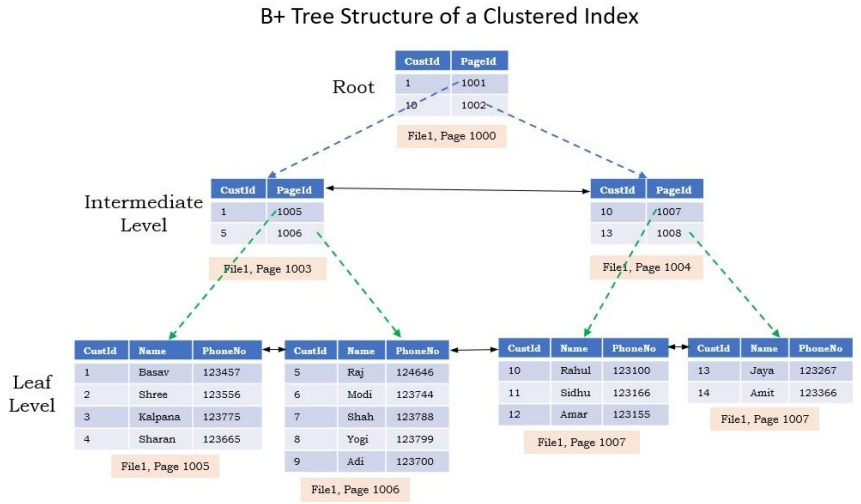
Indexes

Clustered & Non-Clustered

Prof. Leandro Colevati

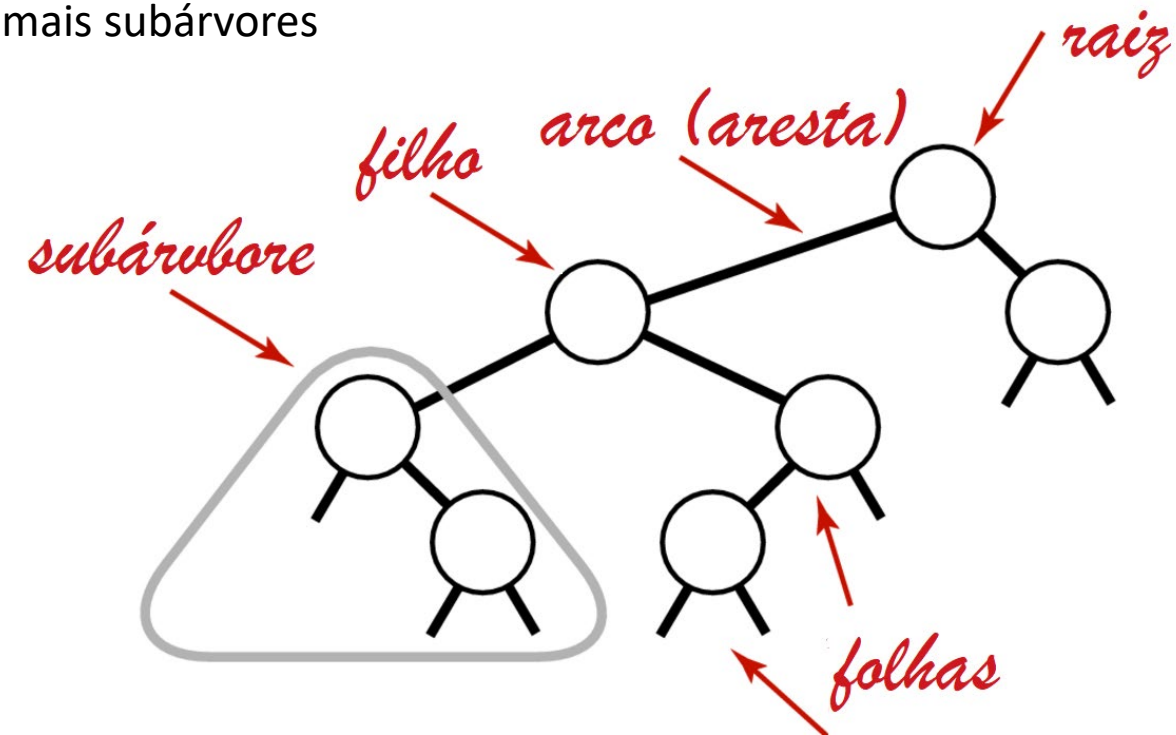
Introdução

- Árvores são estruturas adequadas para representação de hierarquias
- Exemplos:
  - Hierarquia de pastas (Sistemas de Arquivos)
  - Árvore genealógica
- O conceito de árvores está diretamente ligado com recursividade



