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
❖ Consider the relation schema:

\[
\text{Lending-schema} = (\text{branch-name, branch-city, assets, customer-name, loan-number, amount})
\]

<table>
<thead>
<tr>
<th>branch-name</th>
<th>branch-city</th>
<th>assets</th>
<th>customer-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>900000</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>Palo Alto</td>
<td>210000</td>
<td>Smith</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>Horseneck</td>
<td>170000</td>
<td>Hayes</td>
<td>L-15</td>
<td>1500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>900000</td>
<td>Jackson</td>
<td>L-14</td>
<td>1500</td>
</tr>
</tbody>
</table>

❖ Redundancy:
- Data for branch-name, branch-city, assets are repeated for each loan that a branch makes
- Wastes space
- Complicates updating, introducing possibility of inconsistency of assets value
❖ Null values
- Cannot store information about a branch if no loans exist
- Can use null values, but they are difficult to handle.

Decomposition of a Relation Scheme
❖ Suppose that relation R contains attributes A1 \ldots 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 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., Can decompose SNLRWH into SNLRH and RW.
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 3 or more relations in a straightforward way.
- It is essential that all decompositions used to deal with redundancy be lossless!

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 cannot 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

- 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!
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - \( ssn \rightarrow did, \ did \rightarrow lot \) implies \( ssn \rightarrow lot \)
- An FD \( f \) is **implied by** a set of FDs \( F \) if \( f \) holds whenever all FDs in \( F \) hold.
  - \( F^* = \text{closure of } F \) is the set of all FDs that are implied by \( F \).
- Armstrong’s Axioms (\( X, Y, Z \) are sets of attributes):
  - **Reflexivity:** If \( Y \subseteq X \), then \( X \rightarrow Y \)
  - **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.)

- Couple of additional rules (that follow from 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: \( \text{Contracts}(cid,sid,jid,did,pid,qty,value) \), 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 \( SDJ \rightarrow CSJDPQV \)
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 $X \rightarrow Y$ 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 $X \rightarrow A$ is in $F^+$
    - There is a linear time algorithm to compute this.
  - Check if $Y$ is in $X^+$

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

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:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$

- In particular, the decomposition of $R$ into $UV$ and $R - V$ is lossless-join if $U \rightarrow V$ holds over $R$. 
Normalization Using Functional Dependencies

- When we decompose a relation schema $R$ with a set of functional dependencies $F$ into $R_1, R_2, \ldots, R_n$ we want
  - **Lossless-join decomposition**: Otherwise decomposition would result in information loss.
  - **No redundancy**: The relations $R_i$ preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
  - **Dependency preservation**: We will talk about it a little later.

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/minimized. This can be used to help us decide whether 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, for all $X \rightarrow A$ in $F^+$
  - $A \subseteq X$ (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.
  - 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).

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y₁</td>
<td>a</td>
</tr>
<tr>
<td>x</td>
<td>y₂</td>
<td>?</td>
</tr>
</tbody>
</table>

Decomposition into BCNF

- Consider relation R with FDs $F$. If $X \rightarrow Y$ 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 $\rightarrow$ C, SD $\rightarrow$ P, J $\rightarrow$ S
  - To deal with SD $\rightarrow$ P, decompose into SDP, CSJDQV.
  - To deal with J $\rightarrow$ 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!
**Example of BCNF Decomposition**

\[ R = (\text{branch-name, branch-city, assets, customer-name, loan-number, amount}) \]

\[ F = \{\text{branch-name} \rightarrow \text{assets branch-city} \]
\[ \quad \text{loan-number} \rightarrow \text{amount branch-name}\} \]

Key = \{\text{loan-number, customer-name}\}

Decomposition
- \( R_1 = (\text{branch-name, branch-city, assets}) \)
- \( R_2 = (\text{branch-name, customer-name, loan-number, amount}) \)
- \( R_3 = (\text{branch-name, loan-number, amount}) \)
- \( R_4 = (\text{customer-name, loan-number}) \)

Final decomposition \( R_1, R_3, R_4 \)

---

**Dependency Preserving Decomposition**

- Consider CSJDPQV, C is key, JP \( \rightarrow \) C and SD \( \rightarrow \) P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP \( \rightarrow \) C requires a join!

- **Dependency preserving decomposition:**
  - 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.
  - I.e., we should be able to check all functional dependencies on individual tables without doing joins
BCNF and Dependency Preservation

❖ In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, CS → Z, Z → C
  - Can’t decompose while preserving 1st FD; not in BCNF.
❖ Similarly, decomposition of CSJDPQV 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.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  ♦ JPC tuples stored only for checking FD! (Redundancy!)

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 in 3NF.
❖ If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no “good” decomp, or performance considerations).
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.
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.