Problem Set #3: Transactions, Cost Models, and Indexes

Due: December 3, 2014 at 11:00am

COMMENTS ON GRADING:

- These problem sets will be graded roughly.
- If your answer is well written and contains a complete, correct answer for a question, you will get bonus points. These bonus points are rare.
- If you get basically everything right, but we find some mistake or gap in your answer, we will give you full credit. We hope to give you essentially full credit for each question.
- Partial credit will be given aggressively–so turn something in!

There will be provided solutions. As a result, problem set material is fair game for tests and quizzes.

Warning: These problems require you to write code, to run software, and to read material. This may be more challenging than previous assignments, start early!

1 Spooky Transactions!

It's never too late for halloween.

Spooky Serializability. This may require you look up locking granularity. Consider the following sequence of two transactions on a table $R(A, B)$:

- T1.
  - START TRANSACTION
  - SELECT COUNT(*) FROM R WHERE A = 6;
  - does some work ;
  - SELECT * FROM R WHERE A = 6;
  - END TRANSACTION
- T2.
  - START TRANSACTION
  - INSERT INTO R VALUES (6,20)
  - END TRANSACTION

Recall that if T1 and T2 are serializable, they should not be able to observe each other.

(a) What is the finest granularity of lock that one needs to guarantee that these two transactions are (conflict) serializable? Use the terminology from the slide on Multiple Granularity Locks.

(b) Suppose there is an index on A. Could you use this to take a lock that is finer granularity than in part (a) of the problem while still guaranteeing serializablity?

(c) What do *phantom tuples* have to do with this problem? Describe how they could occur in the above situation.

(d) Using Google or a text book, describe a *phantom deadlock*, an issue with deadlock detection in a distributed setting, in your own words. Use an example to illustrate how this spooky problem can happen.

2 Don't Drink and Hash Collide

If you don't know about hashing, this may require a brief primer from the book (but it is covered in a prereq).

Hipster Doofus (HD) is at it again. HD has to match a large number of incoming requests against a set of $T = 50000$ tinder profiles, and he wants to use a hash function to partition the profiles into $n = 100$ buckets. HD decides to code up a hash function for the Tinder Profile IDs:

$$\text{hdhash}(s_1, \ldots, s_k, n) = \sum_{i=1}^{k} s_i \bmod n,$$

where s_i represent the characters of a string.
This code can be represented as a Python one-liner:

```
def hdhash(s,n): return sum(map(ord,s)) % n
```

This gives output like:

```
>>> hdhash("TI00312",100)
3
```

The IDs look as follows, which HD thinks are pretty uniformly distributed:

TI00000
TI00001
TI00002
...
TI49998
TI49999

Here, each account starts with 'TI' followed by five digits (padded with 0 as shown). Each of the fifty thousand possible combinations of digits occurs exactly once. If this is a good hash function, it should be the case that each hash bucket has $T/n \approx 500$ values. However, HD's application runs more slowly than he expected!

(a) Help HD understand what is going on, by drawing a *histogram* of the bucket sizes. Submit a histogram of the data generated by any tool (if you don't have a preferred tool, Google Spreadsheets will do this for you). You should write some code to compute the above histogram. Please turn in any auxiliary material (code or scripts) that you used to generate the data.

(b) Using the data you collected from part (a), compute *how much slower* HD's application runs compared to what he expected. Here, we assume the running time for the application is *quadratic* in the number of items in a bucket: that is, if the buckets have $10, 30, 7, \ldots$ items each, then the running time is $10^2 + 30^2 + 7^2 \ldots$

HD expects a running time of $\approx 100 * 500^2$ since he expected all buckets to have the same size. Compute how much larger the actual running time is compared to this expectation.

(c) Design a better hash function that partitions HD's Tinder profiles more uniformly and show the new histogram. Compute the running time at point (b) for your new hash function. Notice that there is no best solution here, feel free to be creative.

Turn in both a program implementing the new hash function, a histogram, and a number indicating the relative running time.

3 No Free Lunch: Cost Models

3.1 Simple Cost Models

Consider the join of two relations $R(A, B)$ and $S(A, C)$.

(a) Describe a scenario where sort-merge join is cheaper than hash join.

(b) Describe a scenario where hash-join is cheaper than block nested loop join (BNLJ).

(c) Describe a scenario where BNLJ is cheaper than sort-merge join.

A scenario above refers to the sizes of R and S in pages and the amount of RAM available in pages. In particular, you should provide a short explanation of the intuition behind your scenario, as well as a short formula to explain your reasoning.

3.2 Cost Models in Postgres

Your goal is to find an SQL query Q, an original data set, and a set of modifications so that the following events can happen in order:

1. Postgres selects hash-join for a join in Q,
2. You apply your modifications to the database (e.g. update, insert, delete),
3. Postgres selects sort-merge join for the same join in Q.

You should turn in the transcript of your actions including the output of all commands, including the EXPLAIN command, as well as any script you write to generate the intial data for your database.

4 Duplicate Duplicates

Assume pages are of size 4096 bytes. You are given a massive file with N integers (8 bytes) that occupy $M = \lceil N/(4096/8) \rceil$ pages. Assume you have B pages of memory. Your job is to output all unique entries in the list. You may not assume that the unique entries fit in memory B. Do not include the output of the result in your IO cost.

(a) Design a sorting-based algorithm for the above problem and write down its cost in IOs in terms of the above parameters.

(b) Design a hashing-based algorithm for the above problem and write down its cost in IOs in terms of the above parameters. You may assume that there are no more than $(B-1) * 4096/8$ of a single integer.

(c) Turn in a screenshot of you running the EXPLAIN command discussed in class on a relation $R(A)$, along with a copy of the transcript creating and populating the table. Is it using one of your algorithms?

5 Clustered, Unclustered, and Who cares

Your goal is to come up with a single data set and a set of two or three queries:

1. A query in which using a clustered index runs the same speed as an unclustered index,
2. A query in which using a clustered index runs faster than an unclustered index,
3. **Bonus**: A query in which using a clustered index runs slower than an unclustered index.

Your data set should include the definition of the relation, and a description of the data. In each case, use a cost model to justify your answer.

6 Submission Instructions

To submit Problem Set 3, save your answers as a PDF (preferably typewritten using LaTeX or Microsoft Word, but a handwritten & scanned PDF is fine as long as it is legible).

When you are ready to submit, copy your files over to the `corn.stanford.edu` machines, and gather all of the following files in a single submission directory:

- Problem_Set_3.pdf
- Any code you wrote for Problem 2, named problem_2.___

Once your submission directory is properly assembled, **with no extraneous files**, execute the following script from your submission directory:

```
/usr/class/cs145/bin/submit-pset
```

Be sure to select "Pset3" when the script prompts you for which assignment you're submitting!

You may resubmit as many times as you like; however, only the latest submission and timestamp will be saved, and we will use your latest submission for grading your work and determining any late penalties that may apply. Submissions via email will not be accepted!