Design Theory Activity - Monday October 21

Problem session -- we'll see how many we can get through

Problem 1 – Easy warm-up

It's a well-known mantra in database circles: "*Every FD is an MVD, but not every MVD is an FD*." Consider a relation R(A,B,C). The statement tells us:

1. If A $\rightarrow$ B then A –>> B
2. If A –>> B it is not necessarily the case that A $\rightarrow$ B

The first property is proven in the lecture video. Prove the second property. You can prove the property by giving just one instance of R such that A –>> B holds but A $\rightarrow$ B does not hold. Try to give the simplest such instance, but do not use trivial dependencies.

Problem 2 – Proof of property that holds (similar to challenge problem)

Prove the *difference rule* for multivalued dependencies. Specifically, consider a relation R, and let AA, BB, and CC be three sets of attributes in R. Prove that if AA –>> BB and AA –>> CC hold for R, then AA –>> (BB – CC) also holds, where – is the standard difference of attribute sets.

- For simplicity you may assume that AA does not intersect BB or CC.
- Do not assume that CC is a subset of BB.
- Don't forget to account for the remaining attributes in R: those that are not in AA, BB, or CC. Call them DD.

Your proof should be based on the formal definition of MVDs, not on other rules. In other words, it should have roughly the following form:

"Suppose AA –>> BB and AA –>> CC hold. To prove AA –>> (BB – CC) , we need to prove that for all tuples t and u in R there exists a tuple v1 in R such that ... [fill in] … From AA –>> BB we know that for all tuples t and u in R there exists a tuple v2 in R such that ... [fill in] … Also, from AA –>> CC we know that for all tuples t and u in R there exists a tuple v3 in R such that ... [fill in] … [Fill in more] We have shown that the required tuple v1 exists, therefore AA –>> (BB – CC) holds."

Problem 3 – Proof of property that doesn't hold

Let AA, BB, and CC be sets of attributes. The transitivity rule for functional dependencies says if nontrivial FDs AA $\rightarrow$ BB and BB $\rightarrow$ CC hold for a relation R, then AA $\rightarrow$ CC holds for R. The transitivity rule for multivalued dependencies is slightly different: It says if nontrivial MVDs AA –>> BB and BB –>> CC hold for a relation R, then AA –>> (CC – BB) holds for R. Motivate the difference by giving a schema and instance for a relation R such that:

- AA –>> BB and BB –>> CC hold
- AA –>> (CC – BB) holds
- AA –>> CC does not hold

Problem 4 – Proof of another type altogether

Consider the Boyce-Codd Normal Form (BCNF) decomposition algorithm as presented in the lecture video. Show that after performing one decomposition step, to determine if the decomposed relations are in BCNF it is necessary to check not only the original functional dependencies (FDs), but also those that follow from the original FDs. Specifically, consider a relation R(A,B,C,D), and give a set F of FDs for R such that:

1) R is not in BCNF.

2) Let R1 and R2 be a decomposition of R according to the algorithm and the FDs in F. The decomposed relations should be such that R1 and R2 are in BCNF according to the FDs in F, but at least one of them is not in BCNF according to the FDs that follow from F. Specify:

- The violating FD from F used to decompose R into R1 and R2
- The schemas of R1 and R2
- The FD implied by F (but not in F) that causes one of R1 or R2 to violate BCNF

Try to find the simplest set of FDs satisfying the conditions.

Problem 5 – Warm-down: Anti-decomposition theory

Suppose a database designer's first task is to design the schema for a company database. Each employee has an ID (unique across employees), a single name, division, location, and salary, and one or more projects. The designer decides to create the following five relations:

EmpName(ID,Name)
EmpLocation(ID,Division)
EmpLocation(ID,Location)
EmpSalary(ID,Salary)
EmpProject(ID,Project)

a) State the completely nontrivial functional dependencies for each relation.

b) Are all five relations in Boyce-Codd Normal Form?

c) Is this a good database design? Why or why not?

d) Can you suggest a theory and/or algorithm for combining relations during the database design process, to complement the methods we learned for decomposing them?