Reference Manual

The Multilevel Grid File (MLGF)

Version 4.0

Manual Release 1

2016 년 7 월

Advanced Information Technology Research Center (AITrc)

KAIST

1. MLGF 4.0의 소개

MLGF 4.0은 다차원 동적 해쉬 파일 구조인 계층 그리드 파일 (Multilevel Grid File) 을 구현한 것이다. 계층 그리드 파일은 레코드의 삽입으로 인하여 페이지가 용량 초과될 때 페이지를 분할하고, 레코드의 삭제로 인하여 페이지가 용량 미달될 때 페이지를 병합함으로써 데이터의 양과 분포의 변화에 적합하도록 자신의 구조를 동적으로 대응시킬 수 있다.

계층 그리드 파일을 구현한 MLGF 4.0은 다음과 같은 기능을 갖는다.

- 계층 그리드 파일을 생성해 주는 기능
- 계층 그리드 파일을 오픈하는 기능
- 오픈되어 있는 계층 그리드 파일을 클로즈하는 기능
- 계층 그리드 파일에 레코드를 삽입하는 기능
- 계층 그리드 파일로부터 레코드를 삭제하는 기능
- 계층 그리드 파일에서 주어진 영역에 속하는 레코드들을 검색하는 기능

2. MLGF 4.0의 응용 프로그램 인터페이스

MLGF 4.0은 MLGF_Init(), MLGF_Final(), MLGF_CreateIndex(), MLGF_OpenIndex(), MLGF_CloseIndex(), MLGF_InsertObject(), MLGF_DeleteObject(), MLGF_Query(), MLGF_Dump() 등의 14개 함수로 구성된 응용 프로그램 인터페이스를 제공한다. 본 절에서는 각 인터페이스 함수에 대하여 설명한다.

MLGF 4.0에서는 프로그램의 이식성을 높이기 위해서 모든 변수 선언을 다음과 같이 하고 있다.

- 32bit 운영체제

```
typedef long            Four;
typedef unsigned long   UFour;
typedef short           Two;
typedef char            One;
```

- 64bit 운영체제

```
typedef int             Four;
typedef unsigned int    UFour;
typedef short           Two;
typedef char            One;
```

따라서 프로그램에서 사용된 One, Two와 같은 타입은 각각 1바이트, 2바이트 변수 임을 나타낸다.

2.1. MLGF_Init

```
Four MLGF_Init( nBuffers)
Four    nBuffers;       /* IN      number of buffers */
```

MLGF_Init()은 계층 그리드 파일에서 buffer manager를 사용할 수 있도록 관련된 데이터 구조들을 위한 메모리 공간을 할당 및 초기화한다. 'nBuffers'는 사용할 buffer의 수이다. Buffer manager를 사용하도록 컴파일된 계층 그리드 파일을 생성 및 사용하기 위해서는 먼저 MLGF_Init()을 이용하여 관련된 데이터 구조들을 위한 메모리 공간을 할당 및 초기화해야 한다. 계층 그리드 파일의 사용이 끝나면 MLGF_Final()을 이용하여 사용했던 메모리 공간을 반환해야 한다.

예제

다음은 계층 그리드 파일에서 buffer manager를 사용할 수 있도록 관련된 데이터 구조들을 위한 메모리 공간을 할당 및 초기화하고, 사용했던 메모리 공간을 반환하는 프로그램의 일부이다.

```
e = MLGF_Init(400);
if ( e < eNOERROR ) {
        /* error handling routine */
}

…


e = MLGF_Final();
if ( e < eNOERROR ) {
        /* error handling routine */
}
```

2.2. MLGF_Final

Four MLGF_Final()

MLGF_Final()은 buffer manager와 관련하여 계층 그리드 파일에서 사용했던 메모리 공간을 반환한다. Buffer manager를 사용하도록 컴파일된 계층 그리드 파일의 사용이 끝나면 MLGF_Final()을 이용하여 사용했던 메모리 공간을 반환해야 한다.

