TRABALHO PRÁTICO 2

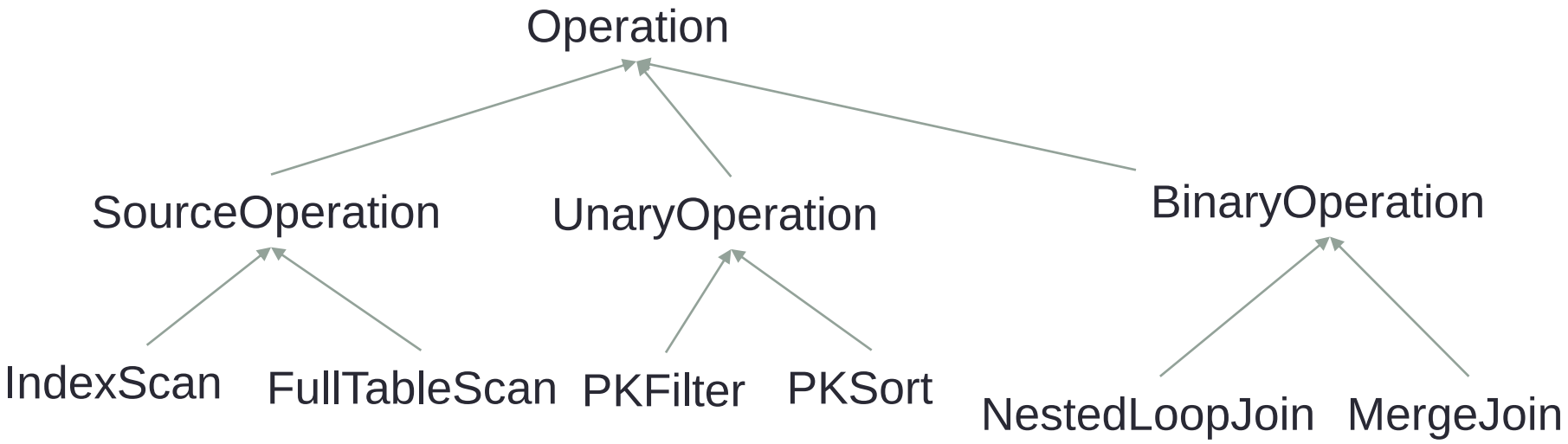
Sérgio Mergen

Código Fonte

- O SGBD criado para a disciplina tem suporte a algumas operações de acesso aos dados
  - Varredura de tabela
  - Filtro
  - Ordenação
  - Junção
  - ...
- Limitação: Na versão atual os registros são pares de <pk, conteúdo>
  - Dessa forma, as operação são limitadas a esses dois campos

Hierarquia de Operações

- A hierarquia abaixo mostra algumas operações para os três tipos básicos de operação



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 FullTableScan

- É uma operação **folha** que provê acesso direto aos registros de uma tabela
  - Assim como um IndexScan
  - Porém, é menos eficiente que um IndexScan na busca por equivalência de pk
    - Foi criado apenas para fins de demonstração

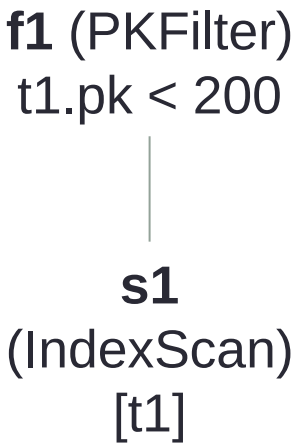
```
Operation scan = new FullableScan(“t1”, table);
scan.open();
Iterator<Tuple> it = scan.run();
while (it.hasNext()){
      Tuple t =  it.next();
      System.out.println(t);
}
```

scan
(FullTableScan)
[t1]

Operação PKFilter

- É uma operação **unária**.
  - Das tuplas que chegam até ela, são removidas as que não satisfaçam à condição de filtragem
- No exemplo abaixo
  - é realizado um filtro que recupera registros(da tabela cujo alias é “t1”) que tenham pk < 200

