Homework 1
Data Storage

Due: *Wednsday, February 11, in-class*

Submission. Please submit your solutions on February 11, in-class. I encourage everyone to use text-processing software for the solutions and use PowerPoint, xfig or other graphical tools to draw pictures/diagrams required in this homework. (I will accept handwritten solutions, but I discourage them. This policy is mainly to ensure expedient and error-free grading).

Problem 1

You will consider two different schemas for storage of data on disk. For each schema, you have to provide descriptions of (a) the page header and (b) records for each of the relational tables described below. You also have to specify both the page header size and the size of the record, and determine how many records can be stored on a single disk page.

In both problems, INT has size 4 and FLOAT has size 8 (bytes).

Storage Schema A.

The size of a disk page is 1Kb. You reserve 128 bytes for the constant-size part of the page header. The file organization is *heap.* You store a bitmap of available slots on the disk page in the header (this is the variable size portion of the page header. The total size of the page header is the sum of the sizes of both parts of the header). The organization of the constant-size part of the header is left up to you, but you need to ensure that the header contains all necessary information to successfully retrieve/maintain records.

All records are constructed by concatenating the field representations of individual attributes. There are no record headers, and you resolve the constraints on the starting positions of the fields by padding. Each record must start at an offset divisible by 8, which entails that the size of the page header must also be divisible by 8.

Deleted records are marked in the bitmap, and nowhere else. You can assume that the relation schema is stored in the file header page, i.e., your pages need not keep this information.

Storage Schema B.

The size of a disk page is 2Kb. You reserve 64 bytes for the page header. The file organization is *sequential*. You use *sliding* to reclaim space during deletions, which means that the free space on your pages can occur only at the end of the block. Your header must contain all information necessary for maintenance of data.

All records are constructed by concatenating the field representations of individual attributes. In addition, an 8-byte header is attached to each record. It includes two timestamp fields - one for the last read access and one - for the last write access to the record. You resolve the constraints on the starting positions of the fields by reordering the fields in the record, and adding and additional padding, if needed to the end of the record, to ensure that all records start with the offsets divisible by 8.

Relations

Below is a list of relations you have to use. Note, that these are standalone relations, NOT a list of relations forming a coherent database.

1. Employee(Id INT, SSN INT, Name VARCHAR(40), Department INT, Salary FLOAT, Bonus FLOAT, StartYear INT, Comment CHAR(80)).
2. Team(Id INT, Name CHAR(20), City CHAR(20), Owner INT, Division CHAR(20), Year INT).
3. Survey(RespondentId INT, Sex CHAR(1), Income INT, Age INT, Race INT, Q1 INT, Q2 INT, Q3 INT, Q4 INT, Q5 INT, Q6 INT).
4. Transcript((Id INT), FirstName CHAR(20), LastName CHAR(20), College CHAR(5), Status CHAR(2), Course CHAR(7), Semester INT, Year INT, Grade INT, CurrentGPA FLOAT).
5. Transaction(Id INT, DateTimePosted DATE, DateTimeOccurred DATE, CardNo CHAR(16), Location INT, Vendor INT, Amount FLOAT, FeeRate FLOAT, Fee FLOAT).
6. Goods(Id CHAR(15), Food CHAR(20), Flavor CHAR(20), Price FLOAT).

