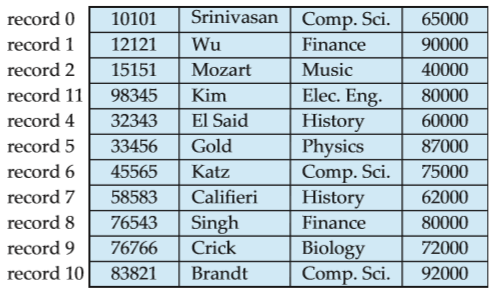
**10.4 Consider the deletion of record5 from the ﬁle of Figure 10.6. Compare the relative merits of the following techniques for implementing the deletion:**

<Figure 10.6>

a.Move record 6 to the space occupied by record 5, and move record 7 to the space occupied by record 6.

When we pull all records at record 6~10 to former record’s space, it will be too costly deletion. However it will not needed free record lists.

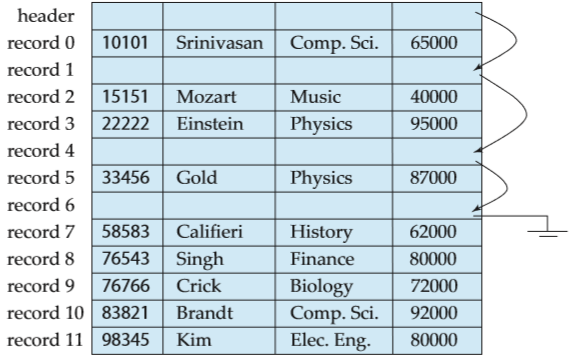
b.Move record 7 to the space occupied by record 5.

Record 7 will be duplicated at line 5 and line 7.

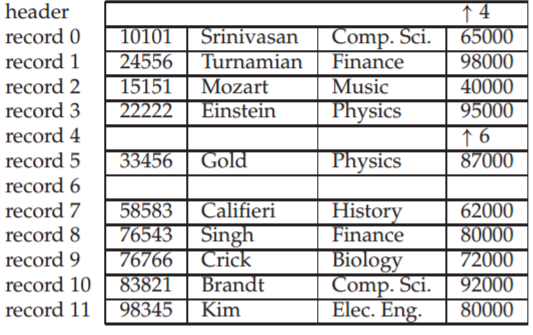
c.Mark record 5 as deleted, and move no records.

We have to remember and manage record 5 as a freelist. However there will not be costly deletion.

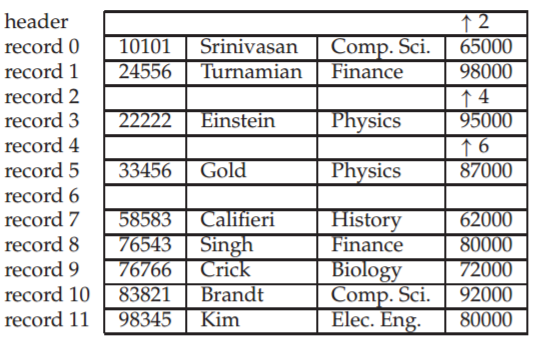
**10.5 Show the structure of the ﬁle of Figure 10.7 after each of the following steps:**

<Figure 10.7>

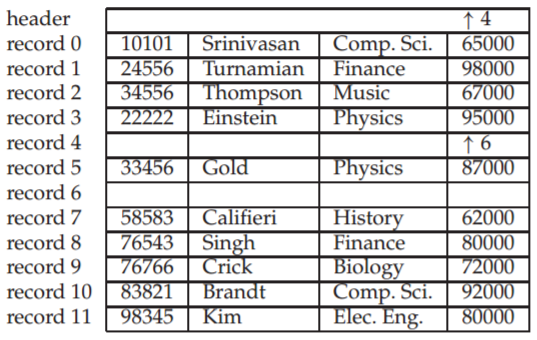
a. Insert (24556, Turnamian, Finance, 98000).



b. Delete record 2.



c. Insert (34556, Thompson, Music, 67000).



**10.14 In the variable-length record representation, a null bitmap is used to indicate if an attribute has the null value.**

a. For variable length ﬁelds, if the value is null, what would be stored in the offset and length ﬁelds?

It does not matter on what we store in the offset and length fields since we are using a null bitmap to identify null entries. But it would make sense to set the offset and lengh to zero to avoid having arbitrary values.

b. In some applications, tuples have a very large number of attributes, most of which are null. Can you modify the record representation such that the only overhead for a null attribute is the single bit in the null bitmap.

We would be able to locate the null bitmap and the offset and length values of non-null attributes using the null bitmap. This can be null attributes, store the value (for fixed size attributes), or offset and length values (for variable sized attributes) in the same order as in the bitmap, followed by the values for non-null variable sized attributes. This representation is space efficient but needs extra work to retrieve the attributes.

**10.19 Give a normalized version of the Index metadata relation, and explain why using the normalized version would result in worse performance.**

Normalized version of index metadata relation needs many separated freelists. So we have to manage a lot of freelist tables consequently.

**10.20 If you have data that should not be lost on disk failure, and the data are write intensive, how would you store the data?**

I would store data at flash memory. Because flash memory is non-volatile and can fast accessed.