CS145 Final Exam Solutions

Index

Problem 1: (a)

Clearly, Q1 produces one of each value of *a* that is associated with at least one *b* value bigger than 10. But so does Q2, because there is only one group for each *a*.

Problem 2: (c)

The set of values produced by Q1 and Q2 are clearly the same. However, Q1 can produce more of something than Q2. For instance, suppose *R* has the tuple (1,2), and *S* has tuples (2,3) and (2,4). Then Q1 produces 1 twice, because there are tuples (1,2,3) and (1,2,4) in the relation that results from the FROM and WHERE clauses. However, Q2 only produces 1 once, because it works from *R* rather than from the join.

Problem 3: (b)

For *a* to be in the result of Q1, there must be exactly one *b* such that (*a*,*b*) is not dangling in *R*, and there must be exactly one *c* such that (*b*,*c*) is in *S*. But if that is so, then *b* is in the result of the subquery of Q2. Thus, Q1 is contained in Q2, but could they be equal? To see that these queries are not equal, consider *R* = {(1,2), (1,3)} and *S* = {(2,4), (3,5)}. Then in Q1, the group of 1 has two members, so 1 is not in the result of Q1. However, 1 is in the result of Q2, in fact twice.

Problem 4: (d)

This was undoubtedly the trickiest problem on the exam, and about 90% chose (c), which is wrong. If the subquery returns a nonempty set, then surely the condition that `b >= ALL` implies `b >= ANY`. However, if the subquery is empty, then you can be bigger than ``all'' yet there does not exist an element than which you are bigger. That's the way logic works; it's the way Oracle works, and it's in the SQL2 definition of the concepts.

Problem 5: (a)

Intuitively, both queries produce every tuple in the natural full outerjoin.

Problem 6: (a)

Both queries produce all the ``R's'' related to Sally. Note that because `name` is a key, there is no worry about duplicates or the lack of them as we saw in Question 2.

Problem 7: (b)

The significant observation is that Q2 does not require the typical objects `rr` and `ss` to be related by the `myRs/mySs` relationship.

Problem 8: (c)

As sets, these two expressions are equal. However, as bags, the number of times an element *x* can appear in the result of Q1 can be larger than the number of times it appears in Q2. Suppose *x* appears *r* times in *R* and *s* times in *S*. Then in Q1, *x* appears *r*+*s*-min(*r*,*s*) times, which is max(*r*,*s*) times. In Q2, *x* appears 0 times in one of the differences and max(*r*,*s*)-min(*r*,*s*) times in the other. Thus, the latter formula is the number of times *x* appears in the union, which is the result of Q2. Thus, unless min(*r*,*s*) = 0, *x* appears more times in Q1 than in Q2.

Problem 9: (d)

The two expressions do not even produce tuples with the same number of components.

Problem 10: (b)

Q2 correctly computes the endpoints of paths in the graph represented by relation `Arc`. However, Q1 only produces the endpoints of the paths of odd length, which might be a proper subset of Q2. It is not hard to prove this characterization of the result of Q1 by induction on the number of times we apply a rule. However, the following example suffices to make the point: let `Arc` = {(1,2), (2,3)}. Then (1,3) is in the result of Q2 but not Q1.

Problem 11: (d)

The result of Q1 is that (10,5) is the one and only tuple with first component 10. In Q2, if there were many tuples of *R* with first component 10, then (10,5) will appear many times. However, it is also possible that *R* had no tuple with first component 10, in which case (10,5) does not appear in Q2.

Problem 12: (b)

The stronger of the conditions represented by the arrows is that for a given *A* entity and *C* entity, there is a unique *B* entity. Since there are only 1000 possible *A*-*C* pairs, there cannot be more than this number of tuples. However, we could also have 1000 triples in the relationship set. Suppose the *A* values are 0-99, and the *C* values are 0-9. Let the associated *B* value be 10**A*+*C*. Then all *B* values from 0 to 999 appear exactly once, so *A* and BB surely determine at most one *C*.

Problem 13: (d)

Remember that when we translate a weak entity set to relations, there is no need for a relationship that supports the weak entity set, such as *R* in our diagram. This relation not only has as attributes a subset of those in the relation for the weak entity set itself, but the meanings of those attributes are the same. Thus, we can safely eliminate the relation for this relationship.

