Schema Refinement and Normal Forms

Chapter 19

Why Is This Important?

- Many ways to model a given scenario in a database
- How do we find the best one?
- We will discuss objective criteria for evaluating database design quality
  - Formally define desired properties
  - Algorithms for determining if a database has these properties
  - Algorithms for fixing problems

The Evils of Redundancy

- Redundancy is at the root of several problems associated with relational schemas:
  - Redundant storage
  - Insert, delete, update anomalies
- Integrity constraints 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 $X \rightarrow Y$ 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 $K \rightarrow R$
  - However, $K \rightarrow R$ does not require K to be minimal.

Example: Constraints on Entity Set

- Consider a 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: $R \rightarrow W$

Example (Contd.)

Are the two smaller tables better?

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

- Problems in single "wide" table due to $R \rightarrow W$:
  - Update anomaly: Can we change W in just the first 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

Reasoning About FDs

- Given some FDs, we can 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 $Y \to X$.
  - Augmentation: If $X \to Y$, then $XZ \to YZ$ for any Z.
  - Transitivity: If $X \to Y$ and $Y \to Z$, then $X \to Z$.
- These are sound (generate only FDs in F^+) and complete (generate all FDs in F^+) inference rules for FDs.

Reasoning About FDs (Contd.)

- Additional rules (that follow from the AA):
  - Union: If $X \to Y$ and $X \to Z$, then $X \to YZ$
  - Decomposition: If $X \to YZ$, then $X \to Y$ and $X \to Z$
- 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

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 $X \to Y$ is in the closure of a set of FDs F. An efficient algorithm:
  - Compute attribute closure of X (denoted X^+) wrt F:
    - Set of all attributes A such that $X \to A$ 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+?

So, What Do We Do Now With FDs?

- Essential for identifying problems in a database design
- Provide a way for "fixing" the problem
- Key concept: normal forms
  - A relation that is in a certain normal form has certain desirable properties

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 or minimized.
  - Helps deciding whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with three 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 all have the same B value.

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for all $X \to A$ in F^+
  - $A \in X$ (called a trivial FD), or
  - X is a superkey for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - R is free of any redundancy caused by FDs alone.
    - No field of any tuple can be inferred (using only FDs) from the values in the other fields in the relation instance
    - For $X \to A$, consider two tuples with the same X value.
    - They should have the same A value. Redundancy?
    - No. Since R is in BCNF, X is a superkey and hence the "two" tuples must be identical.

X	Y	A
x	y1	a
x	y2	?

Problems Prevented By BCNF

- If BCNF is violated by (non-trivial) FD X→A, one of the following holds:
  - X is a subset of some key K.
    - We store (X, A) pairs redundantly.
    - E.g., Reserves(S, B, D, C) with SBD as only key and FD S→C
      - Credit card number of a sailor stored for each reservation
  - X is not a proper subset of any key.
    - Redundant storage of (X, A) pairs as above
    - And 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(S, N, L, R, W, H) with S as only key and FD R→W
      - Have chain S→R→W, hence cannot record the fact that employee S has rating R without knowing the hourly wage for that rating

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 is a superkey 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, is it automatically in 3NF? What about the other direction?
- If R is in 3NF, some redundancy is possible.
  - 3NF is a compromise, used when BCNF is not achievable (e.g., no ``good'' decomposition, or performance considerations).
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is always possible. (covered soon)

What Does 3NF Achieve?

- Prevents same problems as BCNF, except for FDs where A is part of some key
  - Consider FD X→A where X is no superkey, but A is part of some key
  - E.g., Reserves(S, B, D, C) with only key SBD and FDs S→C and C→S is in 3NF
    - Notice: same example as before, but adding C→S made it 3NF
      - Why? Since C→S and SBD is a key, CBD is also a key. Hence for S→C, C is part of a key
    - Redundancy problem: for each reservation of sailor S, same (S, C) pair is stored.
- BCNF did not suffer from this redundancy problem.
- So, why do we need 3NF? Let's look at decompositions first.

Footnote About Other Normal Forms

- 1NF: every field contains only atomic values, i.e., no lists or sets
- 2NF: 1NF, and all attributes that are not part of any candidate key are functionally dependent on the whole of every candidate key
  - 3NF implies 2NF
- 4NF: prevents redundancy from multi-valued dependencies (see book)
- 5NF: addresses redundancy based on join dependencies, which generalize multi-valued dependencies (see book)

Decomposition of a Relation Schema

- Suppose relation R contains attributes A1,..., An. A decomposition of R replaces R by two or more relations such that:
  - Each new relation schema 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.
- Intuition: decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.

Example Decomposition

- Decompositions should be used only when needed.
  - Let SNLRWH have FDs S→SNLRWH and R→W
  - Second FD causes violation of 3NF
    - W values repeatedly associated with R values.
  - Easiest fix: create a relation RW to store these associations and remove W from the main schema:
    - I.e., we decompose SNLRWH into SNLRH and RW
- Each SNLRWH tuple will now be projected into two tuples, SNLRH and RW, each stored in the corresponding relation
  - Are there any potential problems with this approach?

