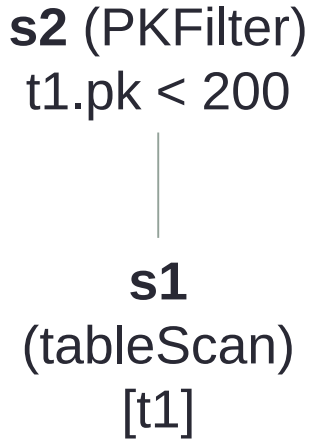
TRABALHO PRÁTICO 3

Operação PKFilter

- É uma operação unária.
  - Das tuplas que chegam até ela, são removidas as que não satisfaçam à condição de filtragem
  - O filtro é sobre a chave primária de uma das fontes que compõe a tupla
    - O nome da fonte deve ser informado

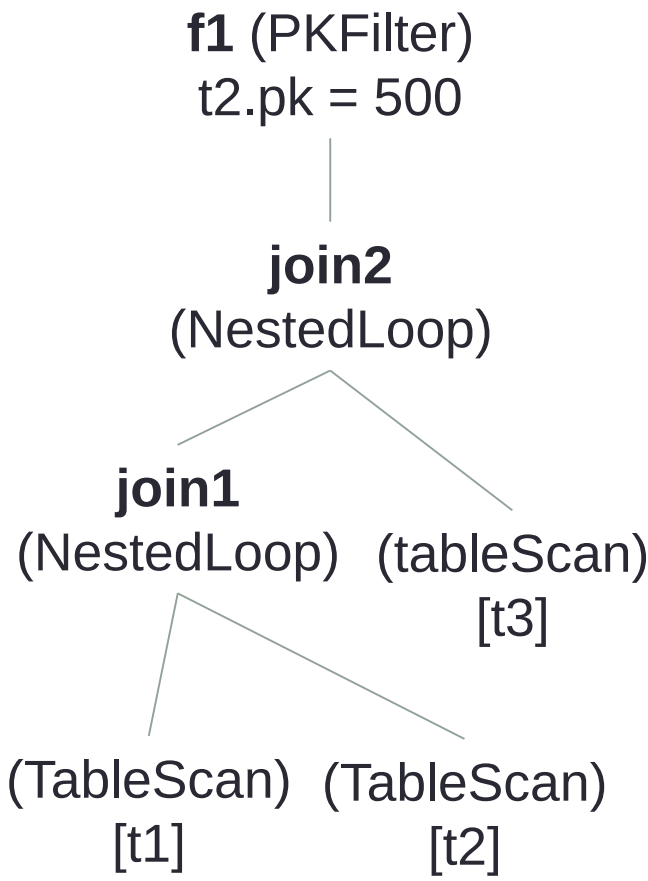
```
Operation s1= new TableScan (“t1”, table);
Operation s2 = new PKFilter(s1, “t1”,
                            LOWER_THAN, 200L);

s2.open();
while (s2.hasNext()){
    …
}
```



Composições de Operações

- Os filtros podem ser combinados em expressões complexas compostas por diversas operações



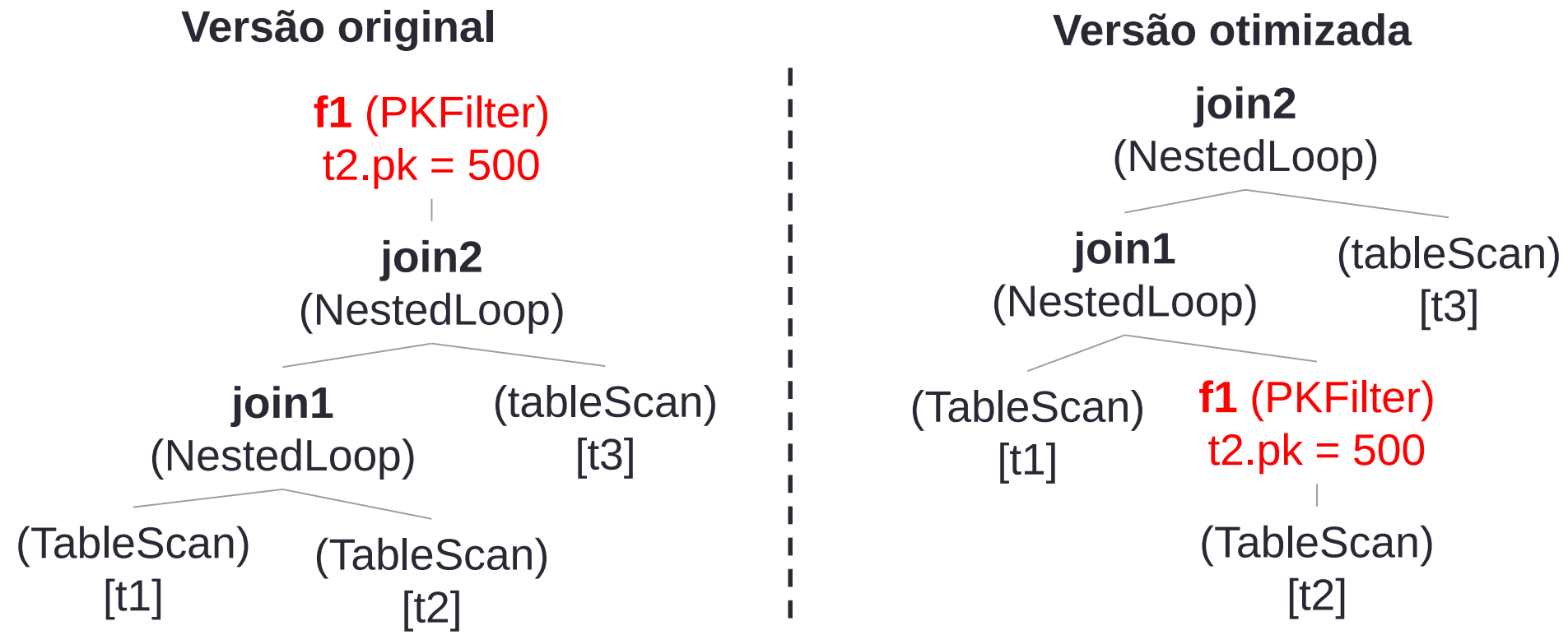

```
TableScan scan1 = new TableScan("t1", table1);
TableScan scan2 = new TableScan("t2", table2);
TableScan scan3 = new TableScan("t3", table3);

Operation join1 = new NestedLoopJoin(scan1, scan2);
Operation join2 = new NestedLoopJoin(join1, scan3);

Operation PKFilter query = new PKFilter(join2,"t2",
                    EQUAL, 500L);
```

Otimização de Consultas

- Como vimos, uma expressão pode ser otimizada movendo as operações de filtragem para baixo
  - de modo que elas sejam realizadas antes das demais operações



Objetivo do trabalho

- O objetivo do trabalho é criar um otimizador de consulta
  - O otimizador deve empurrar as operações de PKFIlter para baixo, de modo que fiquem perto das suas respectivas tabelas
- A classe que implementa o otimizador deve se chamar XXXQueryOptimizer, onde XXX é o nome do aluno
- A função principal desta classe deve obedecer à seguinte assinatura

```
public Operation pushDownFilters(Operation op);
```

- Tanto a entrada como a saída da função correspondem à operação de mais alto nível da árvore

Exemplo de uso

- A otimização pode fazer com que a operação de mais alto nível mude
  - Por isso é necessário recuperar a saída da função de otimização

```
TableScan scan1 = new TableScan(“t1”, table1);
TableScan scan2 = new TableScan(“t2”, table2);
TableScan scan3 = new TableScan(“t3”, table3);

Operation join1 = new NestedLoopJoin(scan1, scan2);
Operation join2 = new NestedLoopJoin(join1, scan3);

Operation PKFilter query = new PKFilter(join2,”t2”, EQUAL, 500L);

XXXQueryOptimizer optimizer = new XXXQueryOptimizer();
query = optimizer.pushDownFiters(query);
```

Formas de Verificação

- A classe ibd.query.Main já traz uma estrutura básica que pode ser usada para testar a implementação

```
...

Main m = new Main();
//coloque o código aqui
m.testPushDownOptimization();

...
```

- Tipos de verificação
  - Formato da árvore
  - Quantidade de blocos carregados
  - Registros gerados

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);

XXXQueryOptimizer optimizer = new xxxQueryOptimizer();
Optimizer.pushDownFilters(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
  - Quando se usa a versão otimizada, a quantidade de blocos carregados tende a se menor em relação à versão original

```
Params.BLOCKS_LOADED = 0;
...
XXXQueryOptimizer optimizer = new xxxQueryOptimizer();
Optimizer.pushDownFilters(query);

query.open();
while (query.hasNext()){
	Tuple r = scan.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

```
Params.BLOCKS_LOADED = 0;
...
XXXQueryOptimizer optimizer = new xxxQueryOptimizer();
Optimizer.pushDownFilters(query);

