The Evils of Redundancy

- Redundancy is at the root of several problems associated with relational schemas:
  - redundant storage
  - insert anomaly
  - delete anomaly
  - update anomaly
- Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.

Schema Refinement

- Main refinement technique: decomposition
  - E.g., replacing ABCD with AB and BCD, or ACD and ABD.
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation? Theory on normal forms.
  - What problems (if any) does the decomposition cause? Properties of decomposition include lossless-join and dependency-preserving.
  - Decomposition can cause performance problems.
### Functional Dependencies (FDs)

- A functional dependency \( X \rightarrow Y \) holds over relation \( R \) if for all allowable instance \( r \) of \( R \):
  - \( t_1 \in r, t_2 \in r, \pi_X(t_1) = \pi_X(t_2) \) implies \( \pi_Y(t_1) = \pi_Y(t_2) \), where \( X \) and \( Y \) are sets of attributes.
- An FD is a statement about all allowable relations.
  - Must be identified based on semantics of application.
  - Given an allowable instance \( r \) of \( R \), we can check if \( r \) 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 relation obtained from Hourly_Emps:
  - Hourly_Emps (seq, name, lot, rating, hrly_wages, hrs_worked)
- **Notation:** We will denote this relation schema by listing the attributes: \( \text{SNLRWH} \)
  - 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:
  - \( \text{seq} \) is the key: \( S \rightarrow \text{SNLRWH} \)
  - \( \text{rating} \) determines \( \text{hrly}_wages: R \rightarrow W \)

### Example (Contd.)

- Problems due to \( R \rightarrow W \):
  - **Redundant storage:** Can we change \( W \) in just the 1st tuple of \( \text{SNLRWH} \)?
  - **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 \( 5 \)!

Will 2 smaller tables be better?
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - Given \( ssn \to did, \ did \to lot \) implies \( ssn \to lot \)
- An FD \( f \) is implied by a set of FDs \( F \), if \( f \) holds for every retn instance that satisfies all FDs in \( F \).
- \( F^+ \) is the 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 \( XZ \toYZ \) for any \( Z \)
  - Transitivity: If \( X \to Y \) and \( Y \to Z \), then \( X \to Z \)

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from AA):
  - Union: If \( X \to Y \) and \( X \to Z \), then \( X \toYZ \)
  - Decomposition: If \( X \toYZ \), then \( X \to Y \) and \( X \to Z \)
- These are sound and complete inference rules for FDs!
  - Soundness: when applied to a set \( F \) of FDs, the axioms generate only FDs in \( F^+ \).
  - Completeness: repeated application of these axioms will generate all FDs in \( F^+ \).

Reasoning About FDs (Contd.)

- Computing the closure \( F^+ \) can be expensive: computes for all FD’s; size of closure is exponential in # attrs!
- Typically, we just want to check if a given FD \( X \to Y \) is in \( F^+ \).
  - An efficient check:
    - Compute attribute closure of \( X \) (denoted \( X^* \)) w.r.t. \( F \), i.e., the largest attribute set \( A \) such that \( X \to A \) is in \( F^+ \).
    - Simple algorithm: DO if there is \( U \to V \) in \( F \) s.t. \( U \subseteq X^* \), then \( X^* = X^*U \)
    - Check if \( Y \) is in \( X^* \).
  - Does \( F = \{ A \to B, B \to C, C \to D \to E \} \) imply \( A \to E \)?
    - i.e., is \( A \to E \) in the closure \( F^+ \)? Equivalently, is \( E \) in \( A^* \)?
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 redundancy related problems are avoided/minimized.

This helps us decide if decomposing the relation will help.

Role of FDs in detecting redundancy:
- Consider a relation R with 3 attributes, ABC.
- No FDs hold: There is no redundancy here.
- Given A \rightarrow B: Several tuples could have the same A value, and if so, they'll all have the same B value!

**Boyce-Codd Normal Form (BCNF)**

- Reln R with FDs F is in BCNF if \forall X \rightarrow A (X is a set of attributes, A is an attribute) in F⁺:
  - A \in X (called a trivial FD), or
  - X is a superkey (i.e., contains a key) for R.
- R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - Can we infer the value marked by '?'?
  - Is the relation in BCNF?
  - If a reln is in BCNF, every field of every tuple records a piece of information that can't be inferred (using only FD's) from values in other fields.

BCNF ensures that no redundancy can be detected using FDs!

**Third Normal Form (3NF)**

- Reln R with FDs F is in 3NF if \forall X \rightarrow A in F⁺:
  - A \in 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, obviously in 3NF.
- If R is in 3NF, some redundancy is possible.
  - E.g., Reserves (Sailor, Boat, Date, Credit_card) with S \rightarrow C, C \rightarrow S is in 3NF. But for each reservation of sailor S, same (S, C) pair is stored.
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible. (not true for BCNF)
What Does 3NF Achieve?

- If 3NF violated by $X \rightarrow A$, one of the following holds:
  - $X$ is a subset of some key $K$:
    - $X \rightarrow A$ is partial dependency.
    - We store $(X, A)$ pairs redundantly.
  - $X$ is not a proper subset of any key:
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, transitive dependency, where $A$ is not part of any key including $K$.
    - It means that we cannot associate an $X$ value with a $K$ value unless we also associate an $A$ value with an $X$ value.

Hierarchy of Normal Forms

- 1st normal form 1NF: no set-valued attributes.
- 2nd normal form 2NF: no partial dependencies.
- 3rd normal form 3NF: 2NF, and no transitive dependencies.
- Boyce-Codd normal form BCNF: 3NF, and no $X \rightarrow A$ s.t. $A$ is part of a key. No redundancy detected by FDs.
- 4th normal form 4NF: BCNF and no multi-valued dependencies (MVD). No redundancy detected by FDs and MVD.
  - We won’t discuss in detail in this class.

Decomposition of a Relation Scheme

- A decomposition of $R$ replaces $R$ 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 new relation.
- As a result, 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.
  - SNLWH has FDs S \rightarrow SNLWH and R \rightarrow W.
  - R \rightarrow W 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:
    - i.e., we decompose SNLWH into SNLRH and RW.
- If we just store the projections of SNLWH tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

- Three potential problems to consider:
  1. Some queries become more expensive.
     - e.g., How much did sailor Joe earn? (salary = W*H)
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
     - Fortunately, not in the SNLWH example.
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, not in the SNLWH example.
- Tradeoff: Must consider these issues vs. redundancy.

Lossless Join Decompositions

- Decomposition of R into X and Y is \textit{lossless-join} w.r.t. a set of FDs F if \forall 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 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

- The 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

- In particular, if U → V holds over R, the decomposition of R into UV and R - V is lossless-join.

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:
  - Intuitively, if R is decomposed into X and Y, and we enforce the FDs that hold on X and Y respectively, all FDs that were given to hold on R must also hold. (Aside: 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 closure F⁺ such that U, V are both in X.

Dependency Preserving Decompositions (Contd.)

- Formally, decomposition of R into X and Y is dependency preserving if (F[X] union F[Y])⁺ = 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., CSJDQV, 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

- 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, CS → Z, Z → C
  - Not in BCNF; can’t decompose while preserving 1st FD.
- 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).
  - However, it is a lossless join decomposition.
  - Adding JPC to the collection of relations gives a dependency preserving decomposition. JPC tuples stored only for checking FD! (Redundancy!)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decom into BCNF can be used to obtain a lossless join decom 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! e.g., consider the 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 $F = \text{closure of } G$.
  - 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., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$

has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ 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.

### Questions