Revision 11/2/2022

ComS 363 Fall 2022
Homework 4

Percentage in your final grade: 10%
Maximum score for the assignment: 150 points
Objectives:

1. Gain deeper understanding about transaction management.
2. Gain deeper understanding about database internals.

Submission instruction:

Submit a zip file named <netid>HW4.zip with the following files: Q1.pdf, Q2.pdf, Q3.pdf, Q3.sql in them. The <netid> is replaced with your university netid.

Questions:

1. (25 points) Consider the following actions taken by transaction T1 on database objects X and Y.
   T1: R(X), W(X), R(Y), W(Y)

   a. (7 points) Give an example of another transaction T2 that, if run concurrently to transaction T1 without concurrency control, does not interfere with T1. Provide reasons to support your answer.

      Hint: Review the lectures about what are considered as conflicts by DBMS.

   b. (9 points) Give an example of another transaction T3 that, if run concurrently to transaction T1 without some form of concurrency control, could create a write-read conflict with T1. Provide a schedule that shows the write-read conflict caused by concurrently running T1 and T3. Do not give a serial schedule as an example. Specify in your schedule where the write-read conflict occurs.

      Assume that both T1 and your T3 want to commit. Show how the Strict Two Phase Locking protocol (Strict 2PL) can disallow the schedule.

   c. (9 points) Give an example of another transaction T4 that, if run concurrently to transaction T1 without some form of concurrency control, could create a read-write conflict with T1. Provide a schedule that shows the read-write conflict caused by concurrently running T1 and T4. Do not give a serial schedule as an example.

      Assume that both T1 and your T4 want to commit. Show how the Strict Two Phase Locking protocol can disallow the schedule.

   To earn a full credit, the description of how 2PL works on your schedule for 1.b and 1.c should describe when a transaction is granted what type of locks using the notation S() for a shared lock and X() for an exclusive lock, when a transaction waits on a lock, when a transaction commits (if not killed by the deadlock handling mechanism), when a lock is released, and what happens if there exists another transaction waiting on the released lock. You can assume that a write action immediately after a read action for the same object in a transaction happens in the same SQL statement. You can assume that a deadlock handling mechanism aborts a younger transaction (starting at a later timestamp) when a deadlock is detected.

   Hint: See the lectures on WK10 practice problems and solutions.

   Put all your answers in Q1.pdf.

2. (60 points) Suppose a relational DBMS maintain a database of two relations with the schemas below.

   Users(screen_name varchar(80), user_name varchar(80), category varchar(80), subcategory varchar(80), state varchar(80), numFollowers int, numFollowing int, primary key(screen_name));

```
Tweets(tid BIGINT, post_day int, post_month int, post_year int, retweetCt int, posting_user varchar(80) not null,
    primary key(tid), foreign key(posting_user) references Users(screen_name));

/* Query 1 */
SELECT u.screen_name
FROM Users u where u.screen_name = 'ajc';

/* Query 2
   Note: that the temp table exists and is empty; it has only one attribute and has no primary key
   temp (tid bigint)
*/
INSERT into temp
SELECT tid
FROM Tweets where posting_user='ajc';
```

a) (30 points) For each of the queries, draw a query execution plan using relational operators when a full table scan is used. Estimate the disk I/O cost (worst-case scenario in terms of the number of pages/disks block) for this plan using the assumptions below.
b) (30 points) For each of the queries, draw all possible single index access path query execution plans.

Assumptions

- An integer takes four bytes; a bigint takes eight bytes; a character takes one byte.
- The fixed-length record format is used. The size of each tuple is estimated as the sum of all bytes for all the attributes.
- A page size (disk block size) is 4,096 bytes where only 4,000 bytes are available for storing tuples. As many tuples as possible are stored per page.
- The Tweets relation has 10,000 tuples. The Users relation has 5,000 tuples. Each relation has a corresponding file to store all the rows in that relation. The rows are not sorted. The file format is the heap file format using the linked list implementation. The page format is the unpacked bitmap format. Ignore the I/O cost of reading the header page of the heap file as the information is small and it is kept in the catalog.
- One B+Tree index is available on each primary key. If the primary key has more than one attributes, one additional B+Tree index is created for each subsequent attribute of the primary key. In addition, one B+Tree index is available for each foreign key. Therefore, the tweets table has two indexes: one on the tid attribute and the other on the posting_user attribute. The users table has only one index on the screen_name attribute.
- The DBMS only supports full table scan and single-index access path plans.
- The selectivity factor for the where clause in Q2 is 0.1. In other words, only 10% of the input rows from Tweets satisfy the where clause.

