CS145 Final Exam Solutions

Index

Problem 1: (d)

The fact that *R* is many-one and onto (i.e., each *A* is associated with exactly one *B* does not constrain *B*, except that it must have at least one entity. All the *A*'s could map to 1 *B* or 100 different *B*'s, or anything in between, and there could be any number of *B*'s that are not associated with any *A*.

Problem 2: (b)

The schema for *R* is (a,b,c), including the full key for *B*.

Problem 3: (a)

Many many people got this wrong. The problem is that people confuse the idea of where the key comes from in a weak entity set with what are the entities themselves. For example, if *B* were ``crews'' as in the text, and *A* was movies, while *R* indicated which crew worked on which movie, a movie would be related to a crew, i.e., a group of people. The fact that we couldn't uniquely refer to that crew without knowing the studio (entity set *C*) they worked for is irrelevant; *R* still connects movies to crews and not to studios.

Problem 4: (c)

The tuples are (1,2,5), (1,2,6), (3,4,NULL), and (NULL,7,8).

Problem 5: (d)

The fourth step (the revoke) effectively cancels the second and third steps. However, the first step gave *B* exactly the privileges described in choice (d), so that correctly describes the situation after step 4.

Problem 6: (d)

You get 2 tuples if *T1* is either completely before or completely after *T2*. The interleaving 2a, 1a, 1b, 2b causes 1b to return only one tuple. The interleaving 1a, 2a, 2b, 1b causes 1b to return 3 tuples: (3,4) because it is always there, (5,6) because it was inserted already, and (1,2), because even though it is no longer there, 1a saw it so 1b must see it. Note that *T1* has no effect on what *T2* sees, so the fact that *T2* is serializable doesn't restrict the order of steps in any way.

Added later: A number of people argued with me that it is impossible for T1 to see one tuple, because the deletion of (1,2) is not committed until after (5,6) is inserted. I sort of sympathize, but the problem is that the standard talks about tuples, as if they were the elementary units of information. Thus, when step 1a asks for committed

tuples, it does not *have* to see (1,2). There is no notion of a ``committed absence of a tuple.''

In practice, DBMS's use disk blocks as the elementary unit of information, e.g., for locking. Here, the reasoning that ``the absence of (1,2) is uncommitted'' makes perfectly good sense. That is, after step 2a, the block containing tuple (1,2) has been updated --- rather than deleted --- and its new value is uncommitted. Therefore, at that time T1 could not see that block and *must* see tuple (1,2), probably on another version of the same disk block (unless T1 gets deferred --- a likely possibility). As a result, answer (b) was also accepted for credit.

Problem 7: (d)

Actually, the instructor and most of the class had answer (b), that the update (I) succeeds in getting tuple (2) out of *R*, but the deletion doesn't.

The deletion (II) will surely be rejected, since it must cause at least one violation of the foreign-key constraint. Thus, it will make no change to either *R* or *S*, assuring that (2) remains in *R*.

However, the following reasoning was shown to me later by one of the students. Since *S.b* is referenced by a foreign key, it must be the primary key of *S*. Therefore, whenever *S* has more than one tuple, the update will violate this unseen primary-key declaration and will be rejected. Thus, in these cases, it will fail to rid *R* of (2).

Problem 8: (c)

What can I say? The subquery produces those departments with an average salary of at least $50K, and the assertion checks that the Toy Dept. is one of these departments.

Problem 9: (a)

View *V(d,c)* consists of the tuples (2,3), (12,2), and (12,1) currently, although the view isn't materialized. Therefore, *W(d,e)* has the tuples (2,3) and (12,3). The query itself groups both tuples, yielding only (7,3).

Problem 10: (c)

There is a ``standard'' trick for getting MAX or MIN into (core) relational algebra or Datalog, where you take the product of a relation with itself, and then do a selection that requires one copy of an attribute like `salary` to be strictly less than the second copy. A projection onto the first copy gives us those that are not maximum, and a set-difference lets us find those that *are* maximum.

Problem 11: (d)

This was a very hard problem for many people, especially those who skipped the Tuesday review session and didn't see me do a similar problem involving inference of MVD's. The key is to verify that A->B *does* follow from A->->B and D->B. To prove A->B, start by assuming there are two tuples in *R* that agree in *A*. We may refer to these tuples as (a,b1,c1,d1) and (a,b2,c2,d2), since we know nothing about components other than *A*.

