

# TRABALHO PRÁTICO 3

---

# Operação IndexScan

- É uma operação **folha** que provê acesso direto aos registros de uma tabela
  - Usa o índice para acelerar buscas por equivalência sobre uma pk

```
Operation scan = new IndexScan ("t1", table);
```

```
scan.open();  
Iterator<Tuple> it = scan.run();  
while (it.hasNext()){  
    Tuple t = it.next();  
    System.out.println(t);  
}
```

**scan**  
(IndexScan)  
[t1]

# Operação IndexScan

- O IndexScan consegue otimizar o filtro que chega da operação de nível superior, desde que essa operação seja
  - Uma PKFilter por equivalência
  - Um NestedLoopJoin

# Operação IndexScan

- No caso do PKFilter, o IndexScan consegue localizar com eficiência os registros que satisfaçam um filtro por equivalência
- Exemplo

```
Operation s1 = new IndexScan ("t1", table);
Operation f1 = new PKFilter(s1, "t1",
                           EQUAL, 200L);

f1.open();
Iterator<Tuple> it = f1.run();

while (it.hasNext()){
    ...
}
```

**f1** (PKFilter)  
t1.pk = 200

|

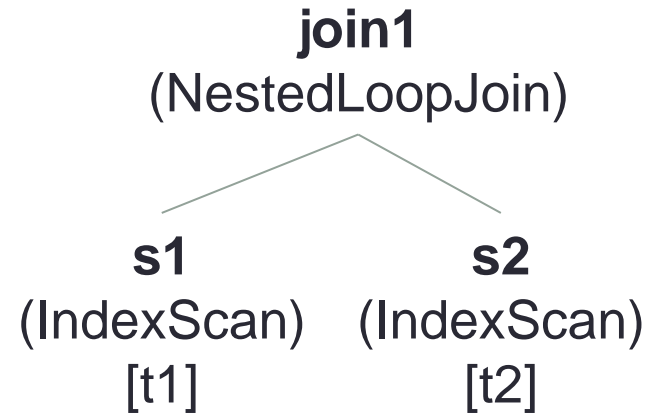
**s1**  
(IndexScan)  
[t1]

# Operação IndexScan

- No caso do NestedLoopJoin, o IndexScan consegue localizar com eficiência os registros que satisfaçam o critério de junção
  - O algoritmo se comporte como um IndexedNestedLoopJoin
- Exemplo

```
Operation s1 = new IndexScan("t1", table1);  
Operation s2 = new IndexScan("t2", table2);  
Operation join1 = new  
    NestedLoopJoin(s1, s2);
```

```
join1.open();  
Iterator<Tuple> it = scan.run();  
  
while (it.hasNext()){  
    ...  
}
```

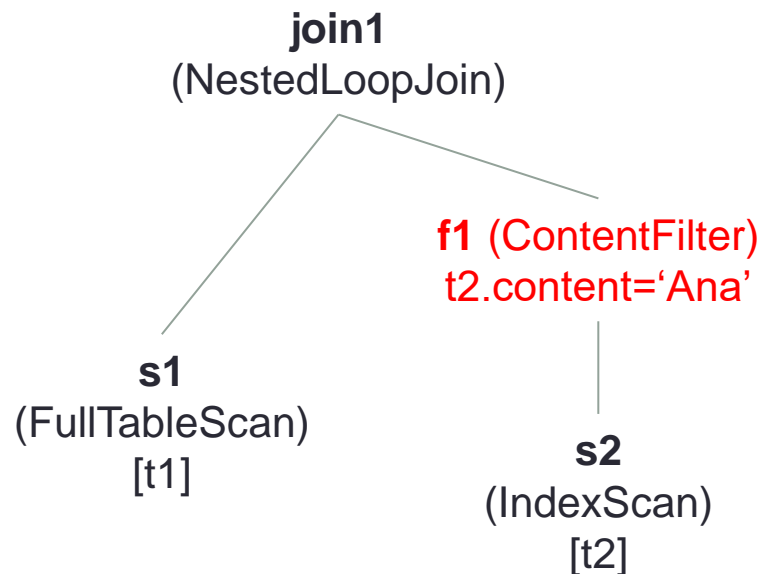


# Operação IndexScan

- Dependendo de como a árvore for construída, o IndexScan não consegue ser usado da maneira mais eficiente
- O problema acontece quando o IndexScan é chamado por qualquer operação que não seja
  - Um PKFilter por equivalência
  - Um NestedLoopJoin
- Nesse caso, o algoritmo se comporta como o FullTableScan

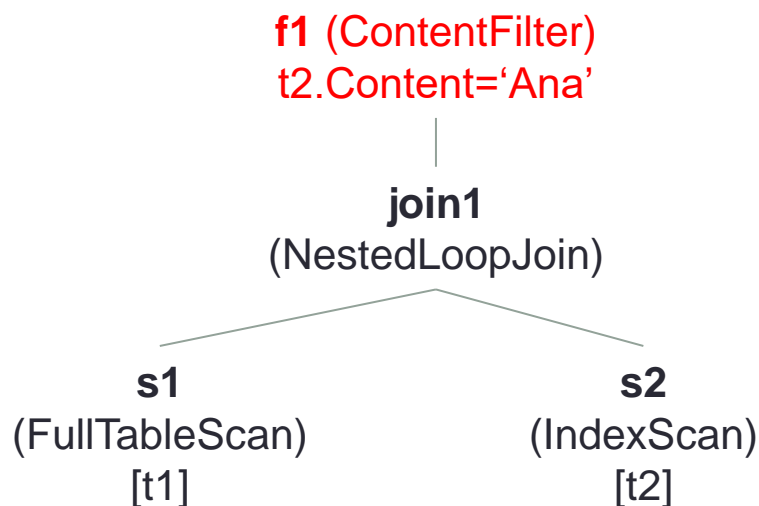
# Exemplos

- No exemplo abaixo, o filtro (= Ana) precisa acessar todos os registros de t2
  - Isso acontece uma vez para cada registro de t1



# Exemplos

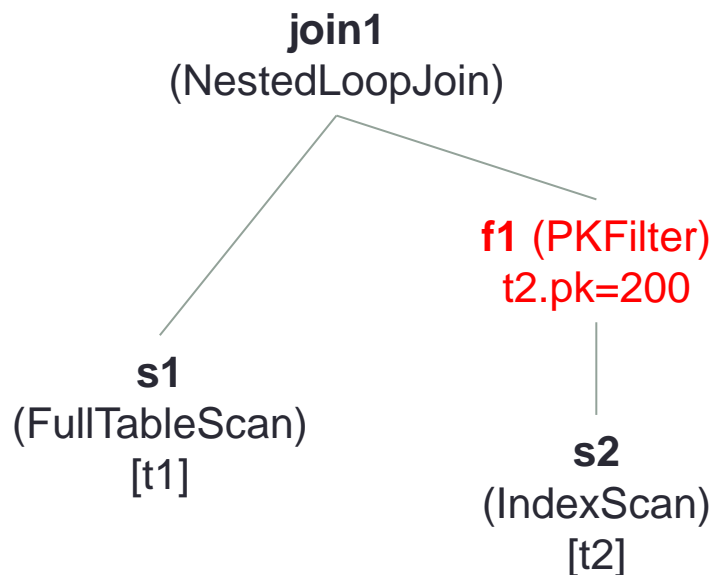
- Neste outro exemplo, o filtro é feito depois
  - Aparentemente isso indicar um plano menos eficiente
  - No entanto, a junção não precisa percorrer todos os registros de t2
    - O IndexScan atinge o registro correto diretamente