Problem 14: (b)

The relation for *E* loses the tuples that belong to the subclass. The relation for *G* keeps the same tuples as it had, but gets the additional attribute *b*, which it now ``inherits'' from *E*. Thus, there are two changes. I suspect that people who got larger numbers of changes were thinking of the more complicated question of what would happen if we first translated the E/R diagram into ODL and then translated the ODL into relations.

Problem 15: (a)

Given a value of *a*, there is at most one associated *b*, because *a* is a key for *A*. The link *L* tells us there is at most one associated *B*, so *a* functionally determines *c* and *d* as well. for this *B* there is at most one associated *C*, so *a* likewise determines *e* and *f*.

Problem 16: (a)

You can check that *AD->CE* does not follow by computing the closure of *AD* with respect to the given FD's. We get *D* because of the FD *D->E*, but that is as far as we can go; we never get *C*.

Problem 17: (d)

(a) is not right because every FD is an MVD.

(b) is not right because *A->B* follows from the given FD; in fact, the given FD is really a shorthand for this FD and the FD *A->C*. Since an FD is an MVD, we see that *A->->B* holds.

(c) is not right because *A->C* is a given FD, this FD implies MVD *A->->C*, and by the complementation rule, *A->->BD*.

Problem 18: (b)

First, note that *A^+ = ABCDE*. Thus, when we project the FD's onto *ABC*, *A* is certainly a key. However, there cannot be any other keys, because neither *B* nor *C* have anything else in their closures, because they are not on the left sides of any FD's. Thus, *A* is the only key.

Problem 19: (c)

In case (c), we can check that the keys are *ACD* and *ABD*. Both FD's violate BCNF, but all the attributes are prime, so there can be no 3NF violation.

In (a), *D* is the only key, so *B->C* is both a 3NF and BCNF violation.

In (b), *AB* is the only key, so both FD's are 3NF and BCNF violations.

In (d), *AD* is the only key, so *B->C* violates both normal forms.

Problem 20: (d)

The decomposition is lossless because the intersection of the two schemas, that is *B*, functionally determines one of the two schemas, namely *B->BCD*. To check, compute *B^+*, which is *BCD*.

To check dependency preservation, note that *A^+ = ABCD*, thus, when we project dependencies onto *AB*, we get *A->B*. The projection onto *BCD* surely gives us the second and third of the given FD's: *B->D* and *D->BC*. We claim that the three dependencies *A->D*, *B->D*, and *D->BC* are equivalent to the three dependencies that we can preserve in the projection: *A->B*, *B->D*, and *D->BC*. The last two in each set are the same. Also, *A->B* is easily seen to follow from the first three, and *A->D* follows from the last three. If we can preserve an equivalent set of FD's in the projections, then surely we can preserve the given set.

Problem 21: (b)

Remember, SQL2 does not allow you to use `NULL` as if it were a value, either in the `WHERE` or `SELECT` clauses. It also does not have a function `ISNULL`, although `b IS NULL` works. However, (b) does not involve null's at all; it is a condition involving a pattern that happens to have the characters `NULL` as part of it; this pattern will not even match a `NULL` value.

Incidentally, although it is irrelevant, the Oracle system accepts (c), and it objects only to `ISNULL` in (a), allowing us to put `NULL` in the `SELECT` clause.

Problem 22: (c)

First, what does the Datalog program do? If we think of relations *R* and *S* as colored arcs in a graph, say ``red'' arcs and ``silver'' arcs, then *Q* contains all those pairs of nodes that are connected by a path of four arcs, colored red-silver-red-silver. Answer (c) produces all pairs (*a,c*) that are related by a red-sliver path of two arcs.

Problem 23: (d)

Although `name` is a key for the objects, the resulting relation is not in BCNF, and there can be many tuples with the same team name. In fact a team 1 with players 2 and 3 and coaches 4 and 5 should convince you that only all three attributes are a key for the relation.