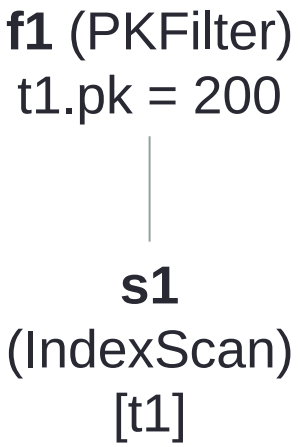
```
Operation s1= new IndexScan (“t1”, table);
Operation f1 = new PKFilter(s1, “t1”,
                            LOWER_THAN, 200L);
f1.open();
Iterator<Tuple> it = f1.run();

while (it.hasNext()){
      …
}
```



Operação PKFilter

- No exemplo abaixo
  - o filtro é por equivalência.
  - Nesse caso, o IndexScan consegue encontrar o registro requisitado pelo PKFilter de forma eficiente.

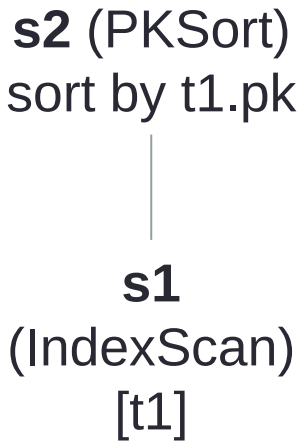
```
Operation s1= new IndexScan (“t1”, table);
Operation f1 = new PKFilter(s1, “t1”,
                                EQUAL, 200L);
f1.open();
Iterator<Tuple> it = f1.run();

while (it.hasNext()){
      …
}
```



Operação PKSort

- É uma operação **unária**.
  - Ordena as tuplas que chegam até ela
  - Usa materialização em memória
- No exemplo abaixo
  - as tuplas são ordenadas com base na PK dos registros que vêm da tabela cujo alias é “t1”

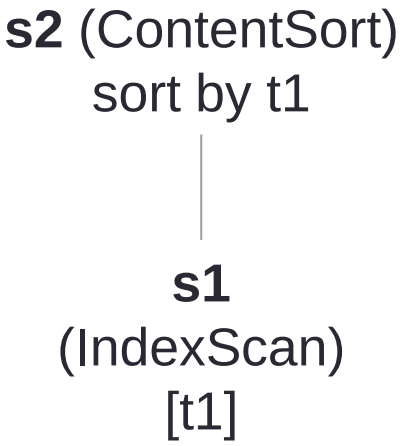
```
Operation s1= new IndexScan (“t1”, table);
Operation s2 = new PKSort(s1,”t1”);
s2.open();
Iterator<Tuple> it = s2.run();

while (it.hasNext()){
      …
}
```



Operação ContentSort

- É uma operação **unária**.
  - Ordena as tuplas que chegam até ela
  - Usa materialização em memória
- No exemplo abaixo
  - as tuplas são ordenadas com base no conteúdo dos registros que vêm da tabela cujo alias é “t1”

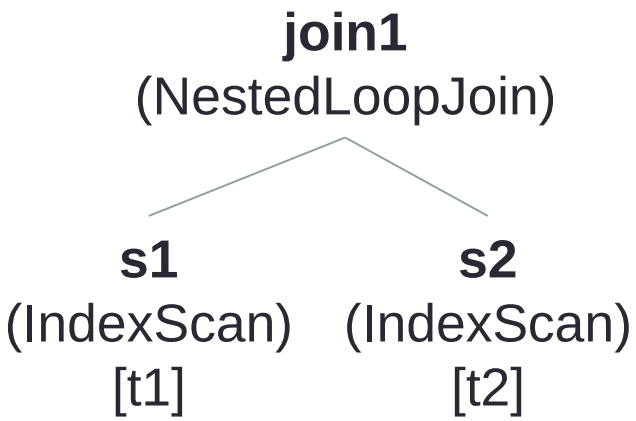
```
Operation s1= new IndexScan (“t1”, table);
Operation s2 = new ContentSort(s1,”t1”);
s2.open();
Iterator<Tuple> it = s2.run();

while (it.hasNext()){
      …
}
```



Operação NestedLoopJoin

- É uma operação **binária**.
  - Realiza a junção com base em igualdade de chave primária

