DBMS Tutorial 14.11.2018

Topics

Merge Sort Exercise

Storing Data

Merge Sort Exercise 1

You have an unsorted file containing 4500 records and a select query is asked that requires the file to be sorted. The DBMS uses an external merge-sort that makes efficient use of the available buffer space.

Assume that: records are 48-bytes long (including a 4-byte sort key); the block size is 512-bytes; each block has 32 bytes of header information in it, rest can be used for records; 4 buffer blocks are available.

1. How many sorted sublists will there be after the initial pass of the sort algorithm? How long will each sublist be?
2. How many passes (including the initial pass considered above) will be required to sort this file?
3. What will be the total I/O cost for sorting this file?
4. What is the largest file, in terms of the number of records, that you can sort with just 4 buffer block in 2 passes? (Hint: use (M/R)*[(M/B)-1] and plug in values in terms of blocks)

Merge Sort Exercise1 - Answers

You have an unsorted file containing 4500 records and a select query is asked that requires the file to be sorted. The DBMS uses an external merge-sort that makes efficient use of the available buffer space.

Assume that: records are 48-bytes long (including a 4-byte sort key); the block size is 512-bytes; each block has 32 bytes of header information in it, rest can be used for records; 4 buffer blocks are available.

1. How many sorted sublists will there be after the initial pass of the sort algorithm? How long will each sublist be? **A. 113**
2. How many passes (including the initial pass considered above) will be required to sort this file? **A. 6**
3. What will be the total I/O cost for sorting this file? **A. 5400**
4. What is the largest file, in terms of the number of records, that you can sort with just 4 buffer block in 2 passes? (Hint: use (M/R)*[(M/B)-1] and plug in values in terms of blocks). **A. 120**

Representing Data Elements

How is data stored?

How are records and relations (aka tuples and files) stored?

How are blocks arranged?

Record Header and Block Header

Schema

Spanned and Unspanned records

Sequencing

Representing Data Elements

- **Fields**: Attributes of relational tuples represented by sequences of bytes called fields (eg EmpName(String), EmpID(int))
- **Records**: Fields grouped together into records. Records are stored in blocks.
- **Record schema** (or type): sequence of field names and their corresponding data types
- **File**: collection of blocks that forms a relation
  - i.e., File: collection of records with the same schema (typically), usually spanning a number of blocks
- Storing blocks of the same file speeds up data access by eliminating some seeks and rotational delays. **Block access is a cost unit for file operation(I/O)**

Representing Data Elements

It is normal for a disk block to hold only elements of one relation, although there are organizations where blocks hold tuples of several relations (*clustering*).

Ultimately everything is stored in bits and bytes, and to find the right block, record or field we use size information and pointers.

Attributes and Record Schema

A schema contains information such as:

- Number of fields (attributes)
- type of each field (length)
- order of attributes in record

The schema is consulted when it is necessary to access components of the record.

Attributes and Record Schema

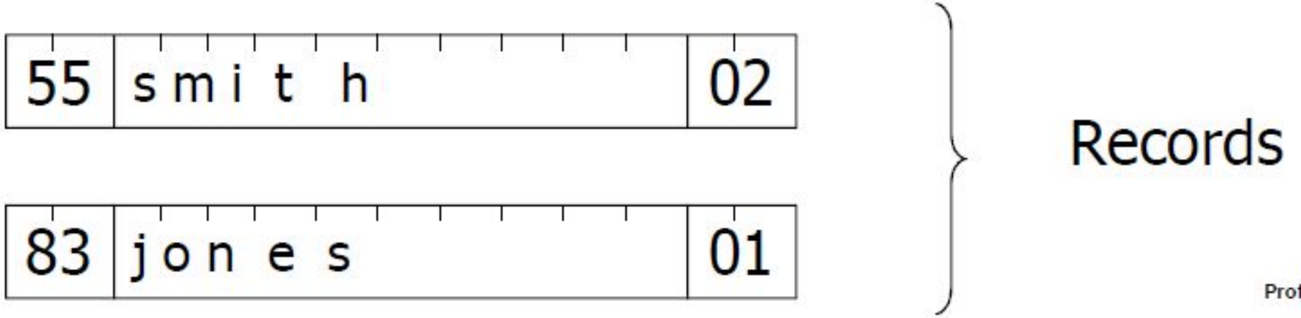
Example: Employee record

(1) E#, 2 byte integer

(2) E_name, 10 char.

