CSE 562 Final Exam

May 8, 2013
Start Time: 11:45 AM
End Time: 2:45 PM

Name:
UBIT:
Person #:
Seat #:

Maximum number of points possible: 180. You have 3 hours (180 minutes). Including scratch pages, there are 12 pages in this exam booklet.
Questions vary in difficulty. Do not get stuck on one question. You may use one double-sided 8.5 x 11 sheet of notes, but no calculators or other computing devices.

Part	Question	Points Possible	Points Earned
A: Indices & Operators	1	10	
	2	10	
	3	5	
B: SQL	1	30	
C: Transactions	1	10	
	2	10	
	3	10	
	4	10	
	5	10	
	6	10	
D: Data Warehousing	1	5	
	2	10	
	3	5	
	4	10	
E: Potpouri	1.a	4	
	1.b	4	
	1.c	4	
	2	6	
	3	6	
	4	6	
	5	5	
	Total	180	

Part A. Relational Indices and Operators

(25 points)

You are the chief architect of the LateDB database system, and you are really running up against the product shipment deadline.

1. (10 points) You only have time to implement one join algorithm. Will you choose tuple-nested-loop join, block-nested-loop join, index-nested-loop join, sort-merge join, or hash join. In no more than 2 short sentences, justify your answer (i.e., clearly explain why you think your choice is the best among the 5 alternatives).

2. (10 points) You only have time to build one type of index structure. Will you choose B+-Trees, Extendible Hashes, or Linear Hashes? Justify your answer.

3. (5 points) You seem to have panicked too soon, and things are not as bad as they seem. So, you have time to implement one other index structure (in addition to the one you chose above). Which one will you choose, and why?

Part B. SQL

(30 points)

Consider the following schema for a stock-market database relation.

```
CREATE TABLE Stock (
  tickersymbol CHAR(4),
  price NUMERIC,
  day DATE,
  PRIMARY KEY (tickersymbol, day)
);
```

```
CREATE TABLE StockSectors (
  tickersymbol CHAR(4),
  sectorID INTEGER,
  PRIMARY KEY (tickersymbol, sectorID)
);
```

Call the *best week* for a stock (identified uniquely by its ticker symbol), the *contiguous* 7 day period during which the stock's average price (over that 7 day period) is at its highest. The *best price* for a stock is its average price over that 7 day period.

Write a SQL query that computes the sum of the best prices of stocks in each sector, and outputs the `sectorID` of the sector with the highest sum.

Answer:

At the heart of this task was a window query (to compute the moving 7-day average). The simplest way to do this was to construct a window and use it using the SQL `OVER` operator.

```
CREATE VIEW avgWeekPrice AS
  SELECT tickersymbol, AVG(price) OVER W
  FROM   Stock
  WINDOW W AS ( PARTITION BY tickersymbol, ORDER BY day
                RANGE BETWEEN CURRENT ROW AND 6 FOLLOWING );
```

For this class of query, the window operator can be simulated. Many strategies exist for doing so, but one of the simplest is to use a 2-way self-join. In the view definition below, S1 establishes one group for each window, while S2 is used to fill in the data for each group.

```
CREATE VIEW avgWeekPrice AS
  SELECT tickersymbol, AVG(groupprice)
  FROM (
    SELECT tickersymbol, S1.day AS group, S2.price AS groupprice
    FROM   Stock S1, Stock S2  WHERE  S1.day <= S2.day AND S1.day+7 > S2.day
  ) W
  GROUP BY tickersymbol, group
```

With the windowed average in hand, we're looking for the maximal window for each tickersymbol, the sum of each maximum for each sector, and then the top-1.

```
SELECT sectorID
FROM ( SELECT sectorID, SUM(bestPrice) as sumPrice
       FROM   StockSector NATURAL JOIN
         ( SELECT tickerSymbol, MAX(bestPrice) AS bestPrice FROM avgWeekPrice; ) bestPrices
) pricesPerSector
ORDER BY sumPrice LIMIT 1;
```

Grading: Use of a windowed aggregate, or successfully simulating one (10 points); A correct outer query (20 points; Partial Credit awarded).

Part C. Transactions

(60 points; 10 each)

Consider the following transaction schedule with numbered operations. C means Commit

T1:		2: R(B)	3: W(A)					8: C
T2:					5: W(A)		7: C	
T3:	1: W(B)			4: W(A)		6: C		

For each of the following concurrency control schemes, describe the final state of the above sequence of operations. Include each of the following:

- Which transactions commit, and which abort.
- For transactions that abort, indicate which operation they abort on.
- In the final state of A and B, which transaction's writes are visible.
- Any other requested scheme-specific metadata

1. 2-Phase Locking with Deadlock Detection (Provide the final dependency graph)

   Answer: No Deadlocks appear in this schedule. T1 blocks on B after operation 2. T2 blocks on A after operation 5. After operation 6, T1 and T2 are unblocked. Depending on how you choose to handle lock reacquisition, either T1 completes fully (T2 blocks on A again) or T2 acquires the lock (T1 blocks on A at op 3).

