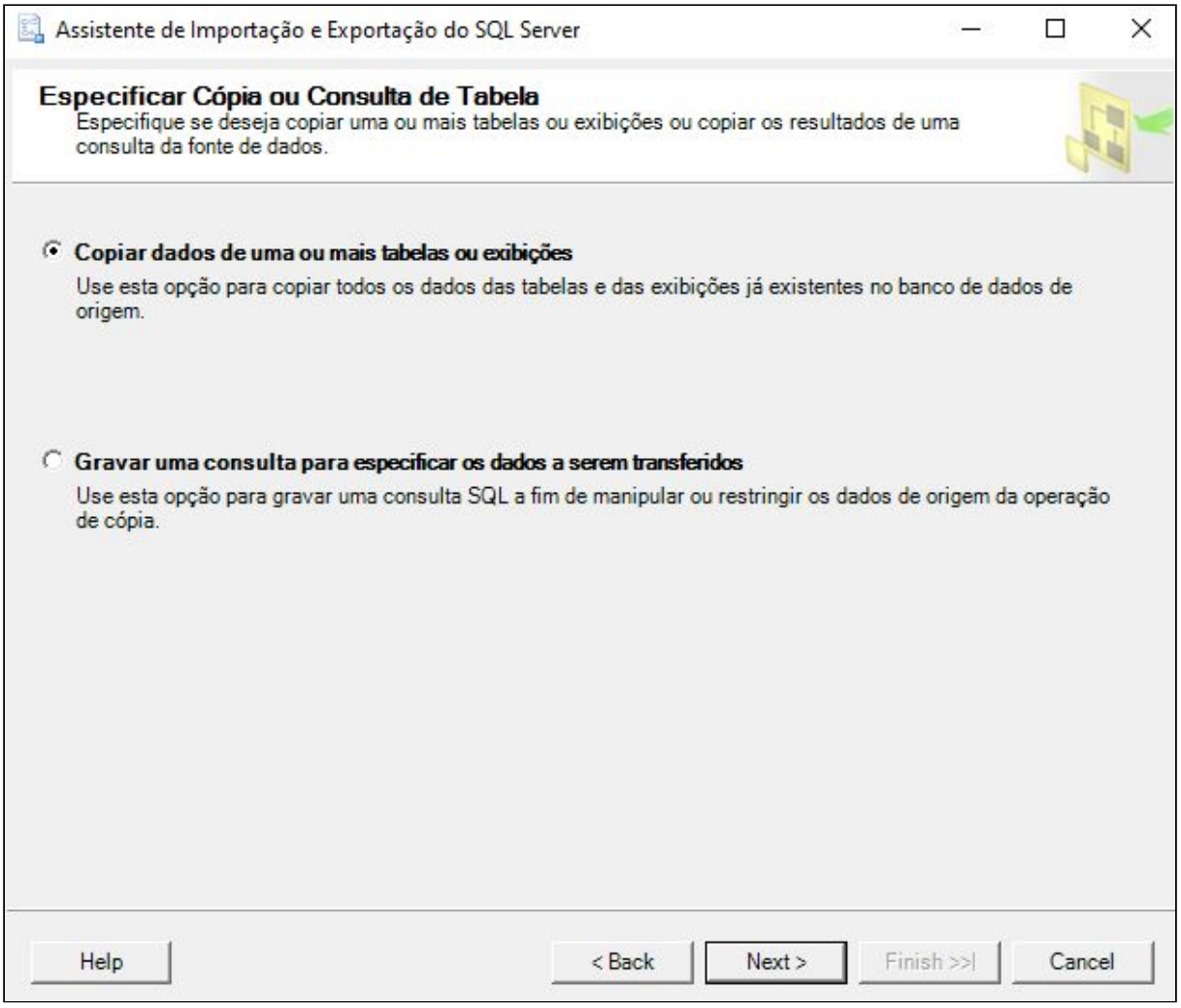
Selecione o banco de dados SQL Server Native e mais abaixo a base de dados que você deseja exportar (geralmente essa opção já vem selecionada, pois você havia clicado com o botão direito sobre a base).



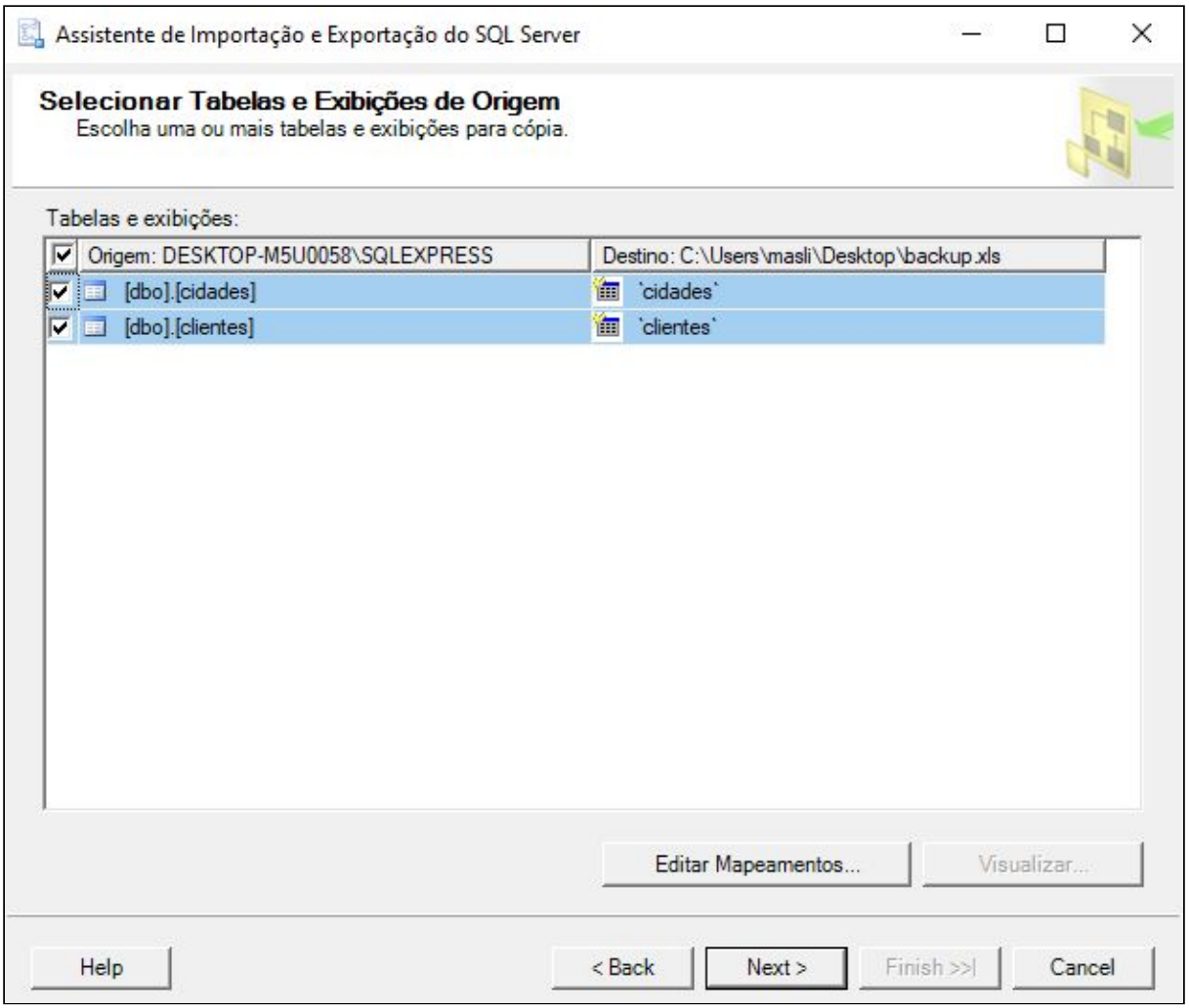
Selecione a maneira que deseja exportar a base de dados, neste exemplo será exportado para Excel, porém há diversas opções.



