6.830 Problem Set 3 (2021)

Assigned: Monday, Apr 12, 2021
Due: Monday, May 10, 2021
Updates: Monday, May 4, 2021 to fix typo in Q6
Sat, May 8, 2021 - Fix definitions of read uncommitted / read commited in Q10/11

Submit to Gradescope `https://www.gradescope.com/courses/229885`

You may work in pairs on this problem set. Only one of you needs to submit on Gradescope, but the other member must be added as a group member on the submission.

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 Transactions and Locks

Consider again the Olympics database we used in earlier assignments. Specifically, consider the following modified version of the athletes database:

```
CREATE TABLE athletes(
  id INTEGER PRIMARY KEY,
  noc_id INTEGER,
  name VARCHAR(255),
  num_medals INTEGER
);

CREATE INDEX num_medals_athletes ON athletes (num_medals, id);

CREATE TABLE noc(
  id INTEGER PRIMARY KEY,
  name VARCHAR(255),
  num_medals INTEGER
);

CREATE INDEX num_medals_noc ON noc (num_medals);

CREATE TABLE athlete_events(
  athlete_id INTEGER,
  medal VARCHAR(255)
);
```

For this question, assume athletes participate in games either by themselves or in teams of 2. Athletes on the same team are always from the same country/territory (noc). We are interested in both the number of medals won by each athlete and the total number of medals won by each noc. However, we need to take care not to double count medals when a medal is given to a team. Suppose we are building an application for the local TV station. The TV station wants to:

- display the current leaderboard of nocs with the most medals.
- for the athlete participating, list her rivals who has won the same number of medals in the past.

- update this information real-time.

Having taken 6.830, you come up with the following list of transactions for this application:

```
CREATE PROCEDURE list_top_noc()
LANGUAGE SQL
AS $$
  SELECT name FROM noc ORDER BY num_medals LIMIT 10;
$$;

CREATE PROCEDURE list_rivals(a_id VARCHAR(255))
LANGUAGE SQL
AS $$
  SELECT
    id, num_medals
  FROM
    athletes
  WHERE
    num_medals = (SELECT num_medals FROM athletes WHERE id = a_id);
$$;

CREATE PROCEDURE record_outcome_single(athlete_id INTEGER, medal VARCHAR(255))
LANGUAGE SQL
AS $$
  BEGIN;
    INSERT INTO athlete_events VALUES(athlete_id, medal);
    UPDATE athletes SET num_medals = num_medals + 1 WHERE id = athelete_id;
    UPDATE noc SET
      num_medals = num_medals + 1
    WHERE
      id = (SELECT noc_id FROM athletes WHERE id = athlete_id);
  COMMIT;
$$;

CREATE PROCEDURE record_outcome_double(athlete_id_1 INTEGER,
                                       athlete_id_2 INTEGER,
                                       medal VARCHAR(255))
LANGUAGE SQL
AS $$
  BEGIN;
    INSERT INTO athlete_events VALUES(athlete_id_1, medal);
    INSERT INTO athlete_events VALUES(athlete_id_2, medal);
    UPDATE noc SET
      num_medals = num_medals + 1
    WHERE
      id = (SELECT noc_id FROM athletes WHERE id = athlete_id_1);
    UPDATE athletes SET num_medals = num_medals + 1 WHERE id = athelete_id_1;
    UPDATE athletes SET num_medals = num_medals + 1 WHERE id = athelete_id_2;
  COMMIT;
$$;
```

Here, we are writing stored procedures, which can be thought of as templates of SQL queries that are executed as a whole. For example, one can execute `record_outcome_single(42, 'gold')` against the database, and the database replaces the above statement with the body of the stored procedure declaration when executing. For this question, you can assume that users will only interact with the database by calling the given stored procedures.

Assume further that the DBMS in question uses B+ tree indexes and strict 2-phase locking for transactions. Locks are page-level. Transactions are executed under `SERIALIZABLE`. All indexes are unclustered.

1. [2 points]: For transaction types `list_rivals` and `record_outcome_single`:

- Describe (informally) the query plan that will likely execute.
- Estimate the number of read (S) and write (X) locks that would be acquired by the execution of your plan. Include locks for both index pages and heap file pages.

Assume for simplicity that each heap page holds 100 tuples (for all three tables) and each index page holds 100 pointers and key values (again for all three tables). You may assume the number of athletes with the same number of medals is small (< 100).

2. [1 points]: After running a few test cases, you notice that your `record_outcome_single` and `record_outcome_double` transactions are sometimes aborted. What might be the cause of this, and are there some ways to lower the abort rate?

3. [1 points]: Can `list_top_noc` ever be aborted by the system? Why or why not?

4. [1 points]: What performance problem(s) may arise with page-level locking if `record_outcome_single` and `record_outcome_double` make up 90% of the workload issued? Name one and explain how changing the locking scheme within the database implementation may mitigate the problem.

2 Serializability

5. [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
```

3 Recovery

6. [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.

```
T1      T2      T3      T4      T5
--      --      --      --      --
RB
                                RE
                                RD
        RB
        RC
        WC
                RD              WE
                                COMMIT
                        RE
WA
                        WE
                        COMMIT
WA
ABORT
                WD
```

Suppose the system crashes after executing WD 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 10 TB of disk storage and 2 GB of RAM. Also, assume that:

- The disk can read at 1 GB/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.
- The network costs to compute the final aggregate are negligible.

For the following query and partitioning strategies of the database, estimate the total runtime of the query. Show your work by drawing or describing the query plan. All joins are in-memory hash joins or Grace hash joins. The database is a row-store. Assume projections and selections are done as early as possible.

```
SELECT COUNT(*)
FROM R JOIN S
ON R.a = S.b //S.b is a primary key, R.a is uniformly distributed
WHERE S.c > 100 //selectivity of .1
```

Here, S.b is a primary key, R contains 3 billion records, S contains 9 billion records. All fields and COUNTs are integers, and integers are 8 bytes (64 bits). Each table contains 10 integer fields, so the size of the tables are:

$$|R| = 3B \times 8 \times 10 = 240GB$$

$$|S| = 9B \times 8 \times 10 = 720GB$$

7. [2 points]: R is hash partitioned across all three on R.a and S is hash partitioned on S.b. Estimate the total query runtime, showing your work.

8. [2 points]: Now suppose R is hash partitioned across all three nodes on R.a and S is hash partitioned on S.c. Estimate the total query runtime, showing your work.

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., `WY ;Y=tx*2` means that a write to Y is done that sets its value to `tx*2`.)

```
T1              T2              T3
----------      ----------      --------
tx = RX         ty = RY         WY ;Y=0
WY ;Y=tx*2      WX ;X=ty+1
```

Suppose you know that the first operation that runs is RX in T1.

9. [2 points]: Assuming there are no deadlocks, list all possible values for X and Y that could result from an execution of these transactions with rigorous two-phase locking at the serialization level SERIALIZABLE?

10. [2 points]: Now suppose you run these transactions at the serialization level READ COMMITTED. List all possible values for X and Y that could result from an execution of these transactions? Assume READ COMMITTED is implemented as described in lecture 13, i.e., transactions only hold short read locks (continuing to hold write locks for the duration of the transaction).

11. [2 points]: Now suppose you run these transactions at the serialization level READ UNCOMMITTED. List all possible values for X and Y that could result from an execution of these transactions? Assume READ UNCOMMITTED is implemented as described in lecture 13, i.e., transactions don't acquire read locks at all (continuing to hold write locks for the duration of the transaction).