Indexes and Indexing Mechanisms

- **So far we talked about how records can be stored in a file and manipulated, and the different file structures that were explored in the past for maintaining records.**

- **Once we have stored records in a file, we need an efficient mechanism for accessing, sorting, searching, and manipulating records.**

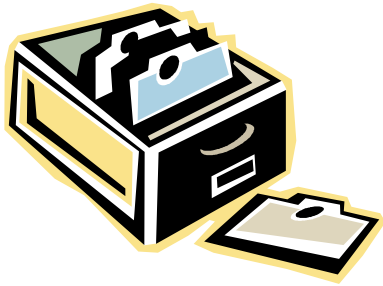
- **There are a number of techniques available for this purpose and each technique must be evaluated on the basis of:**

  Access Time – the time it takes to find a particular record.

  Insertion Time – the time it takes to insert a new data item. This includes the time to find the place to insert the new record and update pointers, etc.

  Deletion Time – this includes the time to find the record and delete it by updating the pointers.

  Space Overhead – the additional space required to maintain and manipulate pointers.



- **Similar to catalogs we have in the library to find where the books are stored, we need a catalog for finding where the records are stored.**

- **Such a catalog is called an "index".**

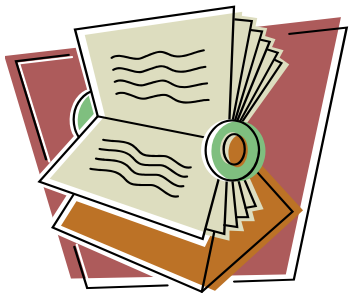
- **An index is a data structure (piece of data) that allows quick access to a record in a data file**

- **Indexes and indexing mechanisms are important to know as they affect database design**

Indexes and Indexing Mechanisms (Cont.)

- **In order to allow fast access to records in a file, an index structure is necessary.**
- **Each index structure is associated with a particular search key.**
- **A search key is a field or set of fields used to look up records in a file quickly (we will define search keys later but for now think of it as your student ID or social security number in the student database system)**
- **Indexes contain the search key field and the address of a record containing that field (similar to table of contents in a book).**
- **Indexes are also kept in a file and loaded onto main memory when a database is opened and accessed.**
- **One of the indexing structures is called the Sequential Index in which the indexes are maintained in a sequential order.**
- **There are two types of sequential indexes**

  1. **Dense Index**
  2. **Sparse Index**

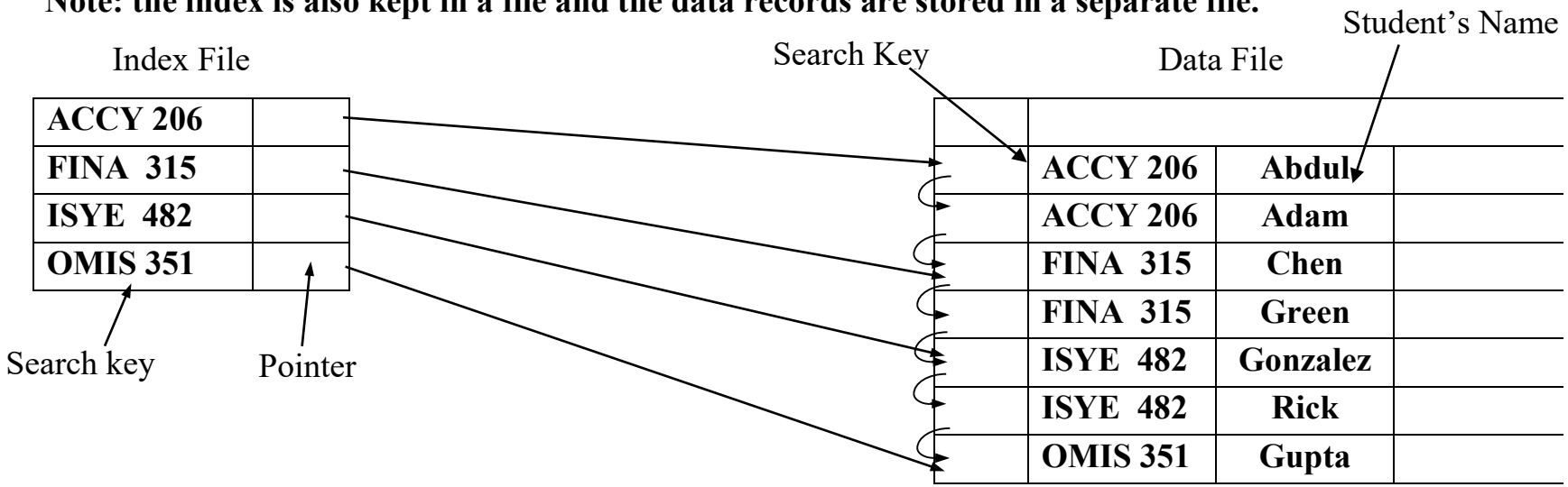


- **In a dense index structure, an index record appears for every search key value in the data file.**

  The index record contains the index or search key value, and a pointer to the record in the data file.

Indexes and Indexing Mechanisms (Cont.)

Note: the index is also kept in a file and the data records are stored in a separate file.



- **The above example illustrates a dense index structure where an index record appears for every search key value in the data file.**

  The search key values in the index file are ordered sequentially in ascending order of key values.

  The data records associated with a particular search key value are also maintained in a sequential order. For example, ACCY206 has two records linked sequentially according to ascending order of names.

  Note: the index records in the index file are not chained together with pointers.

  All the data records in the data file are linked together with pointers and are maintained in a sequential order. The pointers linking empty records to the header record are not shown in this figure.

Indexes and Indexing Mechanisms (Cont.)

- **Accessing records in a dense index structure:**

  Suppose we want to find the names of all students enrolled in ISYE482, how can we find them?

  First we will search the index file by checking each index record sequentially and comparing the index value to "ISYE482" until we find the index record with the search key value "ISYE482".

  Next we will follow the pointer from that index record containing the key value ISYE482 to the data file where we will retrieve the names of students in records with the key value "ISYE482".

  Note: the index record and the data record should contain the same search key value. Otherwise we cannot locate the record quickly.

- **How can we go about retrieving all the records in the data file?**

  For this we will only have to access the first index record, i.e. ACCY 206, and follow its pointer to the data file and retrieve all the records sequentially in the data file.

- **Deleting records in a dense index structure**

  In order to delete a record, we have to find the record first and then we will have to delete it.

  There are several scenarios possible in deleting records within a dense index structure.

Indexes and Indexing Mechanisms (Cont.)

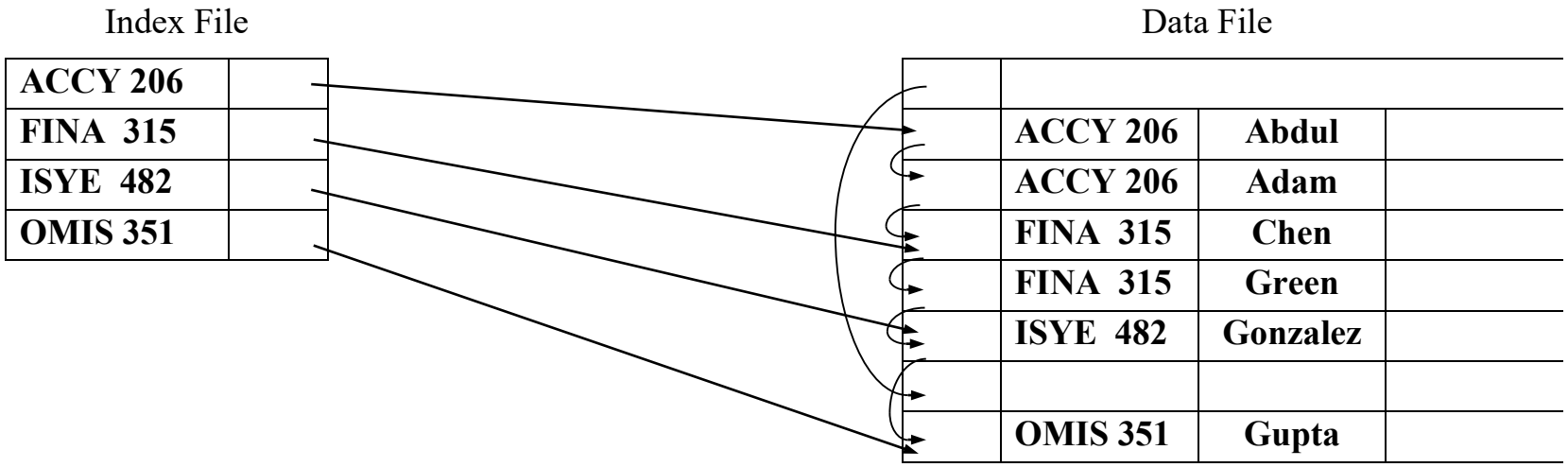
1. **More than one record exists in the data file for an index**

   Example: delete (ISYE482, Rick, …). There are two records with the index value ISYE482

   This requires first searching the index file to find ISYE482 and following its pointer to the data file.

   Next finding the record with key value ISYE482 and then the record with the name Rick.

   Deleting this record will only require manipulating the pointers in the data files as shown on the figure below:



Note: The pointer from the header record points to the deleted record to show that record is empty.

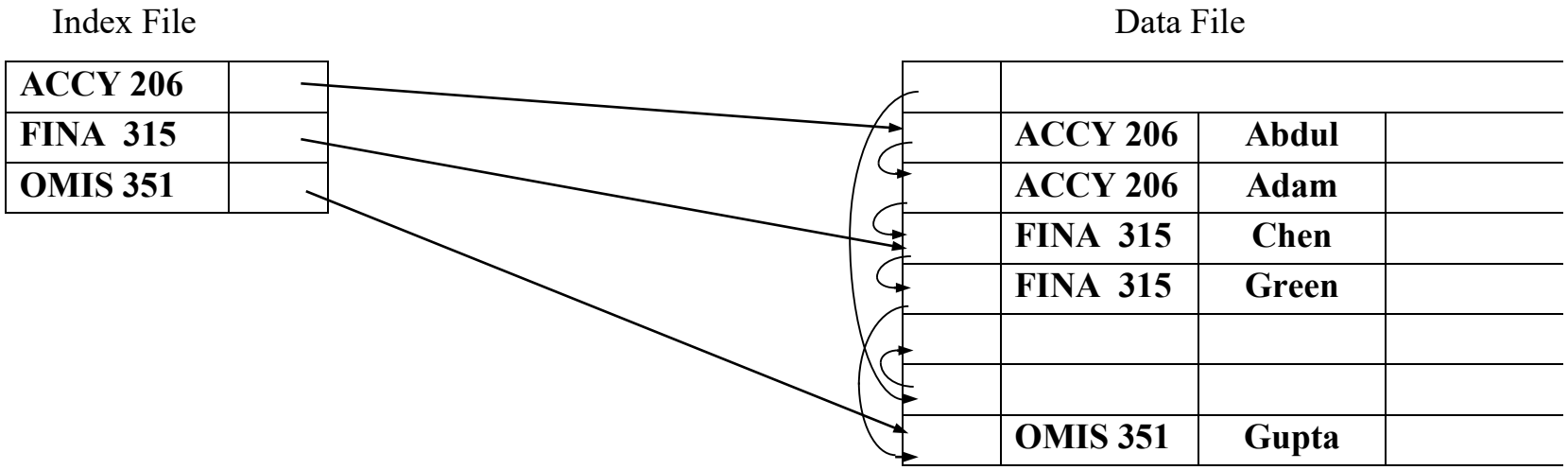
Indexes and Indexing Mechanisms (Cont.)

2. **Only one record in the data file for the index**

   Example: Delete (ISYE482, Gonzalez, …) from the file structure shown in the previous page.

   This requires deleting both the index record and the data record since there is only one data record with the search key value "ISYE482".

   The resulting index file and data file structures are as shown below:



   Note the following:

   The index records have to be moved up to maintain a sequential order since the index file does not have pointers. For the same reason, you cannot leave empty index records within an index file.

Indexes and Indexing Mechanisms (Cont.)

If the last record in the index file has to be deleted, then it can be deleted easily or be replaced with null symbols to show the record is empty.

If the first record in the index file is to be deleted, then the records below have to be moved up and this can be done by recopying the records into a new file and copying it back into the index file.

- **Inserting a new record into a dense index structure**

  There are two scenarios possible in the case:

  1. **Insert a new record into the data file for an existing index.**
  2. **Insert a new record into data file for a new index.**

     (i) Insert the index record at the end of the index file.

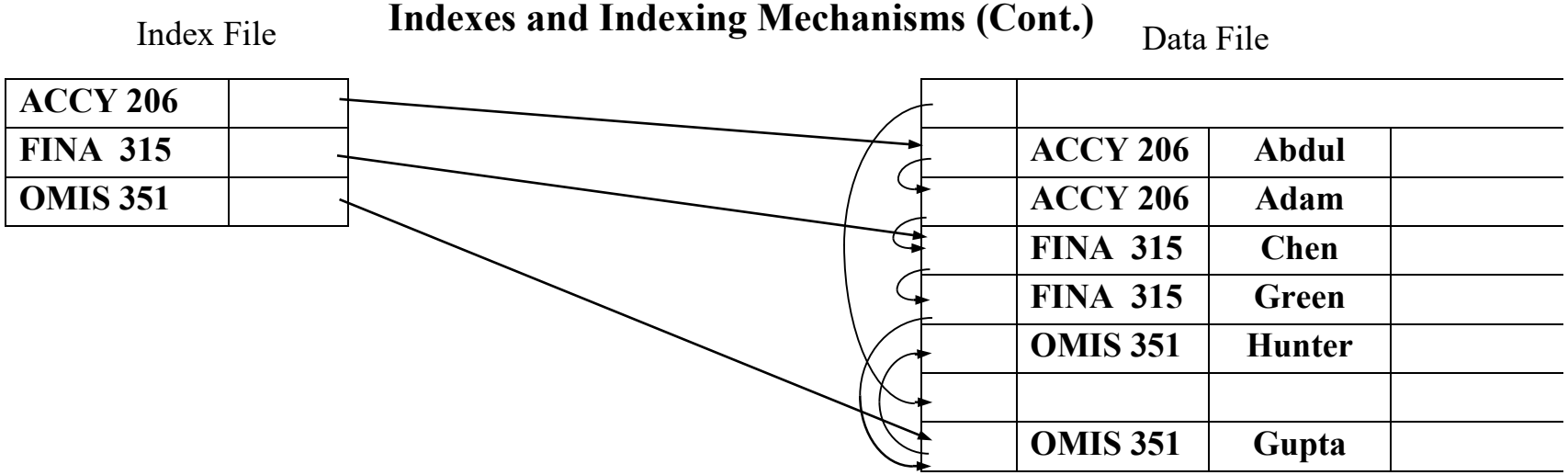
     (ii) Insert the index record at the middle of the index file.

     (iii) Insert the index record at the top of the index file.

- **Example: Insert (OMIS351, Hunter, …) in the file structure shown in the previous page.**

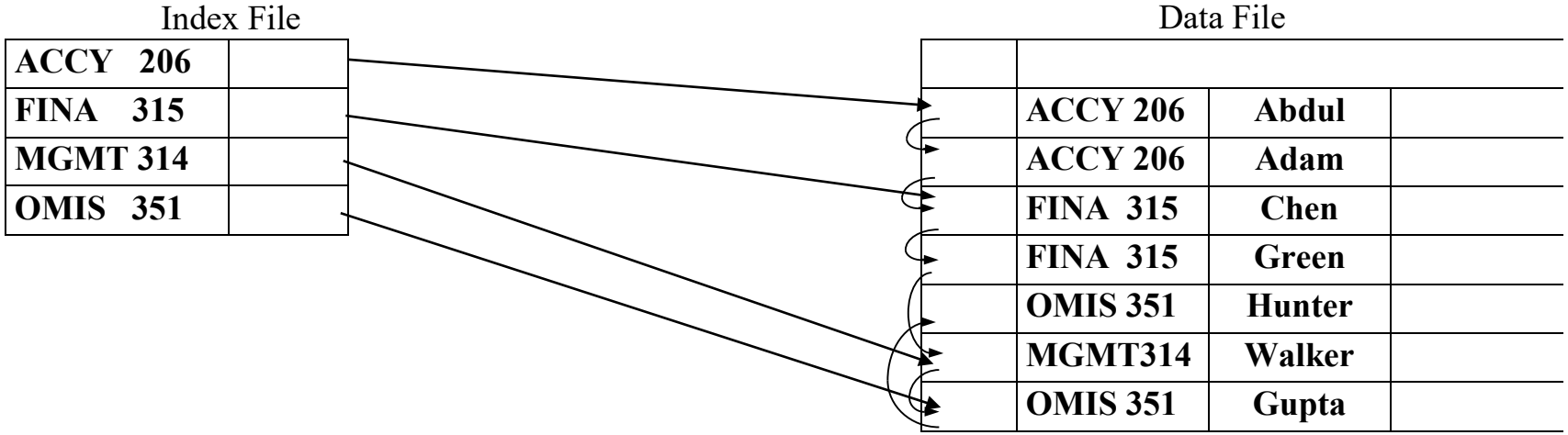
  The resulting structure would look as follows:

Indexes and Indexing Mechanisms (Cont.)



Note: The new record has been inserted into a free record space pointed to by the header record. The records within OMIS351 are sequenced using pointers.

- **Example: Insert (MGMT314, Walker,…)**



Indexes and Indexing Mechanisms (Cont.)

Note: The new record for "MGMT 314" has been inserted in place in the index file. The data records have been sequenced using pointers.

- **What would happen if we do the following:**

  insert (TECH360, Thomas, …)

  insert (ABCD100, Sanchez, …)

- **In a sparse index structure, index records are created for only some of the search key values.**

That is, in a sparse index structure, the index file is sparsely populated.

How sparsely the index file is populated is decided by the software developers and not the user.

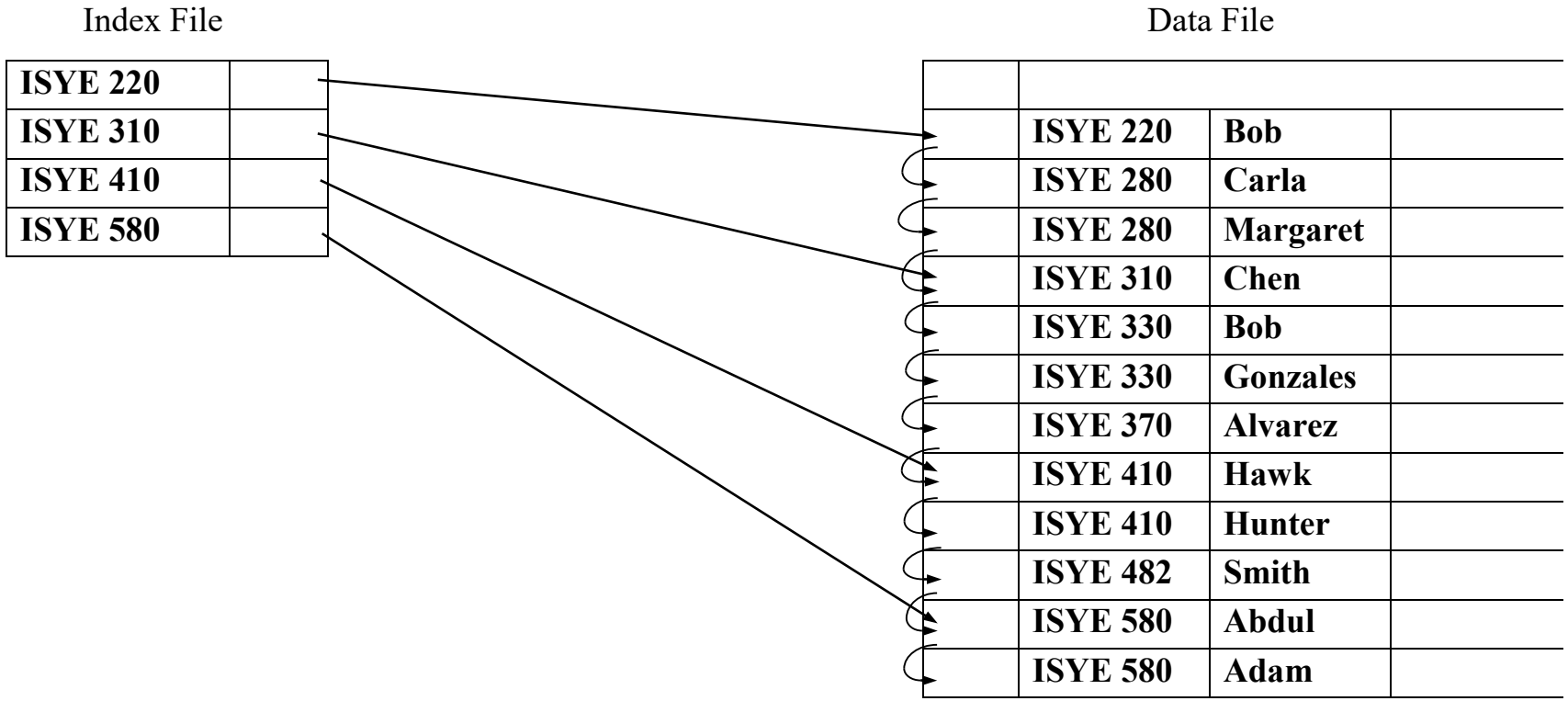
In this course, we will apply some common sense rules to decide how sparse is sparse.

Similar to the dense index structure, the index file is maintained in a separate file for the sparse index.

The index file is maintained in sorted order and does not contain pointers to sequence index records.

The data records are maintained in a separate file and the data records are sequenced using pointers.

Indexes and Indexing Mechanisms (Cont.)



- **Accessing records in a sparse index structure**

  Suppose we want to find the names of students enrolled in "ISYE 310", how would we go about doing it?

  First we will search the index file and find if the index for "ISYE 310" exists.

  Then if the index is there, we will follow its pointer to the data file and retrieve all the records with the key value "ISYE 310".

Indexes and Indexing Mechanisms (Cont.)

Suppose we want to find the names of all students enrolled in "ISYE 330", how can we do this?

When we search the index file we will find that the index for "ISYE 330" does not exist and if it does, it should exist between "ISYE 310" and "ISYE 410".

We will have to follow the pointer for the index with a value lower than "ISYE 330" and proceed to the data file.

In the data file, we will search each record beginning with records that contain "ISYE 310" as the key value and reach those with the key value "ISYE 330".

Then we will retrieve all the records that contain the key value "ISYE 330" one by one.

- **Deleting records in a sparse index structure has several possible scenarios:**

1. **Index exists in the index file and more than one record exists in the data file with the required key value.**

2. **Index exists in the index file and only one record exists in the data file with the required key value.**

3. **Index does not exist in the index file and more than one record exists in the data file with the required key value.**

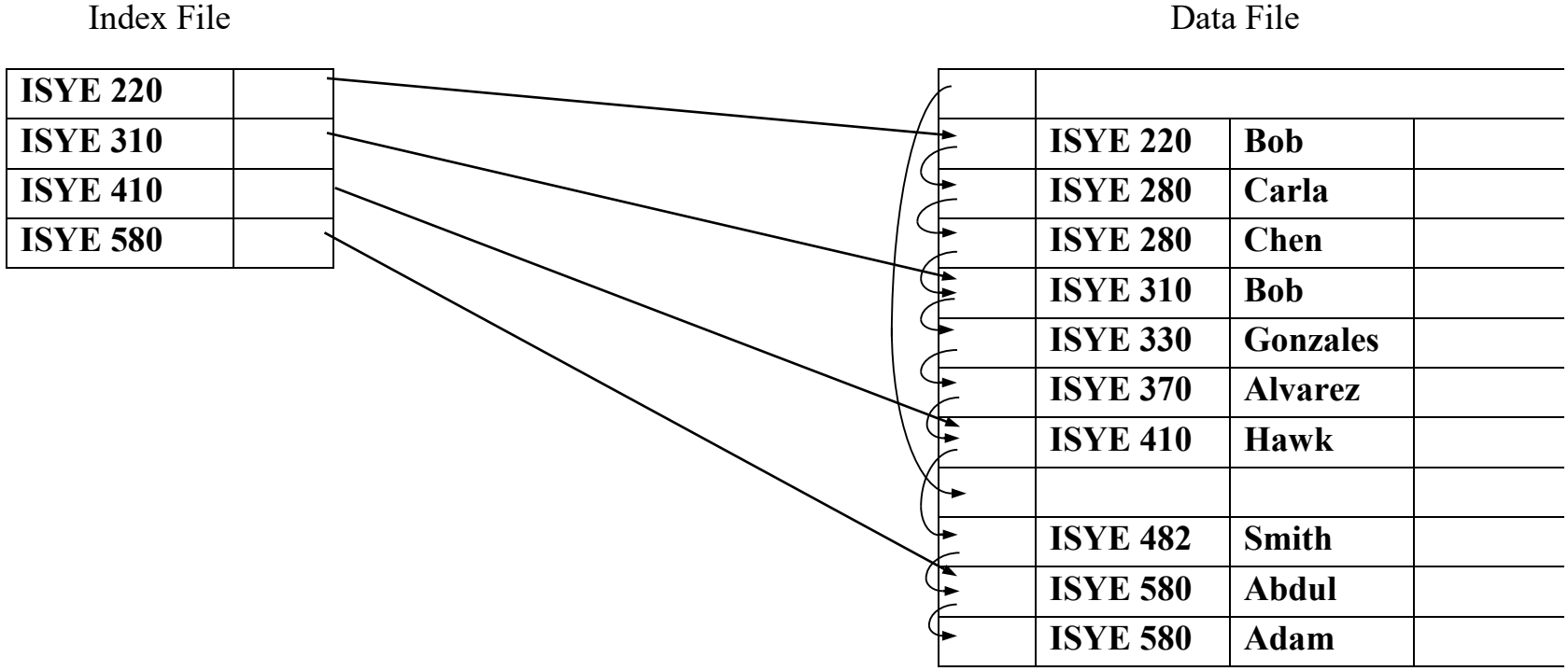
4. **Index does not exist in the index file and only one record exists in the data file.**

Indexes and Indexing Mechanisms (Cont.)

- **Suppose we want to delete (ISYE 410, Hunter, …) from the previous structure how would we go about doing it?**

  First we will have to access the record for (ISYE 410, Hunter, …) as we described in the procedure for accessing a particular record.

  Then we will have to delete the required record by updating its pointers as shown in the figure below:



Indexes and Indexing Mechanisms (Cont.)

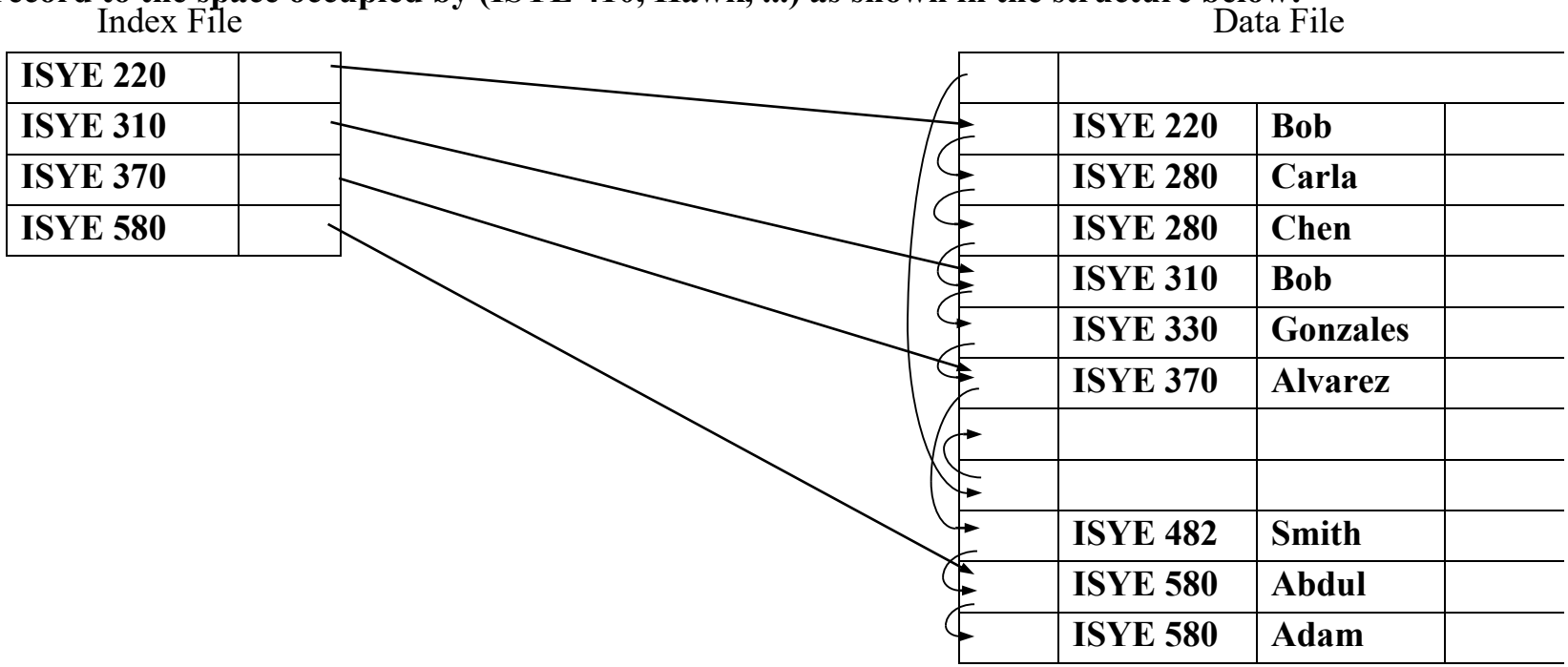
- **Suppose we want to delete (ISYE 410, Hawk, ...) from the previous structure.**

  This record is pointed to by the index record and there is only one record exists in the data file.

  Therefore, if we delete the data record we may also have to delete the index record.

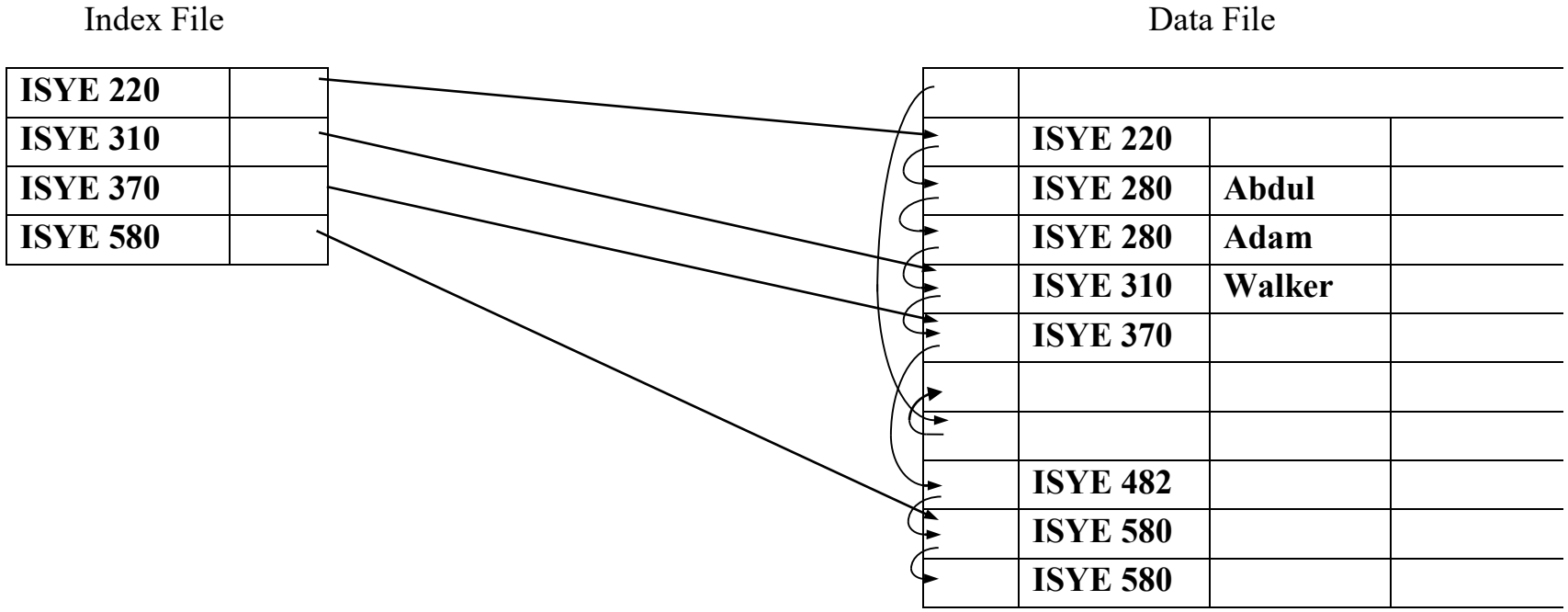
  If we delete the index record, we will have to do something with that index record space.

  The simplest solution would be to copy the index value of the previous record ISYE370 or the succeeding record to the space occupied by (ISYE 410, Hawk, ...) as shown in the structure below:



Indexes and Indexing Mechanisms (Cont.)

- **Index does not exist in the data file and more than one record exists in the data file with the required key value.**



Delete (ISYE 280, Adam, ...) from the above structure (not all names have been shown in the data file)

If we delete this record, the pointer from (ISYE 280, Abdul, ...) should be linked to (ISYE 310, Walker, ...) and the deleted record should be linked to the chain of deleted records.

Now if we delete (ISYE 280, Abdul, ...) it would be done in a similar manner to the previously deleted record and there will be no effect on the index file.

Indexes and Indexing Mechanisms (Cont.)

- **Inserting records into a sparse index file structure**

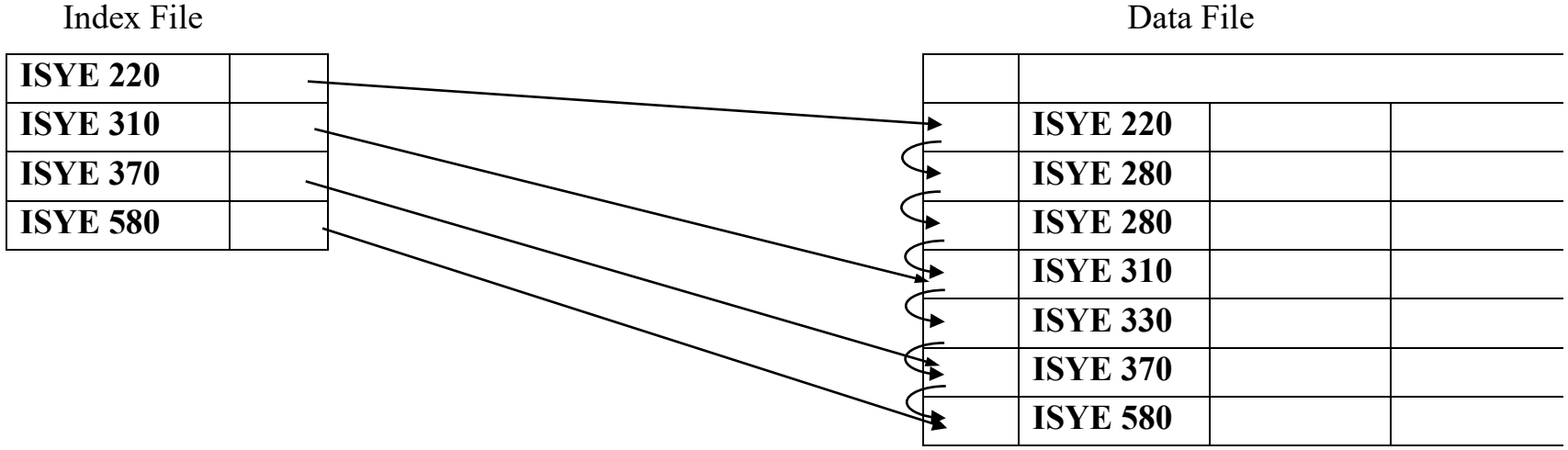
  If a new record has to be inserted into a sparse index structure, no change needs to be made in the file unless a new index is created.

  The first search key value appearing in the new block will be inserted into the index file.

  Case 1: insert new record for an index value that exists in the index file.

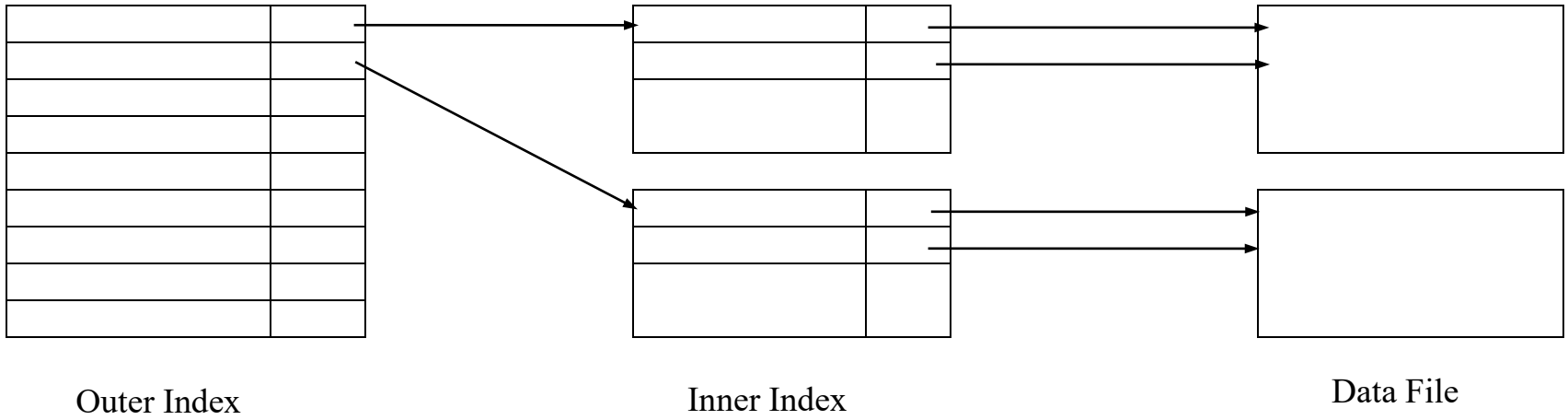
  Case 2: insert new record for a new index in the index file, using a new block.

  Inserting a new record for an existing index is simple. But if the block is full, we will have to start a new block for creating a new record for a new index and this can be cumbersome.



Indexes and Indexing Mechanisms (Cont.)

- **Two Level Sparse Index**



Two level sparse indexes can be used to speed up the search process, but may require more overhead as shown in the figure above.

Remember!

- **After performing a series of insertion and deletion operations, do not sort the final set of data records manually and arrange them in a sequence from top to bottom in a file. This is a mistake!**
- **Sequencing of records in a file should be done using pointers, and the final set of data records after a series of insertion and deletion operations may not be visually appealing but it is not a problem.**
- **In reality the record operations are done using database software and not done manually!**

Indexes and Indexing Mechanisms (Cont.)

- **Degree of Inversion**

  Refers to the extent to which search key values and pointers to corresponding records are placed in the index.

  The higher the degree of inversion, the larger the number of field values inverted (or indexed).

  100% inversion means that every field has been inverted or placed in the index file.

  0% inversion means only one primary index, i.e. only one field has been placed in the index file.

- **Difference Between Dense and Sparse Indices**

Dense	Sparse
Records can be accessed quickly.	Records access can be relatively slow compared to dense index.
More space required since all key values have to be maintained in the index file.	Less space is required since the index file is sparsely populated.

- **Why every field in a record should not be defined as a search key in a database?**

- **Disadvantages of Sequential Index Structures**

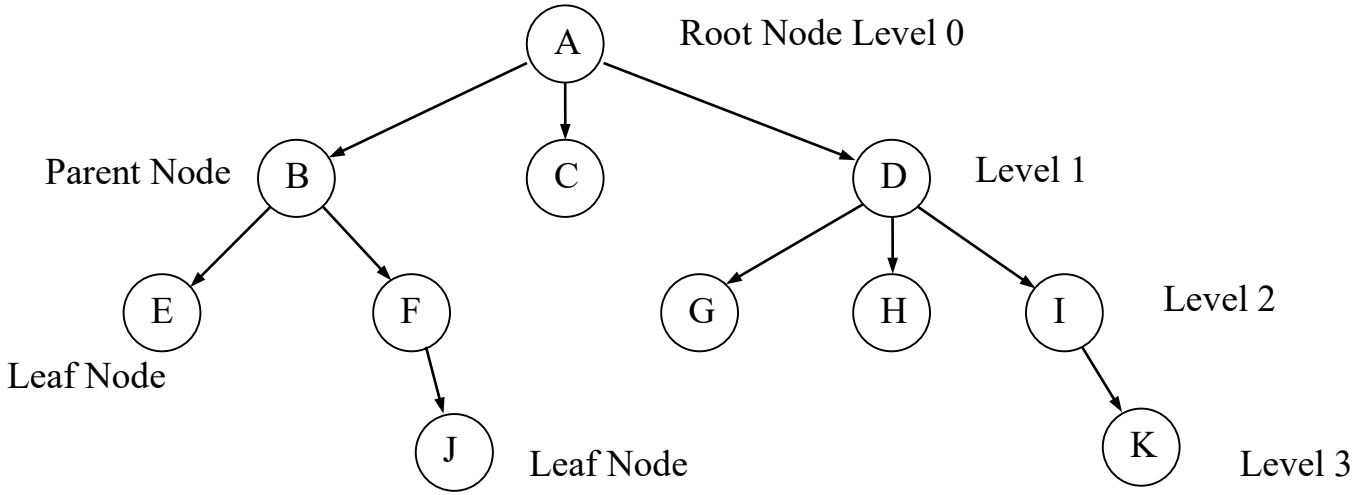
  1. Index file is difficult to maintain insertion and deletion are complicated due to lack of pointers.

  2. Sequential accessing of records slows processing.

Indexes and Indexing Mechanisms (Cont.)

- **To overcome the disadvantages of sequential index file structures, a tree type indexing scheme was developed**

- **Tree Data Structure**



E, J, C, G, H, K are called leaf nodes.

B, F, A, D, I are called parent nodes or non-leaf nodes.

A non-leaf node is also called an internal node.

A root node is also a parent node and a non-leaf node.

