Quiz 2

Deadline	Sunday, 14 March 2021 at 9:00PM
Latest Submission	Sunday, 14 March 2021 at 7:55PM
Raw Mark	4.00/4.00 (100.00%)
Late Penalty	N/A
Final Mark	4.00/4.00 (100.00%)

Question 1 (1 mark)

Consider performing a nested-loop join on two tables as follows:

```
for i in 0..nPages(R)-1 {
   read page i from R
   for j in 0..nPages(S)-1 {
      read page j from S
      check all pairs of tuples in R page and S page
      add those satisfying join condition to output
   }
}
```

If table *R* contains 500 large tuples stored in 10 pages and table *S* contains 1000 small tuples stored in 5 pages, and there is one input buffer for each table and one output buffer to hold result tuples, how many pages will be **read** in executing the above query method?

60

✔ Your response was correct.
Mark: 1.00

Reads 10 pages in the outer table.

For each of those pages, reads all 5 pages of the inner table.

So, pages read = 10 + 10 * 5 = 60

Question 2 (1 mark)

Consider two tables R and S, like the tables in Q1, where nPages(R) = 10 and nPages(S) = 5.

Now consider the execution of a nested-loop on these two tables, this time using a buffer pool:

```
for i in 0..nPages(R)-1 {
   request page i from R
   for j in 0..nPages(S)-1 {
      request page j from S
      check all pairs of tuples in R page and S page
      add those satisfying join condition to output
      release page j from S
   }
   release page i from R
}
```

The buffer pool contains 7 buffers and is initially empty. One buffer will be locked into memory for use as an output buffer, while the other 6 buffers will be available to the buffer-pool manager to handle page requests. Assuming an MRU buffer replacement strategy, where most-recent is determined by "last released", how many pages will be **read** in executing the above nested-loop join? Note: we are asking for the number of pages actually read from disk, not simply the number of page requests.

15

✔ Your response was correct.
Mark: 1.00

One buffer (let's say Buf0) is used for output, leaving 6 buffers.

On the first pass, R0 is read into Buf1, S0 into Buf2, S1 into Buf3, ... S4 into Buf6

After the first pass, the last-released page is R0 (so it becomes the MRU)

On the next pass, the "current" R page in Buf1 is replaced by the next R page.

During all subsequent passes, the S0...S4 pages are already held in buffers.

So, pages read = 10 R pages + 5 S pages = 15

Each page is read exactly once

Question 3 (1 mark)

In the PostgreSQL file manager, a **fork** is ...

(a) ○	a new file access process, spawned when the PostgreSQL server becomes overloaded
(b) ○	a branch of relation data related to MVCC; it allows multiple different versions of the relation to be stored
(c) ◉	an additional data file added to hold tuples for one relation when the original data file for that relation becomes too large
(d) ○	a file that provides information about where free space exists in the data file for some relation; used to aid insertion in heap files
(e) ○	None of the above answers is correct

✔ Your response was correct.
Mark: 1.00

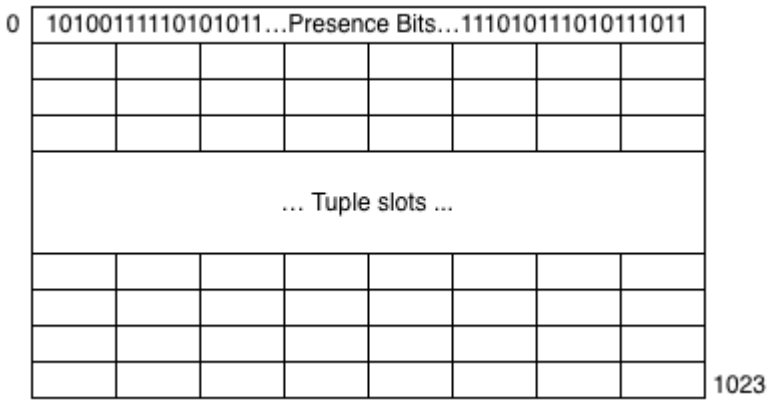
Question 4 (1 mark)

Consider a relational table defined as:

```
create table T (
   rid integer references R(id),
   sid integer references S(id),
   primary key(rid,sid)
);
```

In a highly-optimised database system, this tuple might occupy exactly 8 bytes, with all tuple metadata (attribute types and lengths) being stored in the system catalog. Since tuples are fixed size, each page needs no free space list or a directory. However, each page does contain a "presence vector", with one bit for each possible tuple in the page. If the i^{th} bit in the presence vector is 1, then the i^{th} slot in the page contains a tuple; if the i^{th} presence bit is 0, then there is no tuple in the i^{th} slot.

The following diagram shows the structure of pages:



In a database system where pages are 1024 bytes in length, how many tuple slots would be in each page if we are to maximise space usage?

You can assume that the presence bits array has a size which is a number *N* of whole 32-bit words (i.e. the presence array is either 32 or 64 or 96 or 128 or ... bits in length).

(a) ○	120
(b) ◉	126
(c) ○	128
(d) ○	132
(e) ○	None of the other answers is correct

✔ Your response was correct.
Mark: 1.00

We need to solve the inequality (ceil(c/8) + c*8) ≤ 1024

where c = page **c** apacity = max # tuples in page

ceil(c/8) = number of bytes for the presence bits

c*8 = number of bytes used for tuples

Of the supplied options, 126*8 + ceil(126/8) = 1024