CS 347
Midterm Exam – Spring 2015

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 70 minutes to complete the exam.

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></td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
1. Consider a database table tracking performance stats of web requests. The schema is:

Requests(request_id, url, timestamp, client_ip, web_server_ip, request_size, response_size, http_code, latency_ms)

The key is “request_id”. HTTP code is the return code from the server, like 200, 404, 500, etc.

a. Imagine that by far the most common query against this table is:

   \[
   \text{SELECT url, \( \text{AVG(latency\_ms)\) FROM Requests GROUP BY url;}}
   \]

   Fragment the Requests table into two tables, using Vertical Fragmentation, in order to provide the best performance for this query. Express your fragments as a projection operation over the original table.

   \textbf{Answer:}
   Requests1 = Project(request_id, url, latency_ms) Requests
   Requests2 = Project(request_id, timestamp, client_ip, web_server_ip, request_size, response_size)

b. For each of the following horizontal fragmentations, state whether it is complete, disjoint, neither or both. Note that for strings, the ‘\(>=\)’ operator means “equal to or lexicographically larger than”.

   • Requests1 = Select(url < “http://www.google.com”) Requests; Requests2 = Select(url >= http://www.apple.com) Requests
     o \textbf{Answer: Complete but not disjoint}

   • Requests1 = Select(request_size < response_size) Requests; Requests2 = Select(request_size >= response_size) Requests;
     o \textbf{Answer: Complete and disjoint}

   • Requests1 = Select(http_code = 200) Requests, Requests2 = Select(http_code = 404)
     o \textbf{Answer: Disjoint but not complete}

   • Requests1 = Select(latency_ms >= 0) Requests, Requests2 = Select(latency_ms < 0) Requests
     o \textbf{Answer: Complete and disjoint}
c. Imagine that we have another table:

\[ \text{PageContents(object_id, url, contents)} \]

The “object_id” is the key of this table. The “url” is a foreign key referring to the “url” column of Requests. Imagine we use the following primary horizontal fragmentation of Requests:

\[ \text{Requests1 = Select(latency_ms < 200) Requests} \]
\[ \text{Requests2 = Select(latency_ms >= 200) Requests.} \]

Will the resulting derived horizontal fragmentation of PageContents be complete, disjoint, neither or both? Why?

**Answer:** The resulting horizontal fragmentation will be complete (because the join attribute is a foreign key) but not necessarily disjoint (because url is not a key of Requests.)
2. Consider the following relations:

\[\text{Site1: } \text{Player(player_id, first_name, last_name, email_address, subscription_plan)}\]
\[\text{Site2: } \text{MatchPlayed(match_id, date, home_player_id, visitor_player_id, game_name, home_score, visitor_score, log)}\]
\[\text{Site3: } \text{Games(game_name, description)}\]
\[\text{Site4: } \text{GamePricing(game_name, subscription_plan, fee_per_game)}\]

Keys are underlined. The following are foreign key relationships:

\[\text{MatchPlayed(home_player_id) } \rightarrow \text{ Player(player_id)}\]
\[\text{MatchPlayed(visitor_player_id) } \rightarrow \text{ Player(player_id)}\]
\[\text{MatchPlayed(game_name) } \rightarrow \text{ Games(game_name)}\]
\[\text{GamePricing(game_name) } \rightarrow \text{ Games(game_name)}\]
\[\text{Player(subscription_plan) } \rightarrow \text{ GamePricing(subscription_plan)}\]

Imagine we have the following statistics, where \(|X|\) means “number of bytes in \(X\)”. The asterisk “\(*\)” means “natural join”. TB means “Terabyte”.
\(|\text{Player}| = 0.01 \text{ TB}\]
\(|\text{MatchPlayed}| = 50 \text{ TB}\]
\(|\text{Games}| = 0.001 \text{ TB}\]
\(|\text{GamePricing}| = 0.001 \text{ TB}\]
\(|\text{Games * GamePricing}| = 0.002 \text{ TB}\]
\(|\text{Games * MatchPlayed}| = 70 \text{ TB}\]
\(|\text{MatchPlayed * Player}| = 110 \text{ TB}\]
\(|\text{Player * GamePricing}| = 0.011 \text{ TB}\]

We want to compute the following query, which determines how much a player has spent:

\[
\text{SELECT P.first_name, P.last_name, P.email_address, G.game_name, G.description, SUM(GP.fee_per_game)}
\text{FROM Player P, MatchPlayed M, Games G, GamePricing GP}
\text{WHERE (P.player_id = M.home_player_id OR}
\text{P.player_id = M.visitor_player_id)}
\text{AND M.game_name = G.game_name}
\text{AND G.game_name = GP.game_name}
\text{AND P.subscription_plan = GP.subscription_plan}
\text{GROUP BY G.game_name, G.description;}
\]

In the questions below, our cost metric is “amount of data sent between sites.”
a. What is the cost of the following plan:
   1. Send Games to Site2
   2. Join Games * MatchPlayed at Site2, and send the result to Site1
   3. Send Player to Site4
   4. Join Player and GamePricing at Site4, and send the result to Site1

   Answer: $0.001 + 70 + 0.01 + 0.011 = 70.022$ TB

b. Assume we are considering plans that send all base relations to a single site and compute the full query at that site. Which site is better to have the final results collected at: Site3 or Site4? Why?

