AM Layer: The Access Method Layer

THE ACCESS METHOD (AM) LAYER

You will use a paged file from the PF layer to represent a B+ tree index, and you will use pages in the file to represent the individual tree nodes. You can choose how the index file will be named, but one thought is to make it a combination of the name of the file on which the index is built and the index number (1st, 2nd, 3rd, etc.), e.g., fileName.indexNo. The code that you build for this layer will help to support an indexed file abstraction, where files are organized as heaps and may have any number of single-attribute secondary indices associated with them to speed up selection and equality-join queries. All routine names begin with the prefix AM, which identifies the associated layer.

Access Method Layer Interface Routines

```
(1)  errVal = AM_CreateIndex(fileName, indexNo, attrType, attrLength)
char *fileName; /* name of indexed file */
int indexNo; /* number of this index for file */
char attrType; /* 'c' (character), 'i' (integer), or 'f' (float) */
int attrLength; /* 4 for 'i' or 'f', 1-255 for 'c' */
```

This routine creates a secondary index numbered indexNo on the file fileName. The indexNo parameter will have a unique value for each index created on a file. Thus, it will be used along with fileName to generate a unique name for the PF layer file used to implement the index. The name of the file would be "fileName.indexNo". The type and length of the attribute being indexed are described by the third and fourth parameters. The job of this routine is to create an empty index by creating its PF layer file and initializing it appropriately (i.e., to represent an empty B+ tree). It returns AME_OK if it succeeds, and an AM error code otherwise.

```
(2)  errVal = AM_DestroyIndex(fileName, indexNo)
char *fileName; /* name of indexed file */
int indexNo; /* number of this index for file */
```

This routine destroys the index numbered indexNo of the file fileName by deleting the file that is used to represent it. It returns AME_OK if it succeeds, and an AM error code otherwise.

```
(3)  errVal = AM_InsertEntry(fileDesc, attrType, attrLength, value, recId)

int fileDesc; /* file descriptor */
char attrType; /* 'c', 'i', or 'f' */
```

```
int attrLength; /* 4 for 'i' or 'f', 1-255 for 'c' */
char *value; /* attribute value for the insert */
int recId; /* id of record to insert */
```

This routine inserts a (value, recId) pair into the index represented by the open file associated with fileDesc. The value parameter points to the value to be inserted into the index, and the recId parameter identifies a record with that value to be added to the index. It returns AME_OK if it succeeds, and an AM error code otherwise.

```
(4)  errVal = AM_DeleteEntry(fileDesc, attrType, attrLength, value, recId)
int fileDesc; /* file descriptor */
char attrType; /* 'c', 'i', or 'f' */
int attrLength; /* 4 for 'i' or 'f', 1-255 for 'c' */
char *value; /* attribute value for the delete */
int recId; /* id of the record to delete */
```

This routine removes a (value, recId) pair from the index represented by the open file associated with fileDesc. It returns AME_OK if it succeeds, and an AM error code otherwise. Note that both AM_AddEntry() and AM_RemoveEntry() must maintain the B+ tree occupancy criterion and accordingly pages may have to be split or merged.

```
(5)  scanDesc = AM_OpenIndexScan(fileDesc, attrType, attrLength, op, value)
int fileDesc; /* file descriptor */
char attrType; /* 'c', 'i', or 'f' */
int attrLength; /* 4 for 'i' or 'f', 1-255 for 'c' */
int op; /* operator for comparison*/
char *value; /* value for comparison */
```

This routine opens an index scan over the index represented by the file associated with fileDesc. The scan will return the record ids of those records whose indexed attribute value compares in the desired way with the value parameter. The (non-negative) scan descriptor returned is an index into a table (implemented and maintained by your AM layer code) used for keeping track of information about the state of in-progress file scans. You may assume that no more than MAXSCANS = 20 file scans will ever need to be performed at one time. If the scan table is full, an AM error code is returned in place of a scan descriptor.

The parameter op determines the way that the value parameter is compared to the record's indicated attribute value. The different comparison options are encoded in op as follows.

```
1 for EQUAL (i.e., attribute = value)
2 for LESS THAN (i.e., attribute < value)
```

```
3 for GREATER THAN (i.e., attribute > value)
4 for LESS THAN or EQUAL (i.e., attribute <= value)
5 for GREATER THAN or EQUAL (i.e., attribute >= value)

(6)  recId = AM_FindNextEntry(scanDesc)
int scanDesc; /* index scan descriptor */
```

This routine returns the record id of the next record that satisfies the conditions specified for index scan associated with scanDesc. It returns AME_EOF, if there are no more records that satisfy the scan predicate, and an AM error code otherwise. Please see the note below for further information about this routine.

```
(7)  errVal = AM_CloseIndexScan(scanDesc)
int scanDesc; /* index scan descriptor */
```

This routine terminates an index scan, disposing of the scan state information.