Introdução

- Elementos básicos:
  - Grau
    - Número de subárvores de um nó
  - Raiz
    - Primeiro elemento da árvore, com zero ou mais subárvores
  - Folha
    - Nós sem filhos
  - Filhos
    - Nós raízes de uma subárvore
  - Arco (Aresta)
    - Conexão entre dois nós

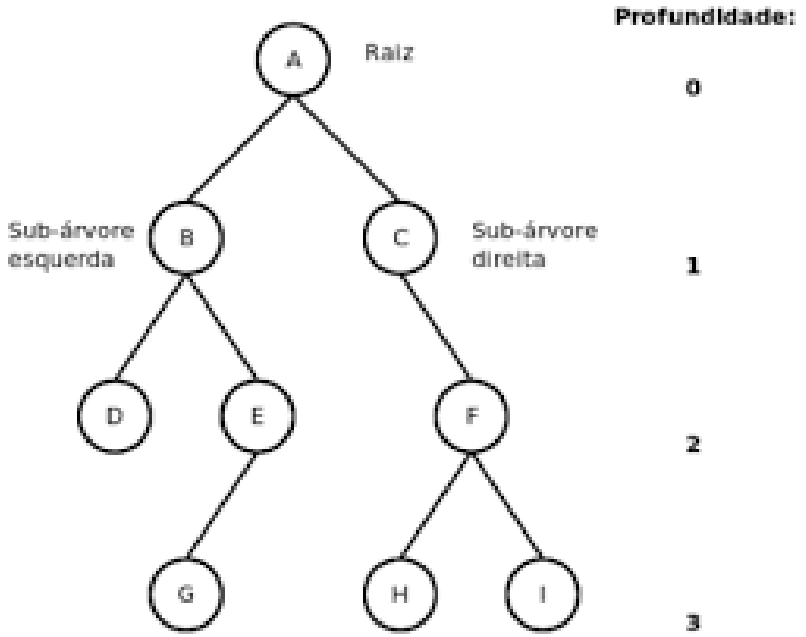


Introdução

- Propriedades:
  - Cada vértice (exceto a raiz) tem exatamente um antecessor (pai)
  - Cada vértice tem nós sucessores imediatos, com exceção das folhas (ou terminais)
  - Filhos de um mesmo pai ➔ irmãos
  - Nós com pelo menos um filho ➔ Internos (ou Não-terminais)
  - Caminho é uma lista de vértices distintos e sucessivos conectados por arcos
  - Existe exatamente um caminho entre a raiz e cada um dos nós da árvore
  - Qualquer nó é a raiz de uma subárvore consistindo dele e dos nós abaixo

