CS 317: DBIS: End Semester Exam, 9 Nov 2016

Time: 5.30 PM – 8.30 PM Marks: 75

Instructions: **You must answer all parts of a question together.**
In case of any ambiguity, make any required assumptions, but state them clearly.

1. Short answers

   (a) Given relations $r(A, B, C)$ and $s(B, D, E)$, where $r.A$ and $s.B$ are primary keys, what non-trivial functional dependencies can you infer on the result of $\sigma_{E=5}(r \bowtie s)$.
   (NOTE: merge functional dependencies with the same LHS into one, and don't include dependencies that can be trivially derived from other ones by adding extra attributes on the LHS.)

   Answer: $\phi \rightarrow E, A \rightarrow BCD, B \rightarrow D$.
   OK to state $A \rightarrow BCDE, B \rightarrow DE$.
   1 mark each. ...3

   (b) Prove the transitivity rule of functional dependencies from first principles, i.e. $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ implies $\alpha \rightarrow \gamma$. You can assume for simplicity that α, β and γ do not have any attributes in common (the rule holds regardless, but this is to make the proof a little bit easier). ...4

   Answer: Straightforward, most people got it right

   (c) Suppose you have an entity Hour, with primary key (start-time, end-time) and no other attributes, and an entity TimeSlot, with primary key timeslotID and no other attribute.
   (i) Draw an ER diagram to show the relationship between Hour and TimeSlot, along with a constraint to ensure each TimeSlot is related to at least one Hour, and possibly many hours. (Added later: state any assumptions you make.) ...1
   (ii) Show the tables generated from the above ER diagram, along with primary/foreign key constraints. ...2

   Answer: Hour(start-time, end-time), TimeSlot(timeslotID), timeslothours(timeslotID, start-time, end-time). (Note: any name can be used instead of timeslothours; for the other two relations also, other names can be accepted.)
   PKs: all attributes are included for all relations; if someone omits end-time, its OK, give full marks.
   FKs: timeslothours.timeslotID references TimeSlot.timeslotID.
   Marks: 1 mark for relations; 1/2 mark if timeslothours is not present.
   1 mark for PK/FK. Full marks can be given if 1 constraint is missing, 1/2 mark if at least 2 are given.
   (iii) Can you translate the ~~above mentioned constraint~~ constraint mentioned in part (i) into primary/foreign key constraints on the generated tables? If so, show how to do it, otherwise explain why not. ...2

   Answer: No, it cannot be translated, since there is no way to specify a foreign-key from

timeslot.timeslotID to timeslothours.timeslotID, since the latter is not a unique field. (Note: any explanation approximately like the above can be accepted).

0 marks for answer that says Yes, 1 mark for answer without acceptable explanation.

(d) Consider the division operator, defined as follows on relations $r(A, B)$ and $s(B)$:

$r \div s$ is the largest subset $t \subseteq \Pi_A(r)$ such that $t \times s \subseteq r$.

For example, given $r = \{(1, a), (1, b), (1, c), (2, a), (2, c)\}$ and $s = \{(a), (b)\}$, $r \div s = \{(1)\}$. Write an SQL query to implement $r \div s$ for the above schema, using aggregation and join. (Hint: use two aggregate queries on r and s ~~respectively~~ in a WITH clause, followed by a join.) You can assume that r and s don't have any duplicate tuples. ...5

Answer:

```
WITH
  scount(num) AS SELECT count(*) from s,
  rmatch AS SELECT * FROM r where r.B in (SELECT * from s)
  rcount AS SELECT A, count(B) AS numB FROM rmatch GROUP BY A
SELECT A
FROM  rcount, scount
WHERE  numB = num
```

VARIANTS: 1. COUNT DISTINCT can be used instead of COUNT

2. rmatch AS SELECT * FROM r where EXISTS (SELECT * from s WHERE r.B = s.B)

3. Final query can be included with previous one using a having clause as follows

```
SELECT A
FROM rmatch
GROUP BY A
HAVING count(B) = (SELECT num FROM scount)
```

Partial marks of 3 for correct query without using aggregation.

2. Query processing

(a) When using sorting to compute a group-by with aggregate sum, what optimization can be done during the run generation phase? Explain very briefly. ...2

Answer: While sorting during run generation, adjacent tuples with same group can be combined into 1 tuple by adding up the aggregated attribute. (Student may also mention this optimization during the merging phase, but not required by the question)

(b) Consider PostgreSQL's bitmap index scan technique. Would the technique be useful if the entire database along with indices is memory resident? Explain your answer briefly. ...3

Answer: If data is memory resident, you may as well access it directly via the index, since access cost is very low. So the bitmap scan technique is not of any use in this case.

