Schema Refinement and Normalization

Schema Refinements and FDs

- *Redundancy* is at the root of several problems associated with relational schemas.
  - redundant storage, I/D/U anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- 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?

Functional Dependencies (FDs)

- A functional dependency $X \to Y$ holds over relation schema R if, for every allowable instance *r* of R:
  - $t1 \in r$, $t2 \in r$, $t1[X] = t2[X]$ implies $t1[Y] = t2[Y]$

    (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that $K \to R$

    Moreover: we require K to be *minimal*!

Example: Constraints on Entity Set

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* {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

Example (Contd.)

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

Problems due to R → W :

- *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!

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

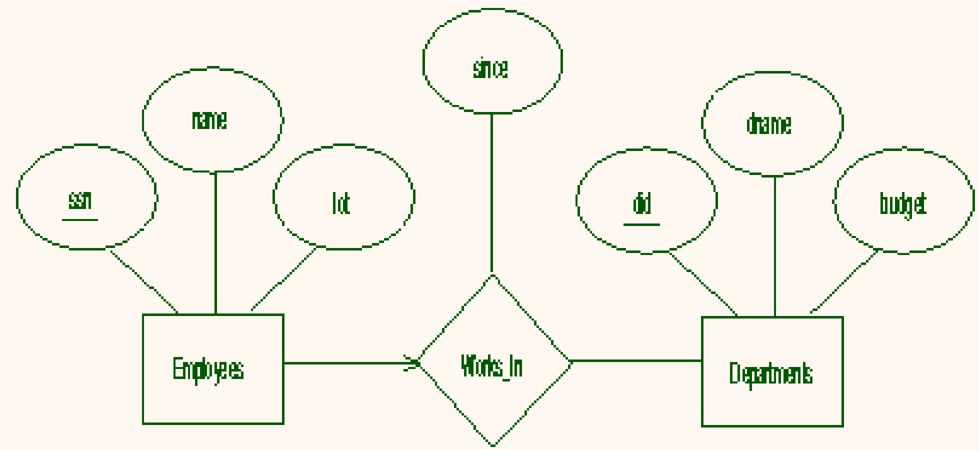
Hourly_Emps2

R	W
8	10
5	7

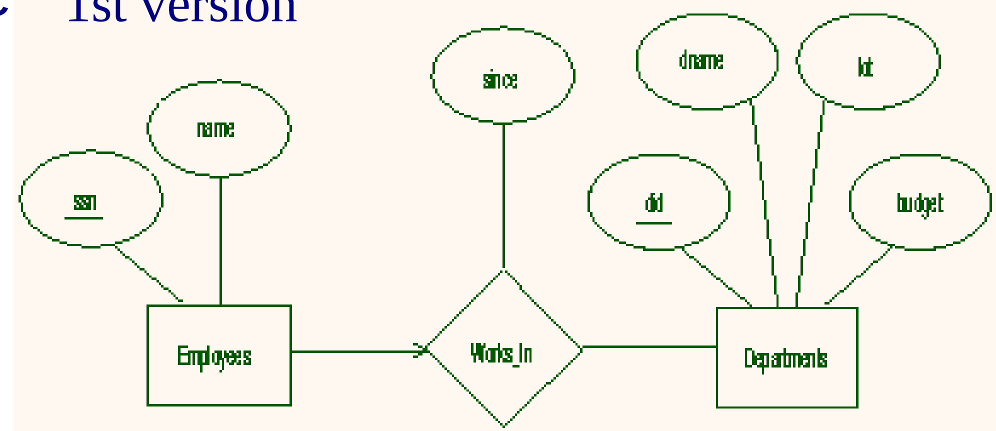
Wages

Anomalies and ER Diagrams

- 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, i.e $D \rightarrow L$
- Redundancy, fixed by: Workers2(S,N,D,S) Dept_Lots(D,L)
- Can fine-tune this: Workers2(S,N,D,S) Departments(D,M,B,L)



1st version



2nd version

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* implies *ssn* $\rightarrow$ *lot*
- An FD *f* is *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 (X, Y, Z are sets of attributes):
  - *Reflexivity*: If X $\subseteq$ Y, then Y $\rightarrow$ X
  - *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
- *Sound* and *complete* inference rules for FDs!
  - Remark: better rules not ``axioms''!

Reasoning About FDs (Contd.)

- 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 $X \rightarrow Z$

Example.: $F = \{ AC \rightarrow B, CB \rightarrow D \}$

Question: $AC \rightarrow D$?

1. $AC \rightarrow B$
2. $CB \rightarrow D$
3. $AC \rightarrow C$ ... By Reflexivity
4. by union from 1. a 3., $AC \rightarrow BC$
5. by transitivity from 2. a 4.,

$\Rightarrow \{AC \rightarrow D\} \in F^+$

Reasoning About FDs (Contd.)

Example: Contracts(*cid,sid,jid,did,pid,qty,value*)

(The contract C is an agreement that supplier S will supply Q items of a part P to project J associated with department D; the value V of this contract is equal to value.)

- C is the key: C $\rightarrow$ CSJDPQV
- Project 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.
  - Check if Y is in X^+
- 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^+?

Quadratic Algorithm for the Membership Problem

Input: R(A), where |A| = n,

F, where |F| = m,

f: $X \rightarrow C$, where **$C \in A$** and $X \subseteq A$.

elementary FDs

Output: Out ... of type Boolean

Data structures :

LS[1:m], RS[1:m] ... arrays of sets, i^{th} element contains attributes from the left and right side of i dependency from F, respectively.