예제

제 2.1절의 MLGF_Init()에 있는 예제 참조

2.3. MLGF_CreateIndex

Four MLGF_CreateIndex(path, numOfKeys, numOfAttrs, attributeType, minMaxTypeVector)

One	*path;	/* IN	full name of the file */
One	numOfKeys;	/* IN	number of keys */
One	numOfAttrs;	/* IN	number of attributes */
AttributeType	*attributeType;	/* IN	types of each attribute */
MLGF_HashValue	minMaxTypeVector;	/* IN	bit vector of flags indicating types of each key's extreme hash value */
One	useAdditionalFunc	/* IN	flag of additional functionalities use */

MLGF_CreateIndex()는 'path'가 가리키는 경로명 (pathname) 을 갖는 새로운 계층 그리드 파일을 하나 생성한다.

파일을 생성할 때 파일에 저장되는 레코드에 대한 스키마 (schema) 정보를 지정해야 한다. 'numOfAttrs'는 레코드에 있는 속성의 개수를 지정하며, 'attributeType'은 각 속성에 대한 데이터 타입을 지정한다. 계층 그리드 파일이 사용하는 키의 개수는 'numOfKeys'에 지정한다. 키로 사용되는 속성들은 다른 속성들보다 레코드의 앞쪽에 위치해야 한다.

속성의 데이터 타입을 지정할 때 사용되는 AttributeType은 다음과 같이 정의되어 있다.

```
typedef struct {
        enum { INT, FLOAT, STRING, VARSTRING } dataType;        /* data type */
        Two                                    length;          /* length */
} AttributeType;
```

'dataType'은 INT, FLOAT, STRING, VARSTRING의 4가지중 한 가지 값을 갖는데 각각 정수, 실수, 고정길이 문자열, 가변길이 문자열 타입을 뜻한다. 'length'는 애트리뷰트의 크기로서 데이터 타입이 INT, FLOAT인 경우에는 각각 int, float 데이터 타입의 크기를 지정한다. 데이터 타입이

STRING인 경우에는 문자열의 크기를 지정하고, 데이터 타입이 VARSTRING인 경우에는 최대 문자열의 크기를 지정한다.

'minMaxTypeVector'는 key들이 가질 수 있는 extreme hash value의 type들을 지정해 주는데 사용된다. 적절한 'minMaxTypeVector'를 지정해 주는 것에 의해 range query의 성능을 향상시킬 수 있다. 지정해 줄 수 있는 extreme hash value의 type으로는 MINTYPE과 MAXTYPE이 있다. 'useAdditionalFunc'는 MLGF의 추가적인 기능을 사용 할지 여부 (TRUE or FALSE) 를 지정한다. 지원하는 MLGF의 추가 기능으로는 1. variable length 레코드 저장, 2. 동일한 hashed key를 가지는 레코드 저장이 있다.

예제

다음은 각각 INT, FLOAT, VARSTRING의 데이터 타입을 갖는 세 개의 속성을 갖고, 이중 처음 두 개의 속성이 키로 사용되는 레코드를 저장할 수 있는 계층 그리드 파일을 생성하는 프로그램의 일부이다. 생성된 파일은 "test.mlgf"라는 이름을 갖는다.

```
AttributeType       attrType[3];                    /* variable declaration */
MLGF_HashValue      minMaxTypeVector;
Four                e;                              /* error code */

attrType[0].dataType = INT;
attrType[0].length = sizeof(int);                   /* first attribute */
attrType[1].dataType = FLOAT;
attrType[1].length = sizeof(float);                 /* second attribute */
attrType[2].dataType = VARSTRING;
attrType[2].length = 20;                            /* third attribute */

MLGF_KEYDESC_SET_MINTYPE(minMaxTypeVector, 0);      /* first key's extreme hash
                                                       type */
MLGF_KEYDESC_SET_MAXTYPE(minMaxTypeVector, 1);      /* second key's extreme hash
                                                       type */

/* create mlgf file */
e = MLGF_CreateIndex( "test.mlgf", 2, 3, attrType, minMaxTypeVector,TRUE);
if ( e < eNOERROR ) {
        /* error handling routine */
}
```

