CS 450

Schema Refinement & Normalization Theory

Functional Dependencies

What's the Problem

- Consider relation obtained (call it SNLRHW)

  Hourly_Emps(<u>*ssn,*</u> *name, lot, rating, hrly_wage*, *hrs_worked*)
- What if we ***know*** rating determines hrly_wage?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Redundancy

- When part of data can be derived from other parts, we say *redundancy* exists.
  - Example: the hrly_wage of Smiley can be derived from the hrly_wage of Attishoo because they have the same rating and we know rating determines hrly_wage.
- Redundancy exists because of the existence of *integrity constraints* ***(e.g., FD: R→ W).***

What's the problem, again

- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

What do we do?

- Since constraints, in particular *functional dependencies*, cause problems, we need to study them, and understand when and how they cause redundancy.
- When redundancy exists, refinement is needed.
  - Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Decomposition

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

=

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

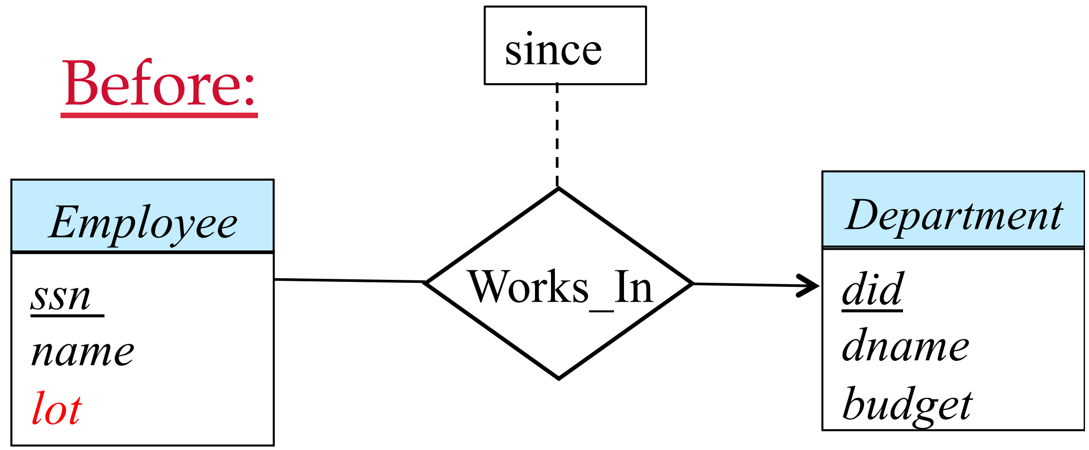
$\bowtie$

R	W
8	10
5	7

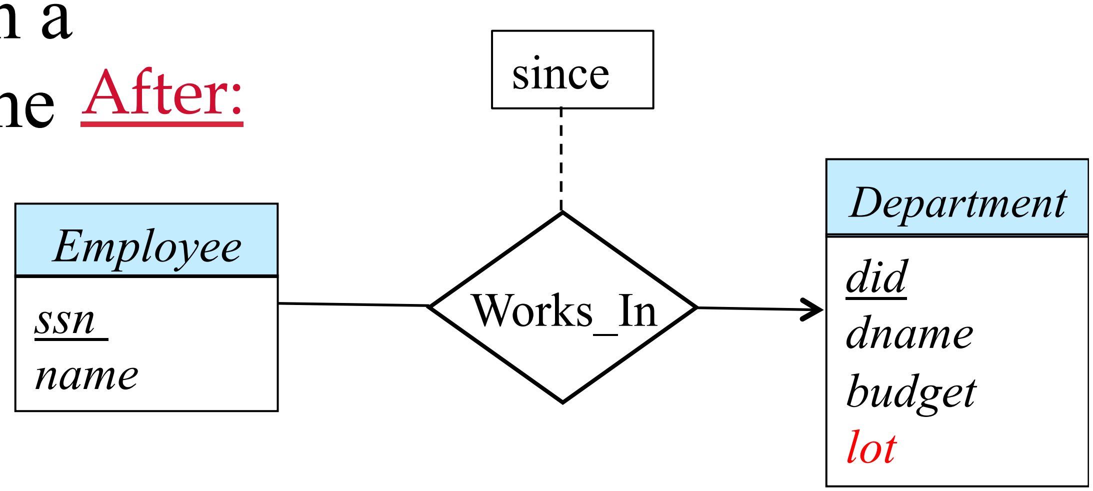
Refining an ER Diagram

- 1st diagram translated:
  Employee(S,N,L,D,S2)
  Department(D,M,B)
  - Lots associated with employees.
- Suppose all employees in a dept are assigned the same lot: D $\rightarrow$ L
- Can fine-tune this way:
  Employees(S,N,D,S2)
  Department(D,M,B,L)

Before:



After:



Functional Dependencies (FDs)

- A functional dependency (FD) has the form: $X \rightarrow Y$, where X and Y are two *sets* of attributes.
  - Examples: rating→hrly_wage, AB →C
- The FD $X \rightarrow Y$ ***is satisfied by a relation instance r if:***
  - for each pair of tuples t1 and t2 in r:

    $t1.X = t2.X$ implies $t1.Y = t2.Y$
  - i.e., given any two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- Convention: X, Y, Z etc denote sets of attributes, and A, B, C, etc denote attributes.

Functional Dependencies (FDs)

- ***The FD holds*** over relation name R if, for every ***allowable*** instance *r* of R, *r* satisfies the FD.
- An FD, as an integrity constraint, is a statement about *all* allowable relation instances.
  - Must be identified based on semantics of application.
  - Given some instance *r1* of R, we can check if it ***violates*** some FD *f* or not
  - But we cannot tell if *f* ***holds*** over R by looking at an instance!
    - Cannot prove non-existence (of violation) out of ignorance
  - This is the same for all integrity constraints!

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  – Hourly_Emps (*ssn, name, lot, rating, hrly_wage, hrs_worked*)
- *Notation*: We will denote this relation schema by listing the attributes: SNLRWH
  – This is really the *set* of attributes {S,N,L,R,W,H}.
  – Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  – *ssn* is the key: S → SNLRWH
  – *rating* determines *hrly_wage*: R → W

One more example

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FDs with A as the left side:	**Satisfied by the relation instance?**
A→A	yes
A→B	yes
A→C	No
A→AB	yes
A→AC	No
A→BC	No
A→ABC	No

How many *possible* FDs totally on this relation instance?

Violation of FD by a relation

- The FD $X \rightarrow Y$ ***is NOT satisfied by a relation instance r if:***
  - There exists a pair of tuples t1 and t2 in r such that

    $t1.X = t2.X$ but $t1.Y \neq t2.Y$
  - i.e., we can find two tuples in *r*, such that X values agree, but Y values don't.

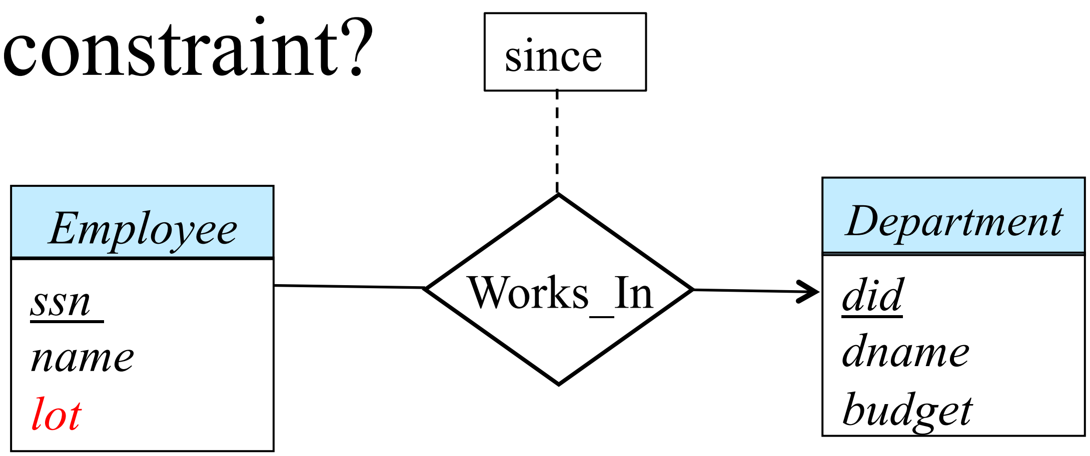
Some other FDs

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FD	**Satisfied by the relation instance?**
C→B	yes
C→AB	No
B→C	No
B→B	Yes
AC →B	Yes [note!]
…	…