Problems with Decompositions

- Three potential problems to consider:
  - Some queries become more expensive.
    - E.g., how much did sailor Joe earn? (salary = W*H)
  - 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.
  - 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.

Reconstructing A Relation

Original table

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

Decomposition

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

Joined back together

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

What went wrong?

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:
  - $\pi_X(R) \bowtie \pi_Y(R) = R$
- It is 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 three or more relations in a straightforward way.
- It is essential that *all* decompositions used to deal with redundancy be lossless. Why?

More on Lossless Join

- The decomposition of R into X and Y is lossless-join w.r.t. F if and only if the *closure* of F contains:
  - $X \cap Y \to X$, or
  - $X \cap Y \to Y$
- Special case:
  - For FD $U \to V$, the decomposition of R into UV and R – V is lossless-join.

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

Dependency-Preserving Decomposition

- Consider CSJDPQV, C is key, JP→C and SD→P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP→C now requires a join.
- Dependency-preserving decomposition (intuition):
  - Can enforce all FDs by examining a single relation instance on each insertion or modification of a tuple (do not need to join multiple relation instances)
- Formal definition requires notion of a projection of a set of FDs F over R:
  - If R is decomposed into X and Y, the projection of F onto X (denoted F_X) is the set of all FDs U→V in F^+ (closure of F) such that U and V both 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 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?)

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 and CSJDQV.
  - To deal with J→S, decompose CSJDQV into JS and CJDQV.
- In general, 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., CSZ with CS→Z and Z→C
  - Not in BCNF, but cannot 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). Why?
  - Note: adding relation JPC gives us a dependency-preserving decomposition into BCNF.
    - Problem: redundancy across relations. Each relation by itself is in BCNF (i.e., no redundancy within relation), but JPC's tuples can be obtained by joining CSJDQV and SDP.

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).
- To ensure dependency preservation, one idea:
  - If X→Y is not preserved, add relation XY.
  - Problem is that XY may violate 3NF.
  - What can we do then?
- Refinement: Instead of the given set of FDs F, work with the 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

Finding The Minimal Cover

- F = {A→B, ABCD→E, EF→GH, ACDF→EG}
- Decomposition to have single attribute on right side
  - A→B, ABCD→E, EF→G, EF→H, ACDF→E, ACDF→G
- Check if any attribute on left side can be deleted without changing closure
  - A→B, A~~B~~CD→E, EF→G, EF→H, ACD~~F~~→E, ACDF→G
- Delete FDs that are implied by others
  - A→B, ACD→E, EF→G, EF→H, ~~ACD→E~~, ~~ACDF→G~~
    - ACDF→G from ACD→E, EF→G

Dependency-Preserving Decomposition into 3NF

- Using minimal cover F of given FD set, we can now achieve a lossless-join, dependency-preserving decomposition into 3NF.

1. Lossless-join decomposition until all smaller relations are in 3NF
2. For each FD X→A in F that is not preserved, add relation XA

- Result is lossless-join (X is superkey of XA) and dependency-preserving (obviously), but is it still in 3NF?
  - All relations after step 1 are in 3NF, but what about XA?
  - X→A is not a problem for 3NF because X is a superkey of XA
  - What if another FD on XA is a problem for 3NF?
    - Any FD on XA can only contain attributes from X∪{A}
    - If right-hand side of FD in F_{XA} contains A, left must be X (otherwise X→A would not have been in minimal cover)
    - If right-hand side does not contain A, it must be a subset of X, i.e., is a subset of a key
      - Why is X a key? It is a superkey, but is it minimal?
      - Yes: if X'⊂X was a key, then X→A would not have been in the minimal cover and X'→A would have been there
- Why not use the same algorithm for lossless-join, dependency – preserving decomposition into BCNF?

Update on DB Design Process

- Create ER diagram
- Translate ER diagram into set of relations
- Check relations for redundancy problems (not in 3NF, BCNF)
- Perform decomposition to fix problems
- Update ER diagram

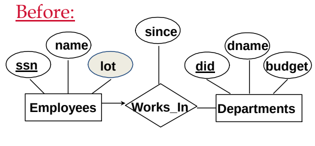
Refining Entity Sets

- Consider Hourly_Emps(ssn, name, lot, rating, hourly_wages, hours_worked)
  - FDs: S→SNLRWH and R→W
- Assume designer created entity set Hourly_Emps as above
  - Redundancy problem with R→W
  - Could not discover it in ER diagram (only shows primary key constraints)
- To fix redundancy problem, create new entity set Wage_Table(rating, hourly_wages)
  - Add relationship to connect Hourly_Emps2(S, N, L, H) and Wage_Table(R, W)
- Similar for refining of relationship sets (see book)

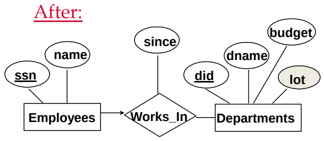
Identifying Entity Attributes

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

Before:



After:



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), consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.