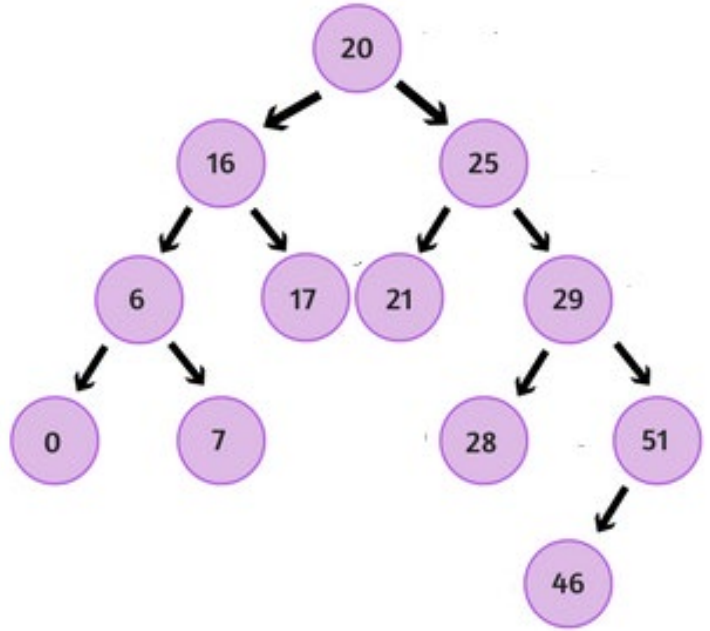
Árvores Binárias

- Árvore nula ou com as seguintes características:
  - Existe um nós especial denominado raiz
  - Nenhum nó tem grau superior a 2, ou seja, nenhum nó tem mais de 2 filhos
  - Existe um senso de posição, ou seja, distingue-se subárvore esquerda e subárvore direita



Árvores de Busca Binária

- Uma árvore é denominada Árvore de Busca Binária se:
  - **Todo** elemento da subárvore esquerda é menor que o elemento raiz
  - **Nenhum** elemento da subárvore direita é menor que o elemento raiz
  - As subárvores esquerda e direita também são de busca binária



SQL Indexes

A maneira mais básica de localizar um registro é procurando-o, na sequência de registros, até encontrá-lo. É o que chamamos tecnicamente de "*table scan*", ou seja, varrer a tabela toda...

Este método pode ser eficiente quando o registro buscado for o primeiro da lista. Mas e se for o último?!

Um jeito interessante de resolver este problema é ordenar a tabela pelo campo em questão.

Neste caso, pode ser realizada uma busca por métodos de "bisecção", ou seja, dividir a tabela ao meio e avaliar em que metade o registro se encontra.

Para isso, compara-se o valor do registro na posição equivalente ao "meio" da tabela com o valor buscado. Se for menor, o dado que procuramos está na metade superior. Se for maior, está na metade inferior.

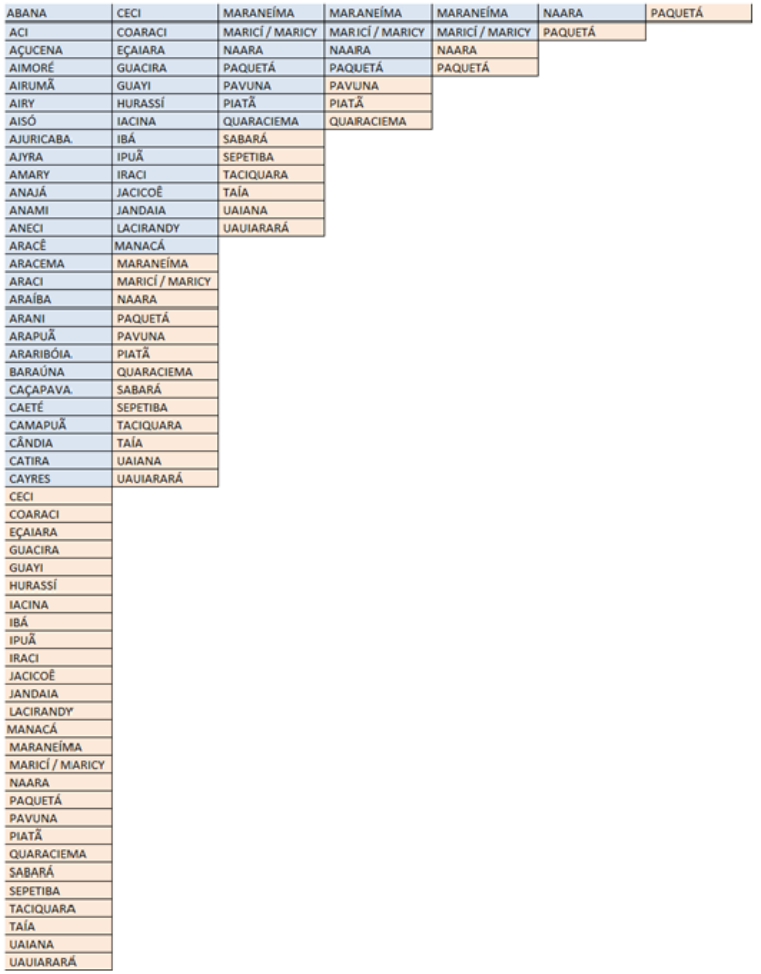
Depois, repetimos o mesmo processo para a metade que sobrou.