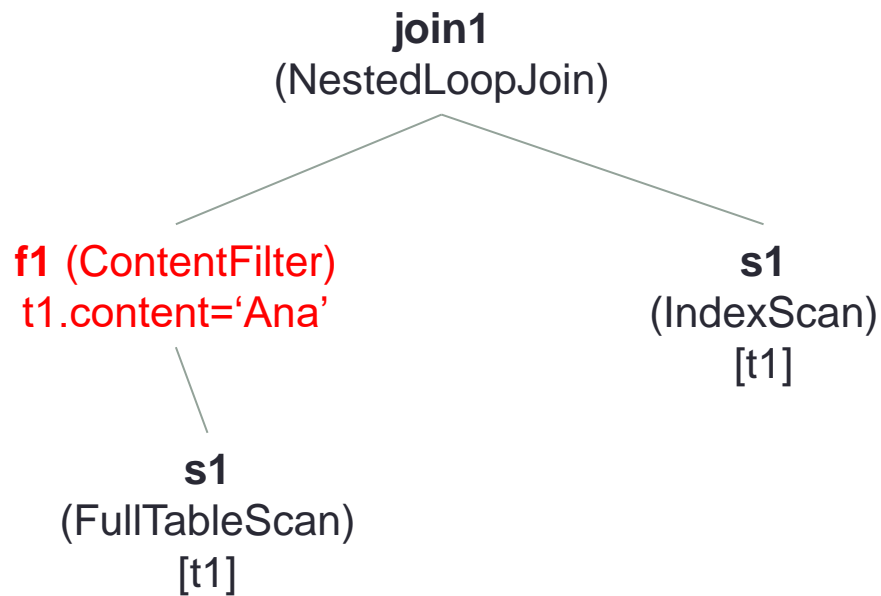
# Exemplos

- Mas se o filtro for equivalência sobre pk, é melhor fazê-lo antes da junção
  - Pois o filtro acessa diretamente o registro que satisfaz o filtro



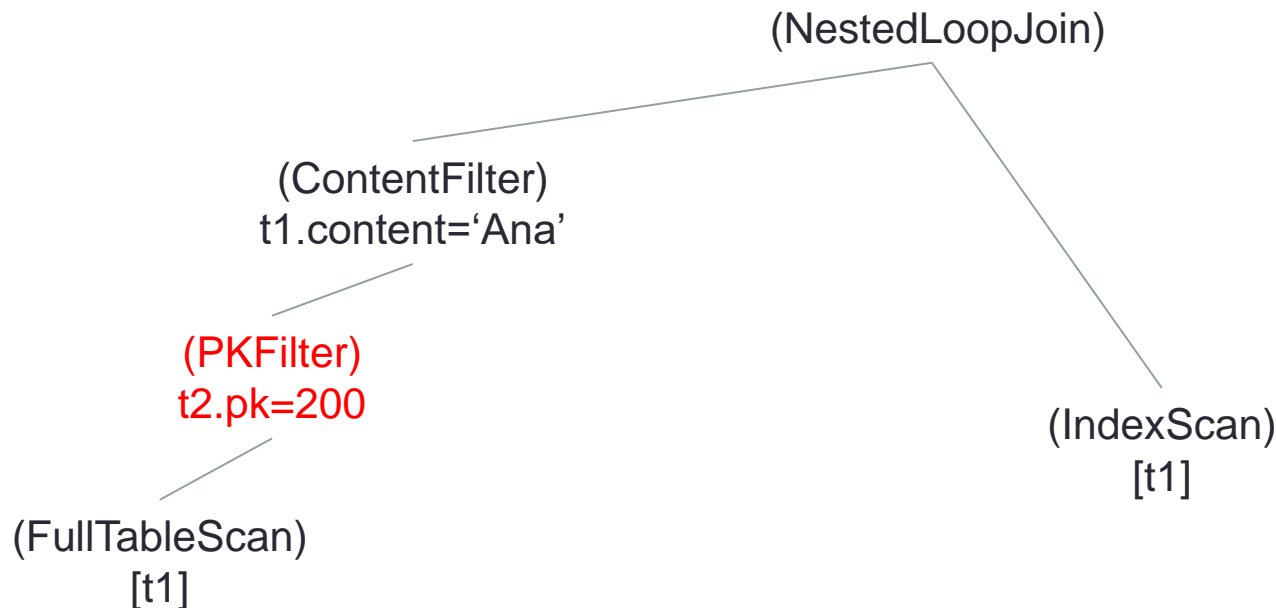
# Exemplos

- Se o filtro for referente ao lado esquerdo da junção, não há problema em fazer o filtro antes da junção.
  - O Join precisa acessar todos os registros que chegam pela esquerda
  - É até melhor que o filtro aconteça antes



# Exemplos

- Mas se houver um PKFilter por equivalência, é melhor que ele seja realizado antes de outros filtros existentes
  - Para poder usar a habilidade do IndexScan em responder consultas por igualdade sobre PK



# Objetivo do trabalho

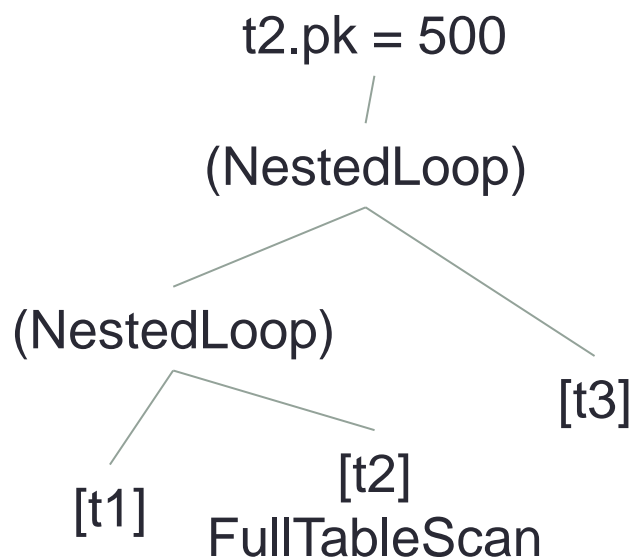
- O objetivo do trabalho é criar um otimizador de consulta
- A classe que implementa o otimizador deve se chamar XXXQueryOptimizer, onde XXX é o nome do aluno
- Pacote da classe: ibd.query.optimizer

# Regras

- As seguintes regras devem ser observadas
  - **Regra 1:** Todo IndexScan deve ser acessado por um PKFilter por igualdade, caso haja um na consulta
  - **Regra 2:** Caso contrário, se o IndexScan estiver do lado direito de um NestedLoopJoin, a junção deve acessar diretamente o IndexScan
  - **Regra 3:** Não se pode trocar a ordem das junções
- Os casos omissos podem ser tratados de qualquer forma

# Exemplos

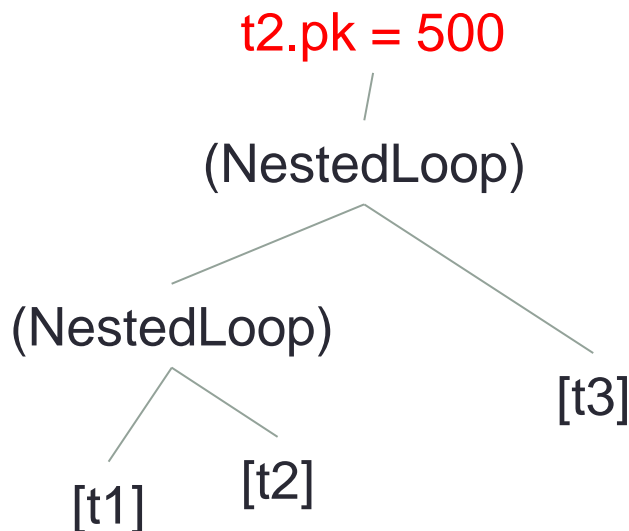
- Nos exemplos a seguir, suponha que o IndexScan seja usado, a menos que seja indicado FullTableScan
  - Por exemplo, na árvore abaixo, t1 e t2 são acessados via IndexScan



