This exam is open book and notes. You can use a calculator and your laptop to access course notes and videos (but not to communicate with other people). You have 140 minutes (2 hours, 20 minutes) to complete it.

Print your name: ________________________________

The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

Signed: ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 (10 points)

(a) Consider the following schedule for transactions $T_1$, $T_2$, $T_3$:
$R_1(A)W_1(C)R_2(B)W_3(C)W_1(A)R_3(B)R_1(A)W_2(B)R_2(C)W_3(B)R_3(A)$

Below are some pairs of transactions. Say 'yes' if the first transaction in the pair must precede the second one in any equivalent serial schedule (if it exists), and 'no' otherwise.

$T_1, T_2$:________________

$T_3, T_2$:________________

$T_2, T_3$:________________

$T_3, T_1$:________________

(b) Consider a schedule:
$S_1 : R_1(B)W_3(A)R_2(C)W_2(C)R_3(C)W_1(B)R_3(B)W_3(C)$

Is $S_1$ serializable? :_____________

Is $S_1$ 2-phase lockable? :_____________

Consider a schedule:
$S_2 : R_1(A)W_1(A)R_2(B)W_2(C)R_2(A)R_3(C)W_3(B)R_1(B)$

Is $S_2$ serializable? :_____________

Is $S_2$ 2-phase lockable? :_____________

2
Problem 2 (10 points)

Consider a multi-granularity locking system, with lock modes S,X,IS,IX,SIX. The object hierarchy is as follows: There is a root R, with blocks A and B under it. Block A contains records a1 and a2, while block B contains records b1 and b2.

(a) The next four questions give a pair of locks each, answer 'yes' if that pair of locks is compatible, and 'no' otherwise.

If a lock is on something other than the root, assume that other suitable locks on parents have been taken, and consider compatibility with those other locks as well. For example, S on B is considered incompatible with X on b1, since the X on b1 will require IX on B.

S on b1, X on B: ________________

IX on R, IX on R: ________________

SIX on A, X on a2: ________________

IS on A, S on a1: ________________

(b) For the next three questions, give the list of locks (both lock type and object being locked) required to do the operation mentioned. Start with the lock on the root, and go downwards in the hierarchy. The answer should have a format like: IS(R),IS(A),S(a1).

Modify a1: ________________

Insert a record into B: ________________

Scan A (read all records in A): ________________
Problem 3 (10 points)

Consider a database that has six elements - A, B, C, D, E and F. The initial values of the elements are:

\[ A = 10, B = 20, C = 30, D = 40, E = 50, F = 60 \]

Let there be three transactions U, V and W that modify these elements concurrently.

- U: A := 5, B := 15, D := 30
- V: C := 25
- W: E := 35, F := 45

While the elements are being modified by the transactions, the database system crashes. The recovery mechanism depends on the logging scheme we use. In the questions below, we present the contents of the log at the time of crash and state the logging scheme used. (For each log entry we give the relevant data, as discussed in class examples.)

For each question, your task is to identify if a combination of values of the elements is possible in the disk at the time of crash and to help recover the database elements.

(a) Pure undo logging is used.

\[ \text{< START } U \text{ >} \]
\[ \text{< } U, A, 10 \text{ >} \]
\[ \text{< START } V \text{ >} \]
\[ \text{< } U, B, 20 \text{ >} \]
\[ \text{< } V, C, 30 \text{ >} \]
\[ \text{< COMMIT } V \text{ >} \]
\[ \text{< START CKPT(U) >} \]
\[ \text{< START } W \text{ >} \]
\[ \text{< } U, D, 40 \text{ >} \]

Is the following state of database elements (on disk) possible at the time of crash?
A = 5, B = 20, C = 25, D = 40, E = 50, F = 60 : (True/False)

What are the values of the database elements after a successful recovery?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
</table>
(b) Pure redo logging

\[
<\text{START } U > \\
< U, A, 5 > \\
< \text{START } V > \\
< U, B, 15 > \\
< V, C, 25 > \\
< \text{COMMIT } V > \\
< \text{START } \text{CKPT}(U) > \\
< \text{START } W > \\
< U, D, 30 > \\
< W, E, 35 > \\
< \text{COMMIT } W >
\]

Is the following state of database elements (on disk) possible at the time of crash?
A = 5, B = 15, C = 25, D = 30, E = 35, F = 60 : (True/ False)