2.4. MLGF_OpenIndex

```
Four MLGF_OpenIndex( path, mlgfd)
One      *path;    /* IN      path */
Two      *mlgfd;   /* OUT     mlgf file descriptor */
```

'path'는 계층 그리드 파일에 대한 경로명이다. MLGF_OpenIndex()은 'path'가 가리키는 파일을 열고, 그 파일에 대한 파일 기술자 (file descriptor) 를 'mlgfd'를 통하여 리턴한다. 계층 그리드 파일을 액세스하기 위해서는 먼저 MLGF_OpenIndex()을 이용하여 파일을 열어야 한다. 파일의 사용이 끝나면 MLGF_CloseIndex()를 이용하여 파일을 닫아야 한다.

예제

다음은 "test.mlgf" 라는 이름을 갖는 계층 그리드 파일을 열고 닫는 프로그램의 일부이다.

```
Two      mlgfd;    /* mlgf file descriptor */
Four     e;        /* error code */

e = MLGF_OpenIndex( "test.mlgf", &mlgfd);
if ( e < eNOERROR ) {
        /* error handling routine */
}

…


e = MLGF_CloseIndex( mlgfd);
if ( e < eNOERROR ) {
        /* error handling routine */
}
```

2.5. MLGF_CloseIndex

```
Four MLGF_CloseIndex( mlgfd)
Two      mlgfd;    /* IN      mlgf file descriptor */
```

MLGF_CloseIndex()는 'mlgfd'가 지정하는 열려진 계층 그리드 파일을 닫는 일을 한다. 'mlgfd'는 MLGF_OpenIndex()에서 리턴된 파일 기술자 (file descriptor) 이다. 계층 그리드 파일의 사용이 끝나면 MLGF_CloseIndex()를 사용하여 파일을 닫아야 한다.

예제

제 2.4절의 MLGF_OpenIndex()에 있는 예제 참조

2.6. MLGF_InsertObject

```
Four MLGF_InsertObject( mlgfd, record)
Two             mlgfd;    /* IN      mlgf file descriptor */
AttributeHeader *attrValues;        /* IN      record array */
```

MLGF_InsertObject()는 계층 그리드 파일에 레코드를 삽입하는 함수이다. ‘mlgfd’는 레코드가 삽입될 계층 그리드 파일에 대한 파일 기술자 (file descriptor) 로서 MLGF_OpenIndex()을 호출했을 때 리턴된 값이다. 삽입할 레코드의 데이터는 속성 값들의 배열 형태로 지정하며 ‘attrValues’는 그 배열의 시작주소를 가리킨다. 속성 값을 지정할 때 사용하는 AttributeHeader는 다음과 같이 정의된다.

```
typedef struct {
        Two     length;             /* length */
        union {
                int     *intPtr;    /*pointer */
                char    *charPtr;
                float   *floatPtr;
        } field;
} AttributeHeader;
```

‘length’는 속성의 길이를 나타낸다. ‘field’는 3개의 포인터의 유니온으로 구성되어 있는데 각 속성의 데이터 타입에 따른 해당 포인터에 실제 속성 값이 저장되어 있는 메모리를 가리키도록 한다. 속성의 데이터 타입이 INT인 경우는 ‘intPtr’을 이용하고, 데이터 타입이 FLOAT인 경우에는 ‘floatPtr’을 이용하여 데이터를 엑세스한다. 데이터 타입이 STRING이거나 VARSTRING인 경우에는 ‘charPtr’을 이용하여 데이터를 액세스한다.