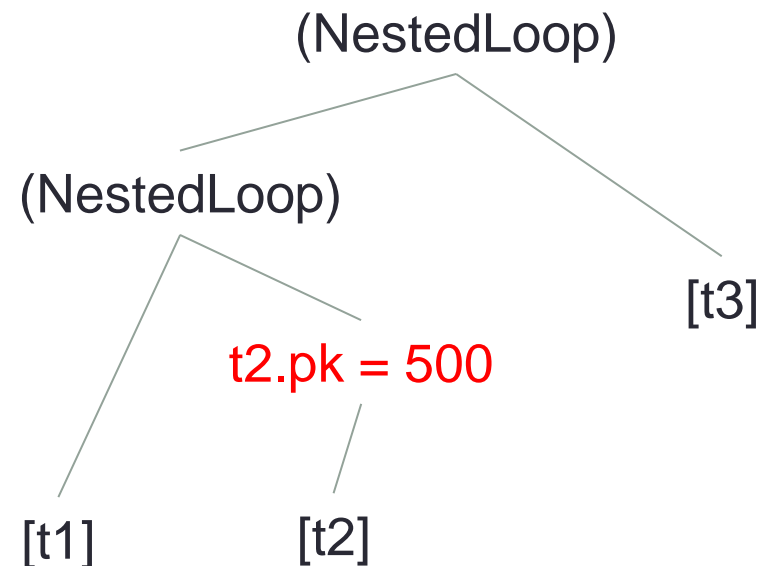
# Regra 1 – PKFilter por igualdade

- Todo IndexScan deve ser acessado por um PKFilter por igualdade, caso haja um na consulta
  - Na versão otimizada, o pk=500 foi movido para perto de t2

**Versão original**



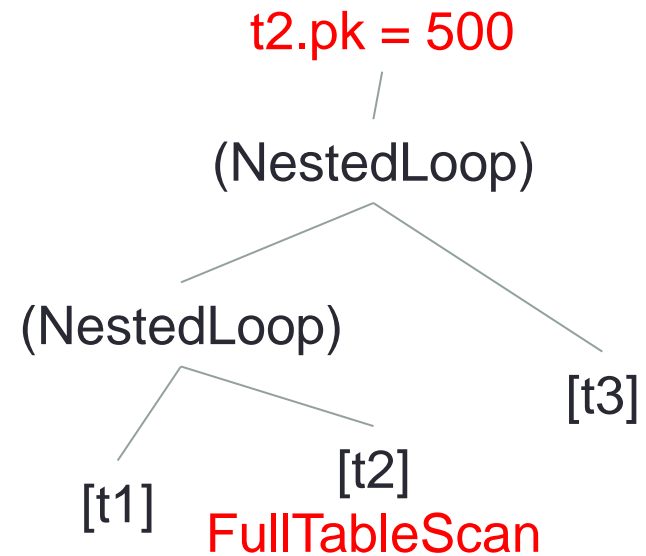
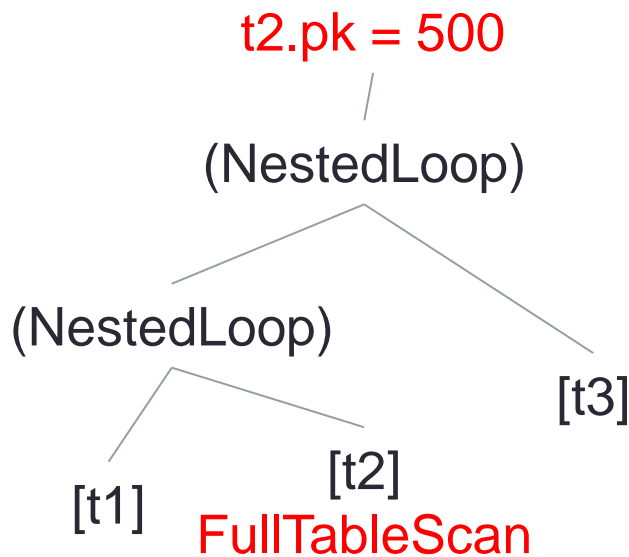
**Versão otimizada**



# Regra 1 – PKFilter por igualdade

- Aqui o filtro não foi movido
  - porque o T2 usa um FullTableScan em vez de um IndexScan

## Versão original

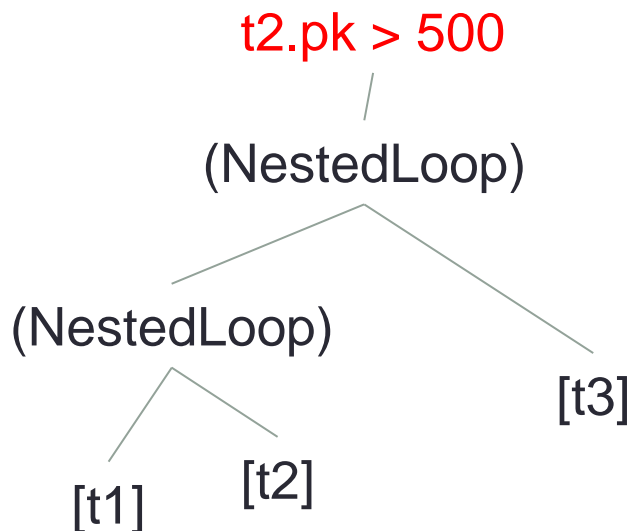




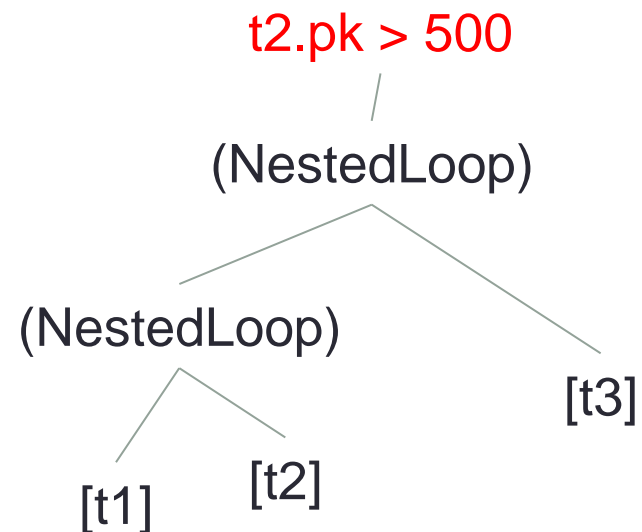
# Regra 1 – PKFilter por igualdade

- Aqui o filtro não precisa ser movido, pois não é por igualdade sobre PK
  - Até poderia, mas para este trabalho essa otimização não é necessária