Because A->->B is given, we know that (a,b2,c1,d1) is also in *R*. Now, we can apply D->B to (a,b1,c1,d1) and (a,b2,c1,d1) to infer that b1=b2. That says that if any two tuples agree in *A*, they must also agree in *B*; i.e., A->B.

Now, C->A and A->B tell us C->B. Also, C->A and C->B tell us C->->AB, and the latter MVD tells us C->->D. Thus, all of (a), (b), and (c) are true, and the answer must be (d).

Problem 12: (b)

The only thing the constraints will allow us to do is insert (4) into *R* or (0) into *S*. Choose the former for the first step. Then, we can add (3) to *S*, then (6) to *R*, (5) to *S*, and (10) to *R*, taking a total of 5 steps. Technically, we need to create a table, using a ``dynamic programming'' algorithm, in which we record for each tuple in *R* or *S*, the least number of steps needed to put it there, but this exploration doesn't go too far before you discover how to add (10) to *R*.

Problem 13: (a)

Choice (a) replaces the first (and only, in this case) question-mark by the value 5. You need to do this step first, in order to have the update make sense when you execute it. Choice (b) makes no sense because there isn't a second question-mark, (c) is nonsense, and (d) is inappropriate because you aren't trying to execute a *query*, but rather an update.

Problem 14: (a)

You need to do all of (b), (c), and (d), but observer methods only read values and do not create or insert them.

Problem 15: (d)

Dots are appropriate here; -> is used only to follow a `REF`.

Problem 16: (c)

If neither *R.a* nor *R.b* are `NULL`, then the expression is a tautology of 2-valued logic and must be true in either 2-valued or 3-valued logic. However, if either or both values are `NULL`, then the 3-valued truth value is at least unknown, and therefore cannot be false. There are, however, values that make this expression unknown, e.g., *R.a* = `NULL` and *R.b* = -10.

Problem 17: (b)

The update replaces (1,2) by (1,3), and then the trigger applies and insert (1,4). Since this insert doesn't wake up the update-trigger, that's all that happens.

Problem 18: (b)

A has to be in any key because it is not on the right of any FD. If we add *E*, the second FD immediately gives us all attributes, so *AE* is a key. To use the first FD, we must add *BC*, and then the first FD immediately gives us all attributes, so *ABC* is another key. However, there are no other keys, since any other key would have to enable us to use at least one of the FD's, and we already have considered the minimum sets needed to get one of their left sides.

Shame on people who are still confusing ``key'' with ``superkey.''

Problem 19: (a)

E->D is a 3NF violation. That is, *E* is not a superkey, and *D* is not prime (in some key).

Problem 20: (d)

The only way a set could not be closed is if it includes the left side of one of the FD's but not the right. The sets that include *AB* but not *C* are *AB* and *ABD*. The sets that include *BC* but not *D* are *BC* and *ABC*. That's it; all 12 of the other subsets of *ABCD* must be closed.

Problem 21: (c)

In this case, with `Answer(x,y)` as the head in each case, a rule is safe only if both *x* and *y* appear in the body in nonnegated, relational subgoals. Only (c) qualifies.

Problem 22: (d)

In (d), the selection equates the attributes of *P*, giving us exactly those tuples that match the subgoal `P(x,x)`. The projection and renaming gives us those values of *x*, but renamed attribute *C*. Thus, the natural join gives us those tuples in *Q* whose first compenents appear ``doubled'' (i.e., a single value, twice) in *P*.

Choice (a) is incorrect; it never forces the tuple from *P* to be ``doubled.'' Choice (b) equates the first component of *P* to the second component of *Q*. Choice (c) fails to equate the components of *P* to the first component of *Q*.

Problem 23: (c)

The join equates the second component of *P* to the first component of *Q*. If we think of *P* as P(x,y,z), as in each of the answer choices, then *Q* should be Q(y,w) to reflect the join. The selection then says that *x<y*, and the projection is onto *x* and *w*. Choice (c) fits all these requirements.

Problem 24: (b)

Let us represent a model by *(R,N)*, where *R* is the set for which Reach is true and *N* is the set for which NoReach is true. Reach is in stratum 0, and NoReach is in stratum 1. We compute the former using the first two rules: *R = {a,c}*. Then, using this value of *R* in the third rule, we have *N = {d}*.