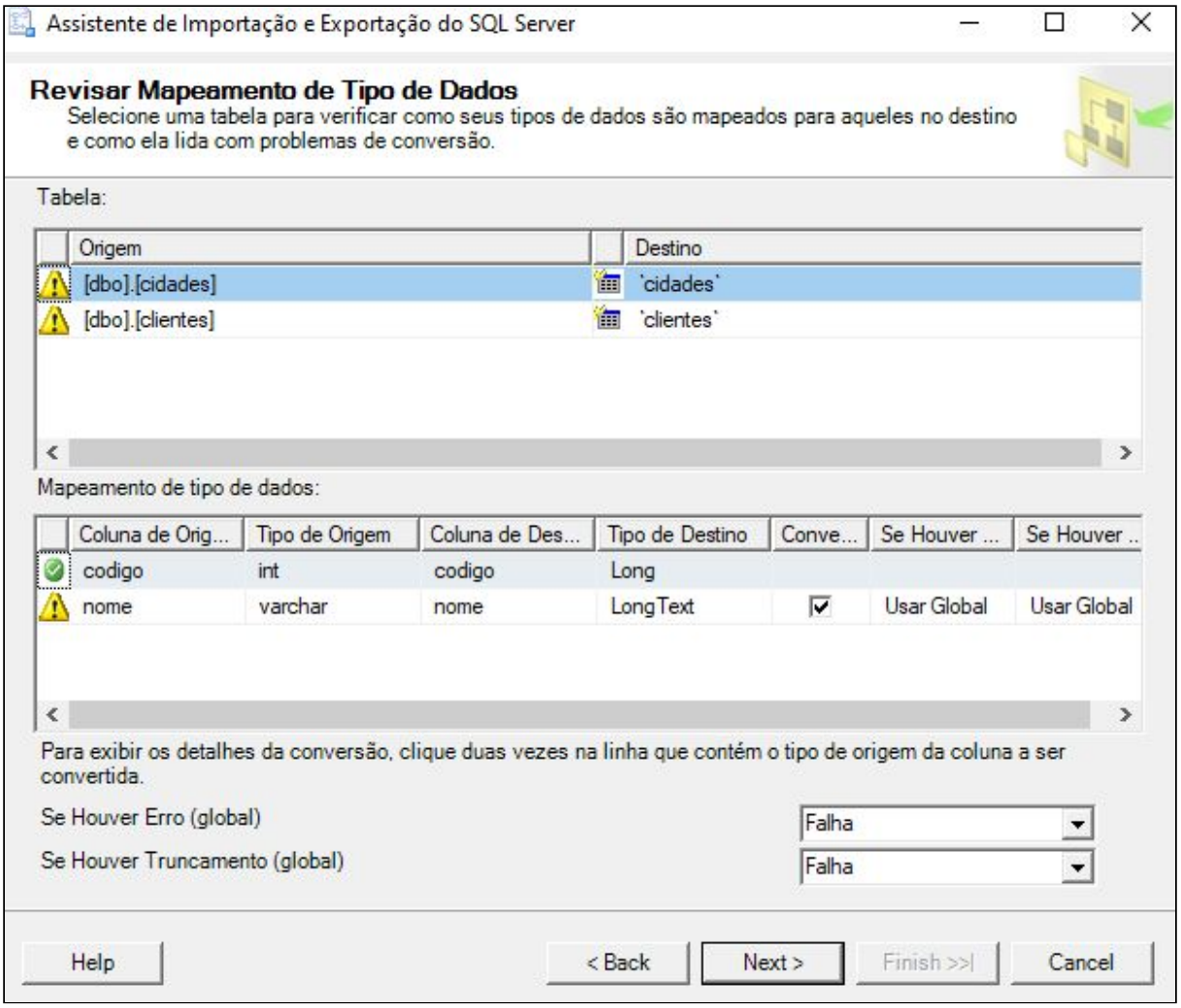
Selecione a opção **Copiar dados de uma ou mais tabelas ou exibições**, assim podemos exportar todas as tabelas de nossa base de dados.



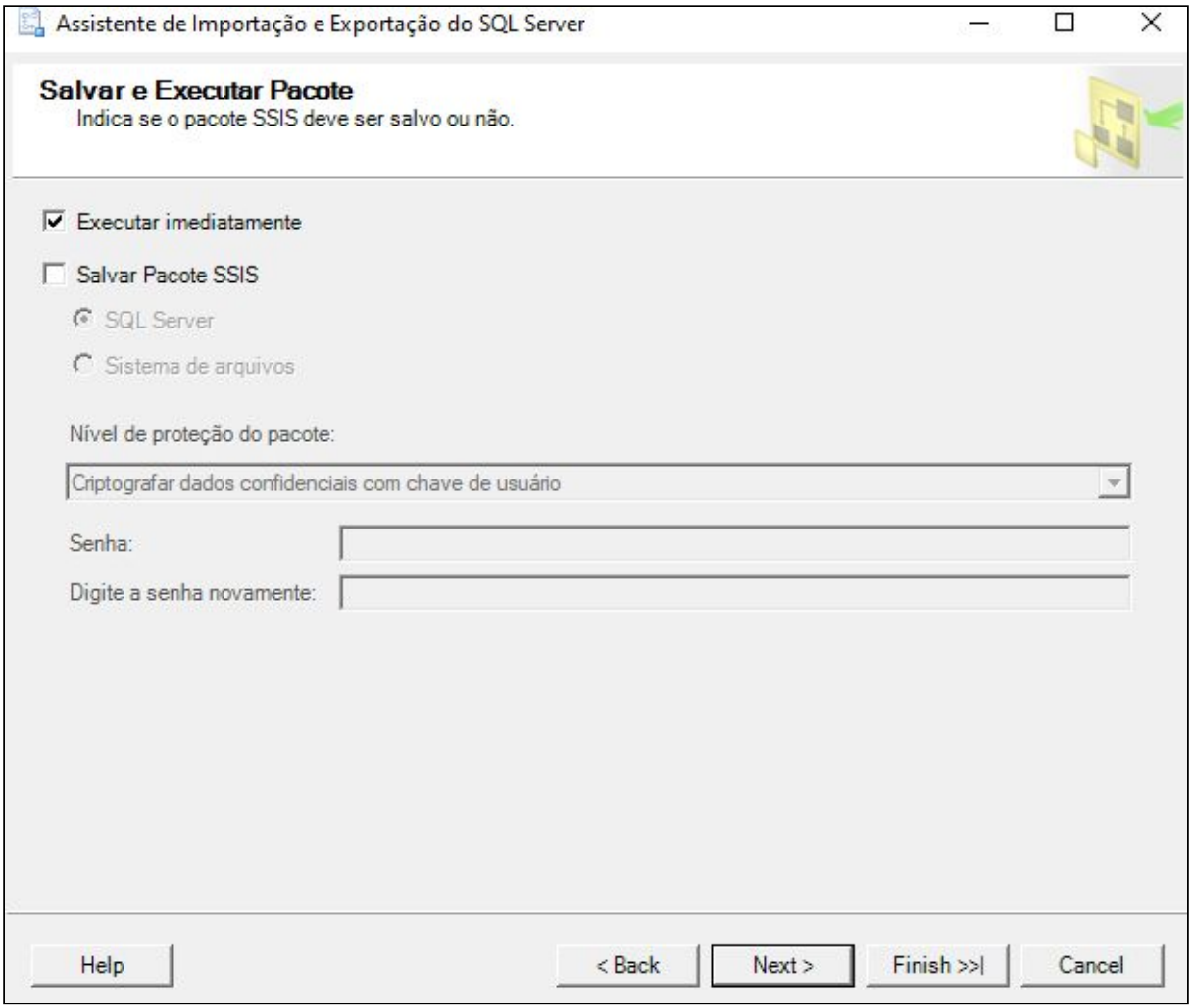
Selecione as tabelas que deseja exportar:



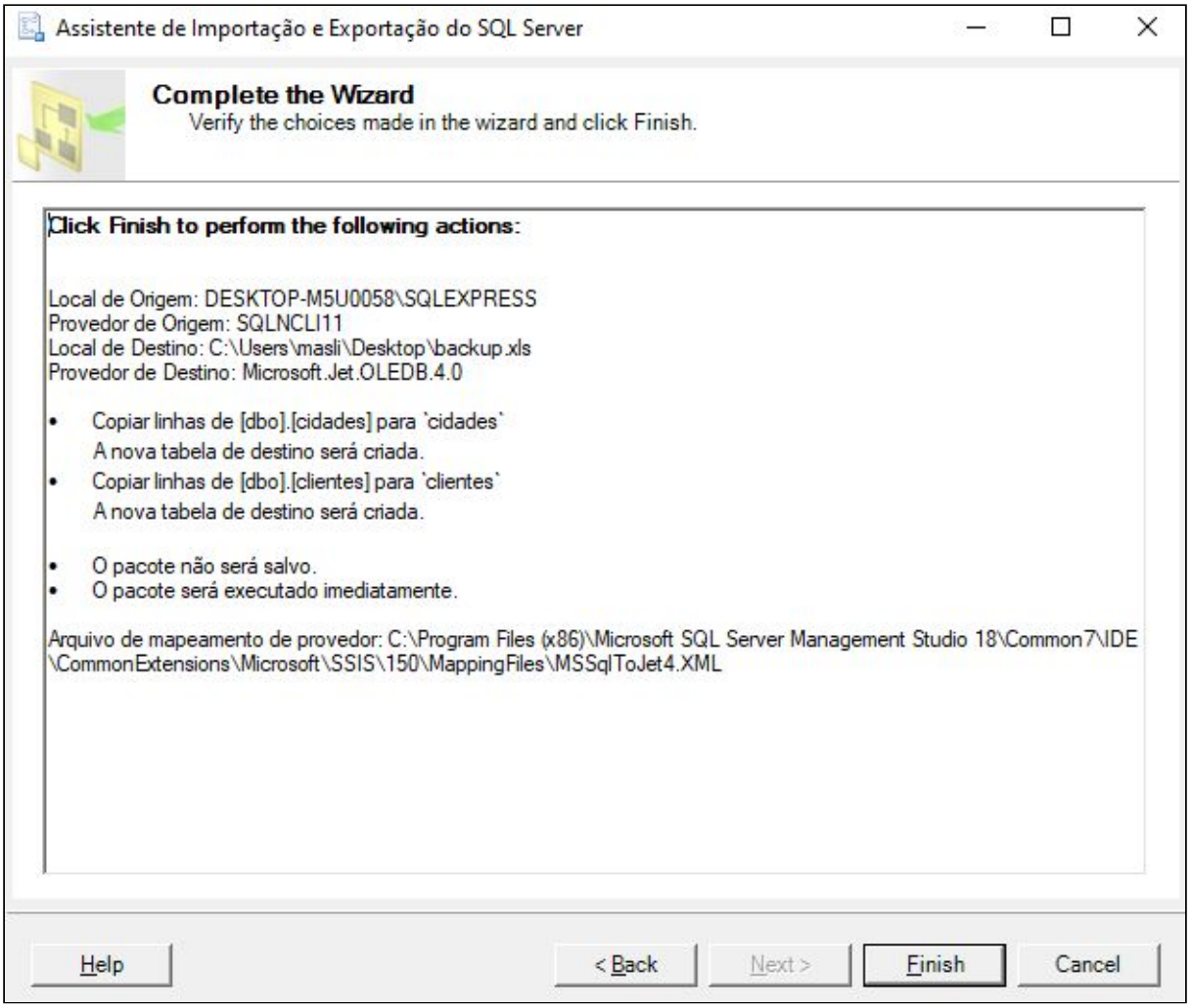
Revise o mapeamento, para isso basta apenas clicar no botão **Next >**



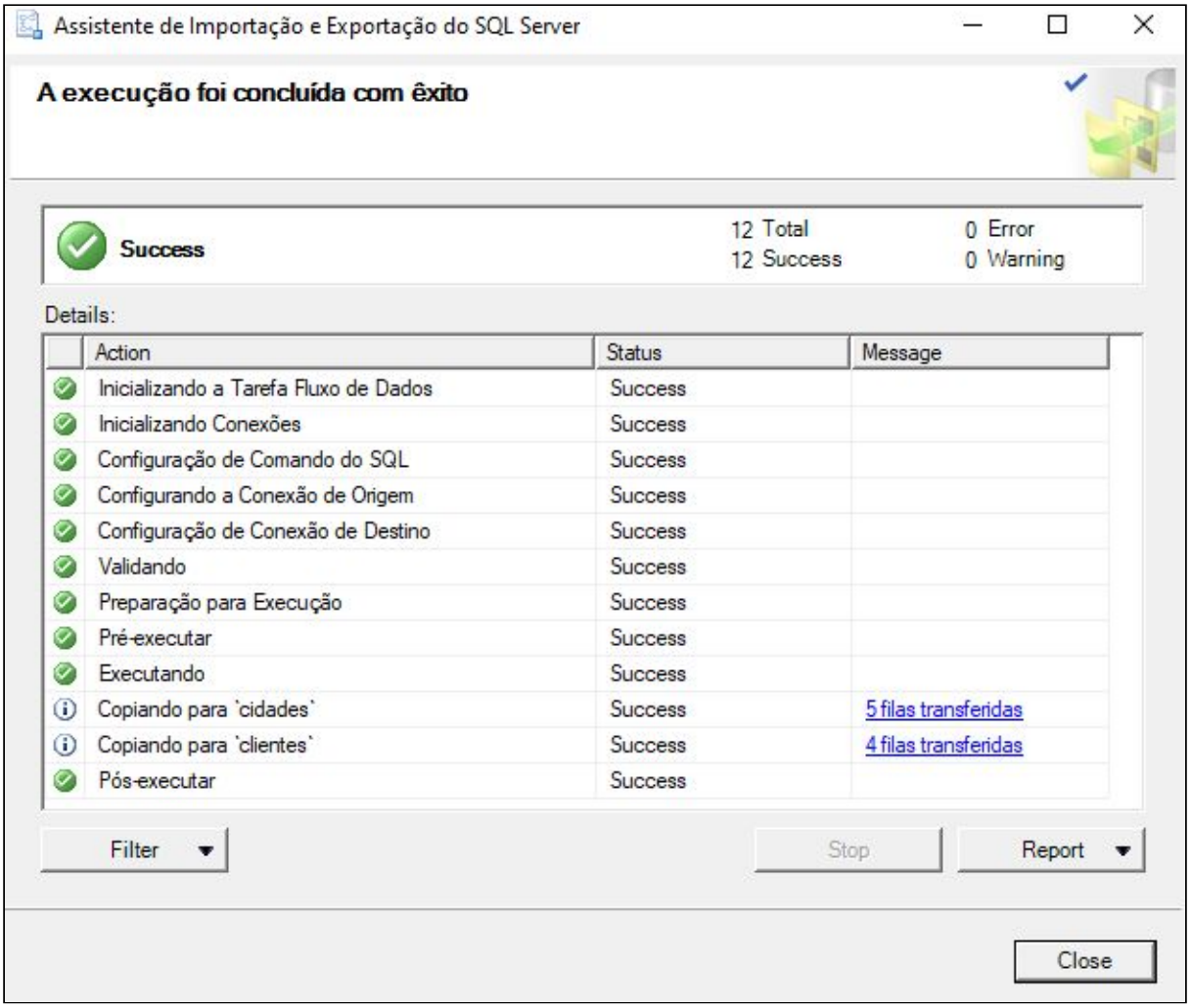
Na próxima tela precisamos apenas apertar o botão **Next >**