SQL Indexes

Vejamos como funciona na prática: vamos procurar, numa lista de 54 nomes femininos de origem Tupi, o nome ***PAQUETÁ***.

A metade na qual o nome se encontra foi copiada à direita da lista anterior, e assim sucessivamente até encontrarmos o nome ***PAQUETÁ***. Repare que foram necessários somente **6 passos**, muito menos que os **45** que seriam necessários no caso de "*table scan*".

SQL Indexes



ABANA
ACI
AÇUCENA
AIMORÉ
AIRUMÃ
AIRY
AISÓ
AJURICABA
AJYRA
AMARY
ANAJÁ
ANAMI
ANECI
ARACÊ
ARACEMA
ARACI
ARAÍBA
ARANI
ARAPUÃ
ARARIBÓIA
BARAÚNA
CAÇAPAVA
CAETÉ
CAMAPUÃ
CÂNDIA
CATIRA
CAYRES
CECI
COARACI
EÇAIARA
GUACIRA
GUAYI
HURASSÍ
IACINA
IBÁ
IPUÃ
IRACI
JACICOÊ
JANDAIA
LACIRANDY
MANACÁ
MARANEÍMA
MARICÍ / MARICY
NAARA
PAQUETÁ
PAVUNA
PIATÃ
QUARACIEMA
SABARÁ
SEPETIBA
TACIQUARA
TAÍA
UAIANA
UAUIARARÁ

CECI
COARACI
EÇAIARA
GUACIRA
GUAYI
HURASSÍ
IACINA
IBÁ
IPUÃ
IRACI
JACICOÊ
JANDAIA
LACIRANDY
MANACÁ
MARANEÍMA
MARICÍ / MARICY
NAARA
PAQUETÁ
PAVUNA
PIATÃ
QUARACIEMA
SABARÁ
SEPETIBA
TACIQUARA
TAÍA
UAIANA
UAUIARARÁ

MARANEÍMA
MARICÍ / MARICY
NAARA
PAQUETÁ
PAVUNA
PIATÃ
QUARACIEMA
SABARÁ
SEPETIBA
TACIQUARA
TAÍA
UAIANA
UAUIARARÁ

MARANEÍMA
MARICÍ / MARICY
NAARA
PAQUETÁ
PAVUNA
PIATÃ
QUARACIEMA

MARANEÍMA
MARICÍ / MARICY
NAARA
PAQUETÁ

NAARA
PAQUETÁ

PAQUETÁ

SQL Indexes - Sintaxe

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX *index_name*

ON(*column* [ASC | DESC] [,...*n*])

[INCLUDE (*column_name* [,...*n*])]

[WITH ([,...*n*])]

[ON { *partition_scheme_name* (*column_name*)