예제

제 2.3절의 예제에서 생성된 MLGF에 레코드 (10, 10.1, “Hello”) 를 삽입하기 위해서는 다음과 같은 과정이 필요하다.

```
AttributeHeader   attr[3];
Four      e;        /* error code */

…

/* allocate memory */
attr[0].length = sizeof(int);               /* first attribute */
attr[0].field.intPtr = (int *)malloc( sizeof( int));
```

```
attr[1].length = sizeof(float);             /* second attribute */
attr[1].field.floatPtr = (float *)malloc( sizeof( float));
attr[2].length = 20;                        /* third attribute */
attr[2].field.charPtr = (char *)malloc( sizeof( char) * 20);

/* assign values */
*(attr[0].field.intPtr) = 10;               /* first attribute */
*(attr[1].field.floatPtr) = 10.1;           /* second attribute */
attr[2].length = strlen("Hello") + 1;       /* third attribute */
strcpy( attr[2].field.charPtr, "Hello");

/* insert record */
e = MLGF_InsertObject( mlgfd, attr);
if ( e < eNOERROR ) {
        /* error handling routine */
}

…

free( attr[0].field.intPtr);
free( attr[1].field.floatPtr);
free( attr[2].field.charPtr);
```

2.7. MLGF_DeleteObject

```
Four MLGF_DeleteObject( mlgfd, record)
Two                 mlgfd;    /* IN      mlgf file descriptor */
AttributeHeader     *attrValues;      /* IN      record to delete */
```

MLGF_DeleteObject()는 'attrValues'가 지정하는 레코드를 계층 그리드 파일에서 삭제한다. 'mlgfd'는 레코드가 삭제되는 계층 그리드 파일에 대한 파일 기술자 (file descriptor) 로서 MLGF_OpenIndex()을 호출했을 때 리턴된다.

예제

제 2.6절의 예제에서 삽입한 레코드 (10, 10.1, "Hello") 는 다음과 같이 삭제할 수 있다.

```
AttributeHeader   attr[2];
Four      e;          /* error code */

…

/* allocate memory */
attr[0].length = sizeof(int);               /* first attribute */
attr[0].field.intPtr = (int *)malloc( sizeof( int));
attr[1].length = sizeof(float);             /* second attribute */
```

```
attr[1].field.floatPtr = (float *)malloc( sizeof( float));
attr[2].length = 20;                    /* third attribute */
attr[2].field.charPtr = (char *)malloc( sizeof( char) * 20);

/* assign values */
*(attr[0].field.intPtr) = 10;           /* first attribute */
*(attr[1].field.floatPtr) = 10.1;       /* second attribute */
attr[2].length = strlen("Hello") + 1;   /* third attribute */
strcpy( attr[2].field.charPtr, "Hello");

/* delete record */
e = MLGF_DeleteObject( mlgfd, attr);
if ( e < eNOERROR ) {
        /* error handling routine */
}
```

2.8. MLGF_Query

```
Four MLGF_Query( mlgfd, region, num, records)
Two             mlgfd;      /* IN   mlgf file descriptor */
QueryRegion     *region;    /* IN   query region */
Four            num;        /* IN   buffer size */
AttributeHeader *records;   /* OUT  buffer to return the records */
```

MLGF_Query()는 지정한 영역에 속하는 레코드들을 검색하여 반환한다. 'mlgfd'는 레코드들을 검색할 계층 그리드 파일에 대한 파일 기술자 (file descriptor) 로서 MLGF_Open()을 호출했을 때 리턴된 값이다. 질의 영역은 키 (key) 속성들에 대하여 최소값과 최대값을 지정함으로써 정의한다. 'region'은 각 키 속성에 대한 최소값과 최대값들의 배열의 시작주소이다. 최소값과 최대값을 지정할 때 사용하는 QueryRegion은 다음과 같이 정의된다.

```
typedef struct {
        One             fullDomainFlag; /* TRUE if region is full domain */
        AttributeHeader minkey;         /* minimum key value */
        AttributeHeader maxkey;         /* maximum key value */
} QueryRegion;
```

'fullDomainFlag'는 질의 영역이 전체 도메인 (domain) 인지 부분 도메인인지를 나타낸다. 전체 도메인이면 TRUE 값을 갖고 부분 도메인이면 FALSE 값을 갖는다. 'minKey'와 'maxKey'는 각각 질의 영역의 최소값과 최대값을 뜻하는 것으로서 질의 영역이 부분 도메인인 경우에만 지정하면 된다.

질의 영역에 존재하는 레코드들은 ‘records’를 통하여 반환된다. ‘records’는 AttributeHeader 데이터 타입의 배열의 시작주소로서, 반환된 레코드들 중 x 번째 record의 y 번째 속성 값은 배열의 (x * # of attributes + y) 번째 원소로서 저장된다.

