CSCI 403 Database Management

12 – Functional Dependencies

CS@Mines

This Lecture

Discuss “goodness” of a database design
- Informal guidelines
- Objective measures

CS@Mines

Informal Guidelines

1. Clear semantics
   - Do your relations make sense as independent units?
   - Do you have a clear separation of concerns?
   - Did you do ER modeling beforehand?
2. Reducing redundancy
   - Data should be stored once and only once (excepting foreign keys)
   - Redundancy leads to modification anomalies
3. Reducing NULLs
4. Disallowing spurious tuple generation

CS@Mines

Example

Figure 1: One possible relation storing Mines course information:

Instructor	Course _id	Section	Title	Office	Email
Painter-Wakefield, Christopher	CSCI403	A	DATABASE MANAGEMENT	BB 280I	cpainter@mines.edu
Painter-Wakefield, Christopher	CSCI262	A	DATA STRUCTURES	BB 280I	cpainter@mines.edu
Painter-Wakefield, Christopher	CSCI262	B	DATA STRUCTURES	BB 280I	cpainter@mines.edu
Mehta, Dinesh	CSCI406	A	ALGORITHMS	BB 280J	dmehta@mines.edu
Mehta, Dinesh	CSCI 561	A	THEORY OF COMPUTATION	BB 280J	dmehta@mines.edu
Hellman, Keith	CSCI 101	A	INTRO TO COMPUTER SCIENCE		khellman@mines.edu
Hellman, Keith	CSCI 101	B	INTRO TO COMPUTER SCIENCE		khellman@mines.edu
Hellman, Keith	CSCI 274	A	INTRO TO LINUX OS		khellman@mines.edu

CS@Mines

Redundancy

- Example has multiple issues of redundancy:
  - Multiple sections, with redundant course id and title information
  - Instructor name and email repeated many times
- Cause:
  - Two (or more) concepts have been combined into one table
    - Instructor
    - Course info
    - Section info
  - These should be (somewhat) independent pieces of data

CS@Mines

Modification Anomalies

- A consequence of bad design
- Goes hand-in-hand with redundancy issues
- Three types:
  - Insertion
  - Update
  - Deletion

CS@Mines

Insertion Anomaly

Insert a new faculty member in example table – no course info yet

- What do we put in for course info?
  - NULL values?
    - Could violate constraints
    - What happens when we want to add a course for this faculty member?
  - Dummy data?

CS@Mines

Deletion Anomaly

Inverse of insertion anomaly:

What happens if we delete the last course taught by an instructor?

Similarly, what happens to a faculty member's courses when they leave/retire?

CS@Mines

Update Anomaly

- When updating redundant data, must remember to update *all* instances
- E.g., suppose you are in an application updating course info for CSCI 403
  - You notice that CPW's office info is wrong (e.g., maybe he moved)
  - You edit the record to correct his office info
  - Now, inconsistent data in different records! Which is correct?

CS@Mines

Spurious Tuple Generation

- Happens when data has been incorrectly factored
  - There is no linking data (foreign keys)
  - The linking data is incomplete
- (Somewhat contrived) example:
  - Table mines_courses (instructor, course_id, section)
  - Table mines_faculty (instructor, course_id, office, email)
  - Joining these tables on instructor and course_id will yield spurious combinations of instructors with sections they do not teach

CS@Mines

Functional Dependencies

- 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 $X \rightarrow Y$ exists if, whenever tuples t_1 and t_2 are two tuples from any relation r(R) such that $t_1[X] = t_2[X]$, it is also true that $t_1[Y] = t_2[Y]$.
- The lingo:

  We say X functionally determines Y, or Y is functionally dependent on X.

CS@Mines

Functional Dependencies 2

- In other words:

  If it is always true that whenever two tuples agree on attributes X, they also agree on Y, then $X \rightarrow Y$.
- Example:

  If we assert that an instructor is always associated with one office and email, then

  { instructor } → { office, email }

  (X = { instructor }, Y = { office, email })

  is a functional dependency (FD) on the example table in figure 1.

CS@Mines

Functional Dependencies 3

Note:
FD's are *properties of the world that we impose on the data*, **not** properties of the data.
That is, finding FD's is a *design activity*.
The result is a constraint on the data that is allowed in our database.

Example:
It may be that we have a particular set of courses data in which each course_id is associated with one instructor. Then, *for that data*, it is true that whenever a tuple agrees on course_id, it also agrees on instructor. However, unless this is required to be true *for any set of data* we can put in our database, we cannot say { course_id } → { instructor}.

CS@Mines

Types of Functional Dependency

- Trivial FD's
  - Trivially, $X \rightarrow X$
  - More generally, if $Y \subseteq X$, then $X \rightarrow Y$
- Non-trivial FD's
  - $X \rightarrow Y$
  - $Y \not\subseteq X$
- Completely non-trivial FDs
  - $X \rightarrow Y$
  - $X \cap Y = \emptyset$ (No overlap between X and Y)

CS@Mines

Non-Trivial FDs

- We are primarily interested in non-trivial and completely non-trivial FD's.
- In our figure 1 example, we might identify the following completely non-trivial FD's:
  - instructor → office
  - instructor → email
  - { course_id, section } → instructor
  - course_id → title
- Can you identify others?

Note the abuse of set notation here. I just find it more readable.

CS@Mines

Functional Dependencies and Superkeys

- FD's can be viewed as a generalization of the notion of a *superkey*
- A superkey is a set of attributes which will contain a unique subset of values for any tuple in a relation.
- Thus, if X is a superkey of R, $X \rightarrow R$.
- Alternately, if $X \rightarrow Y$ and $X \cap Y = R$, then X is a superkey of R.

CS@Mines

Inference Rules

Allow us to infer additional FD's from an existing set of FD's

- Splitting rule:
  If $A \rightarrow \{B_1, B_2\}$ then $A \rightarrow B_1$ and $A \rightarrow B_2$
- Combining rule:
  If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow \{B, C\}$
- Transitive rule:
  If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$

More set notation abuse here. A, B, C, etc. are all sets. {B, C} is the union of sets B and C.

Additional rules can be derived and can be found in your textbook.

CS@Mines

Closures

Definition:

Given some set of functional dependencies F on a relation schema R, and some subset of attributes A, then the set $\{B_i : A \rightarrow B_i\}$ is called the closure of A and is denoted A^+.

Closures are useful in:

- Normalization
- Finding all superkeys of a relation schema

CS@Mines

Computing Closure

Algorithm:

Given set *F* of functional dependencies, and some set of attributes *A*, compute A^+:

Start with S = A. Trivially, A → S.

Repeat until no change:

if there exists an FD X → Y in F such that X ⊂ S,

then let S = S ∪ Y

A^+ = S

Expands S while maintaining the invariant A → S. The step follows from the three inference rules.

CS@Mines

Finding All Superkeys

- In short:
  - Generate the power set of R – all subsets of attributes
  - For each subset, compute the closure
  - If the closure = R, then the subset is a superkey of R
- This algorithm is mostly of academic interest to us, but could be used in automated software to build a normalized database, when the functional dependencies are inputted.

CS@Mines

Next Time

- Normal forms & Boyce-Code normal form
- Decomposition algorithm

CS@Mines