TRABALHO PRÁTICO 5

Código Fonte

- O SGBD criado para a disciplina tem suporte a algumas operações de acesso aos dados
  - Varredura de dados de uma tabela
    - TableScan
    - OrderedScan
  - Algoritmos de junção
    - NesteLoopJoin
    - MergeJoin

Interface Operation

- Todas operações precisam implementar a Interface Operation
- A interface declara as funções necessárias para a comunicação entre operações
  - funções baseadas em iteradores

```
public interface  Operation {
    public abstract void open() throws Exception;
    public abstract Tuple next() throws Exception;
    public abstract boolean hasNext() throws Exception;
    public abstract void close() throws Exception;
}
```

Interface Operation

- Possui duas extensões

- Operações que aceitam uma única operação de entrada devem implementar **UnaryOperation**

```
public interface UnaryOperation extends Operation{
    public Operation getOperation();
}
```

- Operações que aceitam duas operações de entrada devem implementar **BinaryOperation**

```
public interface BinaryOperation extends Operation{
    public Operation getLeftOperation();
    public Operation getRigthOperation();
}
```

Interface Operation

- A chamada ao método next() provoca a geração de uma tupla.
  - public abstract **Tuple** next() throws Exception;

- As operações devem retornar uma instância de Tuple

Classe Tuple

- Representa um registro contendo código e valor

```
public class Tuple {
    public long primaryKey;
    public String content;
}
```

Operação TableScan

- Operação Unária.
  - Provê acesso aos registros de uma tabela

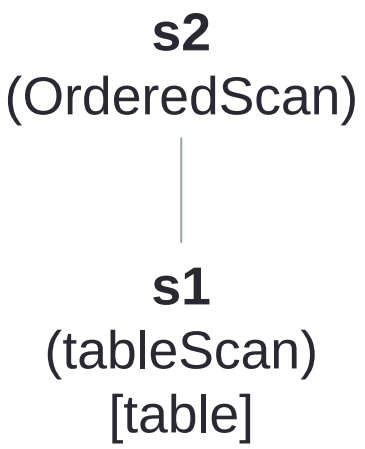
```
Operation scan = new TableScan(table);

scan.open();
while (scan.hasNext()){
      Tuple r =  scan.next();
       System.out.println(r.primaryKey);
       System.out.println(r.content);

}
```

scan
(tableScan)
[table]

Operação OrderedScan

- Operação Unária.
- Dá acesso aos dados ordenados
- Usa materialização em memória

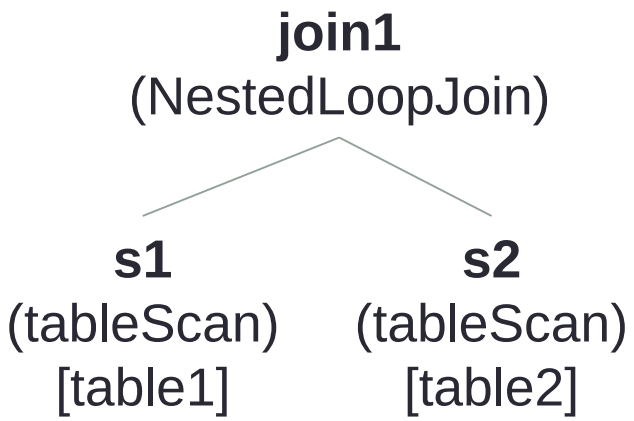
```
Operation s1= new TableScan (table);
Operation s2 = new OrderedScan(s1);

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



Operação NestedLoopJoin

- Operação binária.
- Realiza a junção com base em igualdade de chave primária

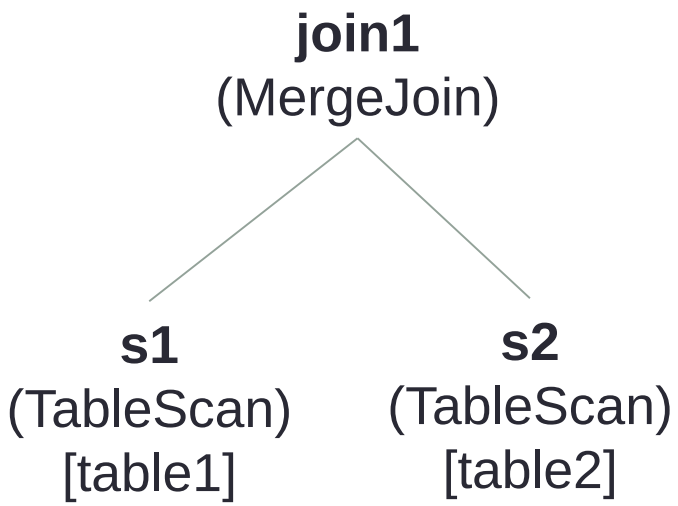
```
Operation s1 = new TableScan(table1);
Operation s2 = new TableScan(table2);

Operation join1 = new
            NestedLoopJoin(s1, s2);

join1.open();
while (join1.hasNext()){
    …
}
```



Operação MergeJoin

- Operação binária
- Os dados já precisam estar ordenados
- Pode ser usado diretamente via **TableScan** se as tabelas já estiverem ordenadas

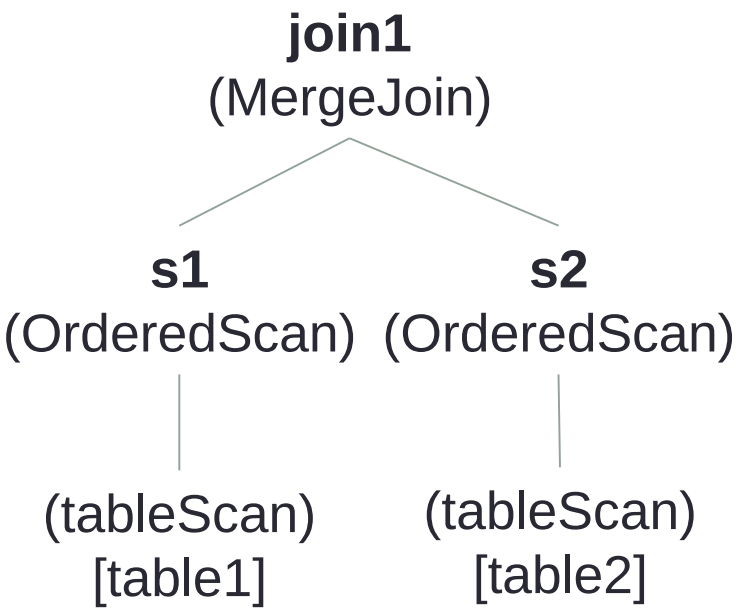
```
Operation s1 = new TableScan(table1);

Operation s2 = new TableScan(table2);

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



Operação MergeJoin

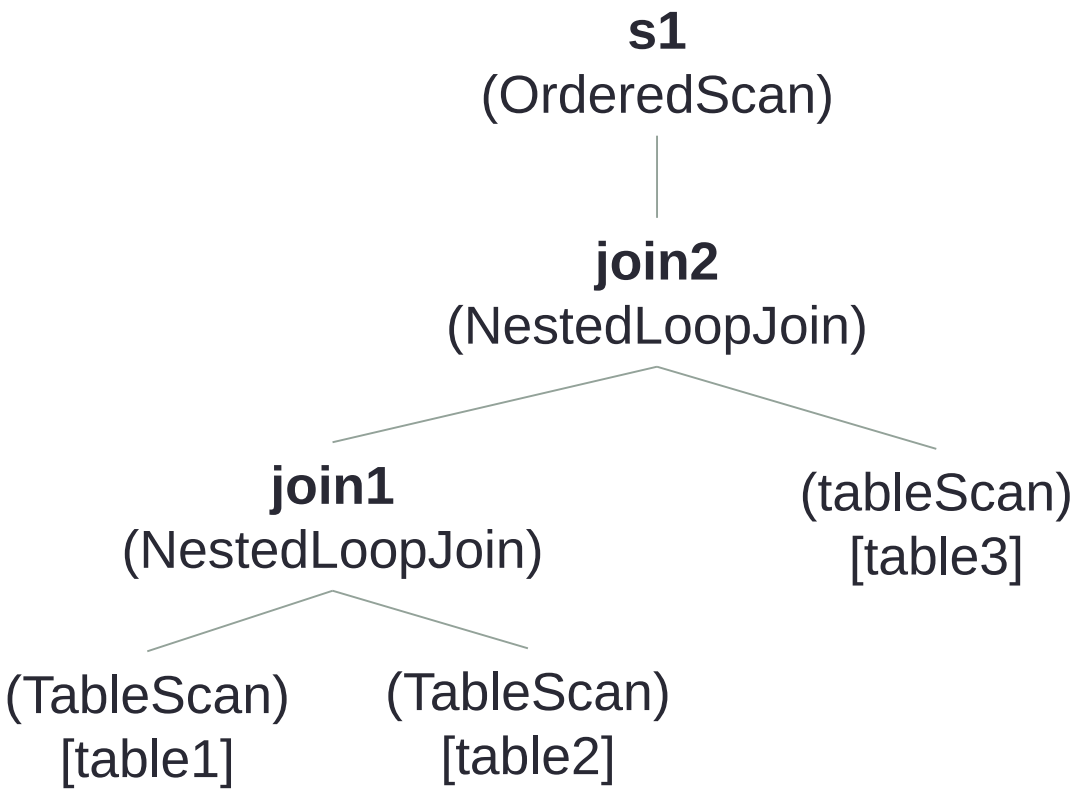
- Operação binária
- Os dados já precisam estar ordenados
- Pode ser alcançado via **OrderedScan** caso os dados estejam desordenados

```
Operation s1 = new OrderedScan (
                    new TableScan(table1));

Operation s2 = new OrderedScan (
                    new TableScan(table2));

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



Composições de Operações

- As operações implementam uma interface comum
  - Isso permite criar composições complexas



OBJETIVO DO TRABALHO

Objetivo do Trabalho

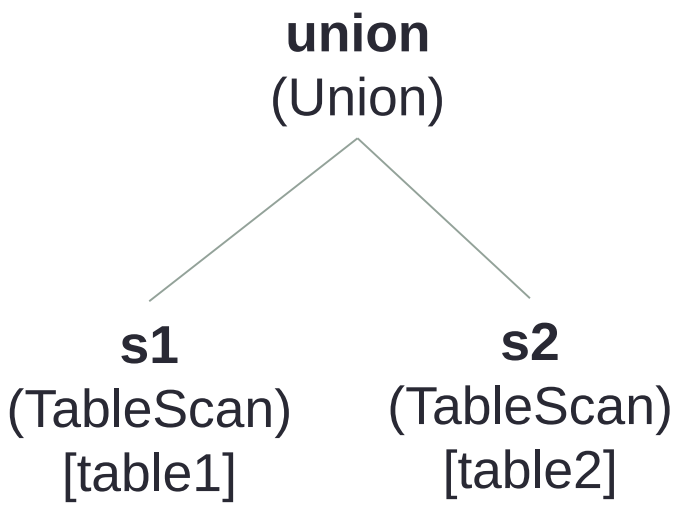
- O objetivo do trabalho é implementar a operação binária de União
  - Criar uma classe chamada xxxUnion, onde xxx é o nome do aluno
  - Implementar as funções herdadas das interfaces

```
public void open() throws Exception;
public Tuple next() throws Exception;
public boolean hasNext() throws Exception;
public void close();
public Operation getLeftOperation();
public Operation getRigthOperation();
```

Operação Union

- Operação binária
- Os dados já precisam estar ordenados
- Pode ser usado diretamente via **TableScan** se as tabelas já estiverem ordenadas

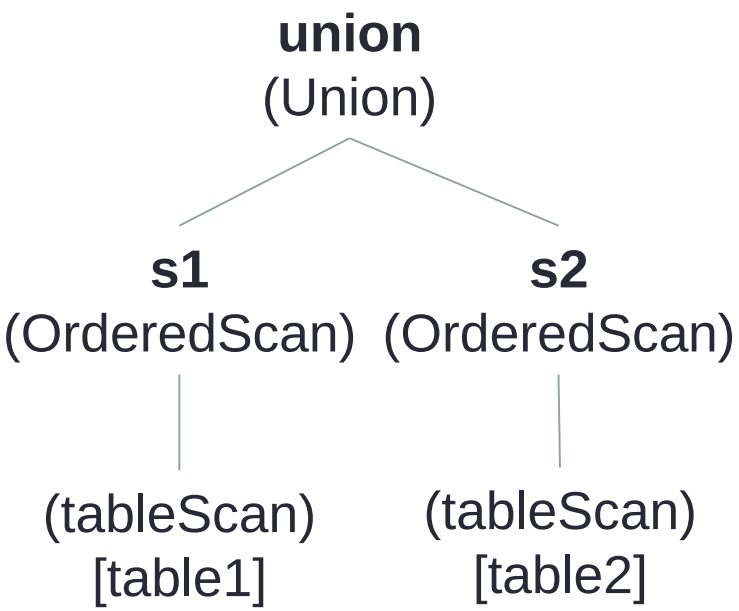
```
Operation s1 = new TableScan(table1);