이 함수는 질의 영역에 있는 레코드의 수를 리턴한다. 만일 질의 영역에 있는 레코드의 수가 ‘num’보다 크면 ‘num’ 만큼만 ‘records’에 복사하고 에러코드로 eTOOMANYRECORDS를 리턴한다. 만일 ‘records’가 NULL이면 레코드는 복사하지 않고 질의 영역에 있는 레코드 수만 리턴한다.

예제

제 2.3절에서 생성한 MLGF에서 첫 번째 키 속성 값은 5부터 15 사이에 있고, 두 번째 키 속성 값은 전체 도메인에 걸쳐 있는 레코드들을 찾는 방법은 다음과 같다.

```
QueryRegion     region[2];
AttributeHeader *attr;
int             nRecords;
int             minKeyValue, maxKeyValue;
int             *intArray;
float           *floatArray;
char            (*stringArray)[20];
Four            e;
int             i, num;


/* set region */
region[0].fullDomainFlag = FALSE;  /* partial domain */
minKeyValue = 5;
region[0].minKey.length = sizeof(int);
region[0].minKey.field.intPtr = &minKeyValue;
maxKeyValue = 15;
region[0].maxKey.length = sizeof(int);
region[0].maxKey.field.intPtr = &maxKeyValue;
region[1].fullDomainFlag = TRUE;    /* full domain */

/*get the number of records in the region */
nRecords = MLGF_Query( mlgfd, region, 0, NULL);
if ( nRecords < eNOERROR ) {
        /* error handling routine */  }

/* allocate memory */
num = nRecords * 3;         /* 3 is the number of attributes */
attr = (AttributeHeader*)malloc(sizeof(AttributeHeader)*num);
intArray = (int *)malloc(sizeof(int)*nRecords);
floatArray = (float *)malloc(sizeof(float)*nRecords);
/* 20 is the maximum length of the third attribute */
stringArray = (char(*)[20])malloc(sizeof(char)*nRecords*20);
```

```
for ( i = 0; i < nRecords; i++) {
        attr[i*3+0].field.intPtr = &intArray[i];
        attr[i*3+1]. field.floatPtr = &floatArray[i];
        attr[i*3+2]. field.charPtr = &stringArray[i][0];    }

/* find records */
e = MLGF_Query( mlgfd, region, nRecords, attr);
if ( e < eNOERROR ) {
        /* error handling routine */  }
```

2.9. MLGF_Dump

```
Four MLGF_Dump( mlgfd, recordOnly, outputfd)
Two     mlgfd;          /* IN     mlgf file descriptor */
Boolean recordOnly;     /* IN     flag that indicates what to print out */
FILE    *fd;            /* OUT    output file descriptor */
```

MLGF_Dump()는 'mlgfd'가 지정하는 계층 그리드 파일의 모든 record들 또는 트리 구조를 'fd'가 지정하는 output 파일에 출력해서 현재의 저장 상태를 보여준다. 만약, 'recordOnly' 값이 TRUE (1) 인 경우에는 모든 record들을 출력하고, FALSE (0) 인 경우에는 계층 그리드 파일의 트리 구조를 preorder 순서로 출력한다.

계층 그리드 파일을 구성하는 페이지에는 디렉토리 페이지, 리프 페이지, 오버플로우 페이지가 있다. 디렉토리 페이지는 트리 구조에서 non-leaf 노드에 해당하는 페이지로 다음 레벨의 디렉토리 페이지나 데이터 페이지를 가리키는 엔트리들을 가지고 있다. 리프 페이지는 leaf 노드에 해당하는데 실제 레코드를 저장하는 곳이다. 그리고 오버 플로우 페이지는 같은 키 값을 갖는 여러 레코드가 저장될 때 그 크기가 리프 페이지 전체 크기의 1/3을 넘는 경우 이 레코드들을 저장하기 위해 사용된다. 같은 키 값을 갖는 레코드들이 데이터 페이지 전체 크기의 1/3 이하이면 데이터 페이지에 저장된다.