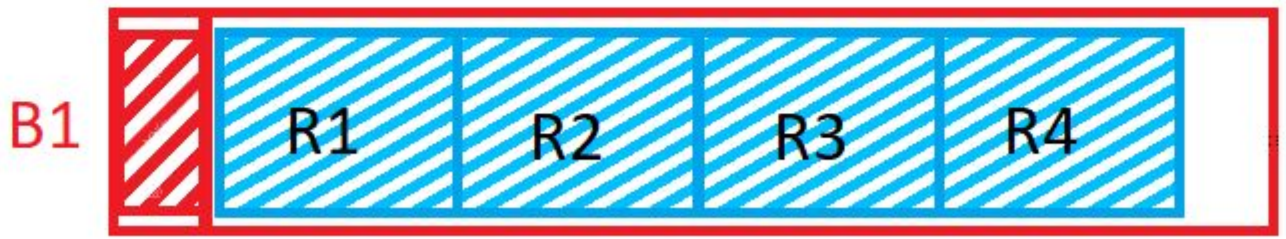
(3) Dept, 2 byte code

Schema



Fixed length Record

- Fixed Length Records : Simplest Records, consists of fixed-length fields, common to begin all fields at a multiple of 4 or 8, *Space not used by the previous field is wasted.* For example: if an attribute needs only 3 bytes, but the length needs to be a multiple of 4, one byte would be wasted.



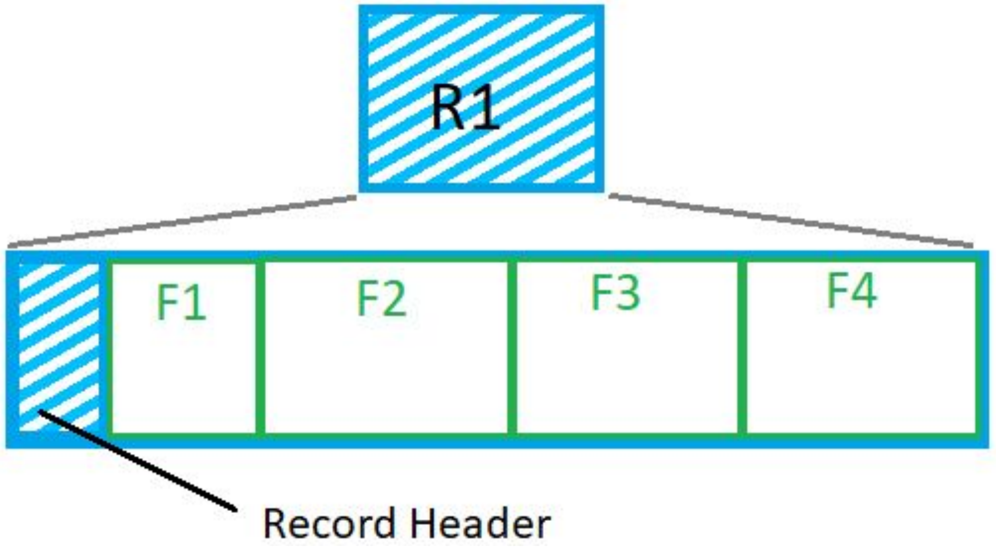
Block filled with fixed length records.

Record Header

Often, the record begins with a header, a fixed-length region where information about the record itself is kept. For example, we may want to keep in the record:

1. A pointer to the schema for the data stored in the record. For example, a tuple's record could point to the schema for the relation to which the tuple belongs. This information helps us find the fields of the record.
2. The length of the record. This information helps us *skip* over records without consulting the schema.
3. Timestamps indicating the time the record was last modified, or last read.
4. Pointers to the fields of the record. This information can substitute for schema information (important when we consider variable-length fields)

And a record



Record Header :

Fixed Length, Info about record such as pointer to schema, record length, timestamp, pointers to fields,

Fields/ Attributes example:

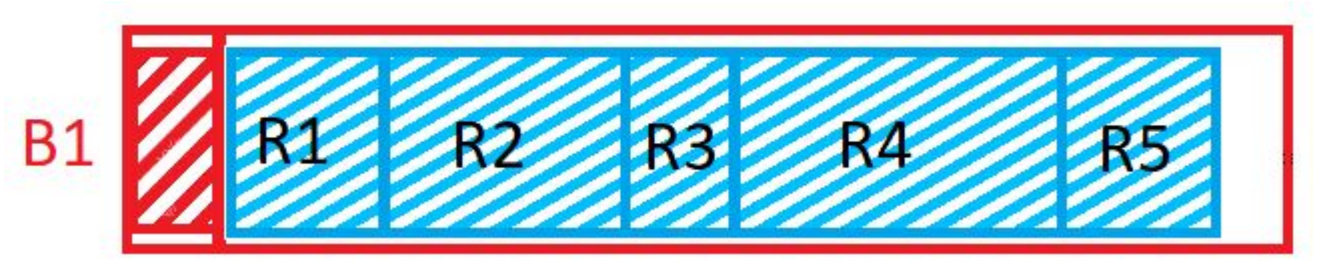
Field 1 = EmpId (2 bytes, int)

Field 2 = Employee Name (eg. String 10 bytes)

