Schema Refinement and
Normal Forms
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?

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 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: \( S \rightarrow SNLRWH \)
  - **rating** determines **hrly_wages**: \( R \rightarrow W \)

Example (Contd.)

- Problems 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 their 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

- 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)
  - Dept_Lots(D,L)
- Can fine-tune this:
  - Workers2(S,N,D,S)
  - Departments(D,M,B,L)

Before:

![ER Diagram Before]

After:

![ER Diagram After]

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 \( XZ \rightarrowYZ \) 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: Contracts($cid, sid, jid, did, pid, qty, value$), and:
  - $C$ is the key: $C \rightarrow CSJD$PQV
  - Project purchases each part using single contract: $JP \rightarrow C$
  - Dept purchases at most one part from a supplier: $SD \rightarrow P$

- $JP \rightarrow C$, $C \rightarrow CSJD$PQV imply $JP \rightarrow CSJD$PQV
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$
- $SDJ \rightarrow JP$, $JP \rightarrow CSJD$PQV imply $SDJ \rightarrow CSJD$PQV

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 $A^+$?
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 → 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 → 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).

<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>
Third Normal Form (3NF)

- Reln R with FDs $F$ is in 3NF if, for all $X \rightarrow A$ in $F^+$
  - $A \in X$ (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, or performance considerations).
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.

What Does 3NF Achieve?

- If 3NF violated by $X \rightarrow A$, one of the following holds:
  - $X$ is a subset of some key $K$
    - We store $(X, A)$ pairs redundantly.
  - $X$ is not a proper subset of any key.
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, 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 reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, $S \rightarrow C$, $C \rightarrow S$ is in 3NF, but for each reservation of sailor $S$, same $(S, C)$ pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.
Decomposition of a Relation Scheme

- Suppose that relation R contains attributes $A_1 ... A_n$. 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 at least 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 $RW$.

Example Decomposition

- Decompositions should be used only when needed.
  - $SNLRWH$ has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
  - 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 schema:
    - i.e., we decompose $SNLRWH$ into $SNLRH$ and $RW$

- The information to be stored consists of $SNLRWH$ tuples. If we just store the projections of these tuples onto $SNLRH$ and $RW$, are there any potential problems that we should be aware of?
Problems with Decompositions

- There are 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 in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    Fortunately, not 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. 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! (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:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$

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

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 (Intuitive):
  - 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. (*Avoids 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 $U \rightarrow V$ in $F^+$ (closure of $F$) such that $U, V$ 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 \(F^+, \text{not } F\), 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?)

** 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, CSJDPQV.
    - To deal with \(J \rightarrow S\), decompose CSJDPQV 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, CS→Z, Z→C
  - Can’t decompose while preserving 1st FD; not in BCNF.
- 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.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (Redundancy!)

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.