Operator Interface


```
interface Operator<Chunk>{

    void open();                    //initializes the operator

    Chunk next();                   //returns the next chunk of data

    void close();                   //performs cleanup work (if necessary)
}
```

Selection Operator

```
class Selection implements Operator<Row>{
    private Operator<Row> input;                                    //internal handle to input Operator
    private Predicate<Row> sel;
    public Selection(Operator<Row> input, Predicate<Row> sel){      //constructor
        this.input = input; this.sel = sel;
    }
    public void open(){input.open();}                               //initializes the operator
    public Row next(){                                              //returns the next row of data
        For (Row tmp = input.next(); tmp != NULL; tmp = input.next(){
            If( sel.execute( tmp ) ){
                return tmp;
            }
        }
        return NULL;                                                //signal end of input
    }
    public void close(){input.close();}                             //performs cleanup work (if necessary)
}
```

Loops in Operators?

```
class Enumerate implements Operator<Integer>{
    private int from, to;
    public Enumerate(int from, int to){                    //return [from;to], i.e. both including
        this.from = from; this.to = to;
    }
    public void open(){}                                   //initializes the operator

    public Integer next(){                                 //returns the next row of data
        For (int current = from; current<=to; current++){  //if still in range
            return current;                                //return and increment afterwards
        }
    }
    public void close(){}                                  //performs cleanup work (if necessary)
}
```

wrong!

[42; 77]

42, 42, 42, 42, ...

Save the State as an Attribute

```
class Enumerate implements Operator<Integer>{
    private int current, from, to;
    public Enumerate(int from, int to){                    //return [from;to], i.e. both including
        this.from = from; this.to = to;
    }
    public void open(){                                    //initializes the operator
        current = from;                                    //initializes the state of the operator
    }
    public Integer next(){                                 //returns the next row of data
        If (current <= to){                                //if still in range
            Integer nextToReturn = current;                //this is what we return
            current++;                                     //need to increment internal state
            return nextToReturn;                           //return next element
        }
        Else return NULL;                                  //signal end of input
    }
    public void close(){}                                  //performs cleanup work (if necessary)
}
```

[42, 77]