   All transactions complete fully. The final write to A is either T1 or T2. The final write to B is T1

   Dependencies: T1 depends on T3, T2 depends on T3, Optionally, either T1 depends on T2 OR T2 depends on T1.

2. 2-Phase Locking with Wait-Die

   Answer: If you followed the priority order given during the exam ($T1 > T2 > T3$), then the sequence of operations is as in (1) above. All blocking lock acquisitions are on locks held by lower priority transactions, so no transactions are prematurely aborted.

   Several people declared an inverted priority order that caused both T1 and T2 to abort. This answer was accepted so long as the priority order was clearly specified.

3. 2-Phase Locking with Wait-Wound

   Answer: If you followed the priority order given during the exam (T1 ¿ T2 ¿ T3), then after operation 2, T1's read forces (the lower priority) T3 to abort. T2 subsequently blocks on A (without aborting it, since T2's priority is lower).

   The final write to A is from T2. No committed operation has written to B.

   In the inverted priority order, T1 and T2 block on T3. Depending on how the transaction operations are scheduled after T3 unlocks A and B, T1 either blocks on its read for A first, or is aborted by T2. The final write to A comes from T1 or T2, respectively.

4. Timestamp-Concurrency (Provide the final read- and write-timestamps for A and B)

   Answer: T1 reads a value written by a later timestamp, and aborts after operation 2 (credit was also given if multiversion concurrency control was explicitly referenced as a reason for T1 not aborting). T2's write to A is silently discarded (as per the Thomas Write Rule). The final writes to both A and B come from T3. The WTS for each is correspondingly 3. If T1 did not abort, A has an RTS of 1. All other RTSes are unchanged from their initial values.

5. Optimistic Concurrency Control with Conflict Serializability (Assume the ARIES recovery algorithm is being used. Provide the final state of the write log)

   Answer: T3 commits first, precluding conflicts. T2 commits second, overwriting T3. T1 commits last, but is not conflict-serializable, so it aborts on operation 8.

   The final write to B is T3's, and the final write to A is T2's.

   Log: Commit(T3), Update(T3, B), Update(T3, A), End(T3), Commit(T2), Update(T2, A), End(T2), Abort(T1), End(T1)

6. Optimistic Concurrency Control with View Serializability (Assume the ARIES recovery algorithm is being used. Provide the final state of the write log)

   Answer: To effectively use view-serializability, it is necessary to discard some write operations (this is analogous to the Thomas Write Rule in timestamp concurrency). View serializability is valid only if T2's subsequent write of A to 'overwrites' T1's write (effectively giving T1 a timestamp between that of T3 and T2).

   T3 commits first, precluding conflicts. T2 commits second, overwriting T3. T1 commits last, and is allowed to drop its write to A.

   The final write to B is T3's, and the final write to A is T2's.

   Log: Commit(T3), Update(T3, B), Update(T3, A), End(T3), Commit(T2), Update(T2, A), End(T2), Abort(T1), End(T1)

Part D. Data Warehousing

(30 points)

Consider the relation R(A, B, C, V) with dimension attributes A, B, and C, and Value attribute V. R contains tuples (1, 1, 1, 9.1), (2, 1, 1, 8.2), (3, 2, 1, 8.3), (4, 3, 0, 0.9), (5, 3, 0, 8.1), and (6, 3, 0, 6.9)

1. (5 points) Show an example of a *horizontal partitioning* of this data (i.e., a row-store).

2. (10 points) Show an example of a *vertical partitioning* of this data (i.e., a column-store).

3. (5 points) Constructing the data-cube for R requires the union of how many group-by queries?

4. (10 points) Assume we have functional dependencies: $A \rightarrow B$, $B \rightarrow C$ (e.g., A, B, and C represent City, Country, and Continent, respectively). Which group-by queries can we eliminate from the data-cube without losing information (the answer to this question is often referred to as the ROLLUP of A, B, C).

Part E. Potpouri

(35 points)

1. (12 points; 4 each) Label each of the following statements, true/false.
   (a) A Flajolet-Martin Sketch can precisely compute the number of distinct elements in a set.
   (b) The size of a Flajolet-Martin Sketch should be scaled with the logarithm of the number of distinct elements to keep it from overflowing.
   (c) A Bloom Filter never produces false positives

2. (6 points) Consider a relation R with five attributes $ABCDE$. Assume that R is decomposed into two smaller relations ABC and CDE. Let S be the relation $ABC \bowtie CDE$. Assume that the decomposition is lossless, but not dependency preserving. Indicate which of the following statements you can infer to be *always* true: (1) $R = S$, (2) $R \subset S$, (3) $R \supset S$, (4) $R \subseteq S$, (5) $R \supseteq S$

3. (6 points) Repeat the above question when the decomposition is dependency preserving, but not lossless.

4. (6 points) Using no more than 3 sentences, define the terms scale-up and speed-up in the context of a parallel database system.

5. (5 points) *Nearly* all of the classroom lectures throughout the term consisted of Oliver presenting in using a slide deck and/or the blackboard. Two lectures broke this pattern. State how for at least one of the two lectures.

(Scratch Page 1/3)

(Scratch Page 2/3)

(Scratch Page 3/3)