| *filegroup_name*

| default

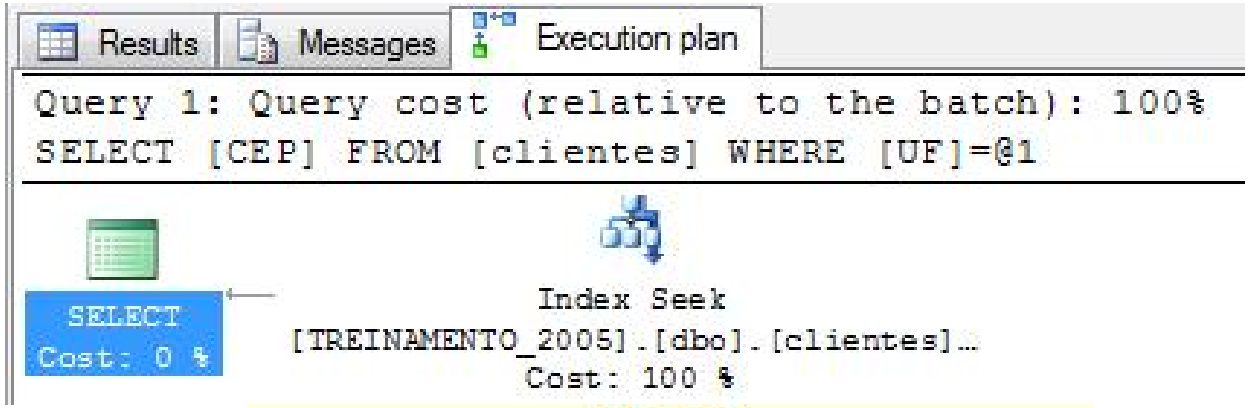
]

SQL Indexes - Acesso

Os acessos aos dados das tabelas e índices podem ser de duas formas, SEEK ou SCAN.

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

SQL Indexes - Desempenho



SELECT

Cached plan size	9 B
Degree of Parallelism	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,0032831
Estimated Number of Rows	1