예제

이 예제에서는 MLGF_Dump() 함수를 이용했을 때 그림 1과 같은 구조의 계층 그리드 파일이 어떤 형식으로 출력되는지 보여준다. "test.mlgf" 파일은 3개의 속성 (INT형 속성 2개와 FLOAT형 속성 1개) 을 갖는 계층 그리드 파일이다.

이 파일 구조를 좀 더 자세히 살펴 보면 루트 페이지에는 2개의 엔트리가 있고 각각 리프 페이지 10, 14를 가리키고 있다. 리프 페이지 10에는 3개의 엔트리가 있고 이중 0번째 엔트리에는 키

값이 같은 2개의 레코드가 저장되어 있다. 리프 페이지 14에는 3개의 엔트리가 있는데 0번째 엔트리에는 키 값이 같은 3개의 레코드가 저장되어 있으므로 오버 플로우 페이지를 만들어 별도로 관리하고 있다.

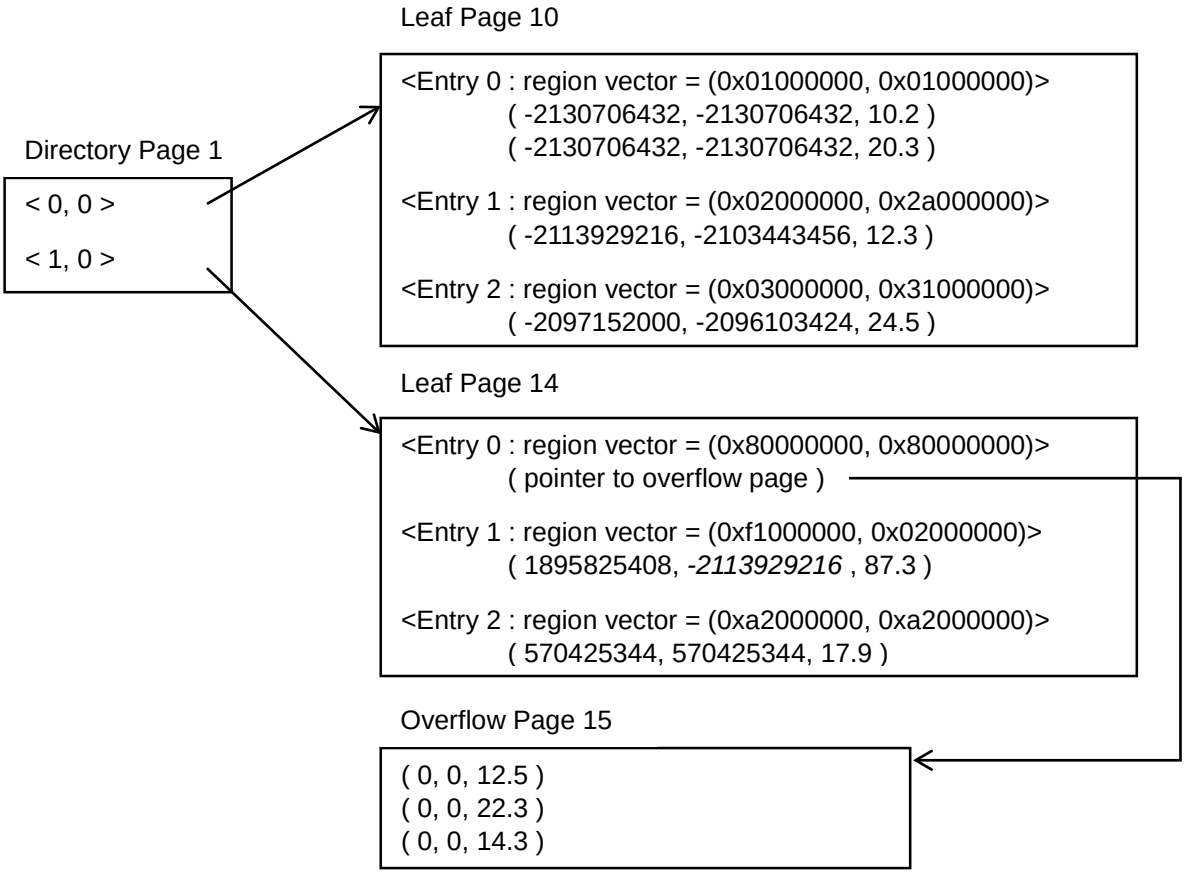

그림 1: Tree structure of “test.mlgf”

```
Two   mlgfd;    /* mlgf file descriptor */
Four  e;        /* error code */

      e = MLGF_OpenIndex( “test.mlgf”, &mlgfd);   /* open MLGF file */
      if ( e < eNOERROR ) {
            /* error handling routine */
      }

      …

      e = MLGF_Dump( mlgfd, FALSE, stdout);      /* dump MLGF file to screen */
      if ( e < eNOERROR ) {
            /* error handling routine */
      }
```

```
        …
        e = MLGF_CloseIndex( mlgfd);                    /* close MLGF file */
        if ( e < eNOERROR ) {
                /* error handling routine */

        }
```

출력 결과:

```
 <depth : 1> Directory Page : 1
-------------------------------------------------------
|    directory page dump
-------------------------------------------------------
|     # of entries              : 2
|   Entry 0 |                   Page Ptr 10 |
|          0-th Hash Value|      valid bit   1|      hash value 01000000
|          1-th Hash Value|      valid bit   1|      hash value 01000000