Problem 25: (b)

Any other model has to have *a* and *c* in *R*, because the EDB and the first two rules force these IDB facts to be true. Thus, we need to consider the following four cases: *R = {a,c}, {a,b,c}, {a,c,d}, {a,b,c,d}*.

In the first two cases, the third rule forces *d* to be in *N*. Other nodes could also be in *N*, but all these models are not minimal; they contain *({a,c}, {d})*, which is the stratified model.

The third case gives us the minimal model *({a,c,d}, {})*. In the fourth case, we get *({a,b,c,d}, {})*, and models with some nodes in *N*, but none of these are minimal; they all contain *({a,c,d}, {})*.

Problem 26: (c)

Strange but true: the RETURN statement in PSM sets the return value but doesn't actually return.

Problem 27: (a)

The rule for what attributes can appear unaggregated in a HAVING clause (outside any subqueries) is the same as for a SELECT clause: only those attributes that appear in the GROUP BY list. Attribute *d* is not one of those.

Problem 28: (b)

As sets, both produce the intersection of *R* and *S*. However, SQL is a bag language. Suppose a tuple *t* appears *m* times in *R* and *n* times in *S*. Then *T* appears *min(m,n)* times in the answer to Q1. However, for the join, we pair all tuples of *R* with all tuples of *S* that agree in the common attributes (i.e., all attributes in this case), and we produce of copy for each successful pairing. Thus, Q2 produces *mn* copies of *t*. It is easy to verify that as long as *m* and *n* are nonnegative integers, *min(m,n) <= mn*.

Problem 29: (a)

Each produces exactly the set of tuples *(a,c,d)*, where *c* is the maximum *b* paired with *a* in *R*, and *d* is the minimum such *b*.

Problem 30: (c)

If we think of *Arc* as arcs in a directed graph, then Q1 produces all *(x,y)* such that there is a path from node *x* to node *y* of length 1 or more. Q2 produces all and only the odd-length paths, as can be proved by an easy induction on the number of times the recursive rule is applied. Thus, surely Q2 is contained in Q1.

To check that these queries are not equal, consider a graph with only the arcs 1->2 and 2->3. Q1 produces (1,3), while Q2 doesn't.

Problem 31: (d)

Q1 produces the names of the A's that are connected only to B's that are connected to the C-object ``Joe.'' Q2 looks at all the B's that are connected to C-object ``Joe'' and produces the names of their associated A-object.

Q2 will produce an A-object that is connected to some B's that are connected to ``Joe'' and to at least one B that is not connected to ``Joe.'' However, Q1 will not produce such an A. Thus, Q2 is not contained in Q1.

On the other hand, Q1 will produce an A that is connected to no B's at all, because the FOR ALL is always true when it is applied to an empty collection. However, Q2 will not produce such an A. Thus, neither query is contained in the other.

Problem 32: (b)

Q1 produces the names of those B's that are related to some C and also related to the A named ``Joe.'' Note that since no B is related to more than one C, no B-object has its name produced more than once, but since a B might not be related to any C at all (see p. 141 of the text, or several examples in the notes), a B might not affect the output, even though it is related to the A named ``Joe.''

On the other hand, Q2 produces all the B's that are related to the A named ``Joe,'' regardless of whether those B's are related to any C. Thus, Q2 is properly contained in Q1.

Problem 33: (c)

Q1 produces the *a*-components of those tuples that are in *R* but not *S*. Q2 produces those *a*'s that are the first component of some tuple in *R*, but not of any tuple in *S*. Surely, if *a* is a first component of a tuple *(a,b)* of *R* but not the first component of any tuple of *S* at all, then *(a,b)* is not in *S*, and therefore *a* is in the result of Q1. It follows that Q2 is contained in Q1.

To check that the queries are not equal, let *R = {(1,2)}* and *S = {(1,3)}*. Then Q1 produces (1), but Q2 does not.

Problem 34: (a)

Both queries produce those tuples of R whose b-components are equal to the d-component of some tuple of S.

Problem 35: (b)

Q1 produces those a-components from R that are bigger than all b-components, while Q2 produces those a-components from R that are different from at least b-component. Our first instinct is that therefore Q1 is contained in Q2, since if a value is bigger than all elements of a set, then it is ``surely'' different from at least one of those elements.

However, if we remember a similar question from last year's final, we think ``what if the set is empty.'' Then a value can be bigger than all, but not different from any. But in this case, when R is empty, both queries produce nothing. Thus, here our first thought turns out to be correct, and Q1 really is contained in Q2.