(c) Consider the division operator that we saw earlier in this paper.

Describe how to compute $r \div s$ by extending any one existing join algorithm. The technique should work even if r is a very large relation. (NOTE: you should NOT use the aggregate

technique from the earlier question.) (Added later: You can assume that s fits in memory.) ...5

Answer: Sort r on $r.A, r.B$ and perform a merge with s. For each $r.A$ value, the merge must ensure every value in $s.B$ is present in $r.B$; non-matching tuples in r can be skipped, but non-matchig tuples in s mean the A value is rejected. If all $s.B$ values are matched for a particular A value, the A value is output. The scan on s is restarted when we get a new value for $r.A$.

Other answers are possible. Sort on $r.A$, and counting the number of $r.B$ after verifying existence using an in-memory hash index on $s.B$ is also fine, since we have assumed no duplicates (in earlier question even though not explicit in this question).

Students may give alternative relational algebra expressions based on cross product/set difference. If they express these in the form of a join, partial marks can be given on a case-by-case basis.

3. Indexing

(a) (i) Under what condition does it make sense to construct an bitmap index (not the PostgreSQL bitmap index scan!) on attributes B and C of relation $r(A, B, C)$? ...2

(ii) And if you have bitmap indices on both B and C, how would you use the indices to answer a query $\sigma_{B=3 \wedge (C=2 \vee C=5)}(r)$. ...2

Answer: (i) It is worth constructing these indices if the number of distinct values of B and C are small, so there are a limited number of bitmaps.

(ii) You would first compute an bitwise OR of the bitmaps for $C = 2$ and $C = 5$, and then compute a bitwise AND of the result with the bitmap for $B = 3$.

(b) Suppose you have relation $r(A, B, C)$, which is stored in a B^+-tree file organization on primary key A. Suppose you now want to create a secondary index on B.

(i) How should this index be created, to avoid very high cost for index update when nodes of the B^+-tree file organization are split/merged. ...2

(ii) And how should it be used to answer queries of the form $\sigma_{B=v}(r)$, where v is any given value. ...2

Answer: (i) The secondary index should be on key B but store A values instead of record IDs, so the secondary index does not need to be updated when the nodes in the file organization are split or merged.

(ii) To answer such queries, first the secondary index is loooked up to find all A values matching $B = v$, and then those A values are used to look up the primary key index to find the required tuples.

4. Query optimization

(a) Consider the following potential equivalence rules.

${}_A\gamma_{count(B)}(r \cup s) \stackrel{?}{\equiv} ({}_A\gamma_{count(B)}(r)) \cup ({}_A\gamma_{count(B)}(s))$.

Give example data to show that the rule is incorrect with the set version of union. You can assume for simplicity that the schemas are $r(A, B)$ and $s(A, B)$. ...3

Answer: Many datasets are possible. But the key to look for is that some tuple is common in r and s.

(b) Given relations $r(A, B, C)$, $s(B, D)$ and $t(D, E)$, a query ${}_{r.A}\gamma_{sum(t.E)}(r \bowtie s \bowtie t)$, what are the interesting sort orders for relation r, and for $r \bowtie s$. In each case explain briefly why the order is interesting. ...4

Answer: $r.A$, $r.B$; and $r.A$ and $s.D$

(c) How many left deep join trees are there for a join of n relations? Explain your answer very briefly. ...2

Answer: There are $n!$ left-deep join trees.

(d) Suppose each relation in a join of n relations has a single selection condition on it; the usual heuristic is to push selection conditions down to the relation.

Suppose, instead, that a selection condition can be applied along with any join condition above the relation, as an alternative to being pushed down to the relation. Then, how many extra equivalent trees are possible ~~corresponding to~~ with a given left-deep join order. (Added later: Selection nodes are not to be added, instead selection conditions are to be included with join conditions.) You can give an approximate formula, instead of an exact one. ...4

(e) Given a particular left-deep join order, the selection on the bottom 2 relations each have $n-1$ alternative locations, while the selections on the subsequent relations have $n-1$, ... 2 alternatives. This gives us $(n-1)! \times (n-1)$, or approx $n!$ for a given left deep order. (Students may have mentioned a total of $(n!)^2$ across all possible join orders, which is correct, but it was not asked for in this question.

