

This Lecture

- Normalization
- Boyce-Codd Normal Form

Example Relation Figure 1: One possible relation storing Mines course information: Instructor Course_id Section Title Office Email Painter-Wakefield, Christopher CSCI403 A DITABASE MANAGEMENT BB 2801 cpainter@mines.edu Painter-Wakefield, Christopher CSCI262 A DITA STRUCTURES BB 2801 cpainter@mines.edu Painter-Wakefield, Christopher CSCI262 B DATA STRUCTURES BB 2801 cpainter@mines.edu Mehta, Dinesh CSCI406 A ALGORITHMS RR 2801 dmehta@mines.edu Mehta, Dinesh CSCI 561 A THEORY OF COMPUTATION BB 280J dmehts@mines.edu Mehta, Dinesh CSCI 561 A THEORY OF COMPUTATION BB 280J dmehts@mines.edu CSCI 101 A INTRO TO COMPUTER SCIENCE BB 310F khellman@mines.edu Hellman, Keith Hellman, Keith CSCI 101 B INTRO TO COMPUTER SCIENCE BB 310F khellman@mines.edu CSCI 101 C INTRO TO COMPUTER SCIENCE BB 310F khellman@mines.edu Hellman, Keith CSCI 274 A INTRO TO LINUX OS BB 310F khellman@mines.edu

■ Our primary tool for eliminating redundancy and modification anomalies ■ A kind of constraint between two sets of attributes in a relation schema ■ Definition: Given a relation schema R and sets of attributes X and Y, then we say a functional dependency X → Y exists if, whenever tuples t, and t, are two tuples from any relation r(R) such that t₂(X) = t₂(X), it is also true that t₂(Y) = t₂(Y). ■ The lingo: We say X functionally determines Y, or Y is functionally dependent on X.

Functional Dependencies Review 2 ■ In other words: If it is always true that whenever two tuples agree on attributes X, they also agree on Y, then X → Y. ■ Example: If we assert that an instructor is always associated with one office and email, then { instructor } → { office, email } / X is a functional dependency (FD) on the example table in figure 1.

Developed to define "good" design for a database Several forms: First normal form (1NF), Second (2NF), etc. Each normal form describe certain properties of a database E.g., 1NF eliminates multivalued and compound attributes Mostly later normal forms subsume earlier normal forms 1 NF - 3NF are not terribly interesting stepping stones to the forms we care about: Boyce-Codd Normal Form (BCNF) Fourth Normal Form (4NF) Intere exist even stronger normal forms (5NF etc.), but BCNF and 4NF suffice for most purposes.

Boyce-Codd Normal Form

A relation R is in Boyce-Codd Normal Form (BCNF) if for every nontrivial functional dependency $X \to A$ on R, X is a superkey of R.

BCNF Example

Consider our example relation schema in Figure 1:

One of the (non-trivial) functional dependencies we identified was instructor → office

Clearly, instructor is not a superkey of the relation.

Therefore, we say that the FD instructor → office violates BCNF, and the relation schema is not in BCNF.

Moving to BCNF

Our goal is a database in which every relation is in BCNF.

Fortunately, there is a straightforward algorithm for getting there.

- Find a relation schema not in BCNF
- Decompose it into two relation schemas, eliminating one of the BCNF violations

(Details on next page)

Decomposition Algorithm

while some relation schema is not in BCNF:

choose some relation schema $\it R$ not in BCNF choose some FD $X \rightarrow Y$ on R that violates BCNF

(optional) expand Y so that $Y = X^+$ (closure of X)

let Z be all attributes of R not included in X or Y

replace R with two new relations:

R1, containing attributes (X, Y)

R2, containing attributes {X, Z}

Decomposition Notes

- The final step above is accomplished simply by projection onto the attributes in R1 and R2. (Recall that this may result in fewer tuples.)
- After decomposing, you will need to establish which FDs now apply to R1 and R2, as well as determine their superkeys, in order to determine if they are now in BCNF.
- The optional step of expanding Y is recommended, as it tends to result in fewer, larger relation schemas, and may reduce the problem faster e.g., consider decomposing on instructor → office.

Decomposition Walkthrough

Let's use the relation schema in Figure 1 as an example.

For this schema, we listed the following FD's:

■ instructor → office

■ instructor → email

■ {course_id, section} → instructor

■ course id → title

What superkeys do we have?

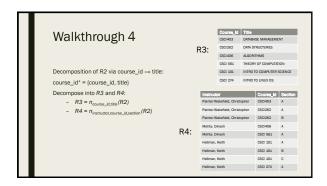
Answer: any superset of our only key, which is {course_id, section}.

Which FD's violate BCNF?

■ Let's pick our first violating FD to work with first: instructor → office ■ Next, expand the RHS as much as possible (we want the closure of instructor): instructor → (instructor, office, email) ■ Now we decompose into two new tables, shown on the next slide: - R1 = π_{estructoc (office, email}(R)) - R2 = π_{estructoc (office, email}(R)) - We can now discard the table from figure 1.



Walkthrough 3 ■ Table R1 is now in BCNF (yay!) - Note this was not guaranteed by the algorithm – we could have had other violating FDs ■ Table R2 has a violating FD, though: course_id → title



■ Done! - Three tables remain: R.1, R.3, R.4 - All non-essential redundancy has been removed - Each table now represents a fundamental entity: ■ R.1 = instructor info ■ R.3 = course info ■ R.4 = section info ■ As a final note: this algorithm is not deterministic – you can different decompositions following different choices of FD to work with.