**Versão original**



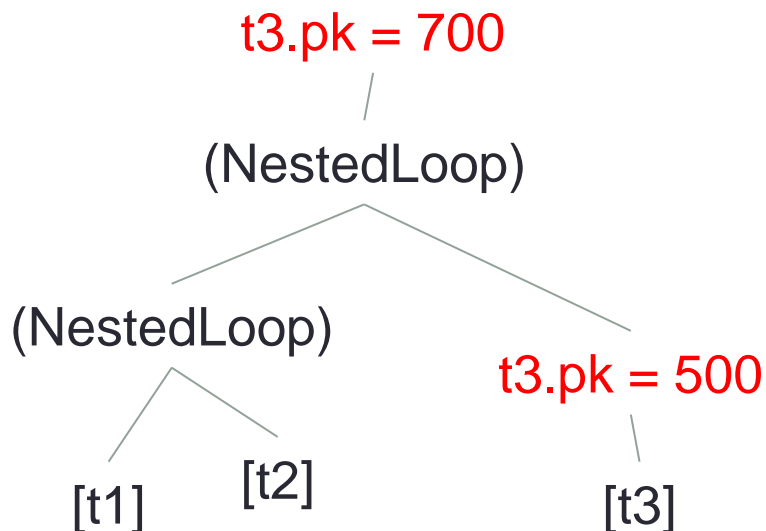
**Versão otimizada**



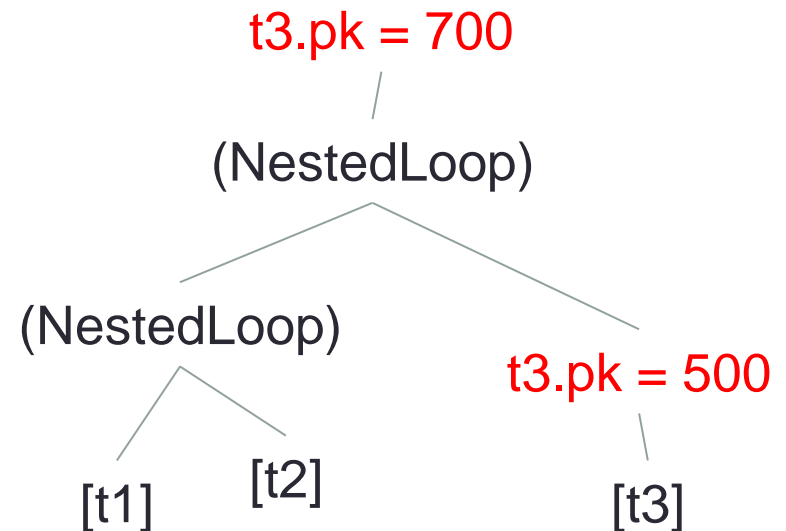
# Regra 1 – PKFilter por igualdade

- Como já há um filtro acima de T3, o outro filtro não precisa ser movido.
  - Até poderia, mas para este trabalho não é necessário

**Versão original**



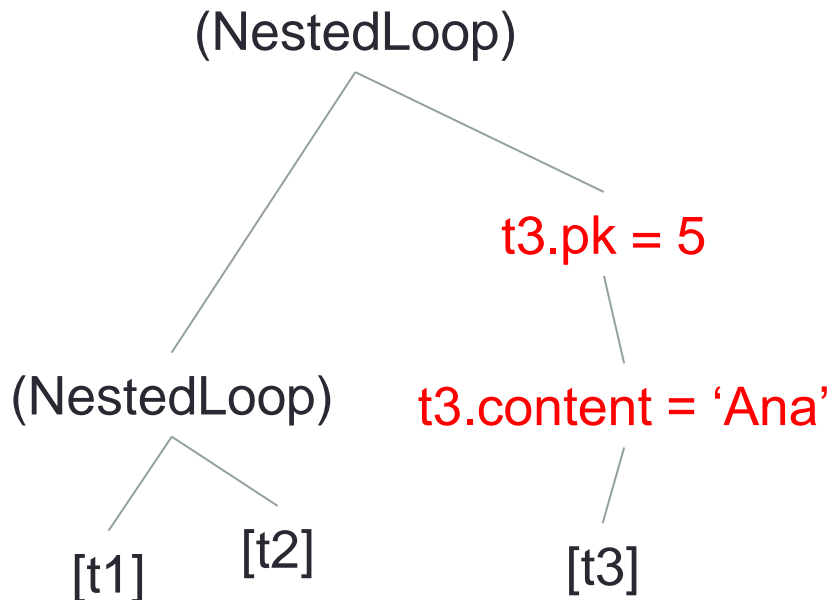
**Versão otimizada**



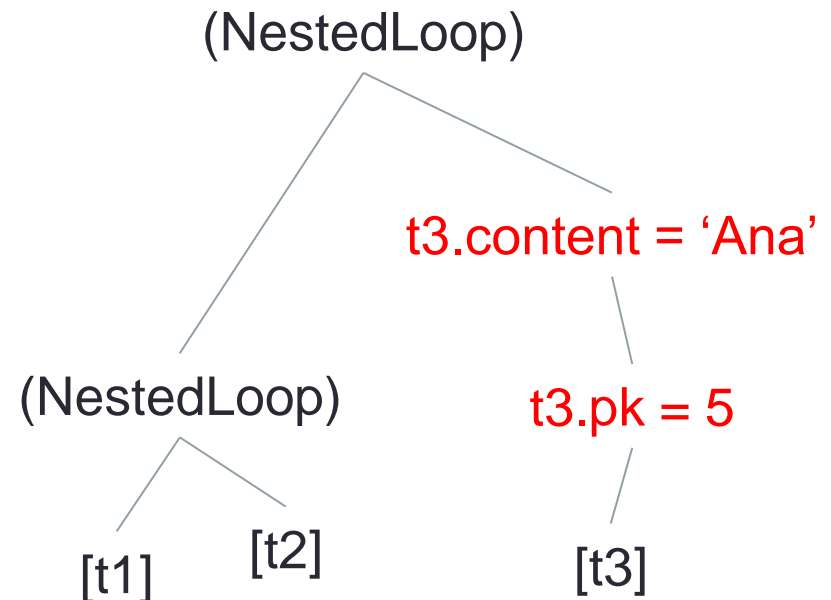
# Regra 1 – PKFilter por igualdade

- Aqui, o filtro mais próximo não é equivalência sobre PK.
  - Foi necessário jogar o filtro sobre PK para baixo de modo a fazê-lo explorar o índice.

## Versão original



## Versão otimizada



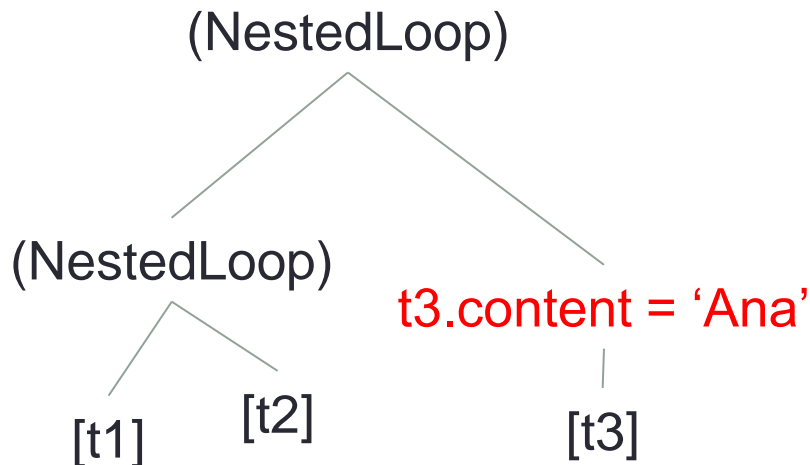
# Regras

- As seguintes regras devem ser observadas
  - **Regra 1:** Todo IndexScan deve ser acessado por um PKFilter por igualdade, caso haja um na consulta
  - **Regra 2:** Caso contrário, se o IndexScan estiver do lado direito de um NestedLoopJoin, a junção deve acessar diretamente o IndexScan
  - **Regra 3:** Não se pode trocar a ordem das junções
- Os casos omissos podem ser tratados de qualquer forma

