

Design Theory Activity - Monday October 24

Menu of three activities

1. Proof similar to upcoming 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 $AA \twoheadrightarrow BB$ and $AA \twoheadrightarrow CC$ hold for R , then $AA \twoheadrightarrow (BB - CC)$ 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 .
- Feel free to introduce additional names for sets of attributes. For example, you may find it convenient to give names to $(BB - CC)$, $(BB \cap CC)$, and $(CC - BB)$.

Your proof should be based on the formal definition of MVDs, not on other rules. It should have roughly the following form: "Suppose $AA \twoheadrightarrow BB$ and $AA \twoheadrightarrow CC$ hold. Then for all tuples t and u there exists a tuple v such that ... [fill in] ... To prove that $AA \twoheadrightarrow (BB - CC)$ holds, we need to prove that for all tuples t and u there exists a tuple v such that ... [fill in] ... Therefore $AA \twoheadrightarrow (BB - CC)$ holds.

2. Proof dissimilar from any challenge problem

Consider a relation R and a set F of functional dependencies for R . Show that in order to determine whether R is in Boyce-Codd Normal Form, we only have to check whether any of the FDs in F violate BCNF (i.e., don't contain a key on the left-hand side); we don't have to check other FDs that follow from the FDs in F . Formally, prove: If R has a nontrivial FD $AA \rightarrow BB$ that:

- violates BCNF,
- follows from the FDs in F , and
- does not appear in F

then R must also have an FD $CC \rightarrow DD$ that violates BCNF and does appear in F .

(over)

3. 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 four 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?