Statement

SELECT [CEP] FROM [clientes] WHERE [UF] =@1

SQL Indexes - Exemplo

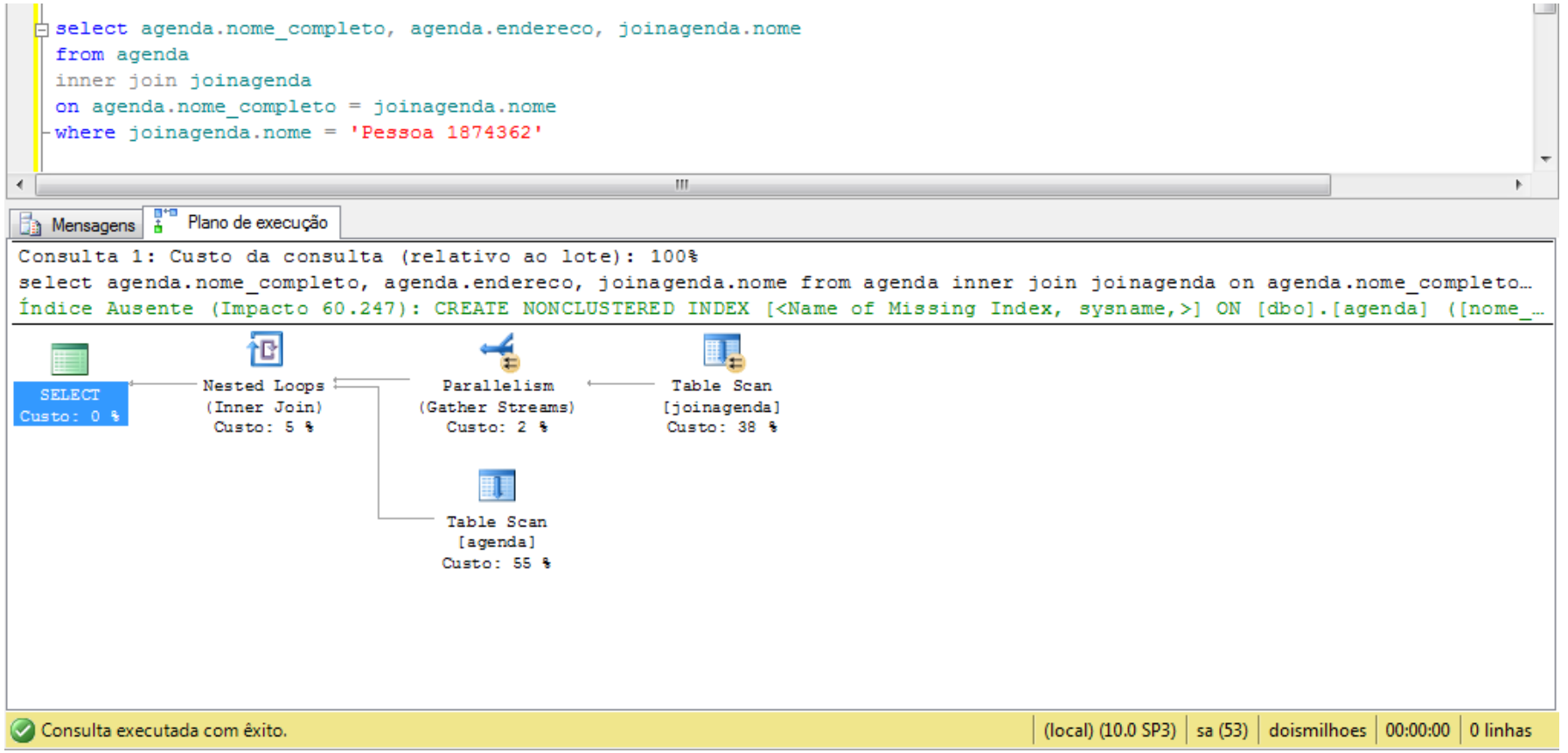
```
CREATE DATABASE doismilhoes
GO
USE doismilhoes
GO
CREATE TABLE agenda(
nome_completo               VARCHAR(100),
endereco                    VARCHAR(100)
PRIMARY KEY (nome_completo))
GO
CREATE TABLE joinagenda(
codjoinagenda               INT          IDENTITY     NOT NULL,
nome                        VARCHAR(100)
PRIMARY KEY (codjoinagenda))
GO
DECLARE @cont AS INT
SET @cont = 1
WHILE @cont <= 2000000
BEGIN
             INSERT INTO agenda VALUES
             ('Pessoa ‘+CONVERT(VARCHAR(10),@cont),'Rua
             ‘+CONVERT(VARCHAR(10),@cont))
             INSERT INTO joinagenda VALUES
             ('Pessoa ‘+CONVERT(VARCHAR(10),@cont))
             SET @cont = @cont + 1
END
GO
SELECT * FROM agenda
SELECT * FROM joinagenda
```

```
SELECT * FROM sys.sysindexes
WHERE id = OBJECT_ID('agenda')

SELECT * FROM sys.sysindexes
WHERE id = OBJECT_ID('agenda')

CREATE NONCLUSTERED INDEX IX_agenda
ON agenda (nome_completo)

CREATE NONCLUSTERED INDEX IX_joinagenda
ON joinagenda (nome)

CREATE CLUSTERED INDEX IX_agenda
ON agenda (nome_completo)

CREATE CLUSTERED INDEX IX_joinagenda
ON joinagenda (nome)

DROP INDEX IX_agenda
ON agenda

DROP INDEX IX_joinagenda
ON joinagenda
```

```
SELECT endereco
FROM agenda
WHERE nome_completo = 'Pessoa 1557854’

SELECT ag.nome_completo, ag.endereco, ja.nome
FROM agenda ag, joinagenda ja
WHERE ag.nome_completo = ja.nome
AND joinagenda.nome = 'Pessoa 1874362'

INSERT INTO joinagenda
VALUES ('Pessoa 1984784a')

DELETE joinagenda
WHERE nome = 'Pessoa 1984784a'
```