What are the values of the database elements after a successful recovery?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Undo-redo logging

\[
<\text{START } U > \\
< U, A, 10, 5 > \\
< \text{START } V > \\
< U, B, 20, 15 > \\
< V, C, 10, 25 > \\
< \text{COMMIT } V > \\
< \text{START } \text{CKPT}(U) > \\
< \text{START } W > \\
< U, D, 40, 30 > \\
< W, E, 50, 35 > \\
< \text{END } \text{CKPT} > \\
< \text{COMMIT } U > \\
< W, F, 60, 45 >
\]

Is the following state of database elements (on disk) possible at the time of crash?
A = 10, B = 20, C = 25, D = 40, E = 50, F = 60 : (True/ False)

What are the values of the database elements after a successful recovery?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 4 (10 points)

Consider a database with six elements A, B, C, D, E and F. There are five transactions T, U, V, W and X that read and write to these database elements. The times at which the transactions start, try to validate and finish are as in the diagram below. The read and write sets of transactions T, U, W and V are also indicated.

(a) Does the transaction U validate? (Yes/ No)  

(b) Does the transaction V validate? (Yes/ No)  

(c) Does the transaction W validate? (Yes/ No)  

(d) If we know that the transaction X validates, give the list of possible elements in the read set and write set of X.

Elements that could be in the read set of X  

Elements that could be in the write set of X  

| = start  
× = validate  
O = finish
Problem 5 (10 points)

Consider a select query over relation $R(A,B,C)$ with condition $(A = a) \land (B = b)$. We are interested in estimating the number of resulting tuples.

(a) Using the simple statistics covered in class, $T(R)$, $V(R, A)$ and $V(R, B)$, what is the expected number of tuples in the result, $E_1$?

Expected result size, $E_1 = \phantom{0}0\phantom{0}$

(b) Suppose that in addition we maintain $V(R, A, B)$, the number of distinct $A$ and $B$ value pairs that appear in $R$. For example, if $R$ contains these three tuples (and only these tuples): $(1,2,20), (1,2,21), (2,3,20), (2,5,40), (3,8,20), (3,8,45), (3,8,55)$, then $V(R, A, B) = 4$.

With this new statistic, what is a better estimate for the result size, $E_2$? (This new estimate handles the correlation between $A$ and $B$ better than the previous estimate.)

Expected result size, $E_2 = \phantom{0}0\phantom{0}$

(c) The ratio $E_2/E_1$ tells us how much larger the improved estimate is. For example, if $E_2/E_1 = 5$, then the improved estimate is 5 times larger than the old estimate. Write expressions (as a function of $T(R)$, $V(R, A)$ and $V(R, B)$) for the minimum and maximum values $E_2/E_1$ can take.

Minimum value for $E_2/E_1 = \phantom{0}0\phantom{0}$

Maximum value for $E_2/E_1 = \phantom{0}0\phantom{0}$

(d) Is it easier to maintain $V(R, A)$ and $V(R, B)$ or the new statistic $V(R, A, B)$?

Circle easier values to maintain: $(V(R, A) \text{ and } V(R, B))$ \text{ or } $(V(R, A, B))$
Problem 6 (10 points)

You have implemented an extensible hash index, and you want to make it perform well when concurrent transactions access the index.

The following pseudo-code implements your extensible hash index. There are two calls you handle, `findKey` and `insertRecord`. (Ignore deletions for this problem.) Variable `d` represents the directory.

```plaintext
Procedure findKey(k) -> record r
  [ b <- findBucket(d, k);
    r <- findRecord(b, k);]

Procedure insertRecord(d, k, r)
  [ b <- findBucket(d, k);
    if isThereSpace(b) = true then
      insert(b, k, r)
    else
      [ doubleDirectory(d);
        splitBucket(k, b);
        b <- findBucket(d, k);
        insert(b, k, r) ] ]
```

The meaning of the functions in this pseudo-code should be clear to anyone who has taken CS245. For instance, `isThereSpace(b)` returns true if there is room in bucket `b` to insert one more key.

Our goal now is to add lock and unlock actions to this code so that we follow the tree locking protocol. In our case the tree has only two levels: The root is the directory `d` and the leaves are the hash buckets. (Assume that a bucket and all of its overflow buckets constitute a single tree leaf node.)

