The RandomAccessFile

There's another way to access data from a file, and this is with a **`RandomAccessFile`**.

This class provides the ability, to directly access and modify data, at any specific location within the file.

A random access file behaves like a large array of bytes, stored in the file system.

There's a kind of cursor, or index into the implied array, called the ***file pointer***.

A RandomAccessFile both reads and writes binary data, using special methods, which keep track of how many bytes will be read or written.

This class can be used for both read and write operations.

The RandomAccessFile File Pointer

When you open a RandomAccessFile, it's file pointer is at 0, or the start of the file.

To move the file pointer, you execute a method on the file, called seek, passing it a long value, the position in the file, you wish to go to.

To get the file pointer, you execute `getFilePointer`.

Depending on the type of read or write method you're using, the file pointer will move a certain number of bytes when these operations complete.

There are a lot of these methods, and I'll show them to you, by pulling up the Java API documentation, for this class's methods.

Still confused about why you'd use this?

Let's say you have a file with many millions of records, and at any one time, you really need to access about 50 of those.

Instead of loading a million records into memory, you can load a simple array or small map, which will tell you how to locate records of interest in the big file.

You wouldn't want to start reading from the beginning of the file, and read 10 million records, checking each one to see if it's a match.

The RandomAccessFile lets you fast forward or backward, to a position in the file, using the seek method.

From this position, you can read in only the data that matters, for your application.

To do this though, you need to understand how many records are in your file, what it's record length is, and how you want to identify each record, to retrieve it.

Understanding the Random Access File's index

A RandomAccessFile needs an index, which houses a file pointer position, to each record of interest.

This index could be implied, for a file with fixed width records, if you only need to get data by a row number, for example.

This means it's very easy to do a little math to get the 10 thousandth record, when all the records are 250 characters in length.

10,000 * 250 will point you to the 10,000th record in your file.

Understanding the Random Access File's index

It's much more common though, to retrieve records by a non-sequential id, than a row id.

For this, you'll need an index, that will contain this id, and the position in the file of the record associated with that id.

For fixed width records, your index wouldn't need the file pointer position, just an association between the row id and the record id.

This index might be an array of record IDs for example, in row id sequence.

Understanding the Random Access File's index

On this slide, you can see that record id 10000 is the first record in the file, so it's at index 0.

The highlighted rows on this table, represent the indexed data you'd need, to locate your records by record id.

Row Index	Record Id	Position in File	Fixed Size Record (250 chars)
0	100000	0	First Record
1	1	250	Second Record
2	543210	500	Third Record
3	777	750	Fourth Record

Understanding the Random Access File's index

For variable length records, the row id alone, isn't enough information to calculate the file pointer.

You could store the length of each record, or you could just store the starting file pointer position.

It's more common to do the latter.

Understanding the Random Access File's index

Again, the indexed information is represented by the highlighted data, shown in the table on this slide.

Your index should store the record id, and its associated file pointer.

Index	**Record Id**	**File Pointer**	**Variable Size**
0	100000	0	First Record (50)
1	1	50	Second Record (250)
2	543210	300	Third Record (150)
3	777	450	Fourth Record (500)

Where's the index?

In the case of a fixed width file, it may not exist.

If it does, then it will be in the same place as an index for a variable width file.

This may be at the beginning of the data file, before the record data.

It may be at the end of the data file, so after all the record data.

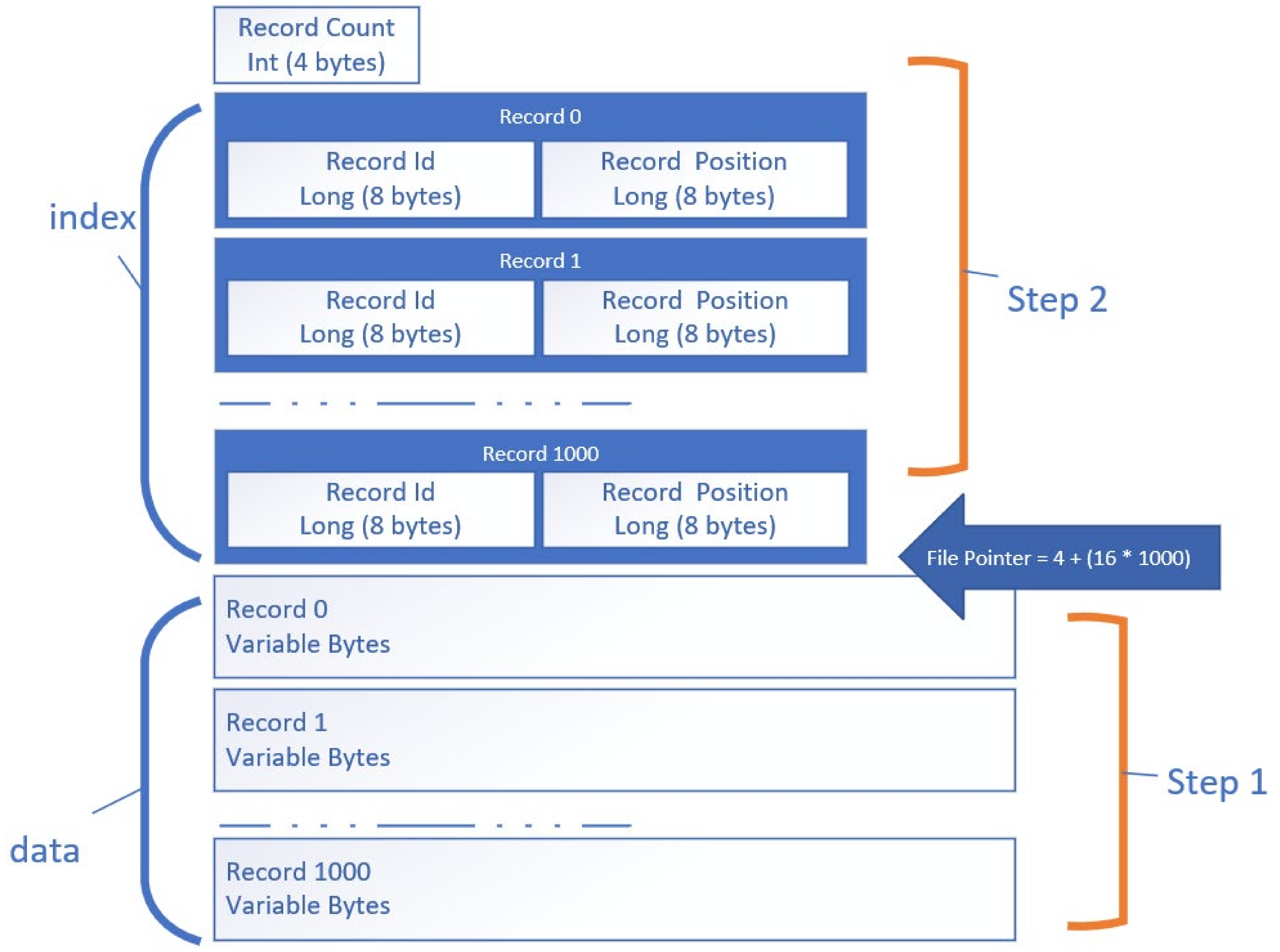
Or the indexed data may be in a separate file altogether.

The .dat file, with an index

This slide represents what the file will look like once it's been completely generated.

I won't be creating this file in top down order however.

In fact, Step 1 will consist of writing the records to the output file, at the file pointer position shown by the arrow.



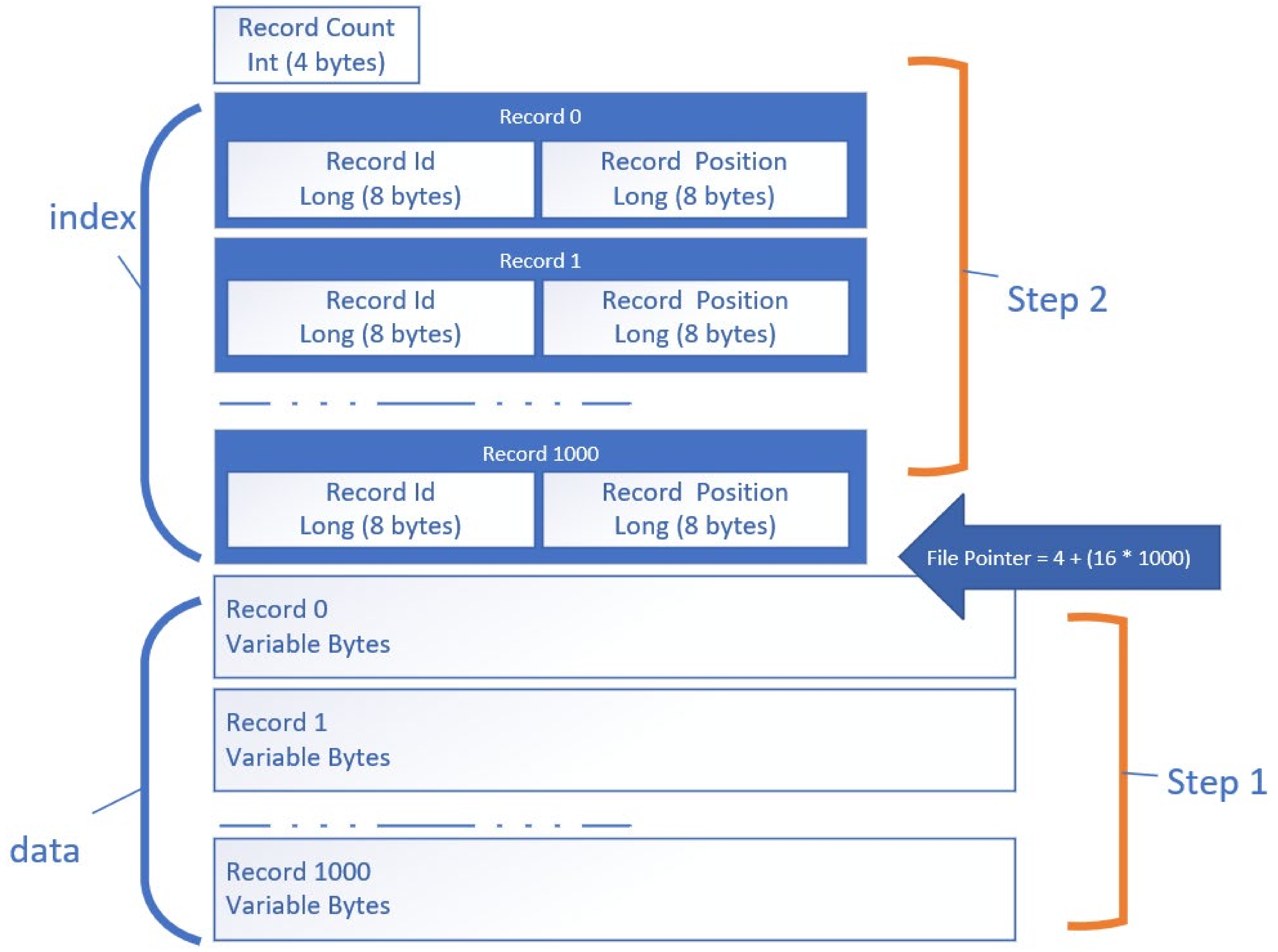
The .dat file, with an index

I'll be keeping track of the file position of each record with my map, as I write out each of the records.

Once I have all the records inserted, and the index map complete, I'll write the total number of records at position 0.

Then I'll start outputting each index key value pair.

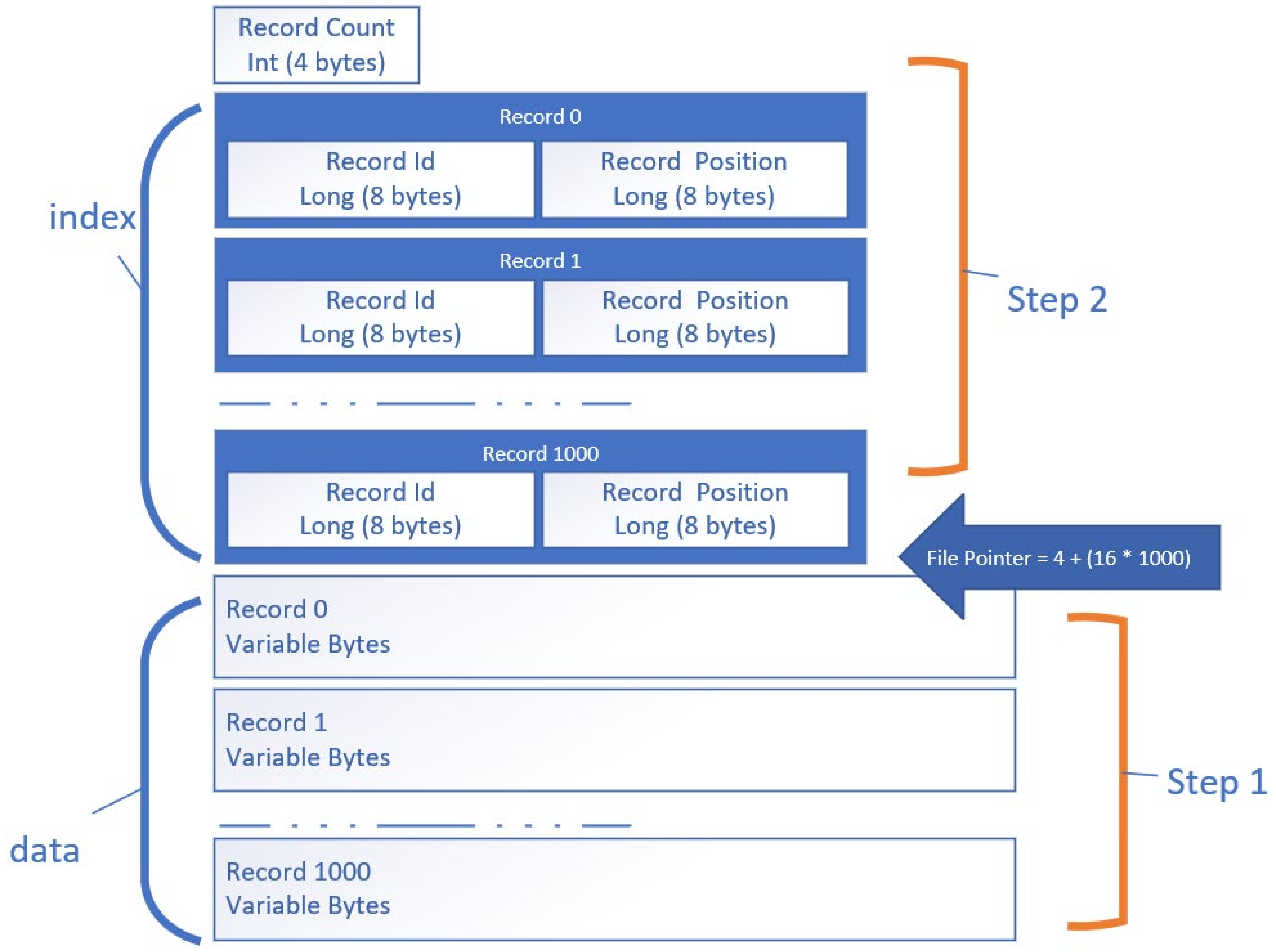
This is Step 2.



The .dat file, with an index

I'm able to start writing my records in step 1, before I insert the index, because I'm using a random access file, and I can move to the position I want to write to.

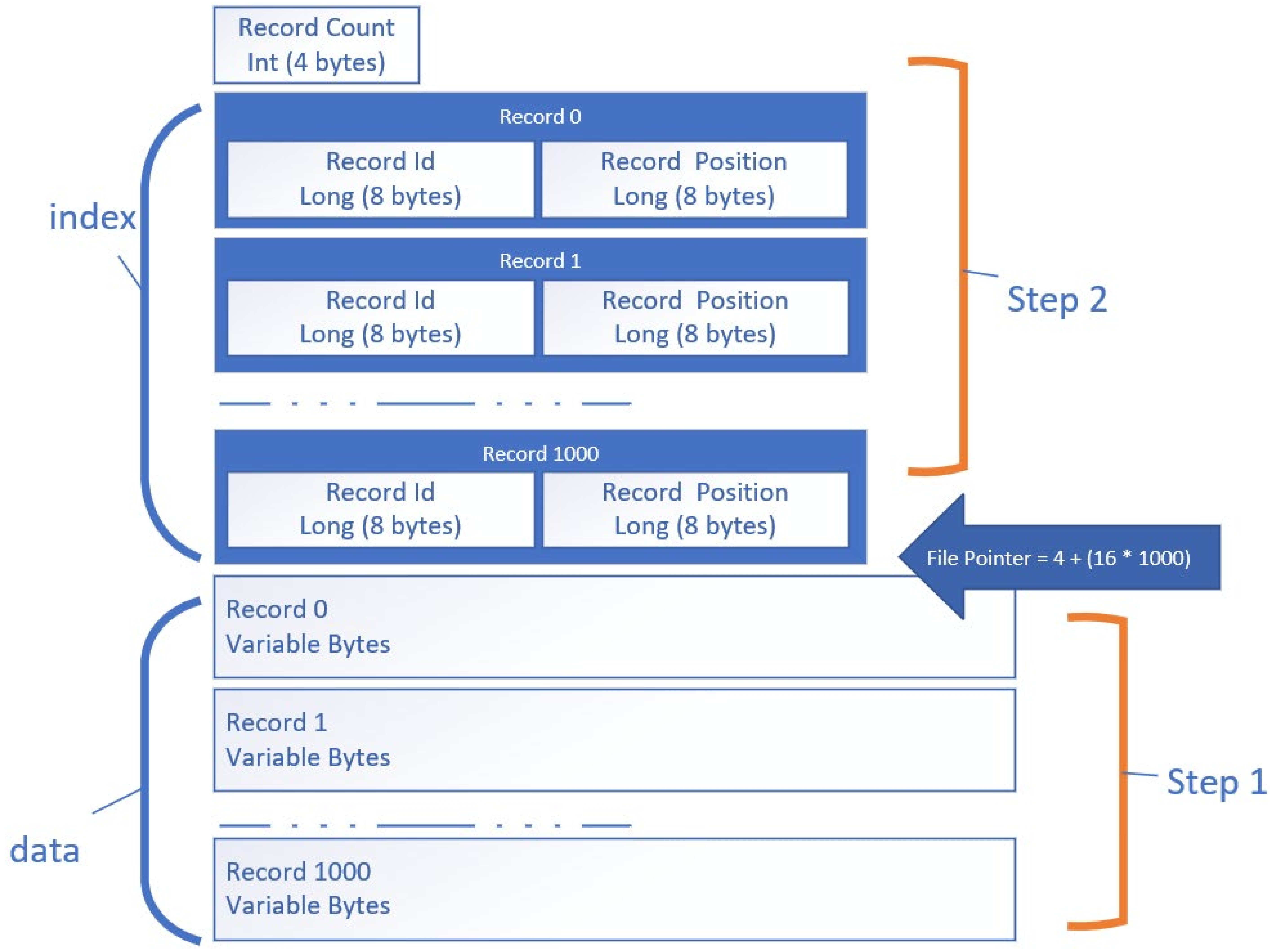
I can derive the position to start, by calculating that I'll use 4 bytes to store the record count, which I'll output as an integer.



The .dat file, with an index

Then I'll have 16 bytes for each indexed entry, because each long takes up 8 bytes.

I can multiply that by the number of records, so 1000 * 16, then plus 4, will be my file pointer position, to start writing the records.


Record Count
Int (4 bytes)
index
Record 0
Record Id
Long (8 bytes)
Record Position
Long (8 bytes)
Record 1
Record Id
Long (8 bytes)
Record Position
Long (8 bytes)
Record 1000
Record Id
Long (8 bytes)
Record Position
Long (8 bytes)
Step 2
File Pointer = 4 + (16 * 1000)
data
Record 0
Variable Bytes
Record 1
Variable Bytes
Record 1000
Variable Bytes
Step 1