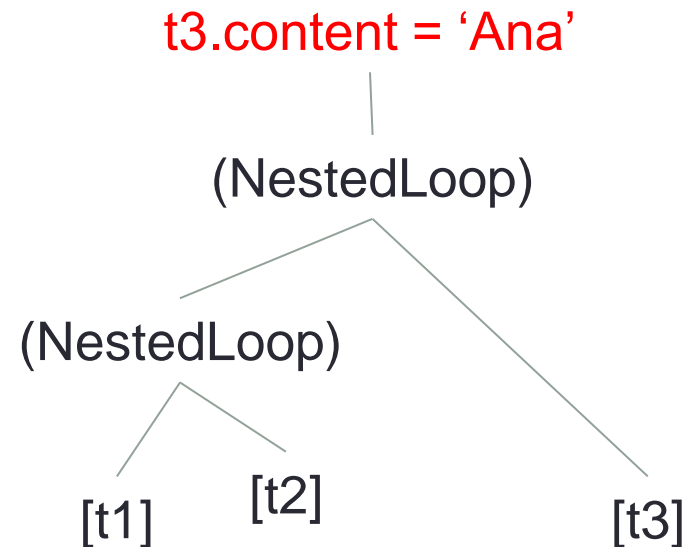
# Regra 2 - NestedLoopJoin

- O IndexScan sobre t3 está do lado direito de um NestedLoopJoin e não há filtro de igualdade sobre pk a ser aplicado. Nesse caso, a junção deve acessar diretamente o IndexScan
  - Para isso, o filtro sobre conteúdo precisou ser movido para cima

## Versão original



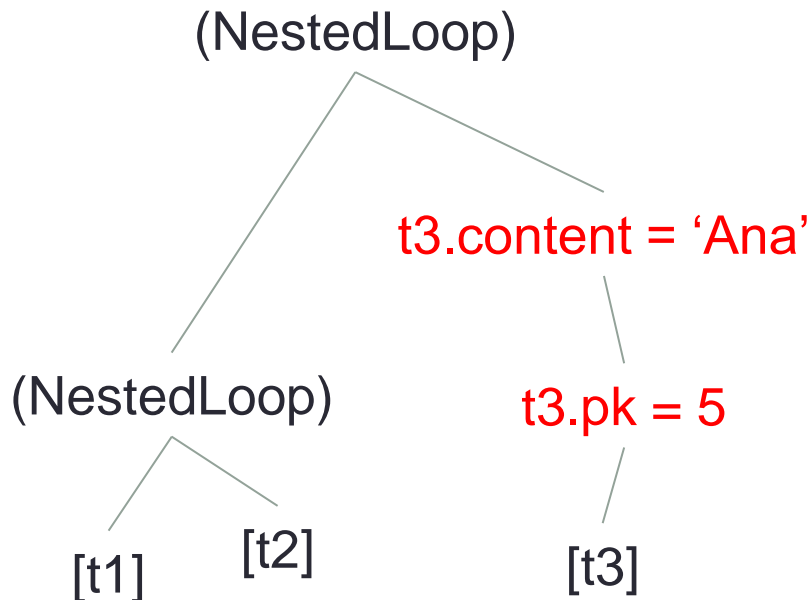
## Versão otimizada



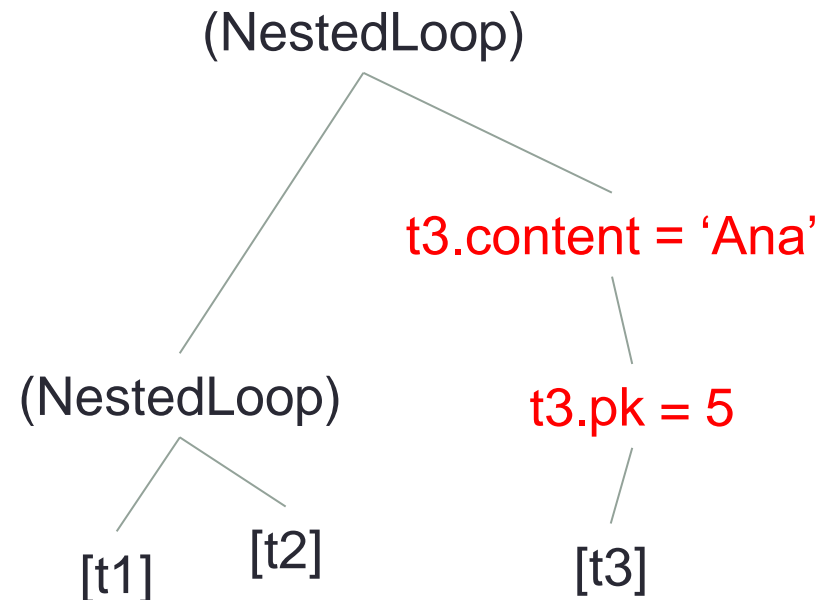
# Regra 2 - NestedLoopJoin

- Neste outro caso, não foi preciso modificar a árvore
  - pois o indexScan em t3 já está sendo usado para realizado um filtro por PK

**Versão original**



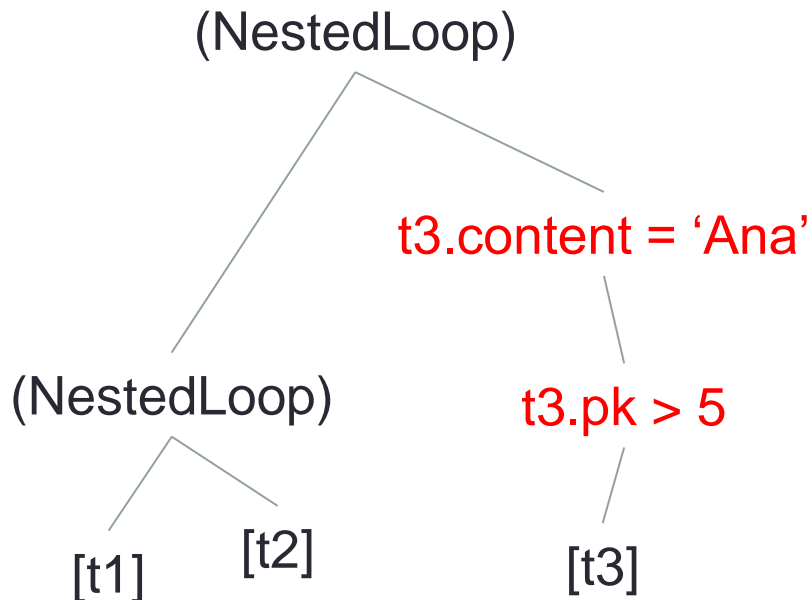
**Versão otimizada**



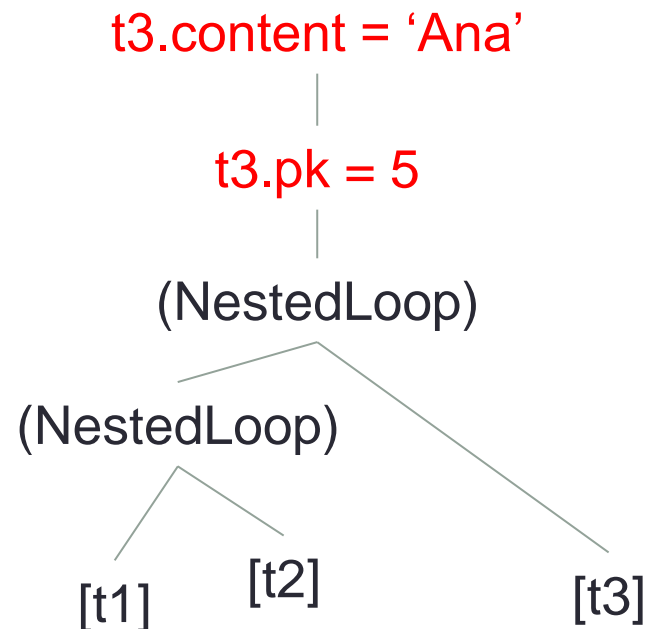
# Regra 2 - NestedLoopJoin

- Aqui foi necessário mover os dois filtros, pois nenhum é pk por igualdade
  - Agora o NestedLoop acessa diretamente o IndexScan em t3

