# CS©Mine CS©Mine CS©Mine

# **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:					
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	в	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	В	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**

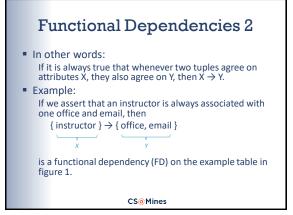
- Happens when data has been incorrectly factored
  - There is no linking data (foreign keys)
  - The linking data is incomplete
- Somewhat contrived) example:
  - Table mines courses (instructor, course id, section)
  - Table mines\_faculty (instructor, course\_id, office, email)
  - Joining these tables 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

- 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 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:

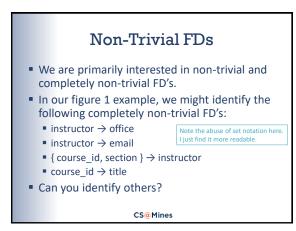
It may be that we have a particular set of courses data in which each course\_id is associated with one instructor. Then, for that data, it is true that whenever a tuple agrees on course\_id, it also agrees on instructor. However, unless this is required to be true for any set of data we can put in our database, we cannot say { course\_id }  $\rightarrow$  { 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 ⊈ X
- Completely non-trivial FDs
  - $X \rightarrow Y$
  - $X \cap Y = \emptyset$  (No overlap between X and Y)

CS@Mines

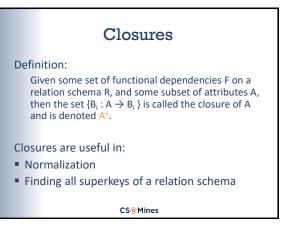


# Functional Dependencies and Superkeys

- 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 RulesAllow us to infer additional FD's from an existing set of<br/>fD's9. Splitting rule:<br/> $[A \rightarrow [B_1, B_2] then A \rightarrow B_1 and A \rightarrow B_2]$ 0. Combining rule:<br/> $[A \rightarrow B and A \rightarrow C then A \rightarrow \{B, C\}$ 1. Transitive rule:<br/> $[A \rightarrow B and B \rightarrow C then A \rightarrow C]$ 1. A b and B $\rightarrow$ C then A $\rightarrow C$ Additional rules can be derived and can be found in your<br/>textbook.CS@Mines



# **Computing Closure** Algorithm: Given set *F* of functional dependencies, and some set of attributes *A*, compute *A*<sup>+</sup>: Start with S = A. Trivially, A $\Rightarrow$ S. Repeat until no change: if there exists an FD X $\Rightarrow$ Y in F such that X $\subset$ S, then let S = S $\cup$ Y A<sup>+</sup> = S Expands while maintaining the furainat A $\Rightarrow$ S. The step follows to the three inference rules.

# 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