Para finalizarmos o processo de exportação, teremos uma janela com as características dos conteúdos exportados:



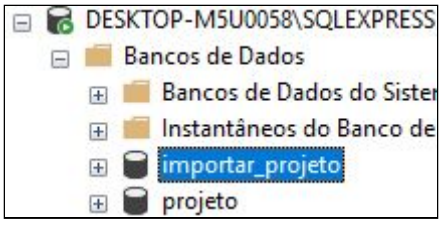
Agora é necessário esperar executar algumas tarefas, dependendo da estrutura poderá levar alguns segundos, minutos ou horas.



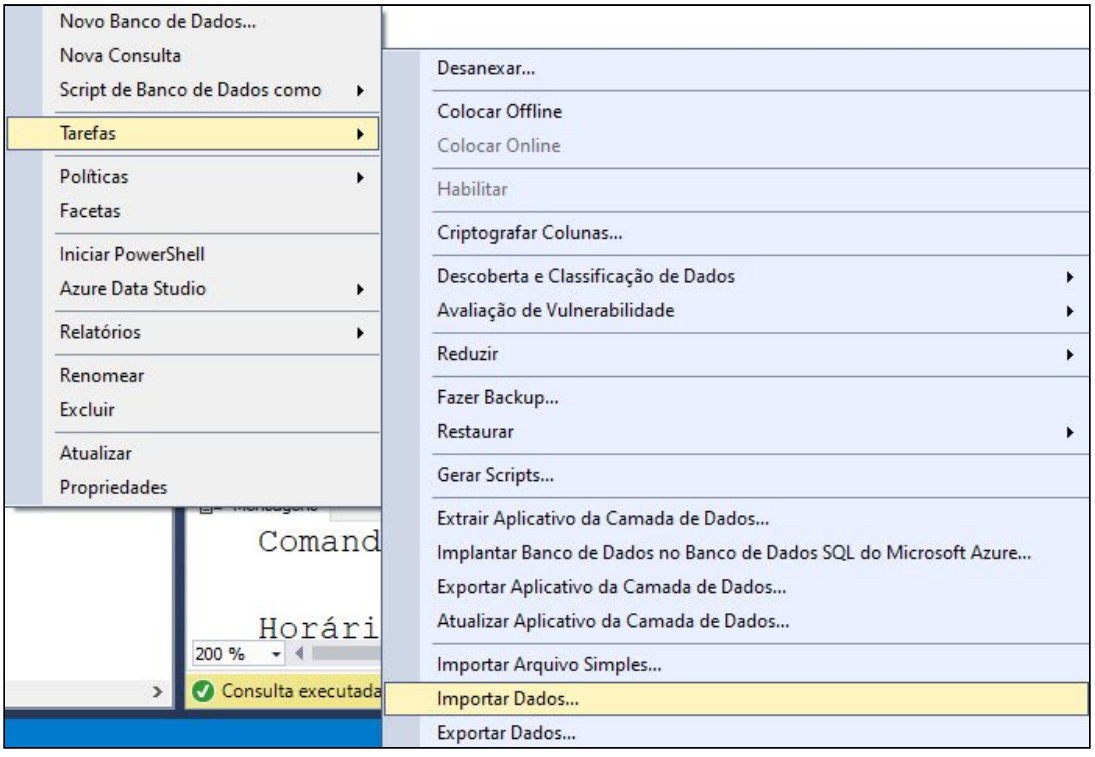
## Importando base de dados

Para podermos importar uma base de dados é bem simples, os passos são muito parecidos com a importação. Antes de iniciarmos o processo, é necessário criar uma base de dados, para esse exemplo a base **importar_projeto** foi criada.