SQL Indexes - Exemplo

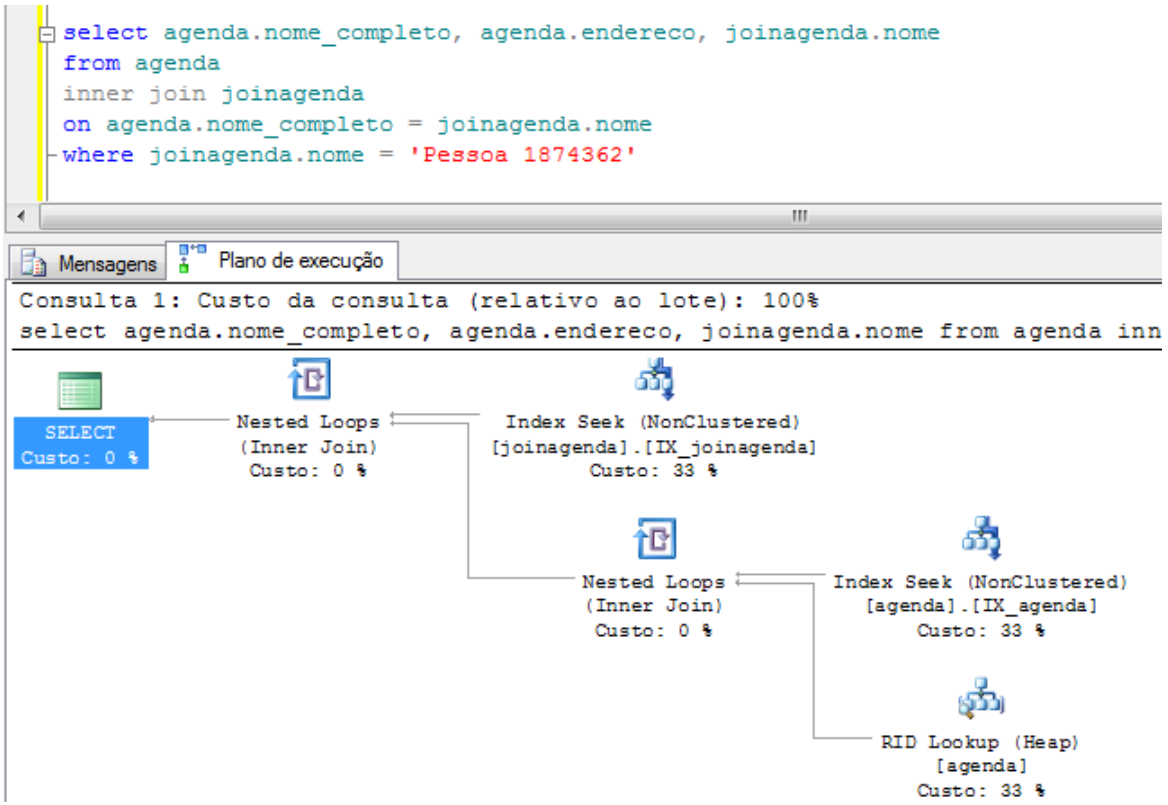
- Sem Index



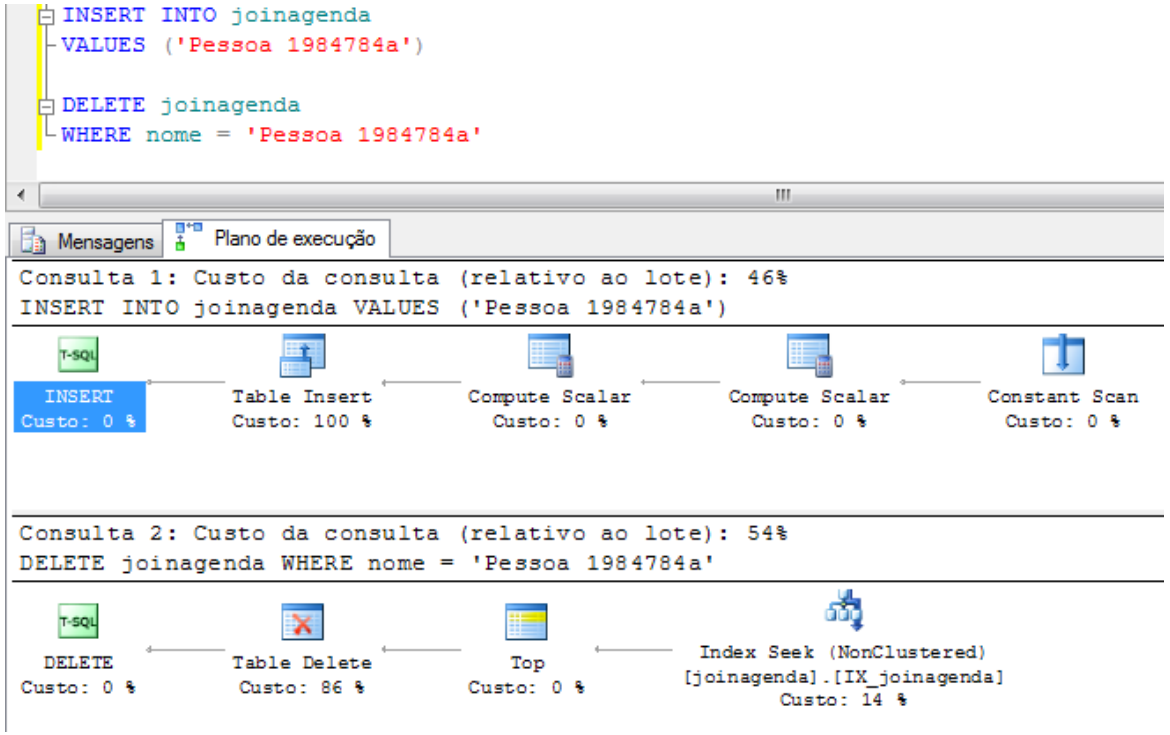
SQL Indexes - Exemplo

- Com INDEX NonClustered nas duas tabelas

```
select agenda.nome_completo, agenda.endereco, joinagenda.nome
from agenda
inner join joinagenda
on agenda.nome_completo = joinagenda.nome
where joinagenda.nome = 'Pessoa 1874362'
```




```
INSERT INTO joinagenda
VALUES ('Pessoa 1984784a')

DELETE joinagenda
WHERE nome = 'Pessoa 1984784a'
```



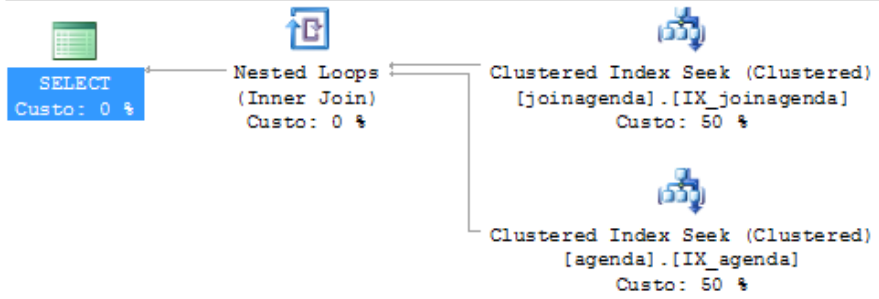
SQL Indexes - Exemplo

- Com INDEX Clustered nas duas tabelas

```
select agenda.nome_completo, agenda.endereco, joinagenda.nome
from agenda