Problem 24: (b)

The definition of a MVD is that when we fix the left side, e.g., the team, then the right side, e.g., the coaches, become independent of the remaining attributes, e.g., the players, and the latter two (sets of) attributes appear in all combinations in the relation. That is exactly what we said to do to construct the relation's tuples from a team object: pair all players and coaches for that team.

You might imagine that several of the other MVD's must hold as well. However, if you believe that, you are probably making the tacit assumption that players can only play for one team, and coaches can only coach for one team. While possibly true in practice (if you ignore things like Deion Sanders playing for both the Falcons and the Braves at one point in his career), nothing of the sort was stipulated in the problem statement. If you add to the relation mentioned in the solution to Question 23 another team 6 with player 2 and coach 4 (only), then the resulting relation violates all three other MVD's among the choices.

Problem 25: (d)

```
R1(c,d) = R(a,b) /* A renamed R */
S1(a,b) = PI_{a,b}(SIGMA_{bd}(R TIMES R1)) /* Nonminimum b's */
S3(a,b) = R - S1 /* a with its maximum b */
S4(a,c) = R - S2 /* a with its minimum b, renamed c */
Answer = S3 NATURAL JOIN S4
```

Problem 26: (d)

(a) is not enforced; the CHECK only requires the price be nonnegative, which allows 0.

(b) is not enforced. Remember that attribute-based checks cannot enforce a foreign-key constraint, such as (b), because they are not triggered on a deletion. Yet a deletion can cause a foreign-key violation.

(c) is not enforced. The assertion only requires that stores in California have sales of at

least a million.

Problem 27: (b)

This constraint is exactly the foreign-key constraint in `Sales`.

Problem 28: (c)

The foreign-key constraint from B to A doesn't cause problems if we delete from A first, since there is a clause that tells how to handle dangling tuples in B. However, the foreign-key constraint from C to A is a problem, since the default policy is to reject a deletion from A that causes a dangling tuple in C. Thus, in a valid sequence, C must precede A. The foreign-key constraint from D to B is not a problem, but the one from D to A is, again because the default policy will cause a rejection of certain deletions from A. Thus, D must also come before A. Of the three sequences, only I has C or D before A, so II and III are OK; I is not.

Problem 29: (c)

First of all, the middle component of `theView`, which has the constant value 'cs145' in all tuples, is a red herring; it is not used by the query and may as well not be there. What the query uses from the view is the fact that a theta-join of *R* and *S* is created, using the condition `R.b=S.e`) and then the *f* component of the join is tested for being greater than 10. Answer (c) does exactly these steps.

(a) makes no sense, since *R* doesn't even have a `class` attribute. (b) is wrong, because the natural join is a product when the relations have no attributes in common. (d) is wrong because it takes the entire projection of `theView`, with the `f>10` test applying only to *S*.

Problem 30: (a)

First, let's see what the PL/SQL does. The query of the cursor joins *R* with itself on the condition that the two second components be the same, and produces the pair of first components. That is, the cursor gives us all pairs of first components that share a common second component.

Then, the body of the PL/SQL statement empties *S*. Next, it looks at each of the cursor pairs and places into *S* those pairs where the first component is less than the second. Datalog query (a) does exactly the same thing. The two *R* subgoals are joined by requiring the same value *z* in the second components, and the comparison subgoal requires that the first component in the head be less than the second.

(b) doesn't work because the join is on first components. The body of (c) looks right, but we take the wrong components for the head. (d) fails to enforce the inequality.

Problem 31: (b)

The longest chain of negations is *a* to *c* to *d*, so there are three strata. Note that the EDB predicates are not assigned to strata.

Problem 32: (a)

(a) correctly takes a tuple variable `xx` representing a tuple of *R*, gets its *d* attribute, which is an object, and then gets the *a* component/attribute from that object.

(b) and (d) use `THE` to apply to the relation of a subquery, rather than to a nested

table (there is none in this problem). (c) is an incorrect version of (a), where we neglect to create the necessary tuple variable.

Problem 33: (a)

Each time the trigger causes an update, it is triggered again, eventually leaving *A* with the tuple (10). The trigger is triggered by the last update, which changed the tuple from (9) to (10), but the `WHEN` condition is not satisfied, so there is no update, and the triggering stops.