CS150A Quiz 2 Solutions

Assume that each page in our system can hold 64 KB (1 KB = 1024 bytes), integers are 32-bits wide, and bytes are 8-bits wide.

Consider the following relation:

```
CREATE TABLE Submissions (
  record_id integer UNIQUE,
  assignment_id integer,
  student_id integer,
  time_submitted integer,
  grade_received byte,

  PRIMARY KEY(assignment_id, student_id)
);
```

Assume the column record_id corresponds to the row's actual record ID.

Q1: How large (in bytes) is a record?

We simply add up the sizes of each field in a record. We have 4 integer fields and 1 byte field, which is 4*4 = **17** bytes.

Q2: Suppose we begin each page with a 32-bytes header plus a bitmap. At most, how many records can fit in an unpacked page?

Let’s convert everything into bits. First, a page holds 1,024 * 64 * 8 = 1,048,576 bits while a record holds 17 * 8 = 136 bits. Now, remember that in an unpacked page, each record needs an additional bit to represent whether or not it is valid (e.g. has been deleted). This means we need 1 more bit per record, so each record in fact requires 137 bits. Finally, we have an extra 32 bytes (32 * 8 = 256 bits) reserved for the page header. This gives us a maximum of (1,048,576 - 256) / 137 ~= 7,651 records.

We add two variable-length fields to our table schema. Now our table looks like this:

```
CREATE TABLE Submissions (
  record_id integer UNIQUE,
  assignment_id integer,
  student_id integer,
  time_submitted integer,
  grade_received byte,

  comment text,
  regrade_request text,

  PRIMARY KEY(assignment_id, student_id)
);
```

We decide to use slotted pages to store the variable length records. Each page begins with a 24-byte header plus a slot directory. (Assume this header contains information such as the number of valid records in the page.) Each pointer inside the slot directory consumes 20 bits/record, while the record header storing field offsets is 32 bits wide.

Q3: What is the maximum number of records that can fit in our slotted pages?

Again, we have 1,048,576 bits per page, the page header consumes 256 bits, and a record is 136 bits wide. Now instead of a bitmap, where each record takes 1 extra bits, we have a slot directory of pointers, so each record requires 20 extra bits. Thus each record costs us 136 + 20 = 156 bits. We need to store field offsets for the variable-length text fields, which is an additional 32 bits per record, bringing us to a total of 156 + 32 = 188 bits per record. Finally, note that we get the maximum possible number of records when all comment and regrade_request fields are NULL; i.e. they both take up 0 bytes. Then the smallest possible memory footprint per record is 188 bits/record, so we get (1,048,576 - 256) / 188 ~= 5,576 records.

Q4: We decide to squash the two text fields together into one field using a semicolon separator character (;), which allows us to shrink the record header from 32 bits to 16 bits at the cost of 8 bits (for the semicolon). For example, the columns ("Submitted late", "Dog ate my homework") get compressed into "Submitted late;Dog ate my homework". Which of the following are true with this new scheme?

Since we're using the semicolon as a field separator, we can't enter comments with semicolons. In our example, the comment "Fantastic work; good job!" would be truncated to "Fantastic work" and " good job!" would be misinterpreted as the regrade_request field. However, since our records become 8 bits shorter, we’ll be able to fit more records per page. As a result, our table will become smaller, and table scans will speed up accordingly (depending on how many pages we save on storage).

C, D