We have available the following locking operations:

- **RL(n)**: read (share) lock tree node `n`.
- **WL(n)**: write (exclusive) lock tree node `n`.
- **U(n)**: unlock tree node `n` (works for either read or write locks)

In the code on the next page please add the locking actions that would implement the tree locking protocol. You want to be as efficient as possible, so do not write lock a node if you can get away with a read lock. Also, release locks as soon as it is safe to do so.

For simplicity you do not have to show locks for a data record `r`.

Finally, note that some of the black lines next page may need no lock, while others may need more than one lock action. If there is more than one lock on a line, write the actions in the correct order (left to right).
Procedure findKey(k) -> record r
[ __________________________
  b <- findBucket(d, k);
  __________________________
  r <- findRecord(b,k);
  __________________________]

Procedure insertRecord(d, k, r)
[ __________________________
  b <- findBucket(d, k);
  __________________________
  if isThereSpace(b)= true then
    [ __________________________
      insert(b, k, r)
    __________________________ ]
  else
    [ __________________________
      doubleDirectory(d);
    __________________________
      splitBucket(k, b);
    __________________________
      b <- findBucket(d, k);
    __________________________
      insert(b, k, r)
    __________________________ ] ]
Problem 7 (10 points)

Consider logical action logging and the recovery strategy discussed in class. The recovery strategy is reproduced at the end of this problem (Notes 10, slides 73-74). Note that step numbers have been added (e.g., 1.1, 2.1), so you can refer to them in your answers below.

(a) Say you want to only record undo logical actions in the log. (You will not write any redo log entries in the log.) Which of the following rules will have to be enforced as you process transactions? (Your answer must include all relevant rules, and may include any number of these rules.)

(R1) Flush database actions by transaction $T$ to disk only after the $T$ commit record has been written to the log.
(R2) Before transaction $T$’s commit record is written to the log, all of $T$’s actions must have been flushed to the database.
(R3) The log record for an action of $T$ is only written to disk after the action has been reflected on the database on disk.
(R4) The log record for an action of $T$ must be written to disk before the action is reflected on the database on disk.
(R5) Before the $T$ commit record is written to the log, all of the action log records of $T$ must be written to the log.
(R6) The $T$ commit record must precede all of the action log records of $T$ in the log.

Rules to enforce with pure undo logical logging:

(b) Is it possible to simplify the general recovery strategy for the case where we only do undo logging? In your answer specify which steps (1.1, 1.2, 1.3, 2.1, 2.2, 2.3) if any can be safely removed. (You can of course assume that the rules you indicated in part (a) are enforced.)

Steps that can be removed with pure undo logical logging:

(c) Questions (c) and (d) are analogous to questions (a) and (b), but now for the pure redo logging case. Say you want to only record redo logical actions in the log. (You will not write any undo log entries in the log.) Which of the following rules will have to be enforced as you process transactions? (Your answer must include all relevant rules, and may include any number of these rules.)

(R1) Flush database actions by transaction $T$ to disk only after the $T$ commit record has been written to the log.
(R2) Before transaction $T$’s commit record is written to the log, all of $T$’s actions must have been flushed to the database.
(R3) The log record for an action of $T$ is only written to disk after the action has been reflected on the database on disk.

...
(R4) The log record for an action of $T$ must be written to disk before the action is reflected on the database on disk.

(R5) Before the $T$ commit record is written to the log, all of the action log records of $T$ must be written to the log.

(R6) The $T$ commit record must precede all of the action log records of $T$ in the log.

Rules to enforce with pure redo logical logging:

(d) Is it possible to simplify the general recovery strategy for the case where we only do redo logging? In your answer specify which steps (1.1, 1.2, 1.3, 2.1, 2.2, 2.3) if any can be safely removed. (You can of course assume that the rules you indicated in part (c) are enforced.)

Steps that can be removed with pure redo logical logging:

Recovery Strategy ******* FOR REFERENCE ***************

[1] Reconstruct state at time of crash
   [1.1] Find latest valid checkpoint, Ck, and let ac be its set of active transactions
   Scan log from Ck to end:
      [1.2] For each log entry [lsn, page] do:
         if lsn(page) < lsn then redo action
      [1.3] If log entry is start or commit, update ac

[2] Abort uncommitted transactions
   [2.1] Set ac contains transactions to abort
   [2.2] Scan log from end to Ck :
      For each log entry (not undo) of an ac transaction,
         undo action (making log entry)
   [2.3] For ac transactions not fully aborted,
      read their log entries older than Ck an undo their actions