At the start of the program operation, a block is read from the file containing document profiles into the block buffer organized in primary storage. Then the records are read in succession from the block buffer and every record is sent to the A&LU through the record buffer. In the A&LU, the record's field 2 (the document profile) is compared with the subrequests of the available query formulation. If all the descriptors in some subrequest are also contained in the document profile, the search is considered successful, and the address (of the document text required to form the output) contained in field 1 is temporarily recorded in primary storage. Then the next record is read from the block buffer and analogous operations are repeated with every record contained in the block buffer. When the last record from the block buffer has been processed, the next block on the disk is read and so on. When the last record (the 10,000th) from the last block is processed, the second part of the program comes into action; that is, an address search begins in the second file (text file) using the addresses found at the first stage. As we have not yet considered the random access file, for the time being we will not discuss the second part of the program.

Thus, we considered the methods of increasing search speed when using the sequential file, which, it should be noted, has a number of advantages over other file organizations (though it has some shortcomings too). The following list presents just a few of these advantages:

1. The sequential file is the most simple and convenient file organization to use.
2. The sequentially organized file takes less space in secondary storage than what would be needed by the same amount of information organized in another manner.
3. Any sorting, which is often needed and important in information retrieval, is feasible if the file being sorted has sequential organization.

Its most serious drawback is the fact that there is no random access in a sequential file in order to retrieve a particular record without reading all the records in the file that precedes it. This means that whenever we need access to only a few records in a sizable file (consisting of 10,000 records in our case), the whole system's performance drastically plummets and in some cases becomes unacceptable. Therefore, we will now consider the most simple file organization allowing for random access. Though being "the most simple," this file organization is sufficient enough to provide the main ideas about the basic techniques used in implementing the BSR structure.

8.4

Relative File and Random Access

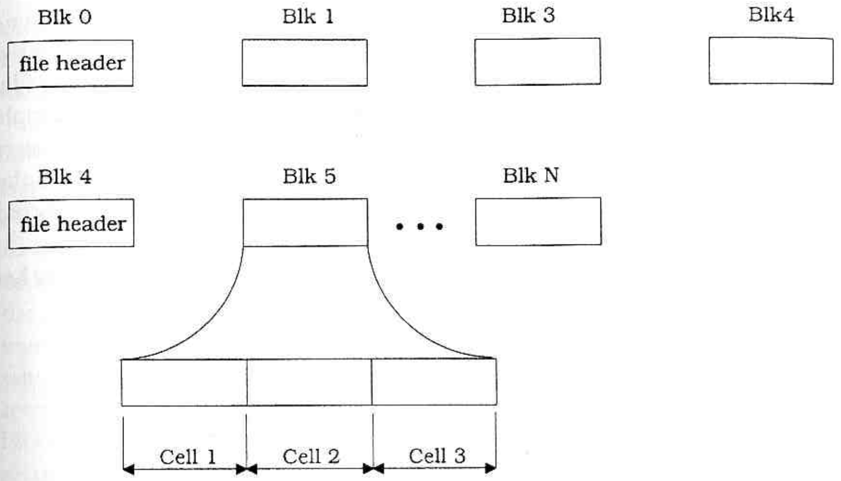

Figure 8.4
Relative file structure.

In mentioning the shortcomings of sequential files we noted that it is impossible to randomly access individual records without reading all records preceding it in the file. It is also true that records can be added only at the end of the file, which lengthens the access to the most recent records. We can design a file organization that has functionality of a sequential file but eliminates the problems that are inherent in sequential file design. This file type is known as the *relative file organization.* The relative file is one in which records can be accessed directly by using their record numbers as an external key (address). A relative file is made up of fixed-length blocks. Inside each block are fixed-length cells. Each cell can hold just one user data record and each cell must fit completely within a block. A relative file is illustrated in Figure 8.4. Indeed, cells are fixed-length containers within a block, much as an egg carton has a dozen fixed compartments each capable of holding only one egg. The cell size is based on the maximum-size user data record. Therefore, any cell can hold a valid data record of any size.

Each data record must have associated with it an integer key value (ordinal number) that will be used both to add the record into the file and to read the record out of the file. Later we will show how the access method converts the integer key (ordinal number) into a precise block and cell number within the entire file. Note, however, that the integer key is not physically part of the data record itself. Thus the responsibility falls on the first part of the search program (see our example) to make the correct association between data record and the record's ordinal number (key value).

The first block of the relative file is the *file header block.* It contains information that allows the relative file access method to process the data records correctly within the file. As mentioned earlier, the key to locating or addressing an individual record is the record's own record number, which is unique within