CS/INFO 330
Schema Refinement and Normal Forms

Mirek Riedewald
mirek@cs.cornell.edu

Overview

- Quick return to data tier
  - Needed for designing database schema
  - Will proceed to middle tier later
- Challenge: How do we design a good database?
  - Recall: Many ER design choices
  - Some resulting databases intuitively better
  - Are there objective criteria?

The Evils of Redundancy

- Redundancy at the root of several problems associated with relational schemas:
  - Redundant storage, insert/delete/update anomalies
- Can use integrity constraints, in particular functional dependencies, to identify such problems and to suggest refinements
- Main refinement technique: decomposition
  - Example: replace schema ABCD with 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)

- Functional dependency X→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 their X values agree, then the Y values must also agree (X and Y are *sets* of attributes)
- 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
- Special case: K is a superkey for R means that K→R
  - However, K→R does not require K to be *minimal*

Example: Constraints on Entity Set

- Consider relation 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→SNLRWH
  - *rating* determines *hrly_wages*: R→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

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

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

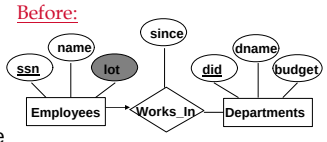
Wages

R	W
8	10
5	7

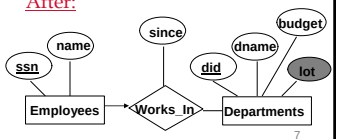
Refining an ER Diagram

- 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→L
  - Redundancy
  - Fixed by:
    Workers2(S,N,D,S)
    Dept_Lots(D,L)
- Can fine-tune this to
  Workers2(S,N,D,S)
  Departments(D,M,B,L)

Before:



After:



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 $X \subseteq Y$, 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.)

- Additional rules (follow from Armstrong Axioms):
  - 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*)
  - 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 $SDJ \rightarrow CSJDPQV$

Example Inference

- Prove the following inference rule using only Armstrong's axioms:
  - If X→Y and W→Z, then XW→YZ

- 1. XW→YW (Augmentation)
- 2. WY→ZY (Augmentation)
- 3. XW→YW→YZ (Transitivity)

Another Example

- Prove the following inference rule for functional dependencies using only Armstrong's axioms
  - If P→QR, Q→S, R→T, then P→ST

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive
  - Size of closure is exponential in # attributes
- 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→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, CD→E } imply A→E?
  - I.e, is A→E in the closure F^+? Equivalently, is E in A^+?

Normal Forms

- When is schema refinement needed?
- If a relation is in a certain normal form, it is known that certain kinds of problems are avoided/minimized
- First Normal Form (1NF)
  - Every field only contains atomic values (no lists or sets)
- Second Normal Form (2NF)
  - Only of historical interest
- Role of FDs in detecting redundancy:
  - Consider relation R with 3 attributes, ABC
    - No FDs hold: No redundancy here
    - Given A→B: Several tuples could have same A value, and if so, they'll all have the same B value

Announcements

- HW2 clarification: Aggregate operators are not considered part of relational algebra for this course
  - General policy: Terminology and definitions are what is on my slides and in the book
  - You can use additional knowledge from the web, BUT if there is the slightest chance that it conflicts with slides/book, do not use it (or at least ask me first)
- First part of project assignment out on CMS
  - We try to make it as realistic as possible
  - Form groups of up to 4
    - Should be a real team effort
    - Part of submission is "what I did for this project and what the others in my team did" report from everybody
  - Decide on an application
  - Write a requirements document
- Later: You will write a design document and implement the application

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all X→A in F^+
  - A∈X (called a trivial FD), or
  - X contains a key for R
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints
  - No redundancy in R that can be detected using FDs alone
    - Given values for attributes in X, there is no other tuple in R with the same value combination
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF and U→B, then U is a key and both tuples must be identical

U	V	B
u	v1	b
u	v2	?

BCNF or "Where are the non-superkey FDs?"