inner join joinagenda
on agenda.nome_completo = joinagenda.nome
where joinagenda.nome = 'Pessoa 1874362'
```

Mensagens | Plano de execução

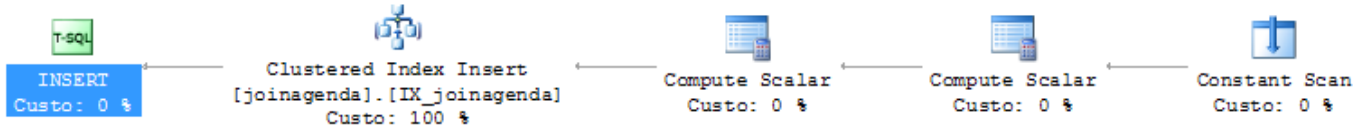
```
Consulta 1: Custo da consulta (relativo ao lote): 100%
select agenda.nome_completo, agenda.endereco, joinagenda.nome f
```




```
INSERT INTO joinagenda
VALUES ('Pessoa 1984784a')

DELETE joinagenda
WHERE nome = 'Pessoa 1984784a'
```

Mensagens | Plano de execução

```
Consulta 1: Custo da consulta (relativo ao lote): 43%
INSERT INTO joinagenda VALUES ('Pessoa 1984784a')
```




```
Consulta 2: Custo da consulta (relativo ao lote): 57%
DELETE joinagenda WHERE nome = 'Pessoa 1984784a'
```