Course Name:
Database Management Systems

Lecture 15
Topics to be covered

- Functional Dependencies

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: *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?

INTRODUCTION TO SCHEMA REFINEMENT

Problems Caused by Redundancy

- Storing the same information **redundantly**, that is, in more than one place within a database, can lead to several problems:
- **Redundant storage:** Some information is stored repeatedly.
- **Update anomalies:** If one copy of such repeated data is updated, an inconsistency
- is created unless all copies are similarly updated.
- **Insertion anomalies:** It may not be possible to store some information unless
- some other information is stored as well.
- **Deletion anomalies:** It may not be possible to delete some information without

- losing some other information as well.
- Consider a relation obtained by translating a variant of the Hourly Emps entity set

Ex: Hourly Emps(*ssn*, *name*, *lot*, *rating*, *hourly wages*, *hours worked*)

- The key for Hourly Emps is *ssn*. In addition, suppose that the *hourly wages* attribute
- is determined by the *rating* attribute. That is, for a given *rating* value, there is only
- one permissible *hourly wages* value. This IC is an example of a *functional dependency*.
- It leads to possible redundancy in the relation Hourly Emps

Use of Decompositions

- Intuitively, redundancy arises when a relational schema forces an association between attributes that is not natural.
- Functional dependencies (ICs) can be used to identify such situations and to suggest revetments to the schema.
- The essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of smaller relations.
- Each of the smaller relations contains a subset of the attributes of the original relation.
- We refer to this process as decomposition of the larger relation into the smaller relations

- We can deal with the redundancy in Hourly Emps by decomposing it into two relations:
- Hourly Emps2(*ssn*, *name*, *lot*, *rating*, *hours worked*)
- Wages(*rating*, *hourly wages*)

rating	*hourly wages*
8	10
5	7

ssn	*name*	*lot*	*rating*	*hours worked*
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

Problems Related to Decomposition

- Unless we are careful, decomposing a relation schema can create more problems than it solves.
- Two important questions must be asked repeatedly:
- 1. Do we need to decompose a relation?
- 2. What problems (if any) does a given decomposition cause?
- To help with the rst question, several *normal forms* have been proposed for relations.
- If a relation schema is in one of these normal forms, we know that certain kinds of
- problems cannot arise. Considering the n

Functional Dependencies (FDs)

- A <u>functional dependency</u> $X \longrightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - 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.
  - 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 $K \longrightarrow R$
  - However, $K \longrightarrow R$ does not require K to be *minimal*!

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 $\longrightarrow$ SNLRWH
  - *rating* determines *hrly_wages*: R $\longrightarrow$ W

Example (Contd.)

→

- 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!

Wages

R	W
8	10
5	7

Hourly_Emps2

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

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

Constraints on a Relationship Set

- Suppose that we have entity sets Parts, Suppliers, and Departments, as well as a relationship set Contracts that involves all of them. We refer to the schema for Contracts as *CQPSD*. A contract with contract id
- *C* species that a supplier *S* will supply some quantity *Q* of a part *P* to a department *D*.
- We might have a policy that a department purchases at most one part from any given supplier.
- Thus, if there are several contracts between the same supplier and department,
- we know that the same part must be involved in all of them. This constraint is an FD, *DS ! P*.

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
- These are *sound* and *complete* inference rules for FDs!

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: Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - 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: S
  - D $\rightarrow$ 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 $X \longrightarrow Y$ 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 $\longrightarrow$ A is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- Does F = {A $\longrightarrow$ B, B $\longrightarrow$ C, C D $\longrightarrow$ E } imply A $\longrightarrow$ E?
  - i.e, is A $\longrightarrow$ E in the closure F^+ ? Equivalently, is E in A^+ ?

Closure of a Set of FDs

- The set of all FDs implied by a given set *F* of FDs is called the **closure of F** and is denoted as *F*+.
- An important question is how we can **infer**, or compute, the closure of a given set *F* of FDs.
- The following three rules, called **Armstrong's Axioms**, can be applied repeatedly to infer all FDs implied by a set *F* of FDs.
- We use *X*, *Y*, and *Z* to denote *sets* of attributes over a relation schema *R*:

Closure of a Set of FDs

- **Reflexivity:** If *X* *Y*, then *X !Y*.
- **Augmentation:** If *X ! Y*, then *XZ ! YZ* for any *Z*.
- **Transitivity:** If *X ! Y* and *Y ! Z*, then *X ! Z*.
- Armstrong's Axioms are **sound** in that they generate only FDs in *F*+ when applied to a set *F* of FDs.
- They are **complete** in that repeated application of these rules will generate all FDs in the closure *F*+.
- It is convenient to use some additional rules while reasoning about *F*+:
- **Union:** If *X ! Y* and *X ! Z*, then *X !YZ*.
- **Decomposition:** If *X ! YZ*, then *X !Y* and *X ! Z*.
- These additional rules are not essential; their soundness can be proved using Armstrong's Axioms.

Attribute Closure

- If we just want to check whether a given dependency, say, $X \rightarrow Y$, is in the closure of a set F of FDs,
- we can do so eciently without computing $F+$. We rst compute the **attribute closure** $X+$ with respect to F,
- which is the set of attributes A such that $X \rightarrow A$ can be inferred using the Armstrong Axioms.
- The algorithm for computing the attribute closure of a set X of attributes is
- *closure* = X;

  repeat until there is no change: *{*

  if there is an FD $U \rightarrow V$ in F such that *U subset of closure*,

  then set *closure* = *closure union of V*

  }