# PS Iterator

Iterators are used by the clients or other high-level programs to instream tuples/ data from the page. It reads the tuples of the relation located on the storage in the background and returns the tuples one by one to the page thus hiding the database low-level page handling.

## What an Iterator consists of?

An iterator consists of 4 crucial functions that are accessible to the client. They are open(), getNext(), hasNext(), close().

a. open(relationName) – Reads the first page of the relation given by “relationName” and positions the pointer to the first tuple located in the first page. It sets “hasNext” to true if it finds a tuple.
b. getNext() – Returns the tuple pointed to by the pointer and sets the pointer to the next available tuple in the page of the relation. It sets “hasNext” to false when it reaches the end of the page allocated to the relation.
c. hasNext() – Returns true if the iterator currently has the pointer pointing to a tuple, else returns false.
d. close() – Closes the iterator

## Base Iterator

Base Iterator is the basic iterator that reads the storage of the relation and iterates through the tuples, returning all the available tuples one-by-one until the end of the page allocated to the relation is reached. When there are multiple pages, the iterator uses the header information to fetch the nextPage information and positions the pointer to the start of the next page.

**Pseudo Code for the Base Iterator**

```
open(relationName) {
    buffer = readPage();      // Read the first page corresponding to the relation
    analyzeHeader();          // Read the header and get currentpage, nextPage, numberOfTuples
    nextTuple = readTuple();// Initialize the pointer to the first available tuple
    if (numberOfTuples==0) hasNextTuple=false;
    else hasNextTuple = true;
}
getNext() {
    currentTuple = nextTuple;
    if end of Page and nextPage==-1
        hasNextTuple = false;
        nextTuple = empty;
    else if endOfPage
        buffer = readPage(); // Read the next page and analyze header
        analyzeHeader();     // Read header and get currentpage, nextPage,  numberOfTuples
        nextTuple = readTuple(); // Read the next available tuple
    return currentTuple;
}
hasNext() {
    return hasNextTuple;
}
close() {
    close the connection;     // close the connection to the storage and release the resources
}
```

## Projection-Selection Iterator

Projection-Selection Iterator uses the Base Iterator and returns the tuples that meet the selection criteria as specified in the XML file of the expression tree for the corresponding relation. Each tuple returned will contain only the attributes specified for projection in the XML file of the Expression tree.

**Pseudo Code for the Projection-Selection Iterator**

```
open(relationName) {
    super.open(relationName);           // Call open() of Base Iterator
    while(super.hasNext())
        tuple = super.getNext();        // Iterate through the tuples evaluating the criteria
        if(evaluateCriteria(Tuple))
            nextTuple = tuple;
            hasNextTuple = true;
            break;
}
getNext() {
    currentTuple = nextTuple;
    if(!super.hasNext())                // super refers to the Base Iterator
        hasNextTuple = false;
        nextTuple = empty;
    while(super.hasNext())
        tuple = super.getNext();
        if(evaluateCriteria(tuple))     // evaluateCriteria() checks if tuple meets the
            nextTuple = tuple;          // specified criteria
            break;
    if nextTuple==currentTuple          // finished iterating through all the tuples
        nextTuple = empty;
        hasNextTuple = false;
    return projection(currentTuple);    // projection() function sends out only the
}                                       // requested attributes
hasNext() {
    return hasNextTuple;
}
close() {
    close the connection;   // close the connection to the storage and release the resources
}
```

In the above pseudo code, the function

a) evaluateCriteria() extracts the filter criteria for the relation in the expression tree XML using XMLparsing library/DOM API and evaluates it against the tuple. It returns true if condition C holds, returns false otherwise. If there are multiple criteria $C_1$, $C_2$.., $C_n$ specified for the relation, it is evaluated as $C_1 \wedge C_2 \wedge \ \dots \wedge C_n$.
b) projection() parses the expression tree XML extracting the attributes to be projected, and sends out only the specified attributes.
c) readTuple() returns the tuple pointed to by the pointer and repositions the pointer to the first byte located after the end of previous tuple.
d) analyzeHeader() reads the header bytes and extracts the currentPageNumber, nextPageNumber, numberOfTuples and currentPageSizeOccupiedByTheRelation that will be used by the iterator to know information about current page and next page.

## Storage and Configuration

The tuples are maintained as bytes in the storage. The format for a page is as follows:

1. For experimental purposes, the size of a page is considered to be 1024 bytes. This can be easily modified by setting the “PAGESIZE” constant.
2. Each page starts with a header sequence of 16 bytes containing
   a. currentPageNumber – 4 bytes
   b. nextPageNumber – 4 bytes
   c. numberOfTuples – 4 bytes
   d. currentPageSizeOccupiedByTheRelation – 4 bytes
3. The header is followed by the sequence of tuples in byte format.
4. The catalog file “catalog.xml” specifies the order and size of attributes in a tuple for each relation.
5. The Expression tree “expTree.xml” obtained from parsing the query specifies the evaluation criteria and the attributes to be projected for a relation.

## Example on how to access the Base Iterator and Projection-Selection Iterator

```
PSIterator iterator = new PSIterator(storageFilePath,"exptree.xml");

// Open the connection to the iterator
iterator.open("Emp");
int recordCount = 0;

// Check if iterator has next tuple. If so, call getNext() and print it.
while(iterator.hasNext()) {
        byte[] empTuple = iterator.getNext();
        System.out.println("Record "+ recordCount +": "+
        EmployeeTuple.convertToEmployee(empTuple));
        recordCount++;
}

// Close the iterator
iterator.close();
```