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?

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps
  - Hourly_Emps (emp_name, lot, rating, hrly_wages, hrs_worked)
- Notation: We will denote this relation schema by listing the attributes: SNLRRWH
  - This is really the set of attributes \{SNLRRWH\}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRRWH)
- Some FDs on Hourly_Emps:
  - emp_name is the key: \( S \rightarrow SNLRRWH \)
  - rating determines hrly_wages: \( R \rightarrow W \)

Example (Cont.)

- Problems due to \( R \rightarrow W \):
  - Undeletion anomaly: Can we change W in just the 1st tuple of SNLRRWH?
  - 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!

Refining an ER Diagram

- 1st diagram translated to Before:
  - 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)
  - DepLot(D,L)
  - Can fine-tune this:
  - Workers2(S,N,D,S)
  - Departments(D,M,BL)
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - sry → did, did → lot implies sry → lot
- An FD is implied by a set of FDs if it holds whenever all FDs in the set hold.
- F+ is 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 ⊆ Y, 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!

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.
- Does F = {A → B, B → C, D → E} imply A → E?
- i.e., is A → E in the closure F+? Equivalently, is E in A+?

Boyce-Codd Normal Form (BCNF)

- Rel 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 dependency in R that can be predicted using FDs alone.
- 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, the 2 tuples must be identical (since X is a key).

Third Normal Form (3NF)

- Rel 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 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).
- Lossless-join dependency-preserving decomposition of R into a collection of 3NF relations always possible.
what does 3nf achieve?

- If 3NF violates by X→A, one of the following holds:
  - X is a subset of some key K.
  - 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.
- But: even if 3NF is in 3NF, these problems could arise:
  - e.g., Reserve SNDC, S→C, C→S is in 3NF, but for each reservation of a sail, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

example decomposition

- Decompositions should be used only when needed.
  - SNLRWWB has FDs S→SNLRWWB 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 scheme.
  - i.e., we decompose SNLRW into SNLR and RW.
- The information to be stored consists of SNLRW tuples. If we store the projections of these tuples onto SNLR and RW, are there any potential problems that we should be aware of?

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:
    \[ \pi_X(r) \land \pi_Y(r) \neq r \]
  - It is always true that \( r \subseteq \pi_X(r) \land \pi_Y(r) \)
  - 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." (Aside Problem 2.)

problems with decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor 10 earn? (salary = WH)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation.
  - Fortunately, not in the SNLRWWB example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
  - Fortunately, not in the SNLRWWB example.
  - Tradeoff: Must consider these issues vs. redundancy.

more on lossless join

- The decomposition of R into X and Y is lossless-join w.r.t.
  if and only if the closure of F contains:
    - \( X \land Y \rightarrow X \), or
    - \( X \land Y \rightarrow Y \)
- In particular, the decomposition of R into UV and R - V is lossless-join
  if U \rightarrow V holds over R.
Dependency Preserving Decomposition

- Consider CS\(\text{JPQV}\), C is key, J\(\rightarrow\) C and S\(\rightarrow\) P.
  - BCNF decomposition: CS\(\text{JPQV}\) and SD\(\rightarrow\) P.
  - Problem: Checking J\(\rightarrow\) C requires a join.
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y, Z, and we enforce the FDS that hold on X, Y and Z, then all FDS that were given to hold on R must also hold. (Amos' Problem (3.2))
- Projection of set of FDS F: If R is decomposed into X, Y, Z, and we enforce the FDS that hold on X, Y and Z, then all FDS that were given to hold on R must also hold. (Amos' Problem (3.2))
  - Projection of F onto X (denoted \(F_X\)) is the set of FDS \(U \rightarrow V \in F^+(\text{closure of F})\) such that U, V are in X.

Decomposition into BCNF

- Consider relation R with FDS F. If X \(\rightarrow\) Y violates BCNF, decompose R into R - Y and X.
  - 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., CS\(\text{JPQV}\): key C, J\(\rightarrow\) C, S\(\rightarrow\) P, J \(\rightarrow\) S
  - To deal with S \(\rightarrow\) P, decompose into SD\(\rightarrow\) P, CS\(\text{JPQV}\).
  - To deal with J \(\rightarrow\) S, decompose CS\(\text{JPQV}\) into JS and CJ\(\text{JPQV}\).
- 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, \(Z \rightarrow\) Z, \(Z \rightarrow\) C
  - Can’t decompose while preserving 1st FD: not in BCNF.
- Similarly, decomposition of CSJ\(\text{DPQV}\) into SD\(\rightarrow\) P, JS and C\(\text{JPQV}\) is not dependency preserving (w.r.t. the FDS J\(\rightarrow\) C, S\(\rightarrow\) P and J \(\rightarrow\) S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - JPC tuples stored only for checking FDI: (Redundancy!)

Minimal Cover for a Set of FDS

- Minimal cover: G for a set of FDS F:
  - Closure of \(F = \text{closure of G}\).
  - 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., \(A \rightarrow B, ABD \rightarrow E, EF \rightarrow GH, AD \rightarrow EG\) has the following minimal cover:
    - \(A \rightarrow B, A \rightarrow D \rightarrow E, EF \rightarrow G\) and \(EF \rightarrow H\)
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.