5. View maintenance.

(a) Given a materialized view v defined as ${}_{A}\gamma_{sum(B)}(r)$, explain how to update it on inserts to r. A textual description will suffice, no need for pseudocode. ...2

Answer: For each new tuple t, first check if there is already a tuple in v with the A value being $t.A$. If yes, add the B value of the new tuple to the sum(B) value of the v tuple. If not add a new tuple $(t.A, t.B)$ to v

Note: wording can be different as long as answer is correct.

(b) Given a materialized view $v = r \rightrightarrows s$, explain how to maintain it on (a) inserts to s, and (b) deletes to s. A textual description will suffice, no need for pseudocode. ...4

Answer: On insert i_s to s, first check what r tuples match the s tuple. If these r tuple had not earlier matched any s tuple, they will be present padded with nulls. Delete such tuples from v and add the matching r, i_s tuples to v.

(Alternative: check if the join attributes of an inserted tuple are already present in other tuples. If yes, there will be no need for deletion, if no, deletion as above will be required.)

On delete, remove $r \bowtie d_s$ tuples from v. For each deleted tuple in d_s if there is another tuple still in s with the same value for join attributes, nothing more needs to be done. Otherwise, add the matching r tuples back, padded with nulls.

Note: wording can be different as long as answer is correct.

Note: 2 marks for each part above.

6. (Schedules) Schedules below should use only read/write operations such as $R(X)$ or $W(X)$.

(a) Give a simple example of a non-serial schedule with two transactions $T1$ and $T2$ that is equivalent to two different serial schedules. ...2

Answer: Many schedules are possible. Make sure that there are no conflicts between $T1$ and $T2$, so both $T1, T2$ and $T2, T1$ are equivalent.

(b) Give an example of a schedule using only 1 data item Q, and two transactions $T1$ and $T2$, such that $T1$ and $T2$ are deadlocked. ...2

Answer: T1: R(Q), T2: R(Q), T1: W(Q) - blocked, T2: W(Q) - blocked and deadlocked
Will usually be written in tabular form but text form above is also ok.

(c) Suppose each of the following transactions execute the specified operations:
$T1: R(X), W(X)$ $T2: R(X), R(Y), W(Y)$ $T3: R(X), R(Y)$.
(i) Is it possible to have any non-serializable schedule involving only $T1$ and $T2$? Explain your answer briefly. ...2

(ii) Give an example schedule for T1, T2 and T3 that is non-serializable (assume no locking is done). (Added later: Explain why it is not serializable.) ...4

Answer: (i) No, there is only one item on which $T1$ and $T2$ conflict. So in any schedule, either $T1$ precedes $T2$ or vice versa, but no cyclic dependency is possible.

(ii) example schedule is below.

The cyclic dependency order is $T2 -> T1 -> T3 -> T2$

3 marks for the schedule, 1 mark for showing cyclic dependency

```
 T1        T2         T3
          R(X)
R(X)
W(X)
                     R(X)
                     R(Y)
          R(Y)
          W(Y)
```

7. (Multigranularity locking). Suppose the lock hierarchy for a database consists of database, relations, and tuples. The goal of multigranularity locking is to allow maximum concurrency, while avoiding acquisition of too many locks.

(a) Suppose a transaction needs to read a lot of tuples from a relation r, what locks should it acquire? ...2
Answer: IS on database, S on r. For each, 1/2 mark if lock is too strong.

(b) Now suppose the transaction wants to update a few of the tuples in r after reading a lot of tuples. What locks should it acquire? ...2

Answer: IX on database, SIX on r, X on required tuples. If any missing/wrong, -1/2 mark; If unnecessarily strong locks on database, -1/2 mark.

(c) If at run-time transaction finds that it needs to actually update a very large number of tuples (after acquiring locks assuming only a few tuples would be updated). What problems would this cause to the lock table, and what could the database do to avoid the problem? ...1+2

Answer: The lock table in memory would get full due to a large number of locks. (1/2 mark for just saying there will be too many locks.)

To avoid this problem, upgrade lock from SIX to X on the relation.

8. Recovery

(a) Give a small example of log records for a transaction, showing why redo should be performed in forward order of the log records; i.e., with your example, performing redo using log records in backwards order should give a wrong final state. ...2

Answer: Many answers are possible, the key to look for is two updates on the same data item in the same transaction, which if redone in the backwards order will result in a wrong final value

(b) Similar to the above, give a small example of log records for a transaction, to show why undo should be done in backwards order. ...2

Answer: Many answers are possible, the key to look for is two updates on the same data item in the same transaction, which if undone in the forwards order will result in a wrong final value

Total Marks = 75