Midterm Solutions: Introduction to Database Systems

Problem 1: B+trees

(A) (3 points)
Node affected: N7

New Node:

(B) (6 points)

(C)
a. (5 points)
1. Merge N3 and N4. (Move 519 to N3)
2. Merge N10 and N11, and move it to child of N3.

b. (1 point)
Insert any value in [520,600)
Problem 2: Query Languages

a. [4 points] Find blog entries posted by an author with realname ‘Ted’, and return the title, timestamp and body of the entry.

```
SELECT e.title, e.timestamp, e.body
FROM entries e, authors a
WHERE e.authorid = a.authorid
AND a.realname = 'Ted'
```

b. [6 points] For each entry in the blog, return the same fields as in part (a), for those entries with more than 2 comments.

```
SELECT e.title, e.timestamp, e.body
FROM blogdb_entries e
WHERE e.id IN
(SELECT c.entry_id
 FROM blogdb_comments c
 GROUP BY c.entry_id
 HAVING count(*) > 2);
```

OR

```
SELECT e.title, e.timestamp, e.body
FROM blogdb_entries e, blogdb_comments c
WHERE e.id = c.entry_id
GROUP BY e.id, e.title, e.timestamp, e.body
HAVING count(*) > 2;
```

c. [6 points] The “parent_id” field in the comments table tracks the nesting of “comments on comments”: when a new comment is posted in response to an old comment, the parent_id of the new comment is the id of the old comment. (You may assume that comments made directly on blog entries have parent_id = 0).

Find the id of each comment and the id of its “grandparent” comment; if it does not have a grandparent, omit it from the answer.

```
SELECT kid.id, parent.parent_id
FROM comments kid, comments parent
WHERE kid.parent_id = parent.id
AND parent.parent_id <> 0
```

OR

```
SELECT kid.id, grand.id
FROM comments kid, comments parent,
     comments grand
WHERE kid.parent_id = parent.id
AND parent.parent_id = grand.id;
```

B. XML

a. [4 points] Write an XPATH query that returns all comments associated with entries entitled “Midterm”.

```
//entry[@title="Midterm”]/comment
```
b. [4 points] Write an XPATH query to find all comments that are a “grandparent” of some other comment.

```xml
//comment/comment
```

c. [6 points] Using the XQuery language, write a FLWOR query that returns entries that have more than 5 comments (nested or otherwise) and an author name containing "Michael".

```xml
FOR $e IN /entry[contains(.//author, "Michael")]
  LET $c := $e//comment
  WHERE count($c) > 5
RETURN $e
```
3. **Sorting.** [8 points]

We would like to sort the tuples of a relation \( R(\text{column1, column2, column3, column4}) \) on a the sort key \( (\text{column1, column2, column4}) \). The following information is known about the relation.

- The relation \( R \) contains 100,000 tuples.
- The size of a block on disk is 4000 bytes.
- The size of each \( R \) tuple is 400 bytes.
- The size of each field in \( R \) is 4 bytes.
- A record pointer is 4 bytes.

Answer the following questions based on the information above.

**A.** [4 points] If we want to sort in two passes (using only Phase 0 and Phase 1), we need to know the minimum number of blocks \( B \) of main memory required. Provide (i) a formula for computing \( B \) correctly, and (ii) also give an integer value of \( B \) that guarantees a 2-pass sort.

**Part i:**

2 points: either solutions received full credit

\[
B(B-1) \leq 10000
\]

Or

\[
1 + \left\lceil \log_{B-1} \left(\frac{10000}{B}\right) \right\rceil = 2
\]

1 point for minor errors (e.g., forgot the ceiling, # pages did not appear anywhere).

**Part ii:**

2 points:

\( B = 101 \)

**B.** [4 points] Assume we have sufficient memory to perform the sort in two passes. What is the cost, in terms of number of disk I/Os, of sorting relation \( R \)? Include the cost of the writing the sorted file to the disk in the end in your calculations.

**4 points:**

2 passes, read in and write out 10000 pages during each pass.

\( 4 \times 10000 = 40,000 \) I/Os
Problem 4: ER Diagrams

Teaches - a given Faculty member will only be able to teach a given Class during one semester.
Classes - Classes with the same department and course number are no longer distinct over different semesters.
Enrolled - a given Student will only be able to take a given Class during one semester.

Recall that the primary keys of entities taken all together become primary keys in the relation!
Problem 5: Query Optimization

Answer 5(a):
Since fromUrl is a foreign-key reference, the number of tuples in the join result (without any selections applied) is \(|L|\) (referential integrity); thus, the selectivity of the join operator is \(\frac{|L|}{|P|*|L|} = \frac{1}{|P|} = 1/10^6\).
Assuming uniformity (for both the pagesize and author attributes), the selectivities for the pagesize and author conditions in the query are \((5000-0)/(20000-0) = \frac{1}{4}\) and \(1/10^4\), respectively.
Thus, assuming independence for the conditions, the overall selectivity is:
\[
\frac{|L|}{|P|*|L|} \times \frac{5000 - 0}{20000 - 0} \times \frac{1}{10^4} = 25 \times 10^{-12}
\]

Answer 5(b):
The complete scan of webpages costs \(10^5\) page IOs. Assuming uniformity over pagesizes, the number of webpages tuples that satisfy the pagesize condition in the query is:
\[
10^6 \times \frac{20000 - K}{20000} = (1 - \frac{K}{20000}) \times 10^6
\]
Similarly, the number of relevant index pages that must be scanned during the index scan is \((1-K/20000)\times10^3\). Since we have an unclustered index and get no benefit from buffering, we must do a data-page IO for each qualifying webpages tuple. Thus, the crossover value of K is determined by the equation:
\[
(1 - \frac{K}{20000}) \times (10^6 + 10^3) = 10^5
\]
which gives \(K = 18002\). Thus, for \(K > 18002\) the unclustered index scan solution is better (and vice versa).

Answer (Extra Credit):
Consider the \(n=2\) case. Assuming we evaluate selection 1 before selection 2, the overall CPU cost (since the predicates are independent) is: \(|R|^c1 + |R|^s1^c2\). (This is because only \(|R|^s1\) tuples “survive” selection 1.) Similarly, for the plan that evaluates selection 2 before selection 1, the cost is: \(|R|^c2 + |R|^s2^c1\). Thus, evaluating selection 1 first is a better strategy, iff
\[
c2 + s2 \times c1 > c1 + s1 \times c2 \quad \text{or} \quad \frac{c2}{1-s2} > \frac{c1}{1-s1}
\]
In the general case, the optimal strategy is to evaluate the \(n\) selections in increasing order of the \(ci/(1-si)\) ratio. Intuitively, this rule says we should evaluate selections in order of their cost/benefit ratio, where cost is the per tuple cost, and benefit is the percentage of tuples they filter out (i.e. 1-selectivity).