|   Entry 1 |                   Page Ptr 14 |
|          0-th Hash Value|      valid bit   1|      hash value 80000000
|          1-th Hash Value|      valid bit   1|      hash value 80000000

 <depth : leaf> Leaf Page : 10
 Dump leaf page : total 3 entries
Entry 0 : region vector = <0x01000000, 0x01000000>, nRecords 2
    Record 0 : ( -2130706432, -2130706432, 10.2 )
    Record 1 : ( -2130706432, -2130706432, 20.3 )
Entry 1 : region vector = <0x02000000, 0x2a000000>, nRecords 1
    Record 0 : ( -2113929216, -2103443456, 12.3 )
Entry 2 : region vector = <0x03000000, 0x31000000>, nRecords 1
    Record 0 : ( -2097152000, -2096103424, 24.5 )

 <depth : leaf> Leaf Page : 14
 Dump leaf page : total 3 entries
Entry 0 : region vector = <0x80000000, 0x80000000>, Overflow Page Ptr 15

 <Overflow Page> Overflow Page : 15
 Dump overflow page : total 3 records
    Record 0 : ( 0, 0, 12.5 )
    Record 1 : ( 0, 0, 22.3 )
    Record 2 : ( 0, 0, 14.3 )

Entry 1 : region vector = <0xf1000000, 0x02000000>, nRecords 1
    Record 0 : ( 1895825408, -2113929216 , 87.3 )
Entry 2 : region vector = <0xa2000000, 0xa2000000>, nRecords 1
    Record 0 : ( 570425344, 570425344, 17.9 )
```

2.10. MLGF_Error

```
char *MLGF_Error( errorCode)
Four    errorCode;      /* IN      error code */
```

MLGF_Error()는 에러 코드 'errorCode'에 대한 에러 메시지를 리턴한다. MLGF 4.0의 모든 함수들은 수행이 끝나면 에러 코드를 반환한다. 따라서 사용자는 그 값을 읽어서 작업이 제대로 수행되었는지 확인할 수 있다. 본 절에서는 MLGF_Error() 함수의 사용법에 대하여 설명하고, 제 2.13절에서 각 에러 코드에 대하여 설명한다.