Operation s2 = new TableScan(table2);

Operation union = new Union(s1, s2);
```



Operação Union

- Operação binária
- Os dados já precisam estar ordenados
- Pode ser alcançado via **OrderedScan** caso os dados estejam desordenados

```
Operation s1 = new OrderedScan (
                        new TableScan(table1));

Operation s2 = new OrderedScan (
                        new TableScan(table2));

Operation union = new Union(s1, s2);
```



Resumo das principais funções

- Função open()
  - Prepara tudo para que se comece a fazer a junção (abre cursores, limpa variáveis, ...)
- Funções next()
  - Recupera o próximo registro e avança o cursor
- Funções hasNext()
  - Verifica se há mais elementos a recuperar

Observações

- É preciso cuidado na implementação das funções hasNext() e next().
  - Por exemplo, deve-se garantir que o iterador só avance quando for chamado o next().
- Registros com a mesma chave primária devem ser concatenados
- O algoritmo de junção MergeJoin também trabalha com dados ordenados
  - Analise as operações desta operação para auxiliar na criação do algoritmo de união

Testes

- Use a função createTable para testar as operações

```
Table table = createTable("c:\\teste\\ibd","t1.ibd",1000, false, 1);

Operation s1 = new TableScan(table1);

...
```

- Parâmetros:
  - Pasta onde está o banco
  - Arquivo onde está o banco
  - Quantidade de registros
  - Flag indicando se registros ficarão desordenados
  - Distância entre um valor e outro

Testes

```
Table table1 = createTable("c:\\teste\\ibd","t1.ibd",50, false, 1);
Table table2 = createTable("c:\\teste\\ibd","t2.ibd",100, false, 1);
Operation union = new Union(scan1, scan2);


Params.BLOCKS_LOADED = 0;
Params.BLOCKS_SAVED = 0;
union.open();
while (union.hasNext()){
	Tuple r = query.next();
	System.out.println(r.primaryKey + " - "+r.content);
}

System.out.println("blocks loaded " + Params.BLOCKS_LOADED);
System.out.println("blocks saved " + Params.BLOCKS_SAVED);
```

Formas de Verificação

- Verificação de consistência
  - Os registros devem ser retornados ordenados e sem duplicatas
  - Nenhum registro deve ser descartado
- Ex.

1	aaa
3	bbb
5	ccc
7	ddd

1	ggg
2	hhh
10	iii

1	aaa ggg
2	hhh
3	bbb
5	ccc
7	ddd
10	iii

Formas de Verificação

- Verificação de desempenho
  - Durante a união, nenhum bloco é processado mais do que uma vez
  - Sendo assim, uma forma de analisar o desempenho é trabalhar com dados já fisicamente ordenados
  - Nesse caso, o número de blocos carregados durante a união deve ser proporcional ao número total de blocos

- Mantenha o tamanho do buffer de blocos com o tamanho original em BufferManager

Formas de Verificação

- Teste com diferentes consultas, variando
  - quantidade de tabelas
  - quantidade de registros de cada tabela

- Obs.
  - A consulta pode conter apenas operações do tipo TableScan e Union
  - Não é preciso testar com o OrderedScan
    - Mas assegure que os dados estejam fisicamente ordenados pela chave primária

Entrega

- Prazo final de entrega, sem descontos
  - Domingo, 28 de junho às 23:55

- A cada dia de atraso, a nota é decrementada em 50%.

- O que entregar
  - O código fonte da classe de otimização (.java) não comprimido