**Versão original**



**Versão otimizada**

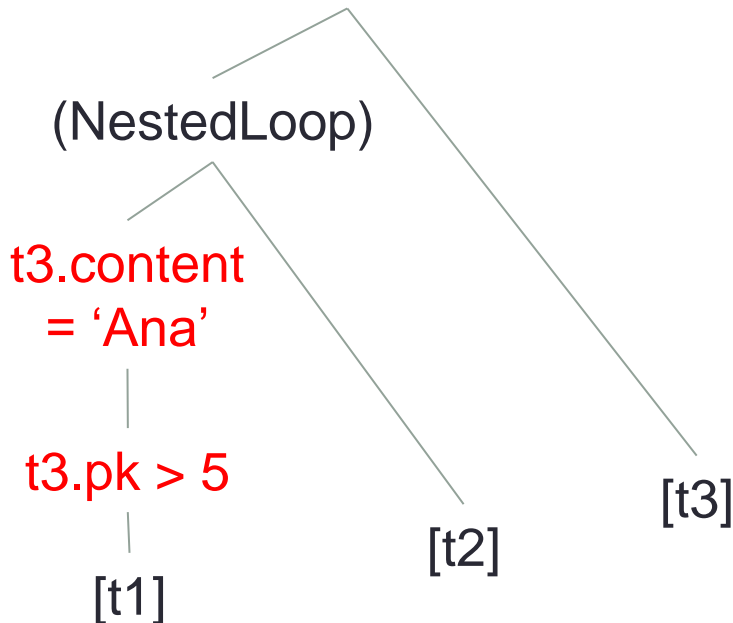


# Regra 2 - NestedLoopJoin

- Nesse caso, não foi necessário mover os dois filtros
  - pois o IndexScan T1 não está à direita de nenhuma operação de NestedLoopJoin

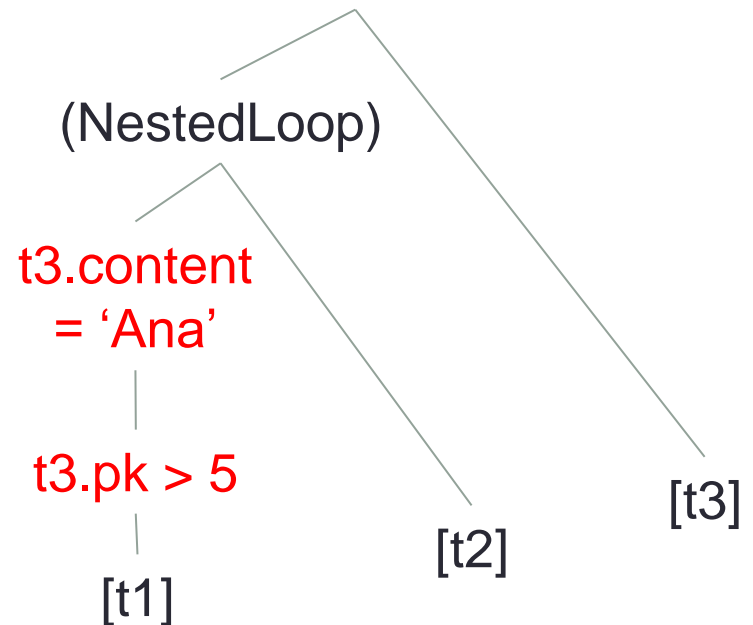
## Versão original

(NestedLoop)



## Versão otimizada

(NestedLoop)





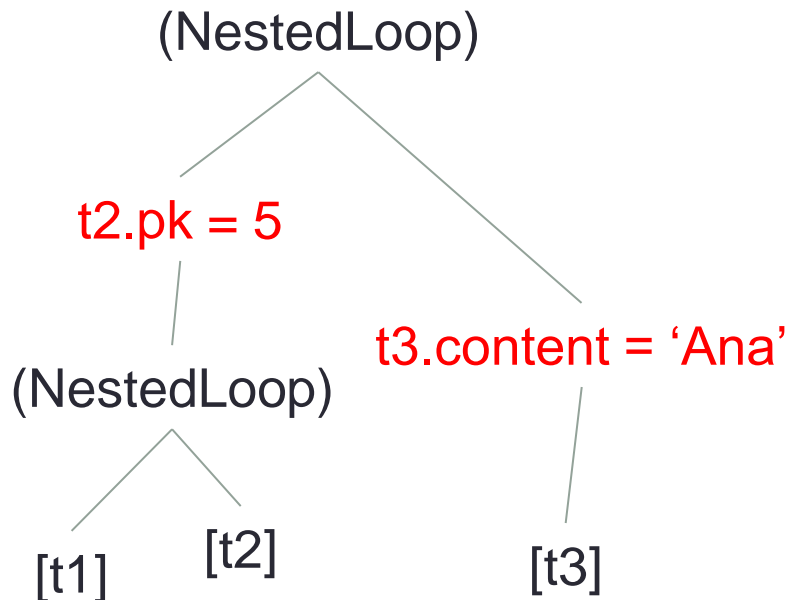
# Regras

- As seguintes regras devem ser observadas
  - **Regra 1:** Todo IndexScan deve ser acessado por um PKFilter por igualdade, caso haja um na consulta
  - **Regra 2:** Caso contrário, se o IndexScan estiver do lado direito de um NestedLoopJoin, a junção deve acessar diretamente o IndexScan
  - **Regra 3:** Não se pode trocar a ordem das junções
- Os casos omissos podem ser tratados de qualquer forma

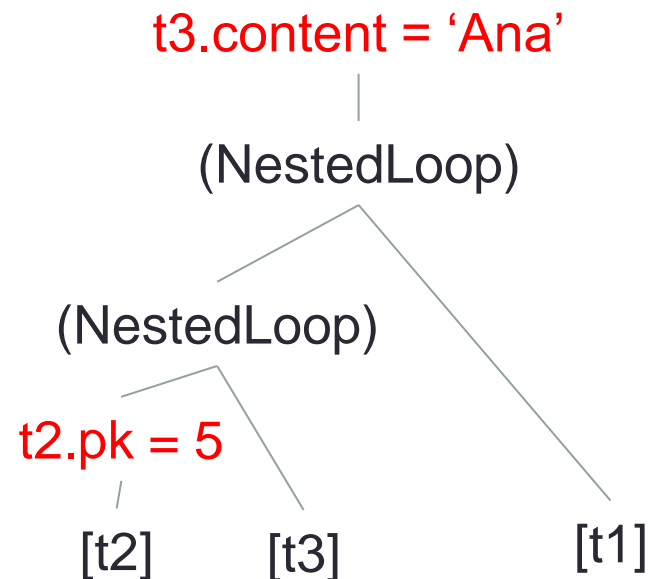
# Regra 3: ordem das junções

- A versão otimizada mexeu na ordem das junções.
- Apesar de neste caso a mudança gerar um plano melhor, alterações desse tipo não devem ser feitas