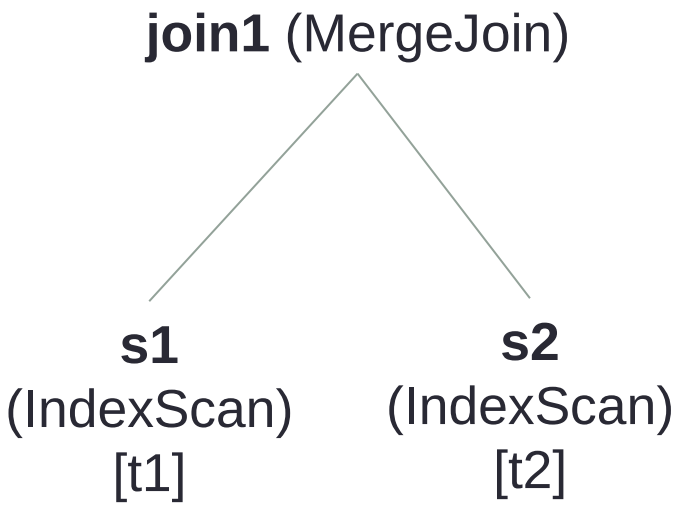
```
Operation s1 = new IndexScan("t1", table1);
Operation s2 = new IndexScan("t2", table2);
Operation join1 = new
            NestedLoopJoin(s1, s2);

join1.open();
Iterator<Tuple> it = scan.run();

while (it.hasNext()){
    …
}
```



Operação MergeJoin

- É uma operação **binária**
  - Os dados já precisam estar ordenados
  - Pode ser usado diretamente via **TableScan** ou **IndexScan** se os dados já vierem ordenados

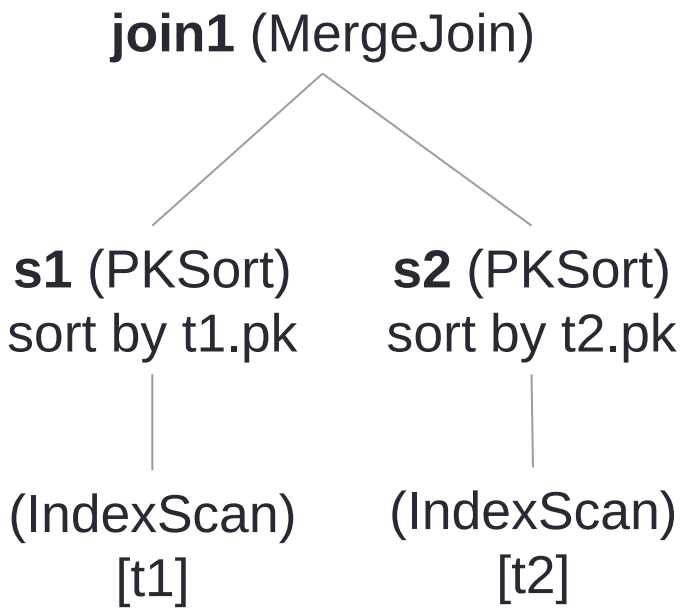
```
Operation s1 = new IndexScan("t1", table1);

Operation s2 = new IndexScan("t2", table2);

Operation join1 = new
            MergeJoin(s1, s2);
```