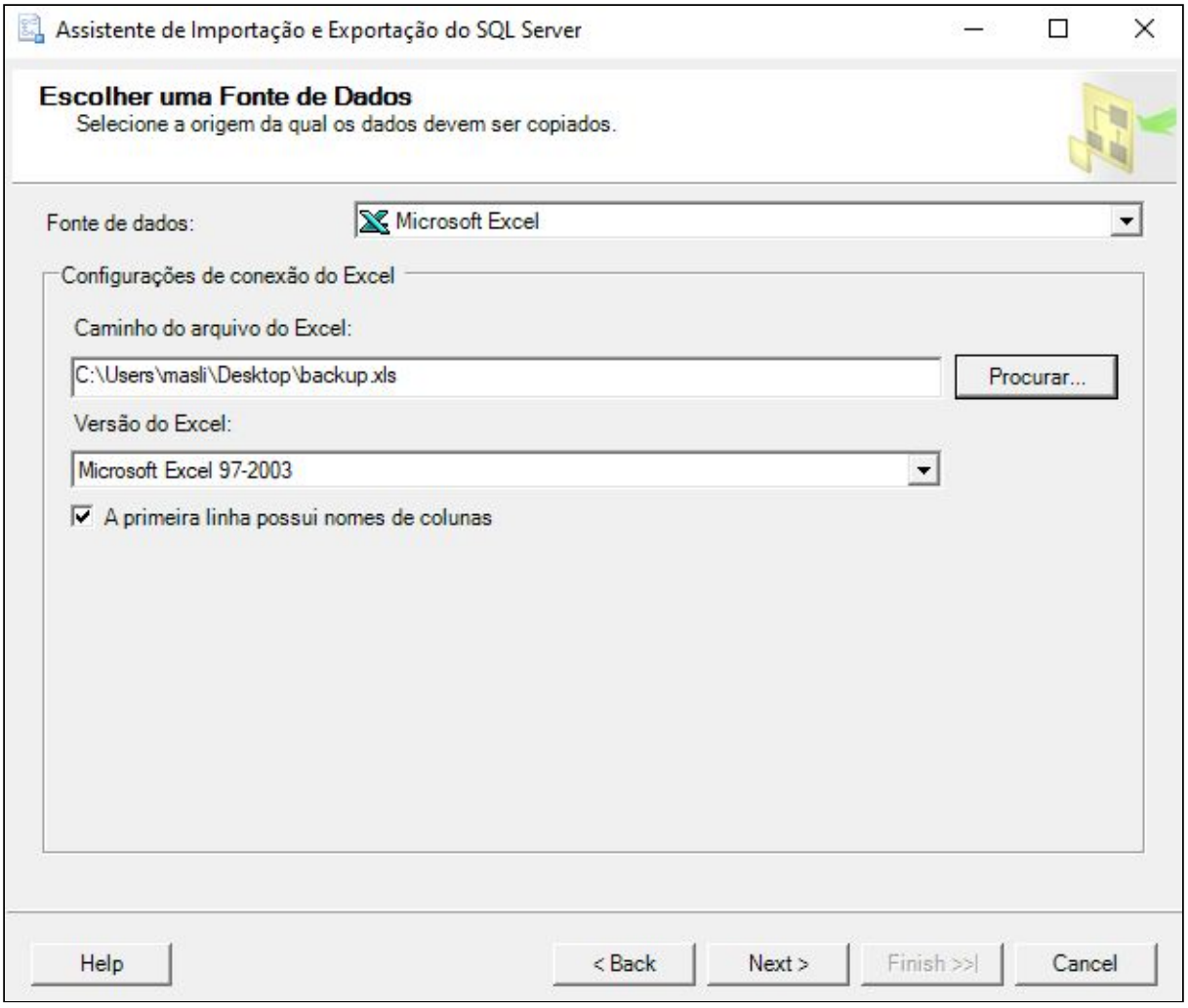
```
CREATE DATABASE importar_projeto;
```

