The purpose of this problem set is to give you some practice with concepts related to recovery, parallel query processing, two-phase commit, and other papers we read during the second half of the course.

1 Locks

You have a database of movies and actors, with a moviesacted table that represents the many-to-many relationship between movies and actors, as follows:

movies : (id int, title char(100), year int)
actors : (id int, name char(100), salary int)
moviesacted : (movieid int, actorid id,)

There are 2 clustered B+Trees: one movies.year and one on actors.name.

There are 20,000 actors and 10,000 movies, the moviesacted table has 100,000 rows, and integers occupy 4 bytes. Data pages hold 1,000 bytes, and each index page holds 100 pointers and key values.

The earliest movie in the database is data 1990, and each year has approximately the same number of movies. For each of the following queries:

• Indicate the query plan that the database would most likely use to answer the query.

• Estimate the number of read (S) and write (X) locks that would be acquired by the execution of your plan assuming the use of page level locking. Assume that locks must be acquired on both index and data pages before they are read or written.

• Estimate the the number of read and write locks that would be required by your plan assuming row locking? Suppose that locks must be acquired for each distinct value read from an index (e.g., if the actor with id 10 is read from the database, a single index lock is acquired.)

1. [3 points]:
   a. SELECT title FROM movies WHERE year > 2004
   b. UPDATE actors SET salary = $10M WHERE name = Brad Pitt

2. [2 points]: Describe a situation in which switching from row to page level locks would likely make processing of some workload much faster. You can use different queries than in the above questions; please explicitly write down the transactions you have in mind. Explain the reasons for the increase in performance.
2 Serializability

3. [6 points]: Below, we provide 3 interleavings of several concurrent transactions (consisting of READ and WRITE statements on objects).

Your job is to indicate whether, for each of the interleavings:

– The interleaving has an equivalent serial ordering. If so, indicate what the serial ordering is.
– Whether the interleaving would be permitted/could arise under strict two-phase locking (where write locks are held until end of transaction).
– Whether the interleaving would be permitted/could arise under rigorous two-phase locking (where read and write locks are held until end of transaction).
– Whether the ordering would be valid using snapshot isolation. If not, indicate which transaction will be aborted.

Assume that snapshot isolation is implemented in the same way as optimistic concurrency control in the paper by Kung and Robinson, except that read sets are not tracked (and read-write / write-read conflicts are ignored.) You can assume that serial validation used in the write phases of optimistic concurrency control happen during the COMMIT statement (and no sooner.)

Assume in all cases that written values depend on previously read values.

Interleaving 1:
1 T1: READ A
2 T2: READ B
3 T1: WRITE A
4 T2: WRITE B
5 T1: READ B
6 T1: COMMIT
7 T2: COMMIT

Interleaving 2:
1 T1: READ B
2 T2: READ A
3 T3: READ C
4 T2: WRITE B
5 T2: COMMIT
6 T3: WRITE A
7 T3: COMMIT
8 T1: WRITE A
9 T1: COMMIT

Interleaving 3:
1 T1: READ A
2 T1: READ B
3 T2: READ A
4 T2: READ B
5 T2: WRITE C
6 T2: COMMIT
7 T1: WRITE B
8 T1: COMMIT
# Recovery

4. **[4 points]**: Suppose you are told that the following transactions are run concurrently on a database system that has just been restarted and is fully recovered, and is running Rigorous 2PL.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>RA</td>
<td>RB</td>
<td>RC</td>
<td>RD</td>
</tr>
<tr>
<td>WA</td>
<td>RE</td>
<td>WB</td>
<td>WD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Suppose the system crashes after executing WB in T3. Assume that each object (e.g., A, B, etc.) occupies exactly one page of memory.

A. Show all of the records that should be in the log at the time of the crash (given your serial order), assuming that there have been no checkpoints and that you are using an ARIES-style logging and recovery protocol. Your records should include all of the relevant fields described in Section 4.1 of the ARIES paper. Also show the status of the transaction table (as described in Section 4.3 of the ARIES paper) after the analysis phase of recovery has run.

B. Suppose you have 3 pages of memory, and are using a STEAL/NO-FORCE buffer management policy as in ARIES. Given the interleaving you showed above, for each of the 5 pages used in these transactions, show one possible assignment of LSN values for those pages as they are on disk before recovery runs. You should use the value “?” if the LSN is unchanged from the prior state of the page before these transactions began. Finally, indicate which pages will be modified during the UNDO pass, and which will be modified during the REDO pass.
4 Parallel Query Processing

The standard way to execute a query in a shared-nothing parallel database is to horizontally partition all tables across all nodes in the database. Under such a setting, distributed joins can be computed by repartitioning tables over the join attributes.

Suppose these tables are partitioned across a 3 node distributed database, where each node has 1 TB of disk storage and 10 GB of RAM. Also, assume that:

- The disk can read at 100 MB/sec (for the purposes of this problem, you may ignore differences between sequential and random I/O),
- The network on each node can transmit data at 1 GB/sec, regardless of the number or rate at which other nodes are simultaneously transmitting,
- A computer cannot send over the network and read or write from its disk at the same time,
- CPU costs are negligible relative to disk or network I/O time.

For each of the following queries, indicate what partitioning strategy you would recommend. Assuming the query in the question is the only query run by the system, your job is to choose the best up-front partitioning for this particular query / setting. Choose between replicating each table on all nodes, or hash, range, or round-robin partitioning it. For hash partitioning, specify the attribute you would use; for range partitioning specify the ranges you would use.

5. [3 points]:

SELECT COUNT(*)
FROM R JOIN S
ON R.a = S.b // S.b is a primary key
WHERE S.c > 200 // selectivity of .01

Here, S.b is a primary key, R is 2 TBs and several billion records, and S is 100 MB and several million records.
Indicate the best partitioning / replication strategy (and partitioning attributes) for R and S, as well as a brief explanation.

6. [3 points]:

SELECT COUNT(*)
FROM R JOIN S
ON R.a = S.b // S.b is a primary key
JOIN T on R.c = T.d // T.d is a primary key
WHERE T.e = 1 // T.e is uniformly distributed in [1..100]
AND S.c > 200

Here, S.b and T.d are primary keys, R is 2 TBs and several billion records, and S is 1 TB and several hundred million records, and T is 1 GB and several million records.
Indicate the best partitioning / replication strategy (and partitioning attributes) for R, S, and T as well as a brief explanation.
5  Locking, Redux

Consider the following transactions, with initial values of $X=3$ and $Y=5$. For writes we also indicate the values that are written (i.e., $\text{WY} ; Y=tx*2$ means that a write to $Y$ is done that sets its value to $tx*2$.)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tx = RX$</td>
<td>$ty = RY$</td>
<td>$WY ; Y=0$</td>
</tr>
<tr>
<td>$WY ; Y=tx*2$</td>
<td>$WX ; X=ty+1$</td>
<td></td>
</tr>
</tbody>
</table>

7. [2 points]: Suppose you know that the first operation that runs is $RY$ in T2. Assuming there are no deadlocks, list all possible values for $X$ and $Y$ that could result from an execution of these transactions with strict two-phase locking?

8. [2 points]: Again, assuming the first operation that runs is $RY$ in T2, list all possible values that could arise for $X$ and $Y$ from an execution of these transactions using snapshot isolation? Assume snapshot isolation behaves identically to serially-validated OCC, except that read sets are not tracked (so there is no need to check that the read set of a validating transaction intersects with the write set of any concurrent transaction).