Operação MergeJoin

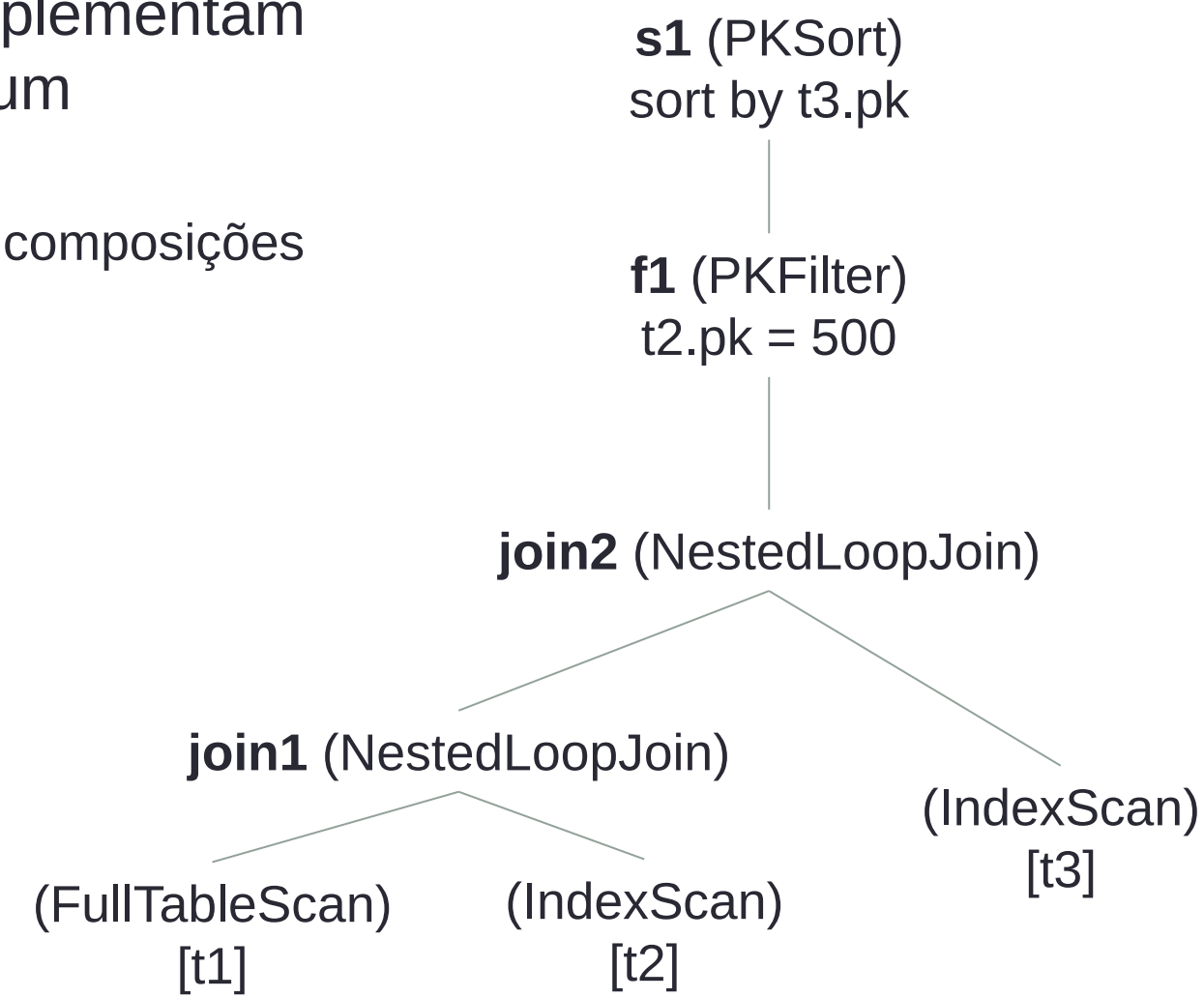
- No exemplo abaixo, supõe-se que os dados não estejam ordenados
  - Nesse caso, deve-se usar o **PKSort** antes do MergeJoin

```
Operation s1 = new PKSort (
            new IndexScan("t1",table1),"t1");

Operation s2 = new PKSort (
            new IndexScan("t2",table2),"t2");

Operation join1 = new
            MergeJoin(s1, s2);
```



Composições de Operações

- As operações implementam uma classe comum (Operation)
  - Isso permite criar composições complexas



Objetivo do trabalho

- O objetivo do trabalho é criar uma operação **unária** de remoção de duplicatas
  - Nome da classe: DuplicateRemovalXXX
    - Onde XXX é o nome do aluno