Field 3 = DepartmentName

Field 4 = Designation

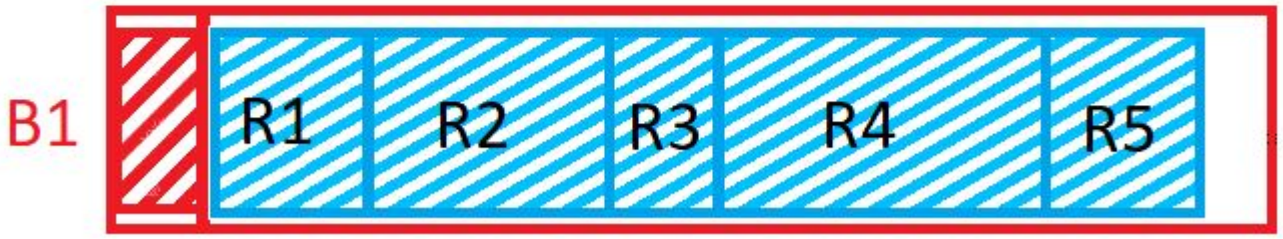
Variable length Record



Variable length could be due to:

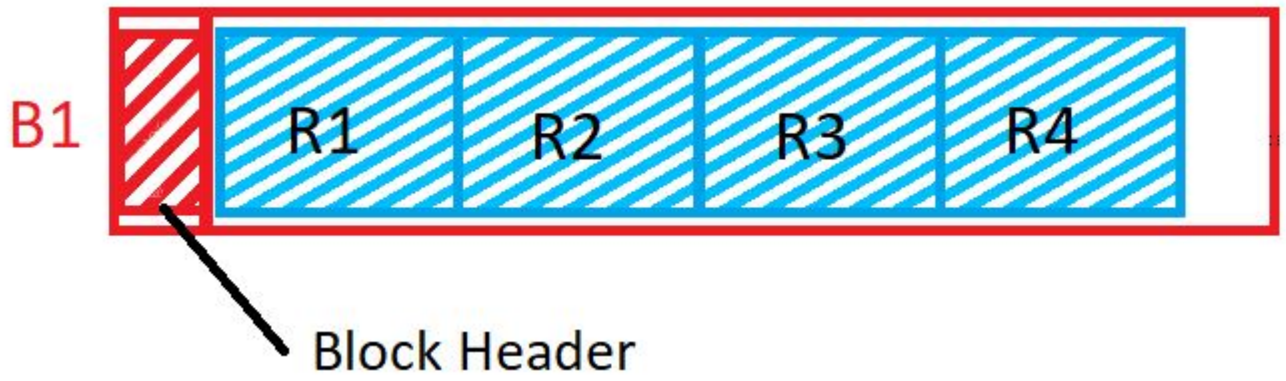
- Data items whose size varies,
- Repeating fields
- Variable format record (sometimes we don't know in advance what the fields will be)
- Enormous fields

Variable length Record



- Schema won't be useful for Record length
- Record itself needs to be "**Self Describing**"
- Record header contains extra infomation such as : # fields, type of each field, order, length, **pointers to (i.e., offsets of) the beginnings of all the variable-length fields other than the first (which we know must immediately follow the fixed length fields).**

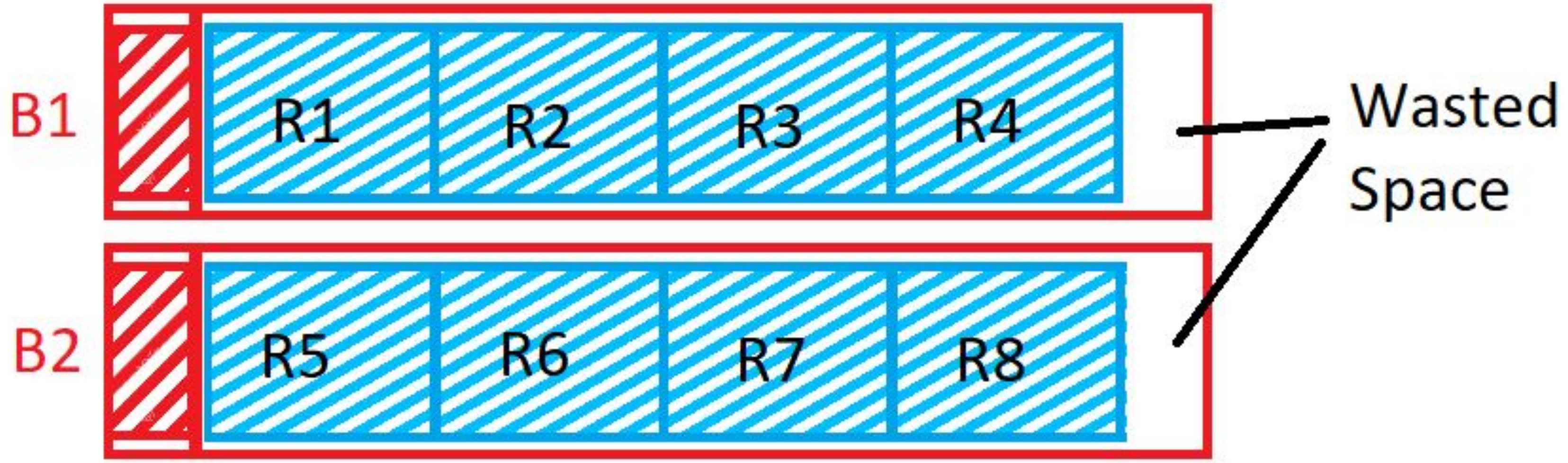
A block



Block Header : block header, with information about that block, consumes some of the space in the block, with the remainder occupied by one or more records. Can include an offset table that has pointers to each of the records in the block.

Other information can include: Links to other blocks of a data structure, which Relation the block belongs to, timestamp, etc

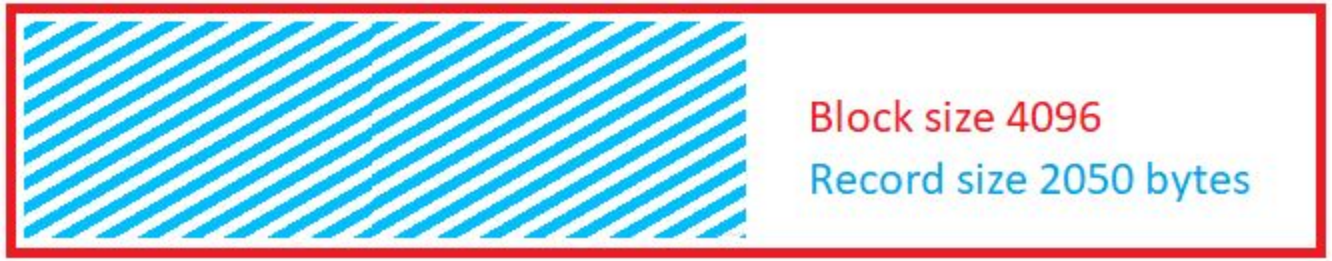
Unspanned Records



Only complete records are stored in a block, even if that means wasting space.

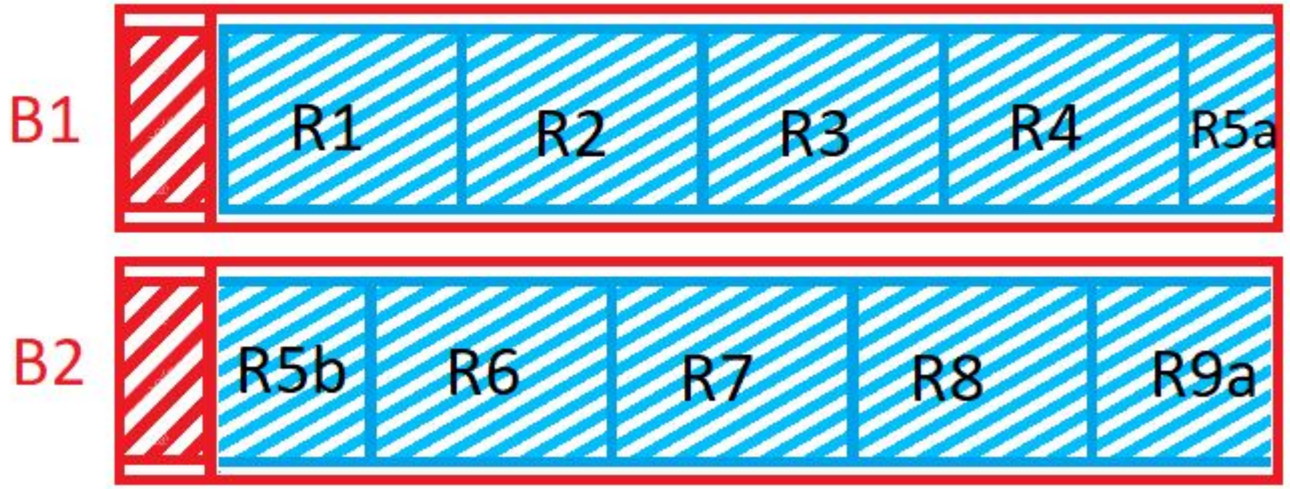
Ie **no record spans two blocks.**

Unspanned Records



Maximum possible space waste in unspanned is approximately 50%

Spanned Records



Better Space Utilization!

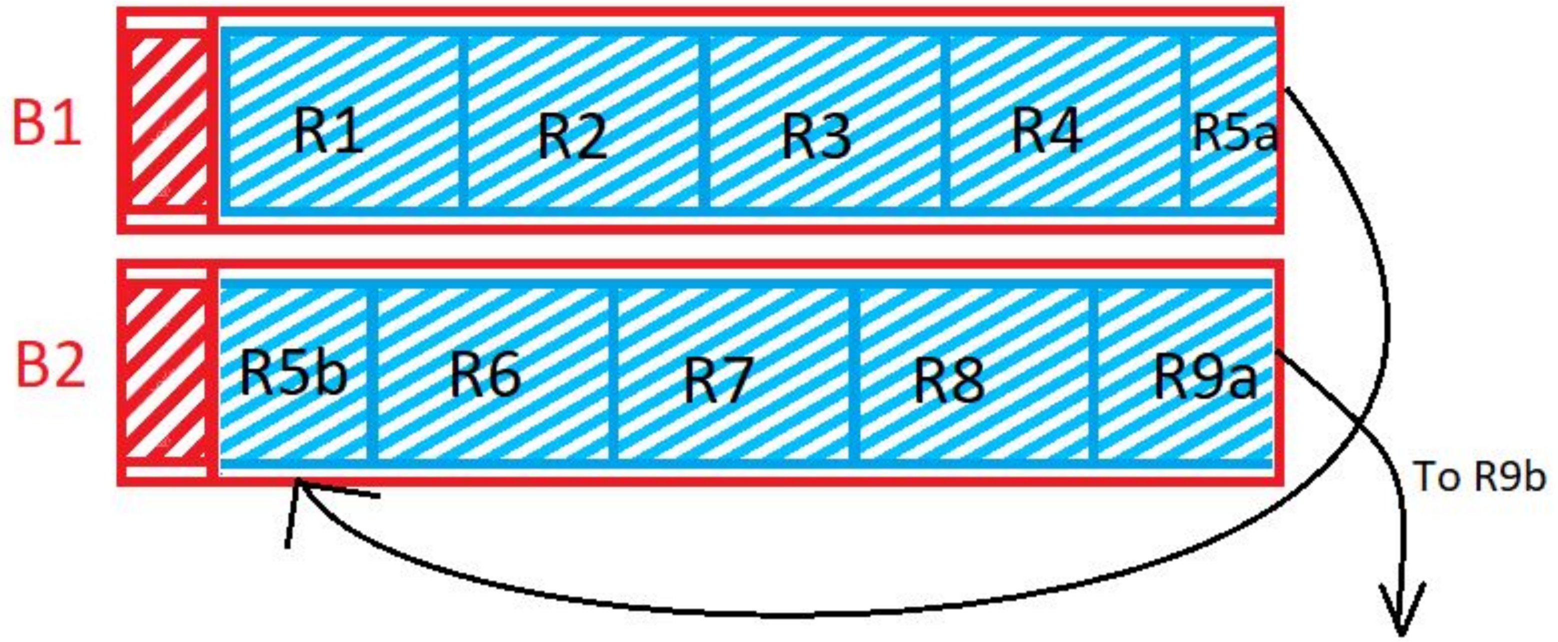
All the space in a block is utilized, even if that means breaking up records. Some **records will be spanned across two blocks**. (For really big records it might be even more than two - forced spanning)

Spanned Records

If records can be spanned, then every record and record fragment requires some **extra header information**:

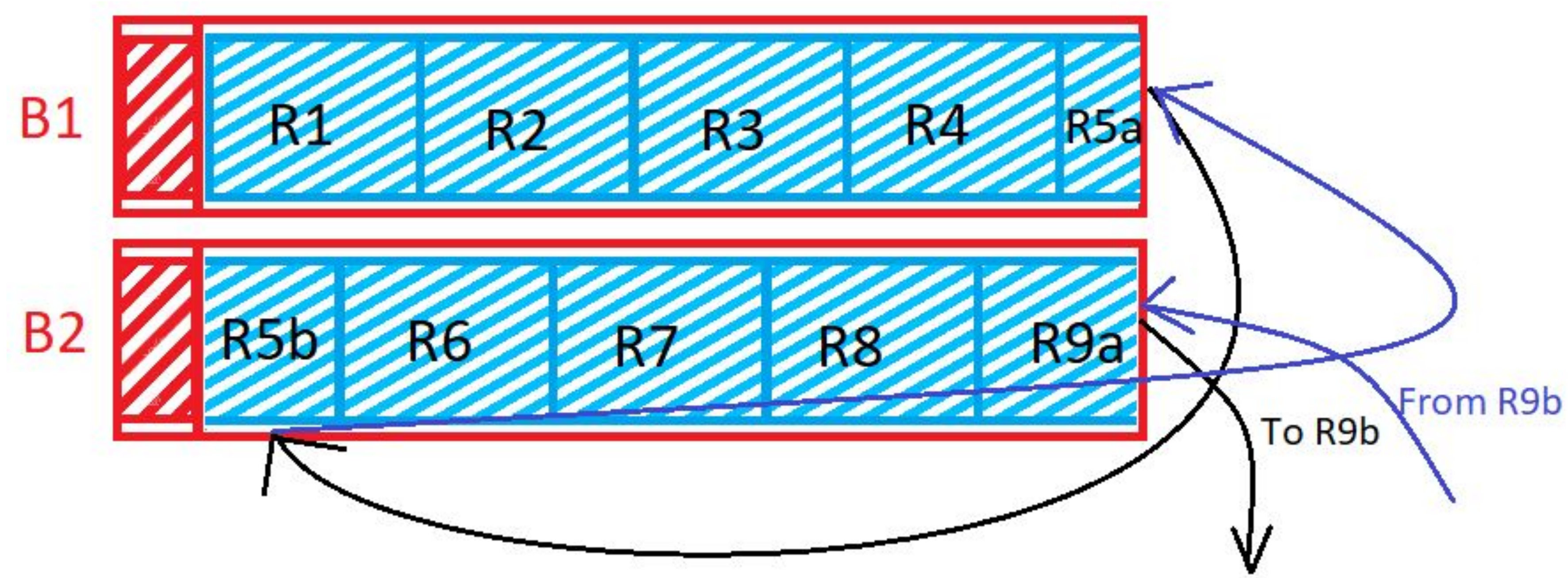
1. Each record or fragment header must contain a bit telling whether or not it is a fragment.
2. If it is a fragment, then it needs bits telling whether it is the first or last fragment for its record.
3. If there is a next and/or previous fragment for the same record, then the fragment needs pointers to these other fragments.

Spanned Records



Spanned Records

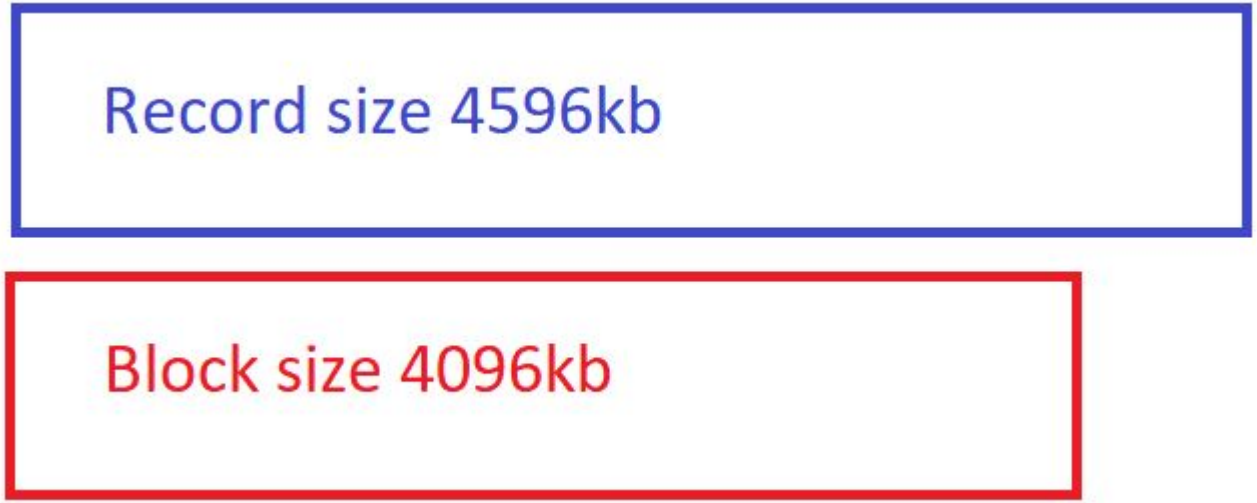
And….



Forced Spanning

Records stored in blocks. ***Blocks contain typically more than one record***.

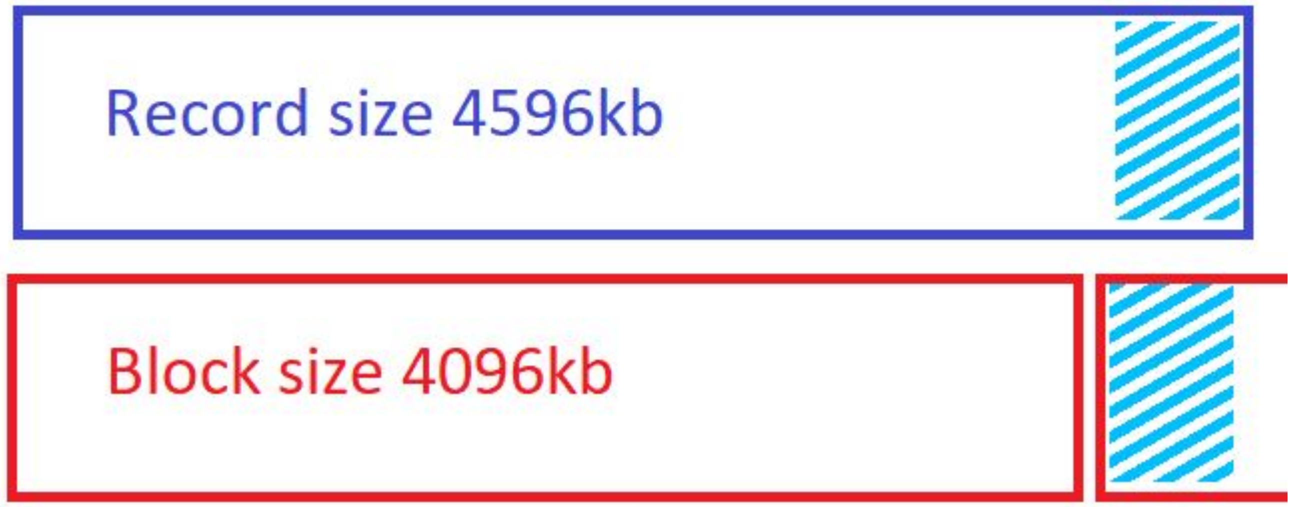
But if the records are bigger than a block, they span more than one block = Forced Spanning



Forced Spanning

Records stored in blocks. ***Blocks contain typically more than one record***.

But if the records are bigger than a block, they span more than one block = Forced Spanning



Since block is too small to hold one complete record, we have to use another block.

Sequencing

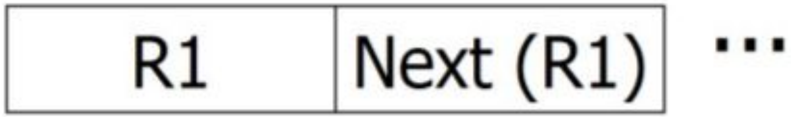
Ordering records in file (and therefore blocks) by some key value.

Sequential file (→ sequenced file)

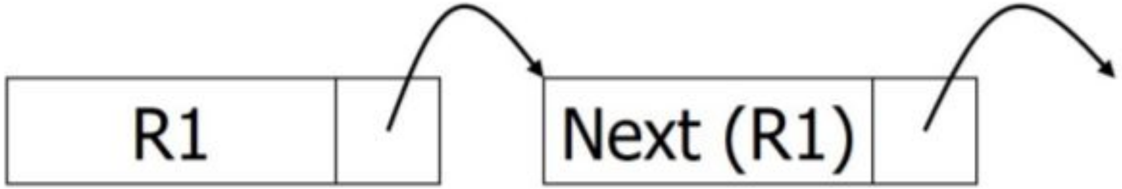
Why sequencing? Typically to make it possible to efficiently read records in order (e.g., to do a merge-join)

Sequencing

(a) Next record physically contiguous



(b) Records are linked

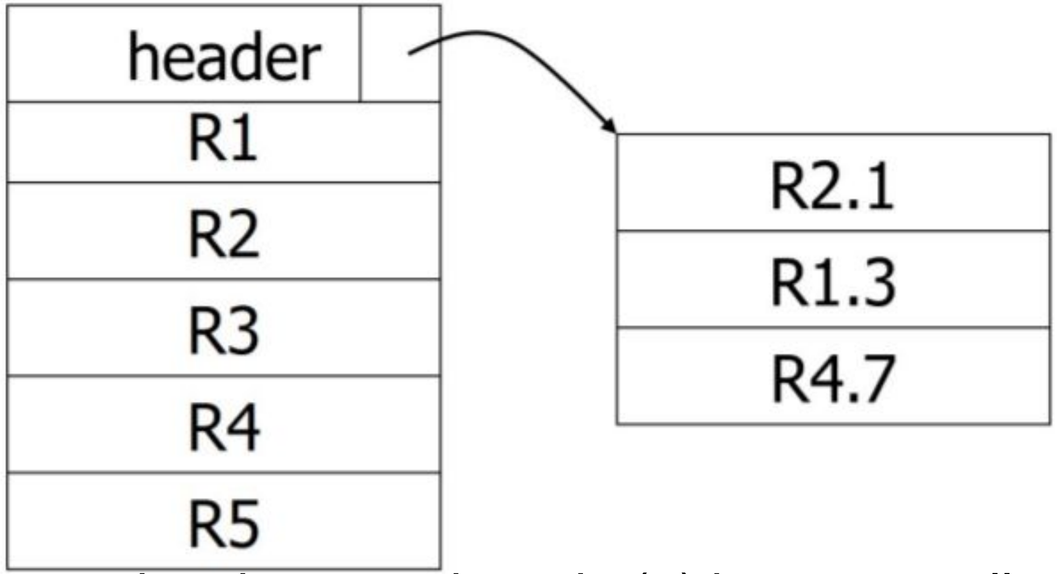


(a) Records physically stored in the right order

(b) Even though records are not next to each other, each record has a pointer to the next record in the sequence (like Linked-List)

Sequencing

(c) Overflow area
Records in sequence



If we started out by storing the records as in (a) but eventually as file grew with inserts and updates we longer have space in the correct position, we store the records in a separate location but provide information about and pointer to such records in the header.

Spanned/Unspanned Exercise

You have a file containing 4500 records. The records are 48-bytes long and have a 4-byte header (excluding the 48 bytes); The block size is 512-bytes; each block has 12 bytes of header information in it, rest can be used for records.

1. How many blocks do you need for a spanned file.Assume you need two extra bits on average per record due to spanning.

2. And for an unspanned one?

Spanned/Unspanned Solution

You have a file containing 4500 records. The records are 48-bytes long and have a 4-byte header (excluding the 48 bytes); The block size is 512-bytes; each block has 12 bytes of header information in it, rest can be used for records.

Record size = 48 + 4 = 52 bytes, Space available for records in Block = 512-12 = 500 bytes.

1. How many blocks do you need for a spanned file? Assume you need two extra bits on average per record due to spanning.

   No. of blocks for entire file = (Size of File)/ (Space available in Block for records) = ceil(4500*54/500) = 486

2. And for an unspanned one? This means we won't store fraction of a record, we will only fill a block with as many **complete** records as possible.

   No. of records/ block = **floor(**Space available in block/record size**)** = floor (500/52) = 9, Therefore number of blocks = **ceil(**4500/9**)** = 500

12.2.1 Exercises (DS:CB)

Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, an SQL

10 bytes

date, and an SQL time (8 bytes). How many bytes does the record take if:

a) Fields can start at any byte.

b) Fields must start at a byte that is a multiple of 4.

c) Fields must start at a byte that is a multiple of 8.

12.2.1 Exercise Answer

Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, an SQL date (10 bytes), and an SQL time (8 bytes). How many bytes does the record take if:

a) Fields can start at any byte. The bytes required by each of the fields is 15 + 2 + 10 + 8 = 35.

b) Fields must start at a byte that is a multiple of 4. Round each of the four field lengths up to a multiple of 4, to get 16 + 4 + 12 + 8 = 40.

c) Fields must start at a byte that is a multiple of 8. Round up again, to a multiple of 8, to get 16 + 8 + 16 + 8 = 48.

12.2.3 Exercise (DS:CB)

Assume the fields are as in Exercise 12.2.1. but records also have a record header consisting of two 4-byte pointers and a character. Calculate the record length for the three situations regarding field alignment (a) through (c) in Exercise 12.2.1

12.2.3 Exercise Answers

Assume the fields are as in Exercise 12.2.1. but records also have a record header consisting of two 4-byte pointers and a character. Calculate the record length for the three situations regarding field alignment (a) through (c) in Exercise 12.2.1

Hint : The elements in the header also require rounding.

a) 44.

b)4 + 4 + 4 + 40 = 52.

(4 + 4 + 1) should be mul of 4
(4 + 4 + 4) = 12

c)72.

(4 + 4 + 1) should be mul of 8
(8 + 8 + 8) = 24
From
16 + 8+ 16 + 8 + 24 = 74

12.2.5 Exercise (DS:CB)

Suppose records are as in Exercise 12.2.3, and we wish to pack as many records as we can into a block of 4096 bytes, using a block header that consists of ten 4-byte integers. How many records can we fit in the block in each of the three situations regarding field alignment (a) through (c) of Exercise 12.2.1?

12.2.5 Exercise Answers

Suppose records are as in Exercise 12.2.3, and we wish to pack as many complete records as we can into a block of 4096 bytes, using a block header that consists of ten 4-byte integers. How many records can we fit in the block in each of the three situations regarding field alignment (a) through (c) of Exercise 12.2.1?

For parts (a) and (b), the header will take 40 bytes, while for (c) it takes 80 bytes.

a) (4096-40)/44 = 92.
b) (4096-40)/52 = 78.
c) (4096-80)/72 = 55.

Merge Sort Exercise 2

For each of these scenarios:

1. a file with 10,000 blocks and 3 available buffers.
2. a file with 20,000 blocks and 5 available buffers.
3. a file with 2,000,000 blocks and 17 available buffers.

answer the following questions assuming that external merge-sort is used to sort each of the files:

a. How many sublist will you produce on the first pass?
b. How many passes will it take to sort the file completely?
c. What is the total I/O cost for sorting the file? (measured in #blocks read/written)

Merge Sort Exercise 2 - Answers

For each of these scenarios:

1.a : 3334 i.e. (ceil(10,000/3))

1.b : 4000

1.c : 117648

2.a : 13

2.b : 7

2.c : 6

3.a : $2 * 10{,}000 * 13 = 26 * 10^4$

3.b : $2 * 20{,}000 * 7 = 28 * 10^4$

3.c : $2 * 2{,}000{,}000 * 6 = 24 * 10^6$