Schema Refinement and Normal Forms

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?

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→SNLRWH
  - rating determines hrly_wages: R→W

Example (Contd.)

- Are the two smaller tables better?
- Problems in single "wide" table due to R→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, do we lose the information about the wage for rating 5.
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 \( Y \rightarrow X \).
  - 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 (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 \( X \rightarrow Y \) 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: \( SD \rightarrow P \)
  - \( JP \rightarrow C, C \rightarrow CSJDPQV \) imply \( JP \rightarrow CSJDPQV \)
  - \( SD \rightarrow P \) implies \( SDJ \rightarrow JP \)
  - \( SDJ \rightarrow JP, JP \rightarrow CSJDPQV \) imply \( SDJ \rightarrow CSJDPQV \)

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 \( X \rightarrow A \) in \( F^+ \)
  - \( A \subseteq X \) (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 \( X \rightarrow A \), consider two tuples with the same X value.
  - They should have the same A value. Redundancy?
  - No. Since X is in BCNF, X is a superkey and hence the “two” tuples must be identical.
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
    - 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.

Lossless Join Decompositions

- Decomposition of $R$ into $X$ and $Y$ is lossless-join w.r.t. $F$ if, for every instance $r$ that satisfies $F$:
  - $\pi_x(r) \cong \pi_y(r) = R$
  - It is always true that $R \subseteq \pi_x(r) \cong \pi_y(r)$
    - 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:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$
- Special case:
  - For FD $U \rightarrow V$, the decomposition of $R$ into $UV$ and $R-V$ is lossless-join.

Dependency-Preserving Decomposition

- Consider CSIDPOV, $C$ is key, $JP \rightarrow C$ and $SD \rightarrow P$.
  - BCNF decomposition: CSIDQV and SDP
  - Problem: Checking $JP \rightarrow C$ 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 $U \rightarrow V$ 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, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?
- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
  - And vice-versa. (Example?)
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., CSJDQV, 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.

### 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, ACDF → E, EF → GH

### 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 → GH, ACDF → E, ACDF → G
- Check if any attribute on left side can be deleted without changing closure
  - A → B, ABCD → E, EF → GH, ACDF → E, ACDF → G
- Delete FDs that are implied by others
  - A → B, ACDF → E, EF → GH, ACDF → E, ACDF → G
- ACDF → G from ACDF → E, EF → G

### 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., CSJDQV, 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.

### 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
  - Right hand side of FDs over XA contains X, left must be X (otherwise X → A would not have been minimal cover)
  - Right hand side does not contain X, A 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 minimal?
  - Not X, if X is 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_Emps(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:
  - Workers3(S,N,L,H)
  - Departments(D,M,B,L)

**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.