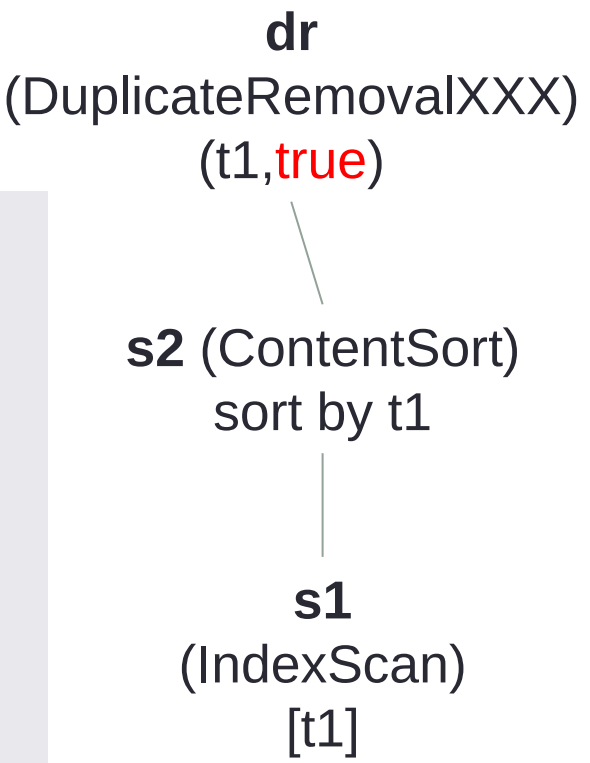
- Das tuplas que chegam até a operação, devem ser removidas aquelas cujo **conteúdo** já foi incluído na resposta

- A operação deve receber um parâmetro que indica se os dados que chegam já estão ordenados
  - Caso estejam, a operação deve eliminar duplicatas conforme elas chegam ao pipeline

Operação DuplicateRemoval

- No exemplo abaixo
  - A operação de remoção de duplicatas pode remover duplicatas conforme processa os registros que vão sendo acessados (solução pelo próprio pipeline)

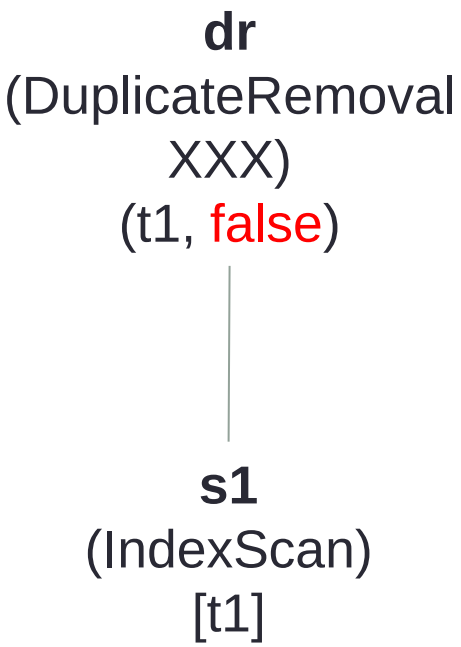
```
Operation s1= new IndexScan (“t1”, table);
Operation s2 = new ContentSort(s1,”t1”);

Operation dr = new DuplicateRemovalXXX(s2, “t1”, true);
dr.open();
Iterator<Tuple> it = dr.run();

while (dr.hasNext()){
      …
}
```



