Schema Refinement and Normalization

1

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - ***redundant storage, insert/delete/update anomalies***
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: <u>*decomposition*</u> (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?

2

Functional Dependencies (FDs)

- A <u>functional dependency</u> $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r$, $t2 \in r$, $t1.X = t2.X$ implies $t1.Y = t2.Y$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Identified by DBA 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 $K \rightarrow R$
  - However, $K \rightarrow R$ does not require K to be *minimal*!

3

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (<u>*ssn*</u>, *name, lot, rating, hrly_wages, hrs_worked*)
- <u>*Notation*</u>: 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*: $R \rightarrow W$

4

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 $R \rightarrow W$:
  - <u>*Update anomaly*</u>: Can we change W in just the 1st tuple of SNLRWH?
  - <u>*Insertion anomaly*</u>: What if we want to insert an employee and don't know the hourly wage for his rating?
  - <u>*Deletion anomaly*</u>: 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

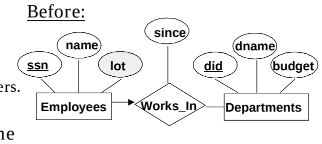
Wages

5

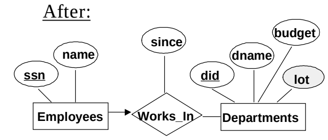
Refining an ER Diagram

- 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: $D \rightarrow L$
- 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)

<u>Before:</u>



<u>After:</u>



6

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* → *did*, *did* → *lot* implies *ssn* → *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 Y ⊆ X, then X → Y
  - *Augmentation*: If X → Y, then XZ → YZ for any Z
  - *Transitivity*: If X → Y and Y → Z, then X → Z
- These are *sound* and *complete* inference rules for FDs!

7

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from AA):
  - *Union*: If X → Y and X → Z, then X → YZ
  - *Decomposition*: If X → YZ, then X → Y and X → Z
- Example: Contracts(*cid*,*sid*,*jid*,*did*,*pid*,*qty*,*value*), and:
  - C is the key: C → CSJDPQV
  - Project purchases each part using single contract: JP → C
  - Dept purchases at most one part from a supplier: SD → P
- JP → C, C → CSJDPQV imply JP → CSJDPQV
- SD → P implies SDJ → JP
- SDJ → JP, JP → CSJDPQV imply SDJ → CSJDPQV

8

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* *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 A is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- Does F = {A B, B C, C D E } imply A E?
  - i.e, is A E in the closure F^+? Equivalently, is E in ?

9

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* (BCNF, 3NF 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 B: Several tuples could have the same A value, and if so, they'll all have the same B value!

10

First Normal Form

- Each field is an atomic (scalar) value
- All relations are in first normal form (1NF)

11

Second Normal Form (2NF)

- A non-key field must not be a fact about a subset of a key (no partial dependencies)
- If X=A ∪ B is a candidate key, A does not determine Y for any non-empty A, non-empty B, and non-prime Y.
- All candidate keys have 1 field ⇒ 2NF
- *Example*: R = (part, wh, qty, wh-addr), with FDs {part, wh} → qty and wh → wh-addr
- Is R in 2NF?

12

2NF (continued)

- No, it's in 1NF only. Only candidate key is {part, wh}, and 2nd dependency violates 2NF.
- So what?
- We're storing wh-addr once for each part stored at that warehouse.
- Solution: R1 (part, wh, qty) and R2 (wh, wh-addr)
- Are R1, R2 in 2NF?

13

Third Normal Form (3NF)

- A non-key field is not a fact about another non-key field.
- Reln R with FDs *F* is in 3NF if, for all X→ A in $F+$
  - A ∈ X (called a *trivial* FD), or
  - X is a superkey for R, or
  - A is prime
- *Example: ED = (enum, ename, sal, dnum, dname, mgr)*
  - FDs: enum → ename, sal, dnum and dnum → dname, mgr
  - Is ED in 2NF? 3NF?
- The problem: dname, mgr stored for all employees in a department

14

3NF (continued)

- One solution:
  - Emp (enum, ename, sal, mgr)
  - Dept (dnum, dname, mgr)
  - Removes the redundancy
  - A problem remains:
- This is an example of a lossy-join decomposition, which is avoidable in 3NF.
- Use Emp1 (enum, ename, sal, dnum) instead of Emp.

15

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all X A in
  - A X (called a *trivial* FD), or
  - X is a superkey
- 3NF plus: no prime field describes a non-prime field
- Example: R = (branch, cust, banker)
  - FDs: banker → branch and cust, branch → banker
  - Is R in 3NF? BCNF?
- Problems:
  - Bankers must have at least one customer
  - Branch stored redundantly for each of a banker's customers

16

BCNF (continued)

- Decomposition: R1 (banker, branch) and R2 (cust, banker)
  - avoids redundancy and other problems
  - A problem remains:
- In this case, no dependency-preserving decomposition into BCNF is possible.

17

Decomposition of a Relation Scheme

- Suppose that relation R contains attributes *A1 ... An*. A <u>*decomposition*</u> of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a proper subset of the attributes of R (and no attributes not 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.

18

Problems with Decompositions

- There are three potential problems to consider:
  - ❶ Some queries become more expensive.
    - e.g., find employee and department names
  - ❷ Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - First 3NF decomposition attempt
  - ❸ Checking some dependencies may require joining the instances of the decomposed relations.
    - BCNF decomposition example
- *Tradeoff*: Must consider these issues vs. redundancy.

19

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:
  - $\pi_X(r) \,|\times|\, \pi_Y(r) = r$
- It is always true that $r \subseteq \pi_X(r) \,|\times|\, \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (avoids Problem 2, previous slide)*

20

More on Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - X $\cap$ Y   X, or
  - X $\cap$ Y   Y
- In particular, the decomposition of R into UV and R - V is lossless-join if U   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

21

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP   C and SD   P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP   C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y 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, ... projection of F onto X (denoted F_X) is the set of FDs U   V in F^+ (*closure of F*) such that U, V are in X.

22

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is *dependency preserving* if $(F_X \text{ union } F_Y)^+ = F^+$
  - 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   B, B   C, C   A, decomposed into AB and BC.
  - Is this dependency preserving? Is C   A preserved?????
- Dependency preserving does not imply lossless join:
  - ABC, A   B, decomposed into AB and BC.
- And vice-versa! (Example?)

23

Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.
  - Various decompositions of a single schema are possible.

24