CLOSUREX contains X^+ when algorithm stops

DONE variable of type Boolean.

Algorithm calculates **X^+**. If **$C \in X^+ \Rightarrow f \in F^+$**.

Quadratic Algorithm for the Membership Problem

```
begin
      CLOSUREX: = X;        {see reflexivity }
      DONE:= FALSE;
      while (not DONE) do
        begin DONE:= TRUE;
               for i = 1 to m do
              begin  if ( LS[i] ⊆ CLOSUREX ) and
                          ( RS[i] ⊄ CLOSUREX )
                  then begin
                  CLOSUREX:= CLOSUREX ∪ RS[i
];
                  DONE:= FALSE
                       end
              end
        end
      Out := (C ∈ CLOSUREX)
end
```

Reasoning About FDs (Contd.)

Example.: $F = \{ AC \rightarrow B, CB \rightarrow D \}$

Question: $AC \rightarrow D$?

Calculate it with the help of the algorithm!

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (3NF, BCNF, etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A $\rightarrow$ B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Third Normal Form (3NF)

- R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key (not just superkey) for R.
- *Minimality* of a key crucial in third condition above!
- If R is in 3NF, some redundancy is possible. It is a compromise (BCNF has better properties!).

What Does 3NF Achieve?

- If 3NF violated by X $\rightarrow$ A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs K $\rightarrow$ X $\rightarrow$ A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.

Decomposition of a Relation Scheme

- Suppose that R contains attributes A_1 ... A_n. A *decomposition* of R consists of replacing R by $R_1 \ldots R_K$, K>1, that:,
  - Each R_i contains a subset of the attributes of R (and no attributes that do not appear in R), and
  - every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S $\rightarrow$ SNLRWH and R $\rightarrow$ W
  - R $\rightarrow$ W causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., SNLRWH decomposed into SNLRH and RW
    - How much did sailor Joe earn? (need join both relations)
  - Given instances of the R_is, we may not be able to reconstruct the corresponding instance of the original relation R!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the R_is.
    - Fortunately, not in the SNLRWH example.
- *Tradeoff*: these issues vs. redundancy.

Lossless Join Decompositions

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
  - $r[X] * r[Y] = r$
- It is always true that $r \subseteq r[X] * r[Y]$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- We consider only binary decompositions.
- *It is essential that all decompositions used to deal with redundancy be lossless!*

More on Lossless Join

- In particular, the decomposition of R into UV and R - V is lossless-join if $U \rightarrow V$ holds over R.

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Decomposition into 3NF

- Consider relation R with FDs F. If X $\rightarrow$ Y violates 3NF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in 3NF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, JP $\rightarrow$ C, SD $\rightarrow$ P, J $\rightarrow$ S
    - To deal with SD $\rightarrow$ P, decompose into SDP, CSJDQV.
    - To deal with J $\rightarrow$ S, decompose CSJDQV into JS and CJDQV
- In general, several dependencies may cause violation of 3NF. The order in which we „deal with'' them could lead to very different sets of relations!

Dependency Preserving Decomposition

- Consider CSJDPQV, JP → C and SD → P.
  - 3NF(lossless) decomposition: CSJDQV and SDP
  - Problem: Checking JP → C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, and Y, and we enforce the FDs that hold on X and on Z, then all FDs that were given to hold on R must also hold.
- *Projection of set of FDs F*:
  - If R is decomposed into X, and Y, then projection of F onto X (denoted F_X) is the set of FDs U → V in F^+ (*closure of F*) such that U, V in X.

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $A \rightarrow C$, decomposed into AB and BC.
  - Is this dependency preserving? Is $A \rightarrow C$ preserved? Yes!
- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
- And vice-versa! (Example? See previous slide)

3NF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into 3NF.
  - e.g., CSZ, CS → Z, Z → C
  - Can't decompose while preserving CS → Z;
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- To ensure dependency preservation, one idea:
  - If X $\rightarrow$ Y is not preserved, add relation XY
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' JP $\rightarrow$ C. What if we also have J $\rightarrow$ C ?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``as small as possible'' in order to get the same closure as F.
- e.g.,F: A $\rightarrow$ B, ABCD $\rightarrow$ E, EF $\rightarrow$ GH, ACDF $\rightarrow$ EG has the following minimal cover G:

  G: A $\rightarrow$ B, ACD $\rightarrow$ E, EF $\rightarrow$ G and EF $\rightarrow$ H

Minimal Cover for a Set of FDs

Proof:

1. F is derivable from G

- Is $ABCD \rightarrow E$ derivable?

$ACD \rightarrow E \Rightarrow ABCD \rightarrow E$

- Is $ABCF \rightarrow GE$ redundant in G?

$ACD \rightarrow E \Rightarrow ACDF \rightarrow EF \rightarrow G \Rightarrow ACDF \rightarrow EG$

1. G is derivable from F

❖ Is $ACD \rightarrow E$ derivable?

$A \rightarrow B \Rightarrow ACD \rightarrow BCD$

$ACD \rightarrow A \Rightarrow ACD \rightarrow ABCD \rightarrow E$

Summary

- If a relation is in 3NF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in 3NF is a good heuristic.
- If a relation is not in 3NF, we can try to decompose it into a collection of 3NF relations.
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.