Relationship between FDs and Keys

- Given R(A, B, C).
  - A→ABC means that A is a key.
- In general,
  - X → R means X is a (super)key.
- How about key constraint?
  - ssn → did



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did,\ did \rightarrow lot$ implies $ssn \rightarrow lot$
  - $A \rightarrow BC$ implies $A \rightarrow B$
- An FD *f* is *logically implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
  - F^+ = *closure of F* is the set of all FDs that are implied by *F*.

Armstrong's axioms

- Armstrong' s axioms are *sound* and *complete* inference rules for FDs!
  - Sound: all the derived FDs (by using the axioms) are those logically implied by the given set
  - Complete: all the logically implied (by the given set) FDs can be derived by using the axioms.

Reasoning about FDs

- How do we get all the FDs that are logically implied by a given set of FDs?
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*:
    - If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*:
    - If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*:
    - If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

A	B	C
1	1	2
2	1	3
2	1	3
1	1	2

Example of using Armstrong's Axioms

- Couple of additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Derive the above two by using Armstrong's axioms!

Derive Union

- Show that

If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

$X \rightarrow YX$ (augment) // Append X on both sides of $X \rightarrow Y$

$YX \rightarrow YZ$ (augment) // Append Y on both sides of $X \rightarrow Z$

Thus, $X \rightarrow YZ$ (transitive)

Derive Decomposition

- Show that

$$\text{If } X \rightarrow YZ, \text{ then } X \rightarrow Y \text{ and } X \rightarrow Z$$

$YZ \rightarrow Y$; $YZ \rightarrow Z$ (reflexive)

Thus, $X \rightarrow Y$, $X \rightarrow Z$ (transitive)

Another Useful Rule: Accumulation Rule

- If X $\rightarrow$ YZ and Z $\rightarrow$ W, then X $\rightarrow$YZW

From Z $\rightarrow$ W, augment with YZ to get YZ $\rightarrow$ YZW

Thus, X $\rightarrow$ YZW (transitive)

Derivation Example

- $R = (A, B, C, G, H, I)$
  $F = \{A \rightarrow B;\ A \rightarrow C;\ CG \rightarrow H;\ CG \rightarrow I;\ B \rightarrow H\}$
- some members of F^+ (how to derive them?)
  - $A \rightarrow H$

    By transitivity from A → B and B → H

  - $AG \rightarrow I$

    By augmenting A → C with G, to get AG → CG, and then transitivity with CG → I

  - $CG \rightarrow HI$

    By augmenting CG → I to infer CG → CGI, and augmenting CG → H to infer CGI → HI, and then transitivity (or use union rule)

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
      then add the resulting functional dependency to F^+
until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Example on Computing F+

- $F = \{A \rightarrow B,\ B \rightarrow C,\ C D \rightarrow E \}$
- Step 1: For each f in F, apply reflexivity rule
  - We get: $CD \rightarrow C$; $CD \rightarrow D$
  - Add them to F:
    - $F = \{A \rightarrow B,\ B \rightarrow C,\ C D \rightarrow E; CD \rightarrow C; CD \rightarrow D \}$
- Step 2: For each f in F, apply augmentation rule
  - From $A \rightarrow B$ we get: $A \rightarrow AB$; $AB \rightarrow B$; $AC \rightarrow BC$; $AD \rightarrow BD$; $ABC \rightarrow BC$; $ABD \rightarrow BD$; $ACD \rightarrow BCD$
  - From $B \rightarrow C$ we get: $AB \rightarrow AC$; $BC \rightarrow C$; $BD \rightarrow CD$; $ABC \rightarrow AC$; $ABD \rightarrow ACD$, etc etc.
- Step 3: Apply transitivity on pairs of f's
- Keep repeating… You get the idea