**Versão original**



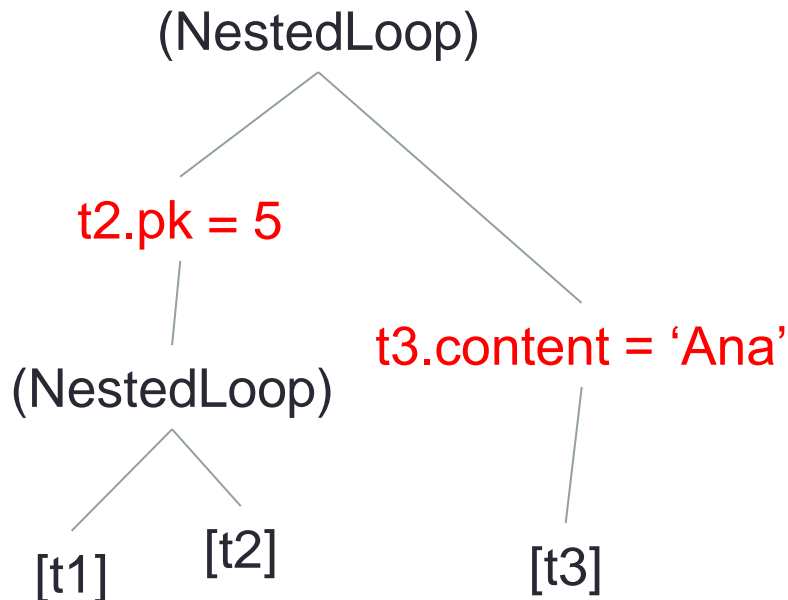
**Versão otimizada da forma não desejada**



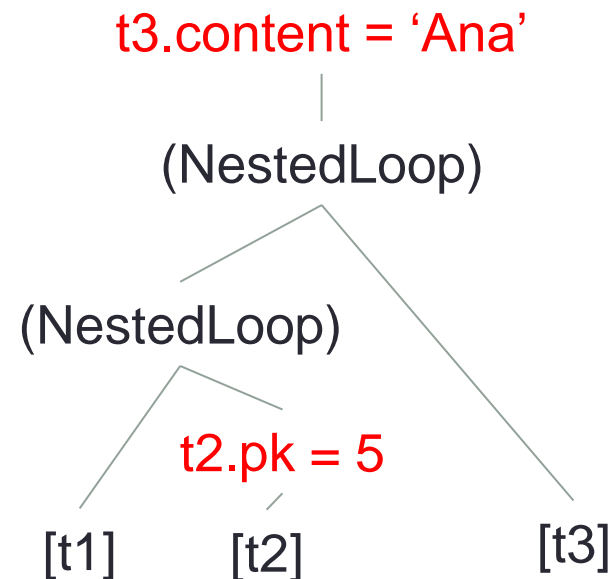
# Não mexa na ordem das junções

- Essa seria a otimização correta, que obedece as regras estipuladas para este trabalho

**Versão original**



**Versão otimizada**



# Implementação

- A classe deve estender **QueryOptimizer** e implementar a função **optimize**
- Tanto a entrada como a saída da função correspondem à operação de mais alto nível da árvore ( a raiz)
- Durante o uso, será necessário recuperar a saída da função de otimização, pois o otimizador pode alterar a raiz da árvore

```
public class XXXQueryOptimizer implements QueryOptimizer{  
  
    public Operation optimize( Operation op) {  
  
        //coloque o código aqui  
        ...  
  
        return ...; // o retorno deve ser a operação raiz da árvore  
    }  
}
```

# Exemplo de uso

- A classe `ibd.query.optimizer.Main` pode ser usada para testes

```
Main m = new Main();
QueryOptimizer opt = new XXXQueryOptimizer();

//cria as tabelas que serão usadas nos testes
createTable("c:\\teste\\ibd", "t1", Table.DEFAULT_PAGE_SIZE, 100, false, 2, 50);
createTable("c:\\teste\\ibd", "t2", Table.DEFAULT_PAGE_SIZE, 100, false, 3, 50);

//executa um teste
m.testOptimization(opt,m.createQuery1(), true, true);
```

# Exemplo de uso

- Função de teste: **testOptimization()**

## Parâmetros

**opt**: otimizador usado

**query**: a consulta a ser otimizada

**showTree** : exibição da árvore das consultas original e otimizada

**runQuery** : execução da consulta

```
Main m = new Main();  
QueryOptimizer opt = new
```

```
//cria as tabelas que serão usadas nos testes  
createTable("c:\\teste\\ibd", "t1", Table.DEFAULT_PAGE_SIZE, 100, false, 2, 50);  
createTable("c:\\teste\\ibd", "t2", Table.DEFAULT_PAGE_SIZE, 100, false, 3, 50);
```

```
//executa um teste  
m.testOptimization(opt,m.createQuery1(), true, true);
```

# Exemplo de uso

- A classe já traz duas consultas prontas
- Crie outras para aumentar a cobertura dos casos de teste

```
Main m = new Main();
QueryOptimizer opt = new XXX()

//cria as tabelas que serão usadas
createTable("c:\\teste\\ibd", "t1",
createTable("c:\\teste\\ibd", "t2",
```


```
private Operation createQuery1() throws Exception{

    Table table1 = Directory.getTable("c:\\teste\\ibd", "t1", D
    Table table2 = Directory.getTable("c:\\teste\\ibd", "t2", D

    Operation scan1 = new TableScan("t1", table1);
    Operation scan2 = new TableScan("t2", table2);

    Operation join1 = new NestedLoopJoin(scan1, scan2);
    Operation filter1 = new PKFilter(join1, "t1", Comparison
    return filter1;
}
```

```
//executa um teste
m.testOptimization(opt, m.createQuery1(), true, true);
```



# Exemplo de uso

- Certifique-se de que as tabelas usadas na consulta tenham sido previamente criadas

```
Main m = new Main();  
QueryOptimizer opt = new XXX()  
  
//cria as tabelas que serão usadas  
createTable("c:\\teste\\ibd", "t1",  
createTable("c:\\teste\\ibd", "t2",
```

```
private Operation createQuery1() throws Exception{  
  
    Table table1 = Directory.getTable("c:\\teste\\ibd", "t1", D  
    Table table2 = Directory.getTable("c:\\teste\\ibd", "t2", D  
  
    Operation scan1 = new TableScan("t1", table1);  
    Operation scan2 = new TableScan("t2", table2);  
  
    Operation join1 = new NestedLoopJoin(scan1, scan2);  
    Operation filter1 = new PKFilter(join1, "t1", Comparison  
    return filter1;  
}
```

```
//executa um teste  
m.testOptimization(opt, m.createQuery1(), true, true);
```



# Formas de Verificação

- Tipos de verificação
  - Formato da árvore (forma principal de verificação)
  - Quantidade de blocos carregados
  - Registros gerados
- Essas três verificações já estão disponibilizadas pela função de teste

# Formato da árvore

- A função `Utils.toString()` do pacote `ibd.query` imprime a árvore a partir de uma operação.
  - Essa função pode ser usada para verificar o resultado de uma otimização

```
...
```

```
//imprime a árvore referente à consulta original  
Utils.toString(query, 0);
```

```
query = opt.optimize(query);
```

```
//imprime a árvore referente à consulta otimizada  
Utils.toString(query, 0);
```

