

# TRABALHO PRÁTICO 2

---

# Sumário

- TableScan
- PKFilterScan
- Definição do Trabalho

# Operação TableScan

- É uma operação que acessa diretamente uma fonte
  - Provê acesso direto aos registros de uma tabela
- É necessário informar
  - Um alias (ex. t1)
  - uma fonte (ex. table)

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

```
scan.open();  
while (scan.hasNext()){  
    Tuple t = scan.next();  
    System.out.println(t);  
}
```

**scan**  
(tableScan)  
[t1]

# Operação TableScan

- A varredura ocorre através do acesso consecutivo aos blocos de uma tabela
  - currentBlock: bloco inicial
  - lastBlock: bloco final
- A configuração do início e fim ocorre dentro de open()
  - Os parâmetros são configurados de modo a percorrer todos os blocos

```
public void open() throws Exception {  
  
    ...  
    currentBlock = 0;  
    lastBlock = Table.BLOCKS_AMOUNT-1;  
    ...  
}
```

# Operação TableScan

Dentro da função next(), o laço while avança registro a registro buscando pelo próximo registro válido que deve ser retornado

```
while (rec == null || !(match(rec))) {  
    currentRecord++;  
    if (currentRecord >= Block.RECORDS_AMOUNT) {  
        currentRecord = 0;  
        {  
            currentBlock++;  
            if (currentBlock > lastBlock) {  
                reachedEnd = true;  
                throw new Exception("No records after this point");  
            }  
        }  
    }  
    Block block = table.getBlock(currentBlock);  
    rec = block.getRecord(currentRecord);  
}
```

# Operação TableScan

O estado da operação de scan é mantido por duas variáveis

- `currentBlock`: o bloco que foi acessado por último
- `currentRecord`: o registro que foi acessado por último

```
while (rec == null || !(match(rec))) {  
    currentRecord++;  
    if (currentRecord >= Block.RECORDS_AMOUNT) {  
        currentRecord = 0;  
        {  
            currentBlock++;  
            if (currentBlock > lastBlock) {  
                reachedEnd = true;  
                throw new Exception("No records after this point");  
            }  
        }  
    }  
    Block block = table.getBlock(currentBlock);  
    rec = block.getRecord(currentRecord);  
}
```

# Operação TableScan


A busca é interrompida caso se tente acessar um bloco que esteja além do último bloco

```
while (rec == null || !(match(rec))) {  
    currentRecord++;  
    if (currentRecord >= Block.RECORDS_AMOUNT) {  
        currentRecord = 0;  
        {  
            currentBlock++;  
            if (currentBlock > lastBlock) {  
                reachedEnd = true;  
                throw new Exception("No records after this point");  
            }  
        }  
    }  
    Block block = table.getBlock(currentBlock);  
    rec = block.getRecord(currentRecord);  
}
```

# Operação TableScan

Para o TableScan, qualquer registro é considerado válido

```
while (rec == null || !(match(rec))) {  
    currentRecord++;  
    if (currentRecord >= Block.RECORDS_AMOUNT) {  
        currentRecord = 0;  
        {  
            currentBlock public boolean match(Record rec){  
            if (currentBlock return true;  
            reachedEnd }  
            throw new Exception("No records after this point");  
        }  
    }  
}  
Block block = table.getBlock(currentBlock);  
rec = block.getRecord(currentRecord);  
}
```

A black arrow originates from the `match(rec)` call in the `while` loop condition and points to the `match` method definition in the `Block` class, which is enclosed in a red rectangular box.



# Sumário

- TableScan
- PKFilterScan
- Definição do Trabalho

# Operação PKFilterScan

- É uma operação que acessa diretamente uma fonte
  - Provê acesso direto aos registros de uma tabela
  - mas só retorna os registros que satisfaçam ao filtro
    - O filtro é sobre a chave primária da tabela

```
Operation scan = new PKFilterScan ("t1", table,  
    GREATER_THAN, 200);
```

```
scan.open();  
while (scan.hasNext()){  
    Tuple t = scan.next();  
    System.out.println(t);  
}
```