query.open();
while (query.hasNext()){
	Tuple r = scan.next();
	System.out.println(r);
}
System.out.println("blocks loaded " + Params.BLOCKS_LOADED);
```

Avaliação

- A nota total é dividida de acordo com a tabela abaixo

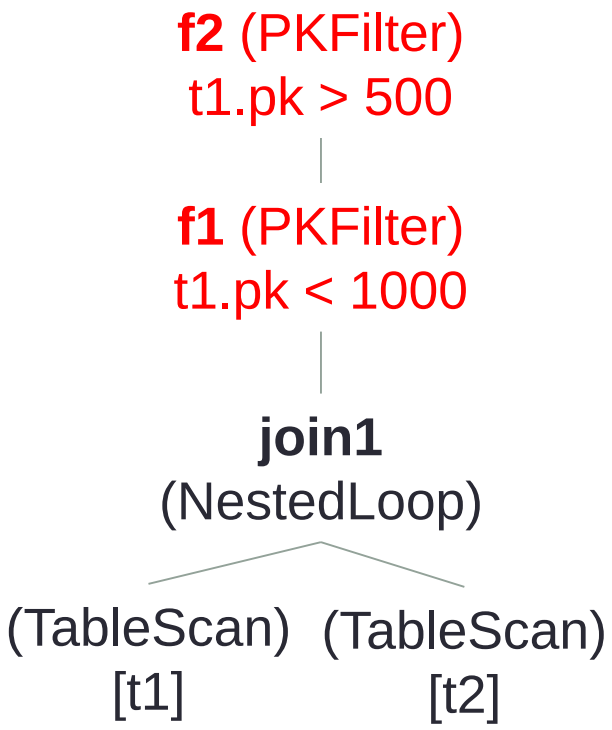
operação	peso
Consultas sem junção e um único filtro	20%
Consultas com uma única junção e um único filtro	20%
Consultas com múltiplas junções e um único filtro	30%
Consultas com múltiplas junções e múltiplos filtros	30%

- Obs. A função de otimização não deve criar novas operações.
  - A ideia é aproveitar as operações existentes, apenas trocando as conexões entre elas

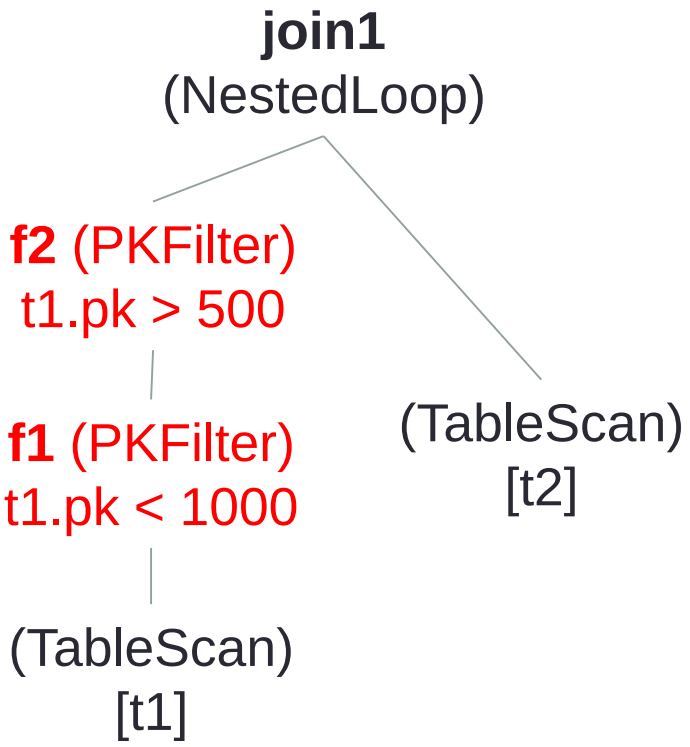
Avaliação

- Teste consultas complexas, com diversas combinações de junções e filtros
  - Exemplo: dois filtros sobre a mesma tabela

Versão original



Versão otimizada



Dicas

- Funções/comandos que podem ser úteis
  - UnaryOperation.getOperation()
    - Recupera a operação de entrada de uma UnaryOperation
  - UnaryOperation.setOperation()
    - 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.getRigthOperation()
    - Recupera a operação de entrada da direta de uma BinaryOperation
  - BinaryOperation.setLeftOperation()
    - Atribui a operação de entrada da esquerda de uma BinaryOperation
  - BinaryOperation.setRigthOperation()
    - Atribui a operação de entrada da direta de uma BinaryOperation

Dicas

- Funções/comandos que podem ser úteis
  - SourceOperation.getSource()
    - Recupera o apelido dado para a fonte de dados referente ao SourceOperation
  - PKFilter.getSourceName()
    - Recupera o apelido dado para a fonte de dados a partir de onde será aplicado o filtro
  - Operation.getParentOperation()
    - Recupera a operação pai de uma Operation
  - instanceof
    - Comando do Java que recupera o tipo de uma classe

Dicas

- A classe ibd.query.Utils 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.getRigthOperation(), tab+4);
     }

     ...
```

Entrega

- Prazo final de entrega, sem descontos
  - Domingo, 1° de agosto às 22:00
- A cada dia de atraso, a nota é decrementada em 25%.
- O que entregar
  - O código fonte da classe criada (.java)
  - A classe deve pertencer ao pacote ibd.query