- Contracts(contractID, supplierID, projectID, deptID, partID, qty, value)
  - Project purchases given part using single contract: projectID, partID → contractID
    - Superkey
  - Dept purchases at most one part from a supplier: supplierID, deptID → partID
    - Do not store partID in Contracts, but in separate table SupplierRelationship(supplierID, deptID, partID)
    - Would not occur in relation in BCNF

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all X→A in F^+
  - A∈X (called a trivial FD), or
  - X contains a key for R, or
  - A is part of some key for R
- Minimality of a key is crucial in third condition above
- If R is in BCNF, obviously also in 3NF
- If R is in 3NF, some redundancy is possible
  - It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomposition, or performance considerations)
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible, not so for BCNF
    - This is the main motivation for 3NF

What Does 3NF Achieve?

- If 3NF violated by X→A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly
    - E.g., Reserves(SBDC) with only key SBD and FD S→C
      - Credit card number stored repeatedly for each sailor
  - X is not a proper subset of any key
    - There is a chain of FDs K→X→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
    - E.g., Hourly_Emps(SNLRWH) with only key S and FD R→W
      - Chain S→R→W, i.e., cannot record fact that employee S has rating R without knowing hourly wage for that rating
- But: even if relation is in 3NF, similar problems could arise
  - E.g., Reserves SBDC, S→C, C→S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored
- Thus, 3NF is indeed a compromise relative to BCNF

Decomposition of a Relation Scheme

- Suppose relation R contains attributes *A1 ... An*
- A decomposition of R consists of replacing R by two or more relations such that:
  - Each new relation scheme 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 at least 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., decompose SNLRWH into SNLRH and RW

Example Decomposition

- Decompositions should be used only when needed
  - SNLRWH has FDs S→SNLRWH and R→W
  - Second FD 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
    - Decompose SNLRWH into SNLRH and RW
- But: Information to be stored consists of SNLRWH tuples
  - 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

- Three potential problems:
  1. Some queries become more expensive
     - E.g., How much did sailor Joe earn? Salary = W*H; now need a join to get the information
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation
     - Fortunately, not the case in the SNLRWH example
  3. Checking some dependencies may require joining the instances of the decomposed relations
     - Fortunately, not the case in the SNLRWH example
- Tradeoff: Must consider 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:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- Always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - 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)

More on Lossless Join

- Decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - X∩Y→X, or
  - X∩Y→Y
- Special case: 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

AB⋈BC

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 (intuition):
  - 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. (avoids Problem 3)
- 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

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: Consider F^+, not F, in this definition
  - Example: 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
  - Example: ABC, A→B, decomposed into AB and BC
- And vice-versa
  - Example?

Decomposition into BCNF

- Consider relation R with FDs F. If X→Y violates BCNF, decompose R into R - Y and XY
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate
  - E.g., CSJDPQV, key C, JP→C, SD→P, J→S
  - To deal with SD→P, decompose into SDP, CSJDQV
  - To deal with J→S, decompose CSJDQV into JS and CJDQV
- Note: Several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations.

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF
  - E.g., Reservations(SBD), SB→D (sailor can reserve given boat at most once), D→B (at most one boat can be reserved on any given day)
    - SBD not in BCNF because D not key
    - Cannot decompose while preserving SB→D
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. 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
    - But: JPC tuples stored only for checking FD (Redundancy!)

Decomposition into 3NF

- Algorithm for lossless join decomposition into BCNF can be used to obtain a lossless join decomposition into 3NF (typically, can stop earlier)
- How to ensure dependency preservation? Idea:
  - If X→Y not preserved, add relation XY
  - Problem: XY may violate 3NF
    - Consider addition of CJP to `preserve' JP→C. What if we also have J→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., A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover:
  - A→B, ACD→E, EF→G and EF→H
- Can obtain lossless-join, dependency preserving decomposition into 3NF based on minimum cover

Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs.
  - 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.
  - Caused by FD X→A where X is not superkey => decompose according to such FD
    - Always lossless-join
  - Must consider whether all FDs are preserved
  - If lossless-join, dependency preserving decomposition into BCNF not possible (or unsuitable, given typical queries), consider decomposition into 3NF
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind