interface Operator<Chunk> {

    void open();  // initializes the operator

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

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

}
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){
        this.input = input; this.sel = sel;
    }
    public void open(){input.open();} //initializes the operator
    public Row next(){
        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){
        this.from = from; this.to = to;
    }
    public void open(){
        //return [from;to], i.e. both including
        //initializes the operator
    }
    public Integer next(){
        //returns the next row of data
        //if still in range
        //return and increment afterwards
        For (int current = from; current<=to; current++)
            return current;
    }
    public void close(){
        //performs cleanup work (if necessary)
    }
}
Save the State as an Attribute

class Enumerate implements Operator<Integer>{
    private int current, from, to;
    public Enumerate(int from, int to){
        this.from = from; this.to = to;
    }
    public void open(){
        current = from;
    }
    public Integer next(){
        if (current <= to){
            Integer nextToReturn = current;
            current++;
            return nextToReturn;
        } else return NULL;
    }
    public void close(){}
}