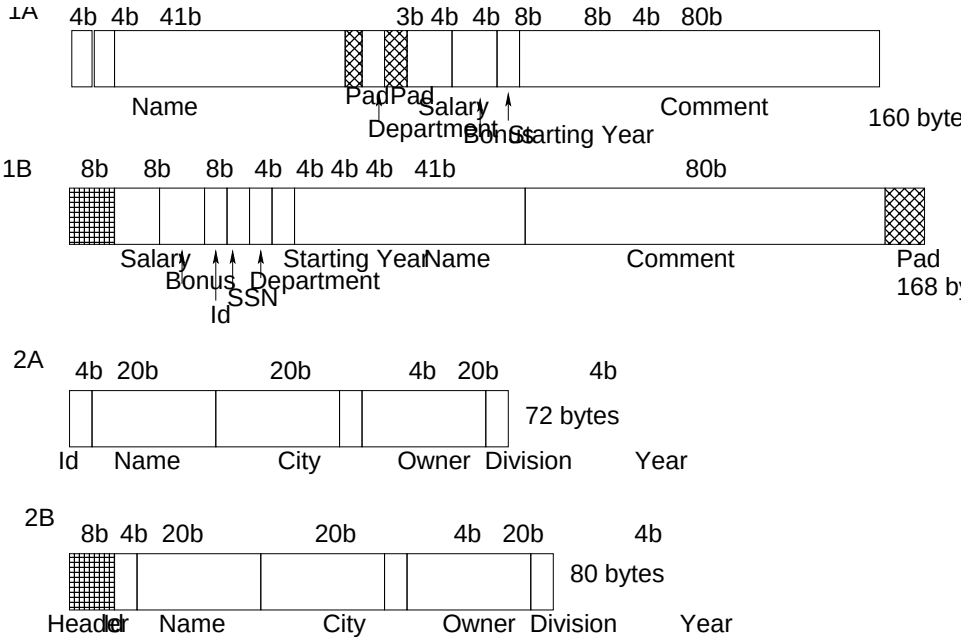
Questions

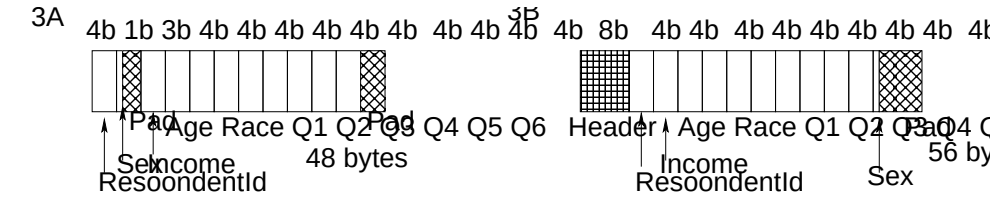
For each storage schema, and for each relation, do the following:

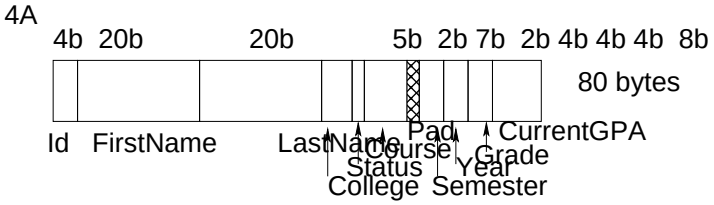
1. Describe the structure of the disk record storing a tuple from the relation. Draw a diagram (see below). Determine the size of the record.

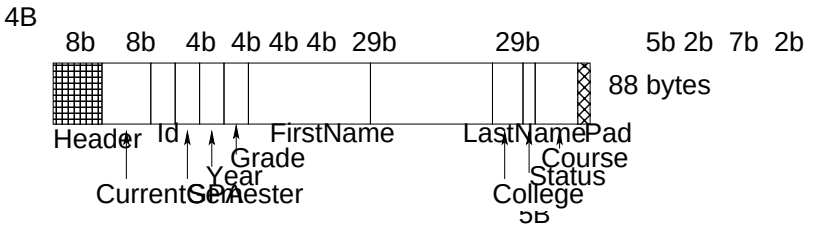
2. Describe the structure of the page header (draw a diagram), determine its size (if not constant).

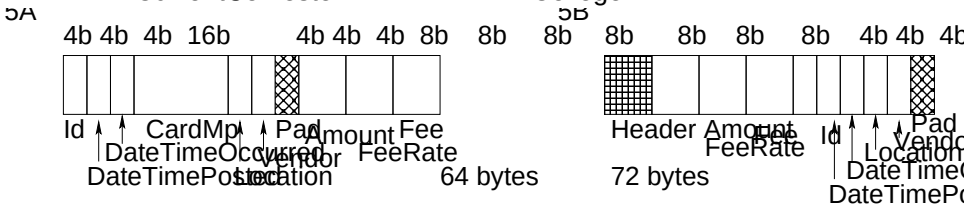
3. Determine the number of records that can be stored on a single disk page.

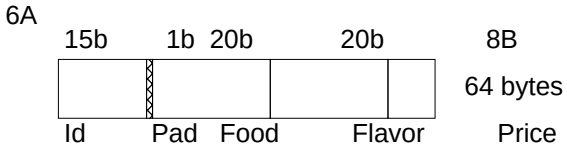


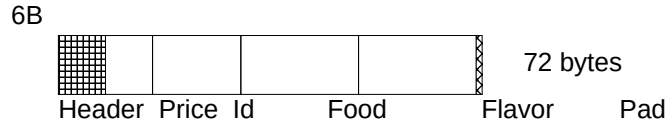












Arithmetics

(1A) Record size = 160 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding) = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 160 = 5. Size of bitmap: 1 byte.

(1B) Record size = 168 bytes. Page header: 64 bytes. Available for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 168 =

11.

(2A) Record size = 72 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding). = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 72 = 12. Size of bitmap: 2 bytes.

(2B) Record size = 80 bytes. Page header: 64 bytes. Avaliable for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 80 = 24.

(3A) Record size = 48 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding) = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 48 = 18. Size of bitmap: 3 bytes.

(3B) Record size = 56 bytes. Page header: 64 bytes. Available for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 56 = 35.

(4A) Record size = 80 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding) = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 80 = 11. Size of bitmap: 2 bytes.

(4B) Record size = 88 bytes. Page header: 64 bytes. Available for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 88 = 22.

(5A) Record size = 64 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding). = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 64 = 13. Size of bitmap: 2 bytes.

(5B) Record size = 72 bytes. Page header: 64 bytes. Avaliable for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 72 = 27..

(6A) Record size = 64 bytes. Page header: 128 bytes (persistent) + 8 bytes (bitmap+padding). = 136 bytes. Available to store records: 1024 - 136 = 888 bytes. Number of records per page: 888 `div` 64 = 13. Size of bitmap: 2 bytes.

(6B) Record size = 72 bytes. Page header: 64 bytes. Avaliable for records: 2048 - 64 = 1984 bytes. Number of records per page: 1984 `div` 72 = 27..

Page Headers

(A) The page header should contain:

Field	size	Comment
PageId	4 bytes (INT)	Id of the page
PtrNext	4 bytes	pointer to next page in the heap
PtrPrev	4 bytes	pointer to previous page in the heap
PtrNextFree	4 bytes	pointer to next page in `FreeSpace` list
PtrPrevFree	4 bytes	pointer to previous page in `FreeSpace` list
RecordSize	4 bytes	size of records on the page
MaxRecords	4 bytes	total number of records that can be stored
CurrentRecords	4 bytes	number of records currently on the page

Total size: 32 bytes. (Additionally, a few timestamps can be added).

(B) The page header should contain:

Field	size	Comment
PageId	4 bytes (INT)	Id of the page
PtrNext	4 bytes	pointer to next page in the heap
PtrPrev	4 bytes	pointer to previous page in the heap
RecordSize	4 bytes	size of records on the page
MaxRecords	4 bytes	total number of records that can be stored
CurrentRecords	4 bytes	number of records currently on the page

Total size: 24 bytes. Additioanlly, a few timestamps can be added.

Problem 2

Consider a database consisting of three data files, P, T and R. Each file consists of a number of disk pages (blocks). We refer to block number i from file F as Fi, e.g., P1 is the first page of file P, T20 is the 20th page of file T and so on.

The DBMS controlling the database is equipped with a buffer manager **B**. The size of the buffer space is **4 disk blocks**.

Consider the following possibilities for **B**.

B1 : LRU (Least Recently Used) buffer manager. This manager uses the least recently used page and flushes it to disk, when a new page needs to be brought from disk.

B2 : FIFO (First In, First Out) buffer manager. This manager flushes the oldest page in the buffer.

B3 : Simple Clock buffer manager. This buffer manager uses the simple clock algorithm (starting with buffer 1 of the buffer space) to determine the next buffer to be flushed.

The query execution layer of the DBMS uses four operations to control data access:

- Read(PageId): results in the page with the given PageId being transfered to the buffer. No action if the page is already in the buffer.

- Write(PageId): results in the page with the given PageId being marked as ready for transfer back to disk. If the page is NOT in the buffer, it is **transferred** back to the buffer, using the buffer manager's management strategy, and is marked *ready* afterwards.
- Pin(PageId): given page is *pinned* in the buffer. If the page to be pinned is NOT in the buffer, Read(PageId) command is executed first.
- Unpin(PageId): given page is *unpinned* in the buffer.

All buffer managers first look find empty slots in the buffer. If no empty slots are available, they look to flush back to disk any pages marked as *ready* by the Write() command. If no such pages found, the respective buffer management strategy is engaged. Filling an empty slot or flushing a *ready* page does not affect the position of the pointer in the clock algorithms.

For each of the following two sequences of commands and each of the buffer management strategies outlined above, show the state of the buffer after each 8 operations. (marked as "`checkpoint`" in the sequence). For each of the four buffers in the buffer space, show the id of the page stored in it, and whether the *ready* and/or *pin* flags have been set. For the clock-based buffer managers show current counter values, and the position of the clock "hand". For LRU and FIFO managers, show the timestamps (use the position in the sequence below as the timestamp).

In all cases assume you start with empty buffer space. For prioritized clock buffer manager, assume that all pages from T and P are added with counter values set to 1, while pages from R are added with counter values set to 2.

Sequence 1

```
Read(T1);
Read(R1);
Read(P2);
Pin(P2);
Read(P3);
Read(T2);
Read(T1);
Read(R4);
     Checkpoint;
Write(R4);
Unpin(P2);
Write(P2);
Read(T3);
Pin(T3);
Write(R3);
Read(R1);
Read(R5);
     Checkpoint;
```

```
Unpin(T3);
Write(T3);
Write(T1);
Read(T2);
Write(P2);
Pin(T1);
Read(R6);
Read(R7);
     Checkpoint;
```

Sequence 2

```
Read(P1);
Read(R1);
Read(T2);
Read(R3);
Read(R4);
Read(P2);
Read(T1);
Read(P1);
    Checkpoint;
Read(R3);
Pin(P1);
Pin(R4);
Read(R5);
Read(T1);
Read(T2);
Write(T2);
Read(T4);
    Checkpoint;
Read(R1);
Unpin(P1);
Read(R2);
Read(R3);
Unpin(P4);
Read(T3);
Read(P1);
Read(T2);
    Checkpoint;
```

LRU

Sequence 1 Checkpoint 1:

1	2	3	4
T2	T1	P2 PINNED	R4

Checkpoint 2: (Read(T3) replaces page in buffer 3. Could also replace page in buffer 4)

1	2	3	4
R5	T1	T3 PINNED	R1

Checkpoint 3:

1	2	3	4
R6	T2	T1 PINNED	R7

Sequence 2 Checkpoint 1:

1	2	3	4
R4	P2	T1 PINNED	P1

Checkpoint 2:

1	2	3	4
T1	R4 PINNED	T4	P1 PINNED

Checkpoint 3: Unpin(P4) has no effect on the state of the buffer.

1	2	3	4
T2	R4 PINNED	P1	T3

FIFO

I believe I actually managed to set up both sequences in a way that makes LRU and FIFO buffer managers behave in the same way. (if your results differ, email me).

Clock Buffer

I used `ClockBuffer` program (now available on the course web page) to run the assignment. The output is below.

Sequence 1

```
------------- CHECKPOINT ----------------
```

```
              PageId     Counter     Pin    Ready
    Slot 1: T2    0  NOT PINNED    NOT READY
    Slot 2: R4    2  NOT PINNED    NOT READY
    Slot 3: P2    1  PINNED       NOT READY
--->Slot 4: T1    1  NOT PINNED    NOT READY

------------- CHECKPOINT ----------------

              PageId     Counter     Pin    Ready
    Slot 1: R5    2  NOT PINNED    NOT READY
    Slot 2: T3    1  PINNED       NOT READY
--->Slot 3: R1    2  NOT PINNED    NOT READY
    Slot 4: T1    0  NOT PINNED    NOT READY

------------- CHECKPOINT ----------------

              PageId     Counter     Pin    Ready
--->Slot 1: R5    0  NOT PINNED    NOT READY
    Slot 2: R6    2  NOT PINNED    NOT READY
    Slot 3: R7    2  NOT PINNED    NOT READY
    Slot 4: T1    1  PINNED       NOT READY
```

Sequence 2

```
------------- CHECKPOINT ----------------

              PageId     Counter     Pin    Ready
--->Slot 1: R4    1  NOT PINNED    NOT READY
    Slot 2: T1    1  NOT PINNED    NOT READY
    Slot 3: P2    0  NOT PINNED    NOT READY
    Slot 4: P1    1  NOT PINNED    NOT READY

------------- CHECKPOINT ----------------

              PageId     Counter     Pin    Ready
    Slot 1: R4    0  PINNED       NOT READY
    Slot 2: T4    1  NOT PINNED    NOT READY
--->Slot 3: T1    1  NOT PINNED    NOT READY
    Slot 4: P1    0  PINNED       NOT READY

------------- CHECKPOINT ----------------

              PageId     Counter     Pin    Ready
    Slot 1: R4    0  PINNED       NOT READY
--->Slot 2: P1    1  NOT PINNED    NOT READY
```

```
Slot 3: T3    0  NOT PINNED    NOT READY
Slot 4: T2    1  NOT PINNED    NOT READY
```