   Answer: It doesn’t matter. In both cases the cost is the same for the best plan, which is to just ship the base relations to the final site.

c. Imagine that we fragment MatchPlayed by match_id into 15 fragments. Now want to sort MatchPlayed by “date” (and that this is not necessarily the same order that match_id sorts the data). The final sorted relation can be fragmented across 10 sites. Briefly describe an efficient algorithm for sorting the data, where “efficient” means “fast time to completion of the sort”, describing what data goes where.

   Answer: There are (at least) two options:

   **Option 1:** Choose 10 date ranges, such that each final site gets one range. Repartition the MatchPlayed data, sending tuples from the source sites to the final sites based on the date range at the final site. Sort each final site partition locally.

   **Option 2:** Choose 10 date ranges, such that each final site gets one range. Sort each initial fragment locally. Send each sorted subrange from the initial site to the final site where it belongs. Each final site performs a merge of the subranges it receives.
3. Consider the basic two-phase commit protocol with timeouts and finish state (e.g. slides 31 and 33 in Notes06.)

a. Imagine a coordinator fails, and upon recovery finds no information for a transaction with id T1 in its log. Which of the following statements might be true? Which of the following statements is definitely true? Why?

   a.1: The transaction T1 committed
   a.2: The transaction T1 aborted
   a.3: The transaction T1 either committed or aborted

Answer: They all might be true. None of them are definitely true. If there is no information in the log, we might have committed or aborted, reached F state, and forgotten the outcome. Or we might have never executed T1 at all.

You could also argue that a.3 is definitely true, if you assume that a transaction id exists only after the transaction starts (e.g. if id T1 exists, then the transaction must have started and entered initial state.)

b. Imagine a participant fails, and finds the following information in its log (assume T2, T3 etc are transaction ids, and this is still 2PC.)

   • T2: Redo(write E1.salary=100000)
   • T3: Redo(write E4.location="NY")
   • T2: Redo(write E1.position="manager-2")
   • T4: Redo(write E9.project="Alpha-migration")
   • T3: W
   • T4: W
   • T3: C

What should the participant do for each transaction? In particular, a) what locks should be acquired; b) what transactions should be in what states; c) what message(s) should be sent?

Answer:

For T2, the participant should consider the transaction aborted and send NOK to the coordinator.

For T3, the participant should consider the transaction committed, and replay the redo log entries. It is acceptable, but not required, for the participant to send C-OK to the coordinator. Note: it is not necessary to acquire locks to replay log entries during recovery for a committed transaction.

For T4, the participant should consider the transaction in W state. It should re-acquire a lock on E9.project, and send OK to the coordinator.

c. Assume that most 2PC transactions commit, rather than aborting. Is presumed-commit or presumed abort better under this assumption? Why?

Answer: Presumed commit is better. Since aborts are rare, keeping track of them until all participants have acked the abort is cheaper than keeping track of commits until all participants have acked the commit.
4. Consider a parallel database with two nodes N1 and N2.

A relation R is fragmented into R1 and R2: R1 resides at N1 and R2 at N2. In addition, R1 and R2 have 12000 tuples each. We want to sort R with respect to an attribute A that has an integer domain. The range of A values at R1 is [5,10] and the range of A values at R2 is [8,17]. Moreover, we do not keep histograms for A in R, hence, we assume a uniform distribution of tuples over the A value-range, both in R1 and R2. N1 is 9 times faster than N2 and the time to sort is proportional to \( K^2 \) where K is the number of tuples being sorted in that node.

In the parallel sort operation using a range-partitioning scheme, we want to decide a splitting point P, such that N1 will sort all tuples with \( A < P \) and N2 all tuples with \( A \geq P \). The best performance for parallel sort happens when the time to sort in each node is exactly the same, since we assume that the time to exchange data between the two nodes is negligible.

Give the optimal splitting point P (tuples with \( A < P \) --> N1, tuples with \( A \geq P \) --> N2).

Answer: In the range [8,17], there are 12000 tuples at R2, hence, 1200 per integer value.
We need to find a P s.t.

\[
(12000 + (P - 8) \times 1200)^2 = 9 \times ((17 - P + 1) \times 1200)^2 \\
10 + P - 8 = 3 \times (17 - P + 1) \\
P = 13
\]