**scan**  
(PKFilterScan)  
pk>200  
[t1]

# Operação PKFilterScan

- É uma extensão de TableScan
- Possui dois atributos novos
  - comparisonType: o operador de comparação.
    - Ex. LOWER\_THAN
  - Value: o valor a ser comparado

```
public class PKFilterScan extends TableScan {  
  
    ...  
    int comparisonType;  
    long value;  
  
}
```

# Operação PKFilterScan

- A função match() foi sobrescrita.
- Agora o registro só é considerado válido se ele satisfaz à condição de filtragem

```
@Override  
public boolean match(Record rec){  
    return Utils.match(rec.getPrimaryKey(), value, comparisonType);  
}
```

# Operação PKFilterScan

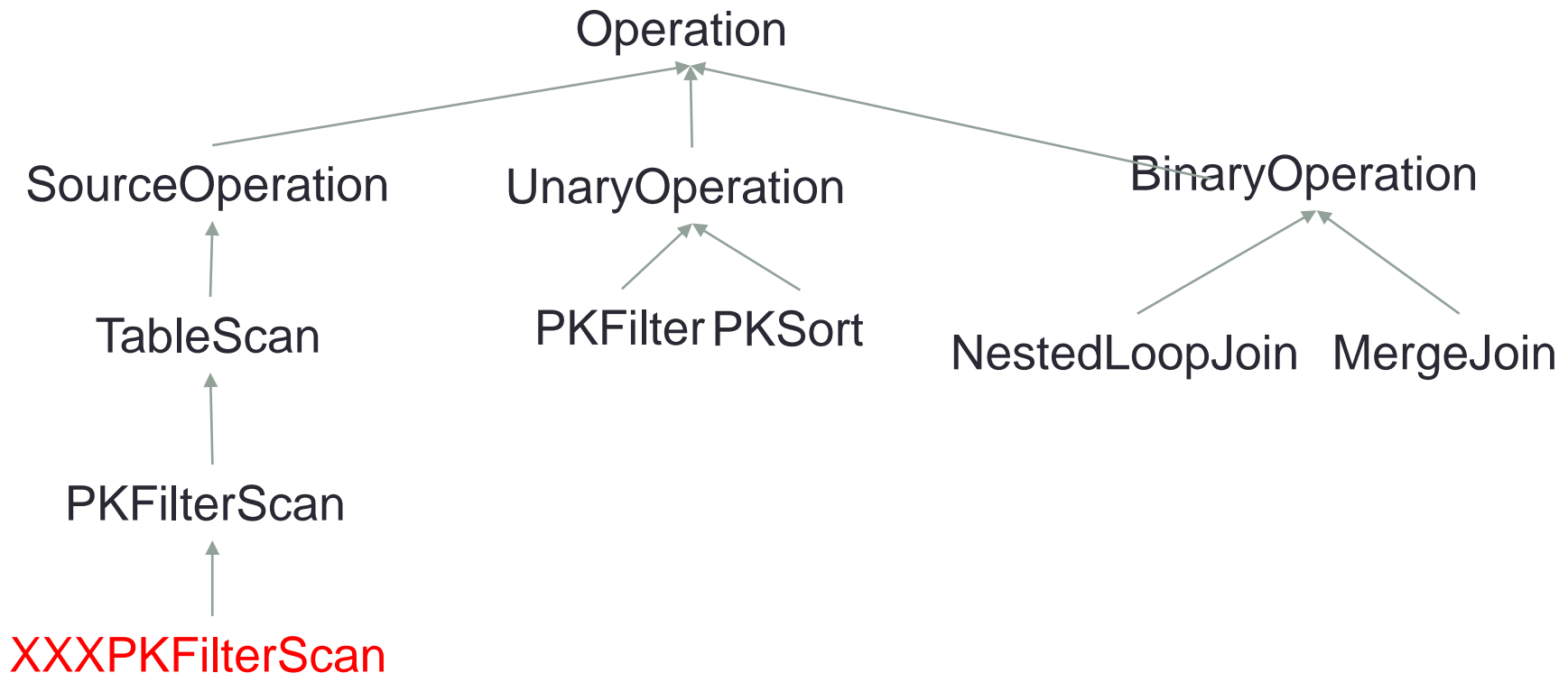
- O PKFilterScan considera que os dados podem estar fora da ordem de chave primária.
  - Sendo assim, todos os blocos precisam ser acessados durante a varredura
- Uma estratégia mais eficiente poderia ser utilizada caso se partisse da presunção de que os registros estejam ordenados por chave primária
  - Isso permitiria usar uma estratégia de busca binária

# Sumário

- TableScan
- PKFilterScan
- Definição do Trabalho

# Objetivo do trabalho

- Criar uma extensão de PKFilterScan
  - A classe criada deve se chamar xxxPKFilterScan, onde xxx é o nome do aluno



# Objetivo do trabalho

- Essa extensão deve considerar que os dados estejam ordenados por chave primária
- Essa presunção deve ser usada para reduzir a quantidade de blocos que precisam ser acessados durante uma varredura
- Em relação à PKFilterScan, a principal diferença são os valores das variáveis que determinam o intervalo de busca
  - currentBlock
  - lastBlock



# Objetivo do trabalho

- Exemplos de intervalos
- $p_k > 50$ 
  - currentBlock: b5
  - lastBlock: b8
- $p_k < 50$ 
  - currentBlock: b0
  - lastBlock: b4
- $P_k = 50$ 
  - currentBlock: b5
  - lastBlock: b5
- $P_k \neq 50$ 
  - currentBlock: b0
  - lastBlock: b8

2,xxx 3,xxx	b0
6,xxx 8,xxx	b1
10,xxx 15,xxx	b2
18,xxx 20,xxx	b3
33,xxx 44,xxx	b4
50,xxx 53,xxx	b5
56,xxx 70,xxx	b6
77,xxx 79,xxx	b7
84,xxx 85,xxx	b8

# Objetivo do trabalho

- A definição do `currentBlock` e do `lastBlock` devem ser feitas dentro da função **`open()`**
- Naturalmente, funções auxiliares podem vir a ser necessárias para se chegar ao resultado

```
@Override  
public void open() throws Exception {  
  
    super.open();  
    //aqui vai o código  
}
```

# Objetivo do trabalho

- **IMPORTANTE**
- Para todas operações, a **busca binária** deve ser usada para encontrar os intervalos
  - Com exceção da diferença, que necessariamente deve percorrer todos os blocos

# Dicas

- Funções/Parâmetros que podem ser úteis
  - `getComparisonType();`
    - Recupera a operação de comparação
  - `getValue();`
    - Recupera o valor de comparação
  - `Table.BLOCKS_AMOUNT`
    - Recupera a quantidade de blocos da tabela
  - `table.getBlock(index);`
    - Recupera um bloco pelo valor de índice (index)
  - `block.iterator();`
    - Recupera um iterador para percorrer os registros de um bloco

# Formas de Verificação

- O objetivo principal é reduzir a quantidade de blocos que são carregadas
  - Com exceção da diferença, que percorre toda a tabela
- O código abaixo mostra um script que pode ser usado para fazer essa verificação

```
Table table1 = createTable("c:\\teste\\ibd","t1.ibd",1980, false, 1);
Params.BLOCKS_LOADED = 0;
Params.BLOCKS_SAVED = 0;
Operation scan = new xxxPKFilterScan("t1", table1, ComparisonTypes.LOWER_THAN, 10L);
scan.open();
while (scan.hasNext()){
    Tuple r = scan.next();
    System.out.println(r);
}
System.out.println("blocks loaded " + Params.BLOCKS_LOADED);
System.out.println("blocks saved " + Params.BLOCKS_SAVED);
```

# Avaliação

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

operação	peso
EQUAL *	30%
DIFF	10%
LOWER_THAN, LOWER_EQUAL_THAN *	30%
GREATER_THAN, GREATER_EQUAL_THAN *	30%

\* É necessário que a definição do intervalo seja feita de forma otimizada (através de uma busca binária)

# Entrega

- Prazo final de entrega, sem descontos
  - Domingo, 11 de julho às 22:00
- A cada dia de atraso, a nota é decrementada em 40%.
- O que entregar
  - O código fonte da classe criada (.java)