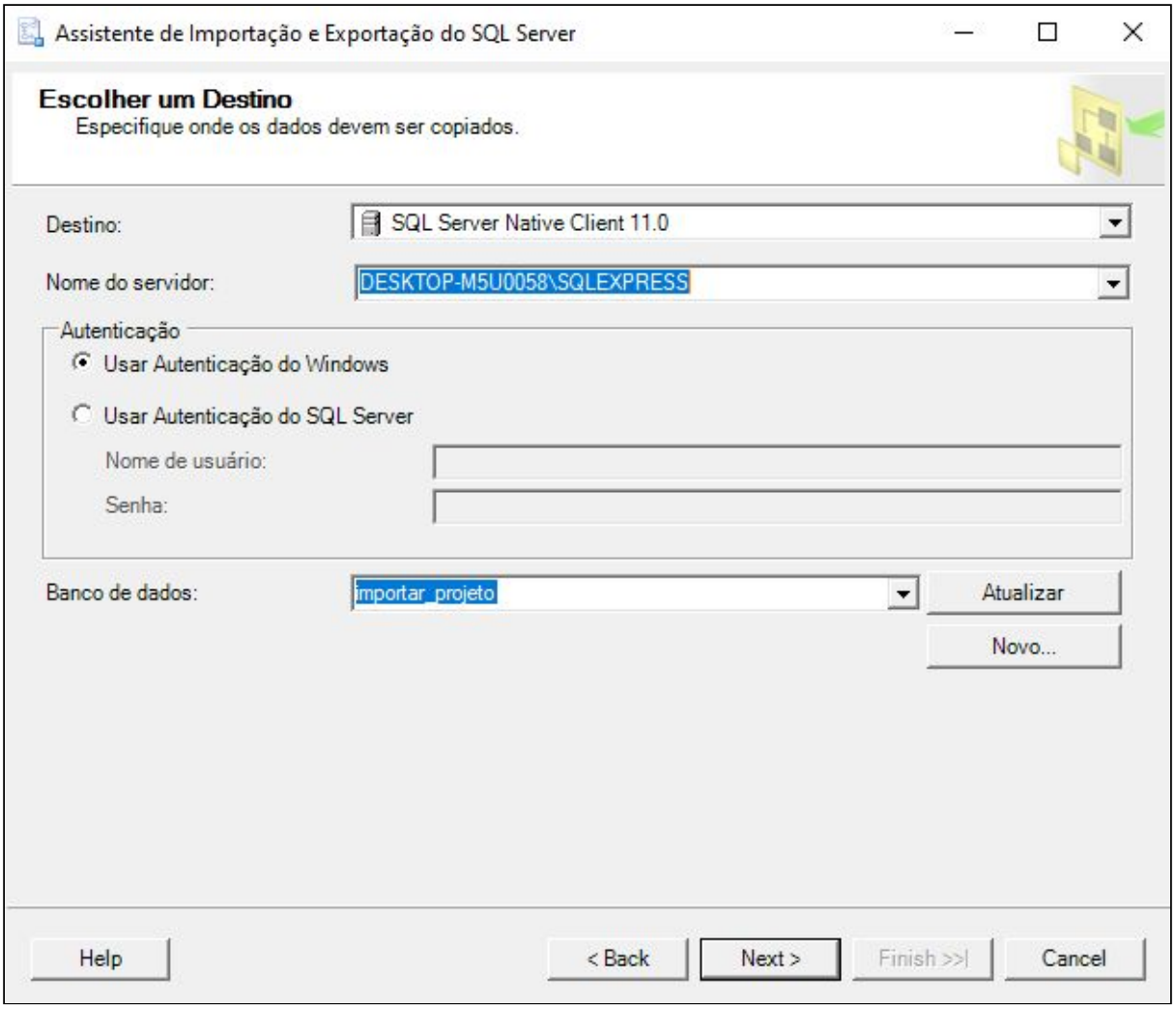


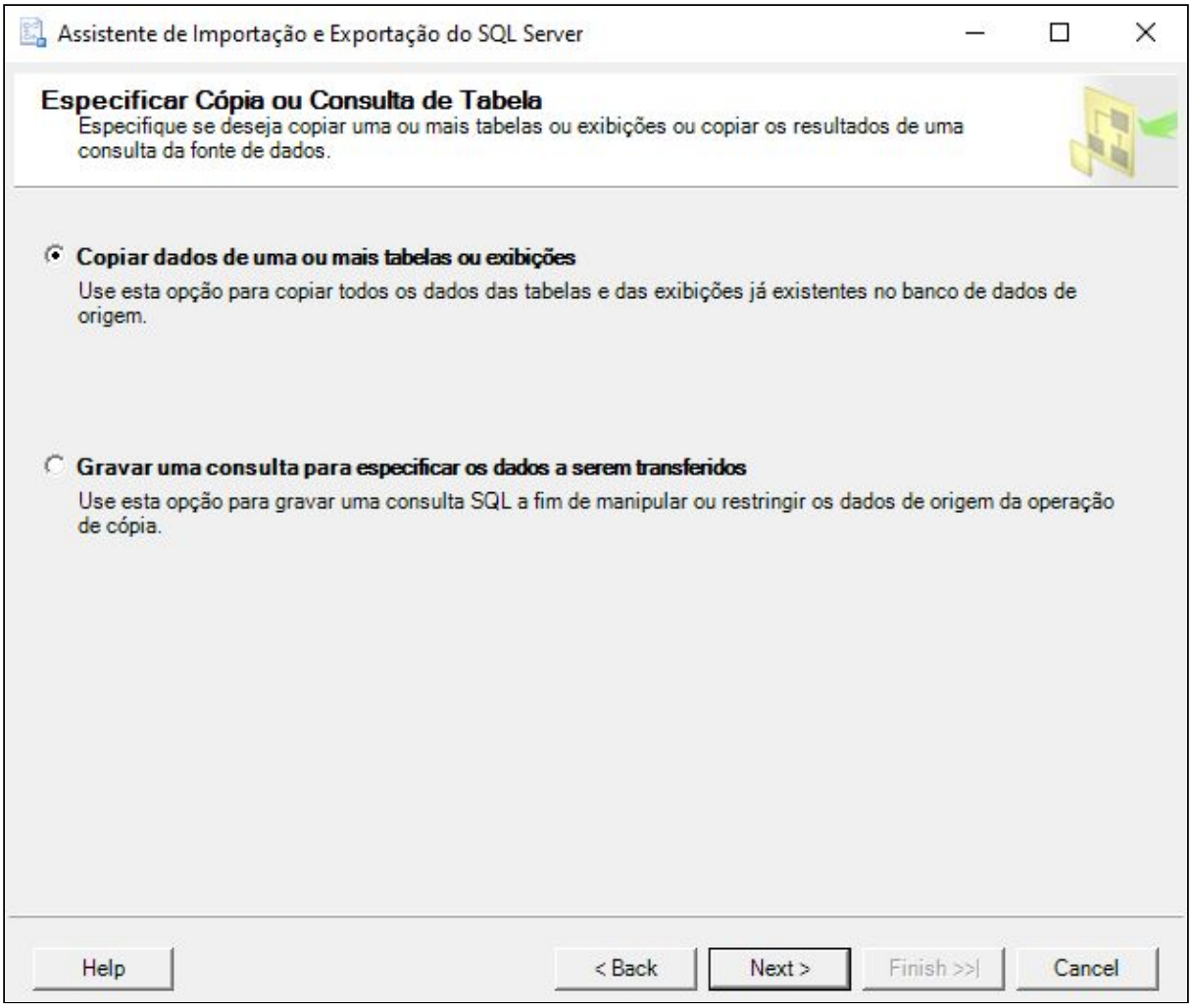

Clique com o botão direito sobre a nova base de dados criada, em seguida selecione **Tarefas**, em seguida **Importar Dados...**

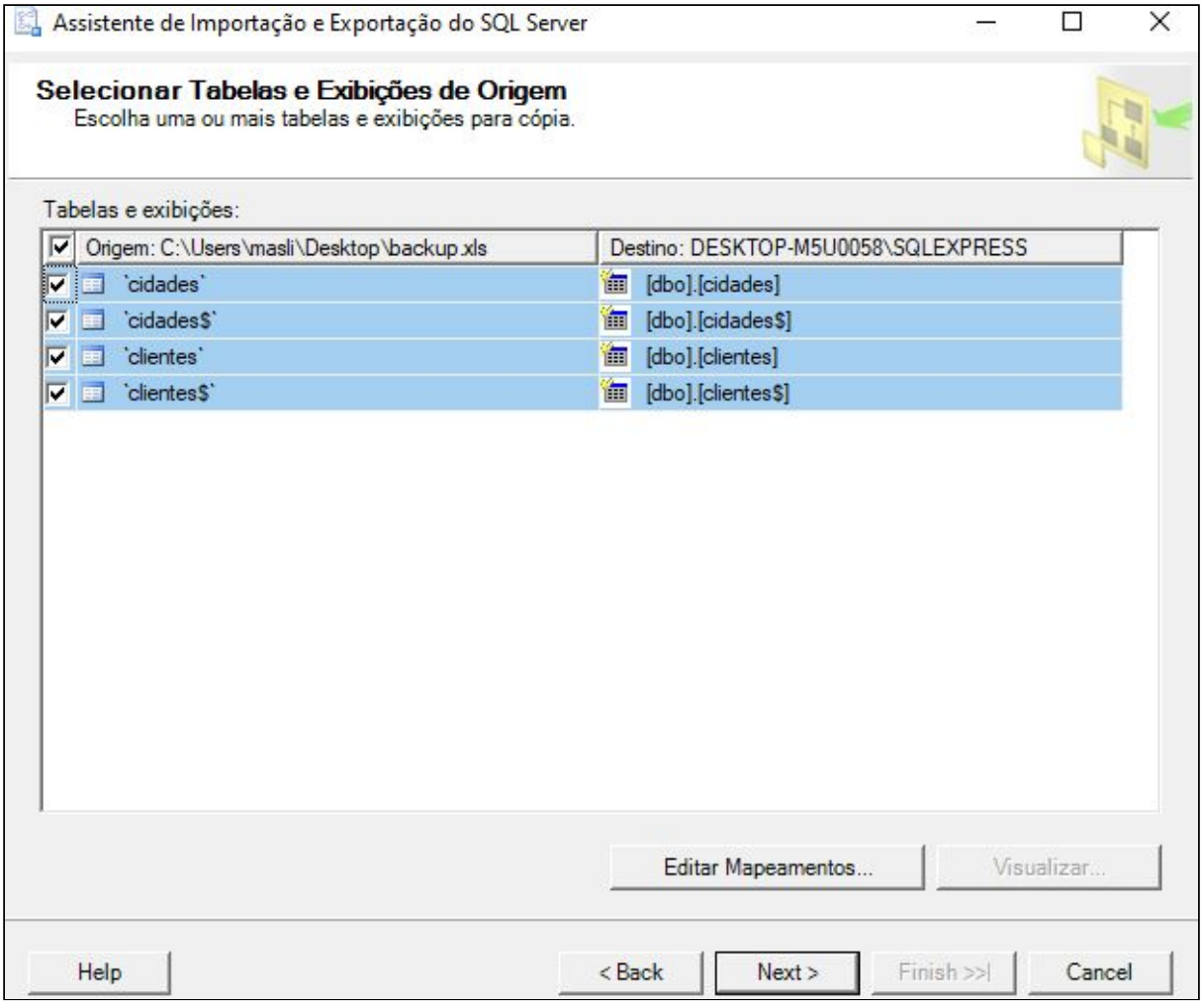


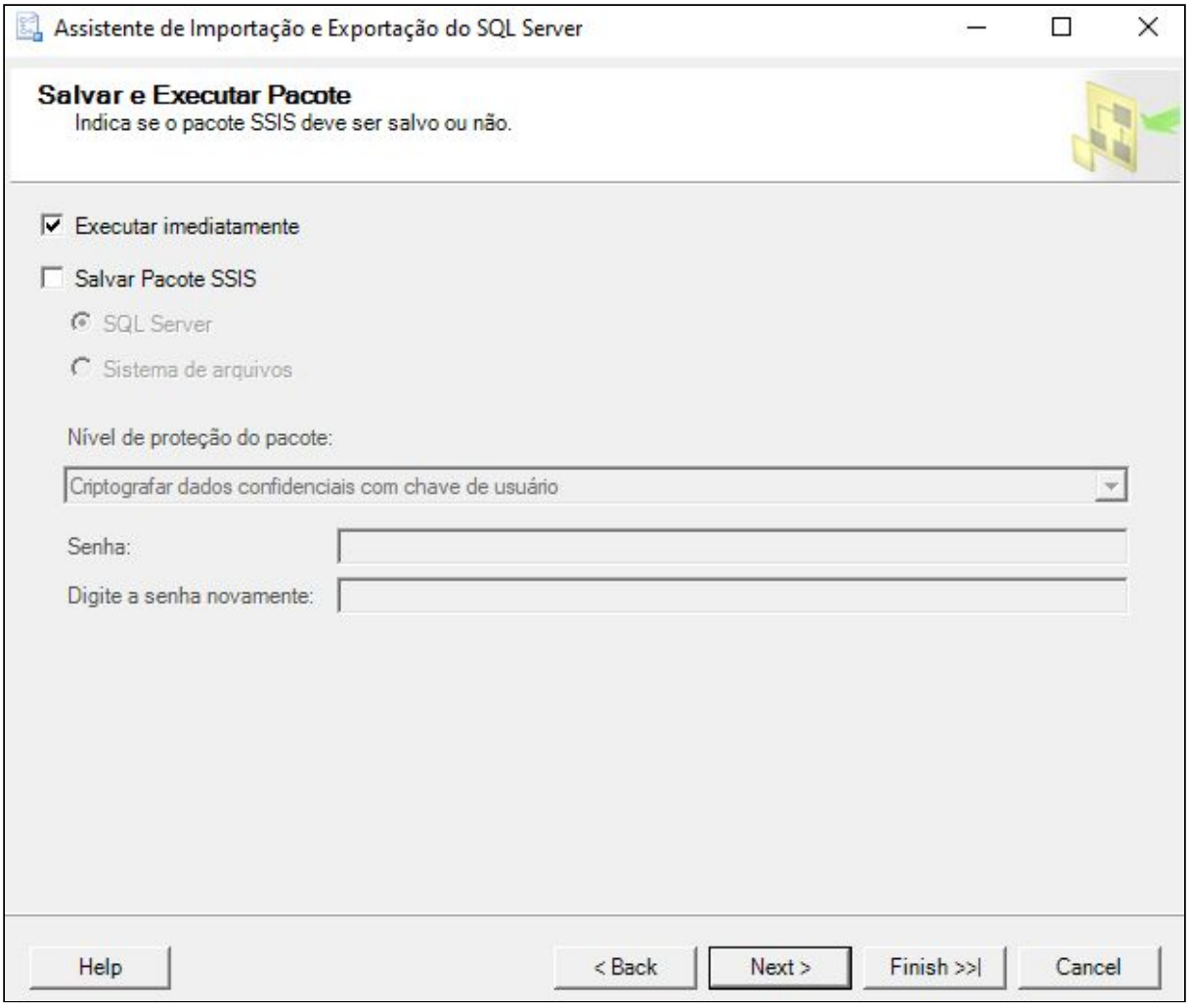
Siga os processos abaixo para importar o arquivo Excel exportado no exemplo anterior:

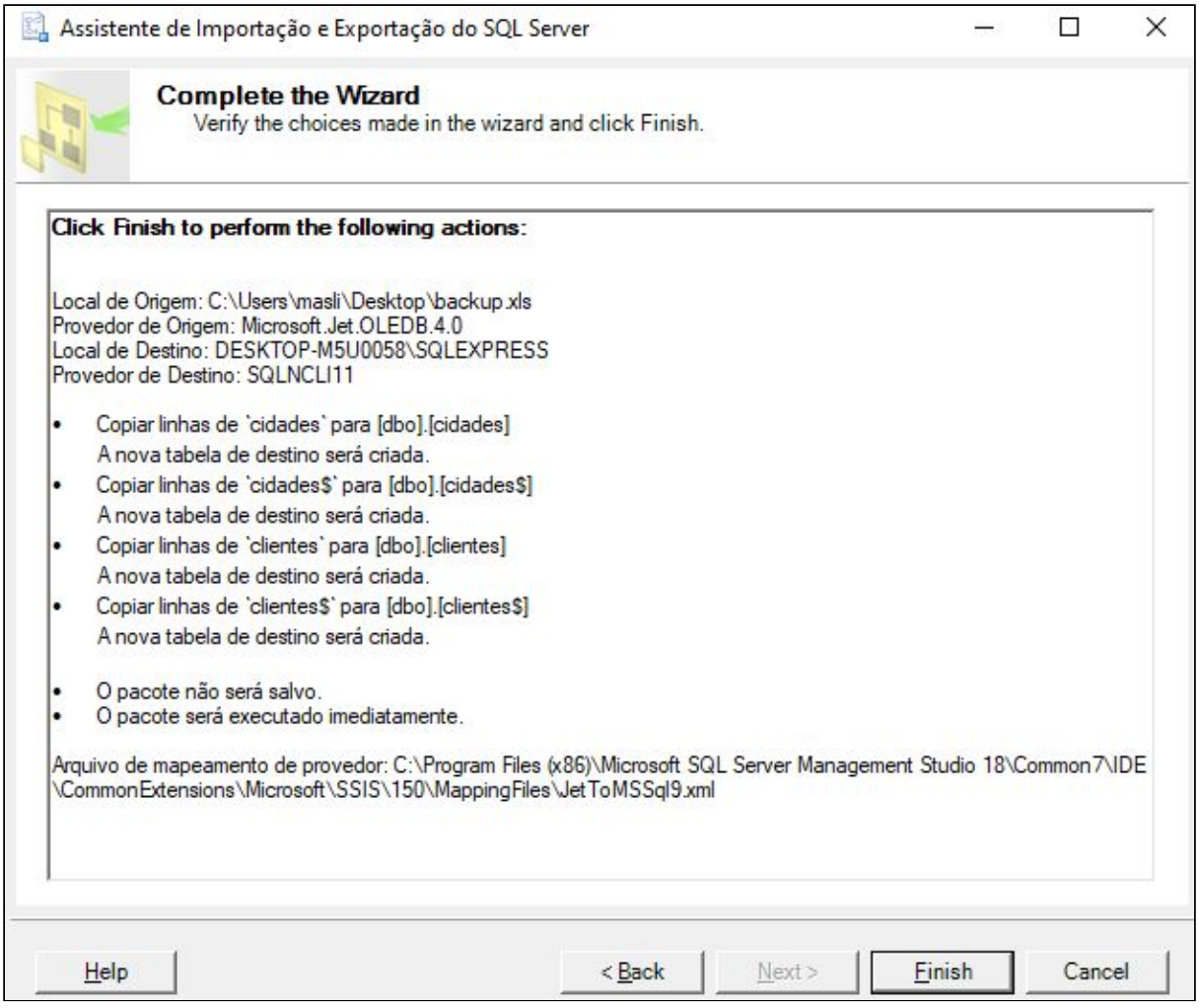

Assistente de Importação e Exportação do SQL Server
Escolher uma Fonte de Dados
Selecione a origem da qual os dados devem ser copiados.
Fonte de dados:
Microsoft Excel
Configurações de conexão do Excel
Caminho do arquivo do Excel:
C:\Users\masli\Desktop\backup.xls
Procurar...
Versão do Excel:
Microsoft Excel 97-2003
A primeira linha possui nomes de colunas
Help
< Back
Next >
Finish >>|
Cancel

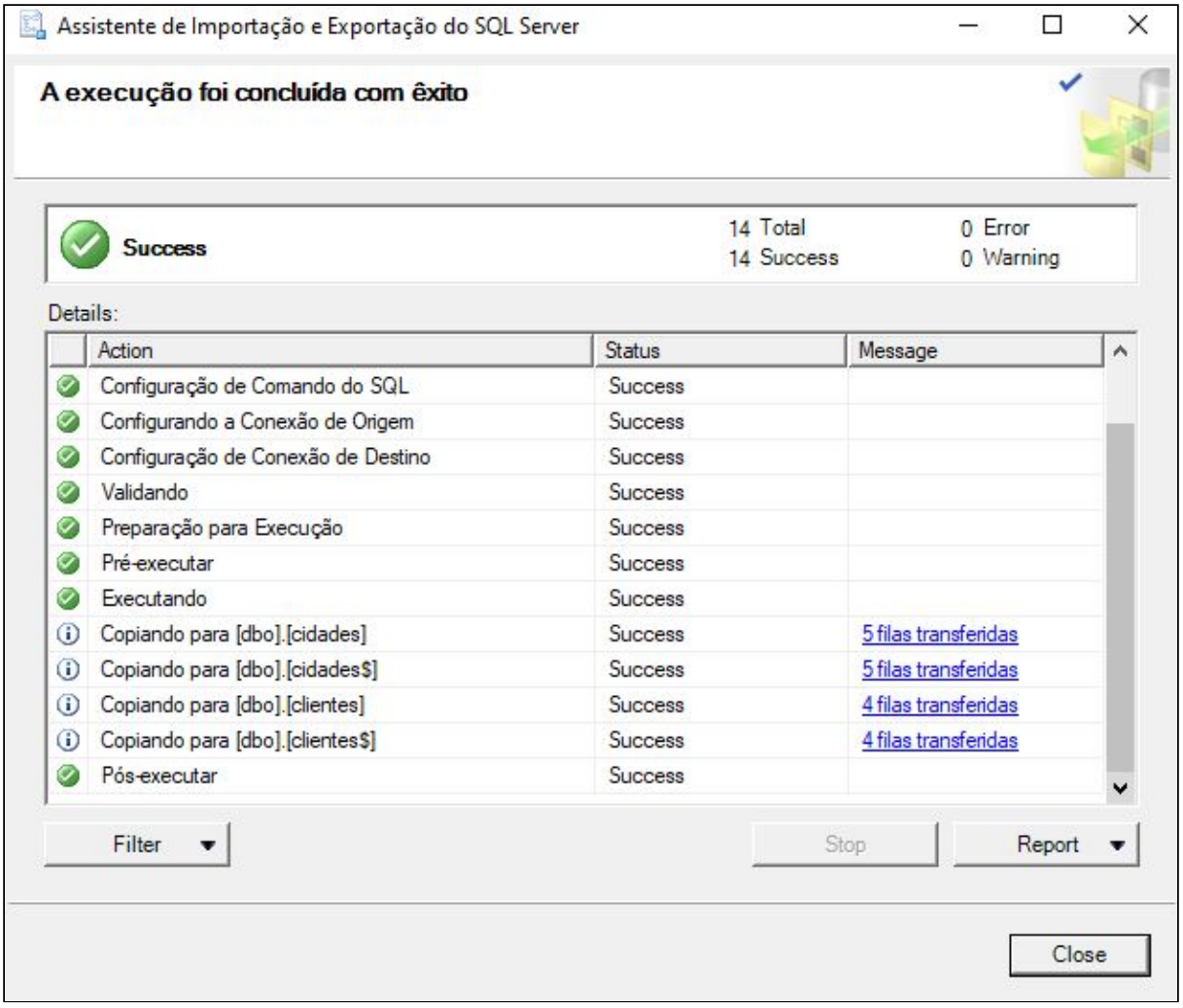

Assistente de Importação e Exportação do SQL Server
Escolher um Destino
Especifique onde os dados devem ser copiados.
Destino:
SQL Server Native Client 11.0
Nome do servidor:
DESKTOP-M5U0058\SQLEXPRESS
Autenticação
Usar Autenticação do Windows
Usar Autenticação do SQL Server
Nome de usuário:
Senha:
Banco de dados:
importar_projeto
Atualizar
Novo...
Help
< Back
Next >
Finish >>|
Cancel


Assistente de Importação e Exportação do SQL Server
Especificar Cópia ou Consulta de Tabela
Especifique se deseja copiar uma ou mais tabelas ou exibições ou copiar os resultados de uma consulta da fonte de dados.
Copiar dados de uma ou mais tabelas ou exibições
Use esta opção para copiar todos os dados das tabelas e das exibições já existentes no banco de dados de origem.
Gravar uma consulta para especificar os dados a serem transferidos
Use esta opção para gravar uma consulta SQL a fim de manipular ou restringir os dados de origem da operação de cópia.
Help
< Back
Next >
Finish >>|
Cancel


Assistente de Importação e Exportação do SQL Server
Selecionar Tabelas e Exibições de Origem
Escolha uma ou mais tabelas e exibições para cópia.
Tabelas e exibições:
Origem: C:\Users\masli\Desktop\backup.xls
Destino: DESKTOP-M5U0058\SQLEXPRESS
`cidades`
[dbo].[cidades]
`cidades$`
[dbo].[cidades$]
`clientes`
[dbo].[clientes]
`clientes$`
[dbo].[clientes$]
Editar Mapeamentos...
Visualizar...
Help
< Back
Next >
Finish >>|
Cancel


Assistente de Importação e Exportação do SQL Server
Salvar e Executar Pacote
Indica se o pacote SSIS deve ser salvo ou não.
Executar imediatamente
Salvar Pacote SSIS
SQL Server
Sistema de arquivos
Nível de proteção do pacote:
Criptografar dados confidenciais com chave de usuário
Senha:
Digite a senha novamente:
Help
< Back
Next >
Finish >>|
Cancel


Assistente de Importação e Exportação do SQL Server
Complete the Wizard
Verify the choices made in the wizard and click Finish.
Click Finish to perform the following actions:
Local de Origem: C:\Users\masli\Desktop\backup.xls
Provedor de Origem: Microsoft.Jet.OLEDB.4.0
Local de Destino: DESKTOP-M5U0058\SQLEXPRESS
Provedor de Destino: SQLNCLI11
Copiar linhas de 'cidades' para [dbo].[cidades]
A nova tabela de destino será criada.
Copiar linhas de 'cidades$' para [dbo].[cidades$]
A nova tabela de destino será criada.
Copiar linhas de 'clientes' para [dbo].[clientes]
A nova tabela de destino será criada.
Copiar linhas de 'clientes$' para [dbo].[clientes$]
A nova tabela de destino será criada.
O pacote não será salvo.
O pacote será executado imediatamente.
Arquivo de mapeamento de provedor: C:\Program Files (x86)\Microsoft SQL Server Management Studio 18\Common7\IDE\CommonExtensions\Microsoft\SSIS\150\MappingFiles\JetToMSSql9.xml
Help
< Back
Next >
Finish
Cancel




Note que temos as tabelas que foram exportadas anteriormente na nova base de dados: