Homework 1 - 2
Data Storage

Due: *Wednesday, October 27, in-class*

Submission. Please submit your solutions on Wednesday, October 27, 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 4Kb. You reserve 256 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 128 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, a 16-byte header is attached to each record. It includes three timestamp fields - one for the last read access, one - for the last write access to the record and one for the first time the record was inserted. Additional four bytes are unused at the moment, but may wind up being used later. You resolve the constraints on the starting positions of the fields by reordering the fields in the record, and adding 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).

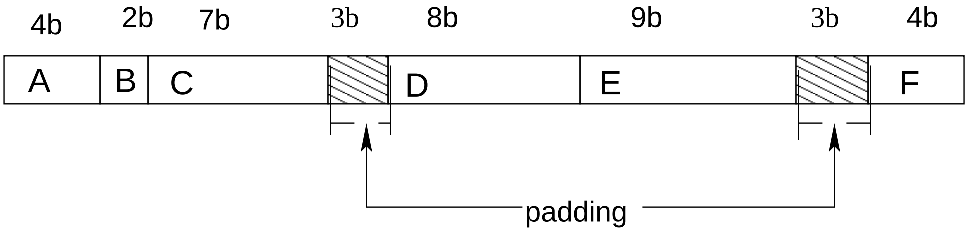
6. Goods(Id CHAR(15), Food CHAR(20), Flavor CHAR(20), Price FLOAT).

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.

Notes. Assume DATE is represented as a VARCHAR(12). For each schema, your descriptions of the page header must be consistent from relation to relation (in fact, you are allowed to specify it just once, and only show the size of the page header for each relation). A diagram of a record and/or header should look similar to the following diagram (created for Schema A storage for the relation R(A INT, B CHAR(2), C CHAR(7), D FLOAT, E VARCHAR(8), F INT)).



Submission Note. Please number your answers as follows: RelationNo Schema.Question. This way 1A.2 represents the answer to the question 2, for relation 1 using storage schema A.

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(R1);
Read(R2);
Read(P1);
Pin(P1);
Read(P2);
Read(R2);
Read(T1);
Read(R3);
```

```
      Checkpoint;
 Write(P1);
 Unpin(P1);
 Write(T1);
 Read(T2);
 Pin(T2);
 Write(R3);
 Read(R1);
 Read(P3);
      Checkpoint;
 Unpin(T2);
 Write(T3);
 Write(P1);
 Read(P2);
 Write(T4);
 Pin(P2);
 Read(P4);
 Read(P5);
      Checkpoint;
```

Sequence 2

```
Read(T1);
Read(P1);
Read(P2);
Read(R1);
Read(R2);
Read(R3);
Read(T2);
Read(P3);
    Checkpoint;
Read(T1);
Pin(P3);
Pin(T1);
Read(R4);
Read(T2);
Read(P2);
Write(T2);
Read(T3);
    Checkpoint;
Read(R1);
Unpin(P3);
Read(R2);
Read(R3);
Unpin(T1);
Read(T3);
```

```
 Read(P1);
 Read(T2);
     Checkpoint;
```

Indexing techniques

Consider a database relation R with an integer key X, which needs to be indexed. X has no duplicate keys. We consider the following two file organizations:

Data File A. Data file A is a sequential file with X as the search key. Each disk page can store four (4) records from the relation R on it.

Data File B. Data file B is a heap file. Each disk page can store three (3) records from the relation R on it.

Consider the following sequence of insertion and deletion operations for the relation R. For simplicity, `Insert(10)` means "insert a record with the value of X attribute set to 10".

```
Create(R);

Insert(100);
Insert(30);
Insert(65);
Insert(10);
Insert(80);

Insert(20);
Insert(135);
Delete(65);
Delete(80);
Insert(75);

Insert(70);
Insert(120);
Delete(15);
Delete(10);
Insert(35);


Insert(60);
Insert(55);
Delete(75);
Insert(110);
Delete(120);

Insert(75);
Insert(10);
Insert(40);
Delete(110);
Insert(95);
```

Problem 3

For each data file structure (A and B), show the state of the data files after each five `Insert/Delete` operations. For each disk page, show only pageID and any pointers you may have in the header (nothing else is needed for this exercise). You can omit the header page - just show the pages with the data.

Problem 4

For Data File A, show the state of the following index files after every five `Insert/Delete` operations:

1. Simple Dense index (10 index records per page) on X;
2. Simple Sparse (5 index records per page) index on X.
3. Dense B+-tree (3 key values per page) index on X;

Problem 5

For Data File B, show the state of the following index files after every five `InsertDelete` operations:

1. Simple secondary index (6 index records per page) on X;
2. B+-tree secondary index (4 key values per page) on X;