예제

다음은 "test.mlgf" 라는 이름을 갖는 계층 그리드 파일을 여는 프로그램의 일부이다.

```
Two     mlgfd;  /* mlgf file descriptor */
Four    e;      /* error code */

e = MLGF_OpenIndex( "test.mlgf", &mlgfd);
if ( e < eNOERROR ) {
        /* error handling routine */
        printf("MLGF_OpenIndex: (error %d) %s\n", e, MLGF_Error( e));
        exit(1);
}
```

2.11. MLGF_GetN_KeyAttribute, MLGF_GetN_Attrribute, MLGF_GetAttributeType, MLGF_GetAttributeLength

```
One MLGF_GetN_KeyAttribute(mlgfd)
Two     mlgfd;          /* IN      mlgf file descriptor */

One MLGF_GetN_Attribute(mlgfd)
Two     mlgfd;          /* IN      mlgf file descriptor */

Four MLGF_GetAttributeType(mlgfd, attributeNum)
Two     mlgfd;          /* IN      mlgf file descriptor */
Two     attributeNum;   /* IN      attribute number */

Two MLGF_GetAttributeLength(mlgfd, attributeNum)
Two     mlgfd;          /* IN      mlgf file descriptor */
Two     attributeNum;   /* IN      attribute number */
```

위 4개의 함수들은 ‘mlgfd’가 지정하는 계층 그리드 파일에 저장된 레코드에 대한 스키마 정보를 참조할 수 있게 해주는 함수들이다. MLGF_GetN_KeyAttribute()와 MLGF_GetN_Attribute()는 각각 레코드에 있는 키 속성 및 모든 속성들의 개수를 반환한다. MLGF_GetAttributeType()과 MLGF_GetAttributeLength()는 각각 레코드에 있는 속성의 타입 및 크기를 반환한다. ‘attributeNum’은 참조할 속성의 번호이다.

예제

다음은 제 2.3절의 예제에서 생성된 MLGF에 저장된 레코드에 대한 스키마 정보를 출력해 주는 프로그램의 일부이다.

```
Two             mlgfd;          /* mlgf file descriptor */
One             nAttributes;    /* number of attributes */
int             i;

…

nAttributes = MLGF_GetN_Attribute(mlgfd);

for( i = 0; I < nAttributes; i++) {
        printf(“Type of attribute #%d: %d\n”, i, MLGF_GetAttributeType(mlgfd, i);
        printf(“Length of attribute #%d: %d\n”, i, MLGF_GetAttributeLength(mlgfd, i);
)

…
```

2.12. 에러 코드

MLGF 4.0의 API 함수가 반환하는 에러 코드는 다음과 같다. 이 값들은 mlgf.h 파일에 기록되어 있다.

```
#define eNOERROR                0       /* no error */
#define eBADPARAMETER           -1
#define eDUPLICATEDRECORD       -2
#define eNOTFOUND               -3
#define eTOOMANYRECORDS         -4
#define eMEMORYALLOCERR         -5
#define eSYSERR                 -6
#define eDUPLICATEDKEY          -7
#define eVARIABLELENGTHRECORD   -8
```

에러 코드가 뜻하는 바는 다음과 같다.

- eNOERROR
  에러 없이 수행을 끝냈다.
- eBADPARAMETER
  함수에 대한 인수 (parameter) 값이 잘못됐다.
- eDUPLICATEDRECORD
  모든 속성 값이 다 같은 레코드가 삽입된 경우이다. Key 값은 같을 수 있으나 모든 속성 값이 다 같은 경우는 에러 처리하고 있다.
- eNOTFOUND
  MLGF_DeleteObject() 함수에서 주어진 레코드를 찾지 못 한 경우 eNOTFOUND를 반환한다.
- eTOOMANYRECORDS
  MLGF_Query() 함수에서 질의 영역에 있는 레코드 수가 주어진 버퍼 크기보다 큰 경우 eTOOMANYRECORDS를 반환한다.
- eMEMORYALLOCERR
  메모리를 동적으로 할당하는데 실패했다.
- eSYSERR
  시스템 호출을 수행할 때 에러가 발생했다.
- eDUPLICATEDKEY
  추가적인 기능을 사용하지 않는 MLGF에서 hashed key 값이 동일한 레코드가 삽입이 될 경우 eDUPLICATEDKEY를 반환한다.
- eVARIABLELENGTHRECORD
  추가적인 기능을 사용하지 않는 MLGF에서 variable length record를 사용하고자 할 경우 eVARIABLELENGTHRECORD를 반환한다.