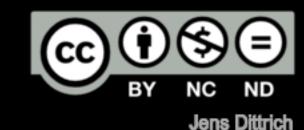
Operator Interface



```
interface Operator<Chunk>{
    void open();
    Chunk next();
```

void close();

```
//initializes the operator

//returns the next chunk of data

//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
                                                                           //returns the next row of data
     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(){}
     public Integer next(){
         For (int current = from; current<=to; current++){
              return current;
     public void close(){}
```

```
//return [from;to], i.e. both including
       //initializes the operator
       //returns the next row of data
       //if still in range
       //return and increment afterwards
       //performs cleanup work (if necessary)
    [45:44]
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){
         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(){}
```

```
//return [from;to], i.e. both including
//initializes the operator
//initializes the state of the operator
//returns the next row of data
//if still in range
//this is what we return
//need to increment internal state
//return next element
//signal end of input
//performs cleanup work (if necessary)
```