Schema Refinement & Normalization Theory

Module 9

Prof. Alex Brodsky

Database Systems

What's the Problem

- Consider relation obtained (call it SNLRHW)

  Hourly_Emps(<u>*ssn*</u>, *name, lot, rating, hrly_wages, hrs_worked*)

- What if we ***know*** rating determines hrly_wages?

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 of the existence of *integrity constraints.*

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!

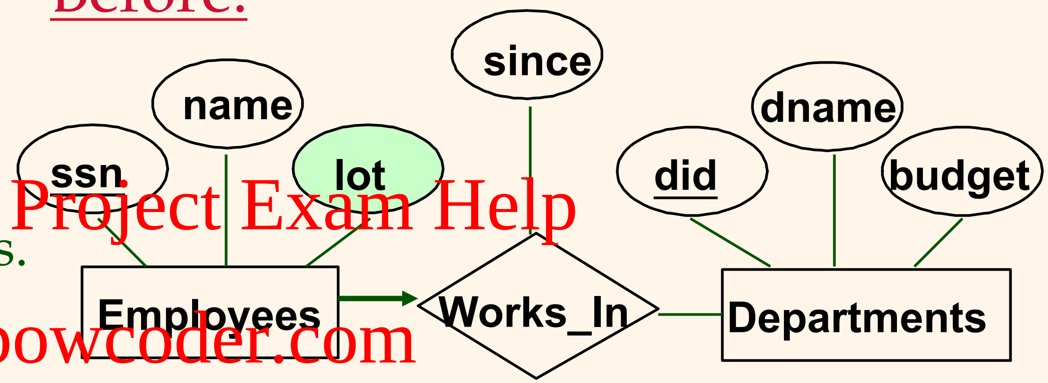
What do we do?

- 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?

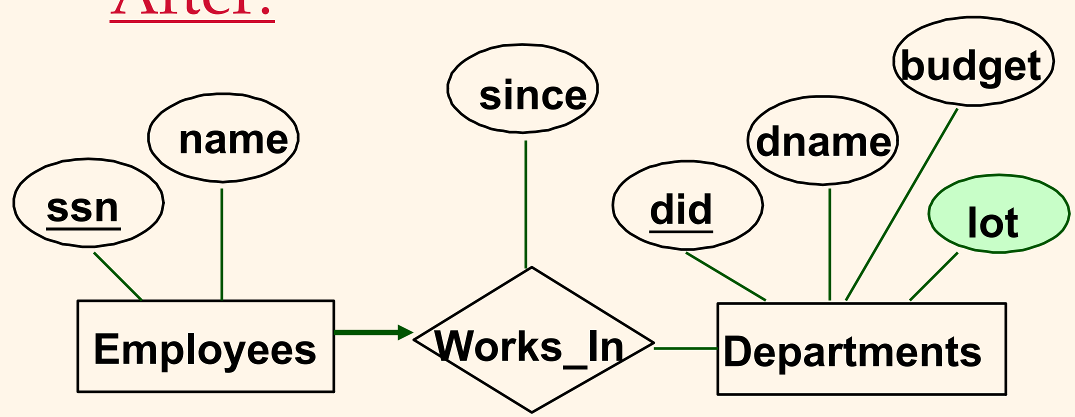
Refining an ER Diagram

- 1st diagram translated:
  Workers(S,N,L,D,S)
  Departments(D,M,B)
  - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot: D $\rightarrow$ L
- Can fine-tune this:
  Workers2(S,N,D,S)
  Departments(D,M,B,L)

Before:



After:



Functional Dependencies (FDs)

- A <u>functional dependency</u> (FD) has the form: X→Y, where X and y are two sets of attributes.
  - Examples: Rating → hrly_Wage, AB → C

- The FD X→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!
  - 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_wages*, *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 $\rightarrow$ SNLRWH
  - *rating* determines *hrly_wages*: R $\rightarrow$ 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?
49.

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.

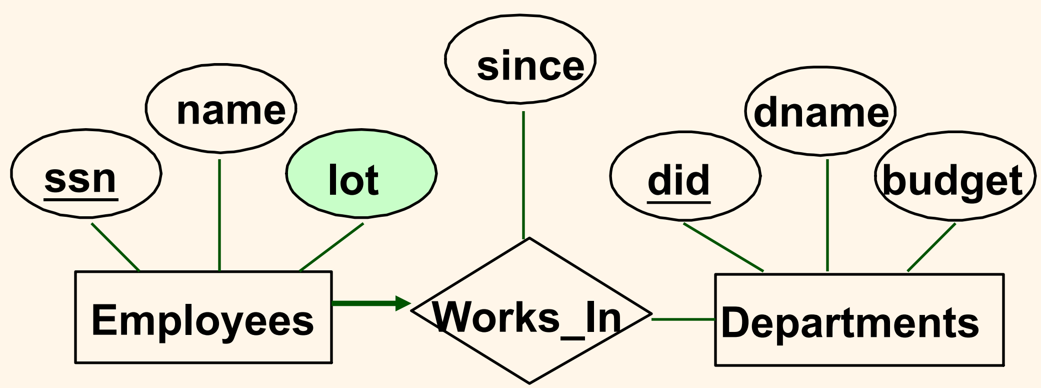
Some other FDs

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).
  - $A \rightarrow ABC$ means that A is a key.
- In general,
  - $X \rightarrow R$ means X is a (super)key.
- How about key constraint?
  - $ssn \rightarrow did$



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - $ssn \to did,\ did \to lot$ implies $ssn \to lot$
  - $A \to BC$ implies $A \to B$

- An FD *f* is *logically implied by* a set of FDs *F*, denoted F |= f, if for every relational instance r, the following holds:
  - if r satisfies all FD's in F,
  - then r satisfies f

- *The closure of F, denoted* F^+ is the set of all FDs that are logically implied by *F*.

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 $X \rightarrow Z$

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.

Example of using Armstrong's Axioms

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

Reasoning About FDs (Contd.)

- Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - C is the key: C $\rightarrow$ CSJDPQV
  - Project (jid) purchases each part using single contract:
    JP $\rightarrow$ C

  - Dept purchases at most one part from a supplier: SD $\rightarrow$ P

- JP $\rightarrow$ C, C $\rightarrow$ CSJDPQV imply JP $\rightarrow$ CSJDPQV

- SD $\rightarrow$ P implies SDJ $\rightarrow$ JP
- SDJ $\rightarrow$ JP, JP $\rightarrow$ CSJDPQV imply SDJ $\rightarrow$ CSJDPQV

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs *F*.

- An efficient check:
  - Compute *attribute closure* of X (denoted X^+) wrt *F:*
    - Set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this.

- Claim: F |= X → Y if and only if Y is in X^+
- Example: Does F = {$A \rightarrow B$, $B \rightarrow C$, $C\,D \rightarrow E$ } imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Computing X^+

- Input F (a set of FDs), and X (a set of attributes)
- Output: Result=X^+ (under F)
- Method:

  - Step 1: Result := X;

  - Step 2: Take $Y \rightarrow Z$ in F, and Y is in Result, do:

    Result := Result *union* Z

  - Repeat step 2 until Result cannot be changed and then output Result.

Example of computing X^+

- $F=\{A \rightarrow B, AC \rightarrow D, AB \rightarrow C\}$
- $X=A$
- Result should be $X^+=ABCD$

Computing F^+

- Given F={ $A \rightarrow B$, $B \rightarrow C$}. Compute F^+ (with attributes A, B, C).

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B		√	√			√	
C			√				
AB	√	√	√	√	√	√	√
AC	√	√	√	√	√	√	√
BC		√	√			√	
ABC	√	√	√	√	√	√	√

Attribute closure
A^+=ABC
B^+=BC
C^+=C
AB^+=ABC
AC^+=ABC
BC^+=BC
ABC^+=ABC

- An entry with √ means FD (the row) $\rightarrow$ (the column) is in F^+.
- An entry gets √ when (the column) is in (the row)$^+$

Check if two sets of FDs are equivalent

- Two sets of FDs are equivalent if they logically imply the same set of FDs.
  - I.e., if $F_1^+ = F_2^+$, then they are equivalent.
- For example, $F_1=\{A \rightarrow B, A \rightarrow C\}$ is equivalent to $F_2=\{A \rightarrow BC\}$
- How to test? Two steps:
  - *Every* FD in F_1 is in F_2^+
  - *Every* FD in F_2 is in F_1^+
- These two steps can use the algorithm (many times) for X^+

Summary

- Constraints give rise to redundancy
  - Three anomalies

- FD is a "popular" type of constraint
  - Satisfaction & violation

  - Logical implication

  - Reasoning