Operação DuplicateRemoval

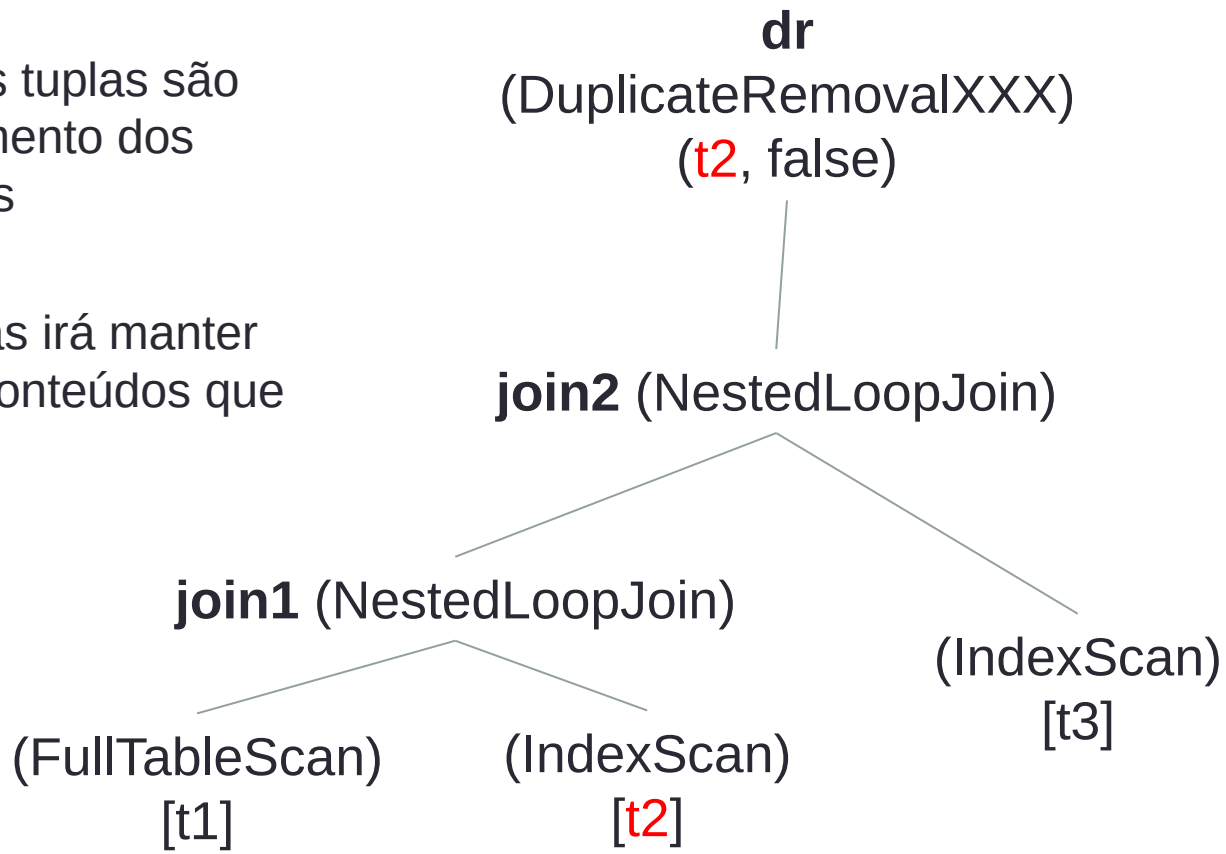
- No exemplo abaixo
  - Como os dados não estão ordenados pelo conteúdo, deve-se encontrar uma forma alternativa para remoção de duplicatas.
    - Essa alternativa necessariamente envolverá algum tipo de materialização

```
Operation s1= new IndexScan (“t1”, table);
Operation dr = new DuplicateRemovalXXX(s1, “t1”, false);
dr.open();
Iterator<Tuple> it = dr.run();

while (dr.hasNext()){
      …
}
```



Operação DuplicateRemoval

- A operação de remoção pode ter que lidar com tuplas que são compostas por vários registros
- No exemplo ao lado, as tuplas são compostas pelo cruzamento dos registros de três tabelas
- A remoção de duplicatas irá manter valores distintos para conteúdos que venham da tabela t2



Dicas de código

- A operação de remoção de duplicatas deve estender de UnaryOperation

- Funções importantes
  - Função open()
    - Prepara a operação para a execução de consultas
    - O uso dessa função depende da lógica implementada
      - Por exemplo, ela pode ser útil para iniciar variáveis
  - Função lookUp()
    - Devolve um iterador que provê acesso às tuplas
  - O iterador deve ser uma instância de uma classe interna que estenda OperationIterator
    - É essa classe interna que percorre as tuplas que entram e decide quais devem ser retornadas

Dicas de código

- Para eliminar duplicatas, deve-se poder acessar o conteúdo de uma tupla:

```
tp.sourceTuples[sourceTupleIndex].record.getContent()
```

- Onde:
  - tp: é uma tupla
  - sourceTupleIndex: é o índice usado para identificar a tabela que será usada para recuperar o conteúdo

- O índice sourceTupleIndex já é calculado pela classe UnaryOperation com base no alias configurado

Dicas de código

- Analise as operações unárias já criadas:
  - **PKHashIndex**: operação de indexação em memória (baseada em materialização)
  - **Filter**: operação de filtragem (baseada em pipeline)
    - Subclasses: PKFilter, ContentFilter
  - **Sorter**: operação de ordenação (baseada em materialização)
    - Subclasses: PKSorter, ContentSorter
-

Testes

- Use a função Utils.createTable() para testes

```
Table table = Utils.createTable("c:\\teste\\ibd","t1",4096,100, false, 1, 100);

...
```

- Parâmetros:
  - Pasta onde está o banco
  - Arquivo onde está o banco
  - Tamanho da página
  - Maior valor possível
  - Flag indicando se registros ficarão desordenados
  - Distância entre um valor e outro
  - Cardinalidade da coluna de conteúdo

Testes

- Exemplo
  - Table table = Utils.createTable("c:\\teste\\ibd","t1", 4096, 8, false, 1,4);

Ordenados e com distância = 1, 4 nomes diferentes

pk	Conteúdo
0	Alexandre
1	Alice
2	Ana
3	Andre
4	Alexandre
5	Alice
6	Ana
7	Andre

Testes

- No exemplo abaixo, cada nome é retornado uma única vez
  - A chave primária pode ser qualquer uma que esteja associada com o nome

Consulta:

dr
(DuplicateRemovalXXX)
(t1,true)

|

s2 (ContentSort)
sort by t1

|

s1
(IndexScan)
[t1]

Dados da tabela t1:

pk	Conteúdo
0	Alexandre
1	Alice
2	Ana
3	Andre
4	Alexandre
5	Alice
6	Ana
7	Andre

Resultado da consulta:

pk	Conteúdo
0	Alexandre
1	Alice
2	Ana
3	Andre

Testes

- Neste outro exemplo
  - estão chegando na operação tuplas compostas por dois registros
  - e a operação deve remover duplicatas do conteúdo referente ao segundo registro (conteúdo 2)

Antes da operação de remoção de duplicatas:

Pk 1	Conteúdo 1	Pk 2	Conteúdo 2
0	Alexandre	0	Ana
4	Alice	4	Laura
8	Ana	8	Rodrigo
12	Andre	12	Laura

Após a operação de remoção de duplicatas:

Pk 1	Conteúdo 1	Pk 2	Conteúdo 2
0	Alexandre	0	Ana
4	Alice	4	Laura
8	Ana	8	Rodrigo

Testes

- Crie diferentes cenários de teste para garantir que a operação funcione corretamente
  - Dados ordenados / dados desordenados
  - Expressões simples / expressões complexas
  - Poucos registros / muitos registros
  - ...

Avaliação

- Caso 1:
  - os dados estão chegando já ordenados pelo conteúdo que se deseja verificar
    - Nesse caso, a resposta deve ser encontrada por meio de **pipeline** (sem materialização)
- Caso 2:
  - os dados não estão chegando ordenados pelo conteúdo que se deseja verificar
    - Nesse caso, a resposta deve ser encontrada por meio de alguma espécie de **materialização**
      - A forma concreta de resolução fica a cargo do aluno
- Cada caso corresponde a 50% da nota

Avaliação

- Os principais aspectos que serão analisados para a avaliação do trabalho são
  - O código deve estar funcional
  - As funções complementares(caso hajam) foram criadas como privadas
  - O arquivo correto foi enviado (.java)
  - O pacote da classe foi especificado da forma correta
    - ibd.query.unaryop
  - O construtor da classe recebe os parâmetros esperados
  - Foi usado o pipeline quando possível
  - A resposta gerada é a correta

Entrega

- 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
- Use a última versão do código disponível no moodle

Entrega

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

Prazo	Nota máxima
06/10 23h59min (sexta)	100%
07/10 23h59min (sábado)	80%
08/10 23h59min (domingo)	60%
09/10 23h59min (segunda)	40%

- Entregas feitas após o dia 09/10 não serão avaliadas