# Quantidade de blocos carregados

- A quantidade de blocos carregados também pode ser comparada
  - Em caso de otimização, a quantidade de blocos carregados deve ser reduzido

```
Params.BLOCKS_LOADED = 0;  
  
...  
query = opt.optimize(query);  
  
query.open();  
Iterator<Tuple> it = query.run();  
while (it.hasNext()){  
    Tuple r = it.next();  
    System.out.println(r);  
}  
System.out.println("blocks loaded " + Params.BLOCKS_LOADED);
```

# Registros gerados

- Também pode-se comparar os registros gerados
  - A versão original e a otimizada devem produzir os mesmos registros

```
...
query = opt.pushDownFilters(query);

query.open();
Iterator<Tuple> it = query.run();
while (it.hasNext()){
    Tuple r = it.next();
    System.out.println(r);
}
query.close();
```

# Dicas

- Funções/comandos que podem ser úteis
  - `UnaryOperation.getChildOperation()`
    - Recupera a operação de entrada de uma `UnaryOperation`
  - `UnaryOperation.setChildOperation()`
    - Atribui a operação de entrada de uma `UnaryOperation`

# Dicas

- Funções/comandos que podem ser úteis
  - `BinaryOperation.getLeftOperation()`
    - Recupera a operação de entrada da esquerda de uma `BinaryOperation`
  - `BinaryOperation.getRightOperation()`
    - Recupera a operação de entrada da direita de uma `BinaryOperation`
  - `BinaryOperation.setLeftOperation()`
    - Atribui a operação de entrada da esquerda de uma `BinaryOperation`
  - `BinaryOperation.setRightOperation()`
    - Atribui a operação de entrada da direita de uma `BinaryOperation`

# Dicas

- Funções/comandos que podem ser úteis
  - PKFilter.getDataSourceAlias()
    - Recupera o nome da fonte de dados usada pela operação de filtragem
  - PKFilter.getComparisonType()
    - Recupera o valor usado na filtragem
- Essas funções recuperam informações de um filtro
  - Útil para decidir o que fazer quando um filtro for encontrado
- Obs.
  - Os tipos de comparação estão descritos em `ibd.table.ComparisonTypes`

# Dicas

- Funções/comandos que podem ser úteis
  - `Operation.getParentOperation()`
    - Recupera a operação pai de uma `Operation`
  - `instanceof`
    - Comando do Java que recupera o tipo de uma classe



# Dicas

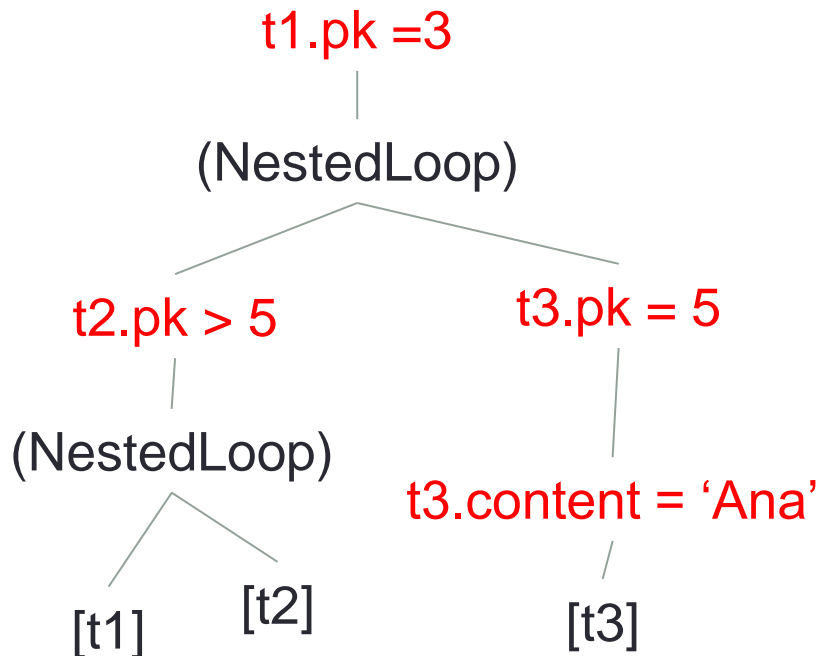
- Algumas dicas podem ser obtidas a partir do código disponibilizado
  - Ex. A classe `ibd.query.Utills` mostra um exemplo de como o comando `instanceOf` pode ser utilizado

```
public static void toString(Operation op, int tab){  
  
    ...  
  
    if (op instanceof BinaryOperation){  
        BinaryOperation bop = (BinaryOperation) op;  
        toString(bop.getLeftOperation(), tab+4);  
        toString(bop.getRightOperation(), tab+4);  
    }  
  
    ...  
}
```

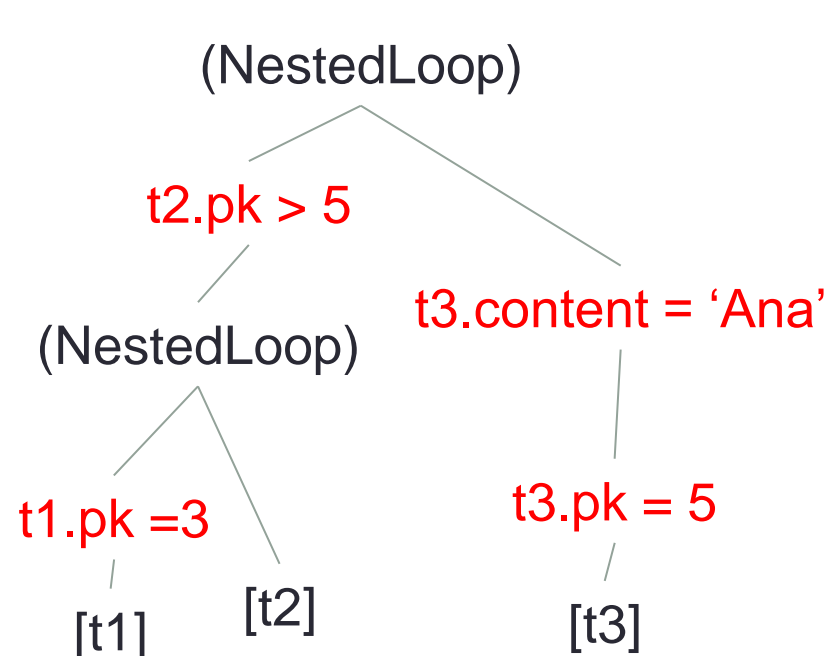
# Avaliação

- Teste consultas complexas, com diversas combinações de junções e filtros

## Versão original



## Versão otimizada



# Avaliação

- Operações que podem ser usadas nos testes:
  - IndexScan
  - FullTableScan
  - PKFilter
  - ContentFilter
  - NestedLoopJoin
- As demais operações não precisam ser testadas

# Importante

- Entrega pelo moodle
- Não entregue o projeto inteiro
  - Apenas a classe java solicitada
- O trabalho é **individual**
  - O compartilhamento de código entre alunos leva à anulação da nota
  - '
- **Obs:** Use o código-fonte disponibilizado no moodle no tópico de divulgação do trabalho
  - É a versão mais recente

# Entrega

- A nota máxima possível depende do dia em que for feita a entrega

Prazo	Nota máxima
20/10 23h59min (sexta)	100%
21/10 23h59min (sábado)	80%
22/10 23h59min (domingo)	60%
23/10 23h59min (segunda)	40%

- Entregas feitas após o dia 23/10 não serão avaliadas