3. (65 points) Optimize the following queries in MySQL by creating an appropriate index such that the DBMS uses the created index when executing the same query. The use of the index must reduce the disk I/O cost, which in many cases results in less query execution time. One index may be useful for more than one query.
See the resources on W3schools for creating an index: https://www.w3schools.com/sql/sql_create_index.asp

```
-- ID7
set @hashtag = "GOPDebate";
set @state_name = "North Carolina";
SELECT count(t.tid) as tweet_count, u.screen_name, u.category
FROM tweets as t, users as u, hashtags as h
WHERE t.posting_user = u.screen_name AND t.tid = h.tid and
h.name = @hashtag and u.state = @state_name and t.post_month = 2 and t.post_year = 2016
GROUP BY u.screen_name, u.category
ORDER BY tweet_count DESC
LIMIT 5;
```

```
-- ID23
SELECT h.name, COUNT(h.tid) as num_uses
FROM Tweets t, Users u, HashTags h
WHERE u.subcategory = 'GOP' AND FIND_IN_SET(t.post_month, '1,2,3')
    AND t.post_year = 2016  AND u.screen_name = t.posting_user AND h.tid = t.tid
GROUP BY h.name
ORDER BY num_uses DESC
LIMIT 5;
```

Fill in Table 1 below. For each query, run it twice before creating the index. Record the I/O cost reported by MySQL each time. See the class participation on WK 10 about how to check for the query execution plan. After you create your index, run the same query two times. Record the I/O cost reported by the DBMS for each run. The same query execution plan may have different costs because some of the data are already stored in a database buffer pool.

In your submission, include the filled Table 1 and two screenshots of the query execution plan for each query, one before the index is created and the other after creating the index. Make sure the screenshot shows the cost and show that the index gets used in the plan. Include Table 1 and these screenshots in Q3.pdf. Put the create index statement for each query in Q3.sql. Add a comment in Q3.sql to indicate which create index is for which query and indicate that you are the author of the solution.

Example: For the query select * from emp where eid=100;

The execution plan example below shows the total disk I/O cost of 1.0 at the top of the figure. It shows that the index on the primary key of emp is used.

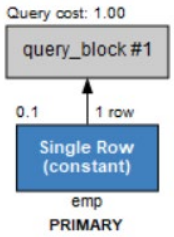

Table 1: MySQL Innodb buffer pool size: _____ Mbytes

Query ID	Create index statement for each query; your execution plan needs to use the created index and reduce disk I/O cost.	Before optimization (disk I/O cost)		After optimization (disk I/O cost)	
		Run1	Run2	Run1	Run2
ID7					
ID23					

Hint:

- To get the memory buffer pool size, run the following statement in MySQL.
  SELECT @@innodb_buffer_pool_size/1024/1024 as "buffer pool size in Mbytes";
- See the explanation in https://dev.mysql.com/doc/refman/8.0/en/innodb-buffer-pool-resize.html.

- To find which attribute to create an index on, look at the where clause that does not involve the join conditions and see whether an index on the attribute used in that where clause may result in a cheaper execution plan.

Practical scenarios: In practice, database administrator (DBA) does not create an index by using just one input values for each query. DBA identifies which query is frequently used and is slow. Then, look at multiple input values for that query to see whether the created index is useful. Also, we look at other options to optimize the query execution time such as increase the DBMS buffer pool size, partitioning the frequently used tables across multiple disks, and partitioning the data across multiple sites in the case of using a distributed DBMS.

Revision 11/2/2022