Limitations of E-R Designs

- Provides a set of guidelines, does not result in a unique database schema
- Does not provide a way of evaluating alternative schemas
- Normalization theory provides a mechanism for analyzing and refining the schema produced by an E-R design
Redundancy

- Dependencies between attributes cause redundancy
  - e.g., all addresses in the same town have the same zip code

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Town</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Joe</td>
<td></td>
<td>11790</td>
</tr>
<tr>
<td>4321</td>
<td>Mary</td>
<td>Stony Brook</td>
<td>11790</td>
</tr>
<tr>
<td>5454</td>
<td>Tom</td>
<td></td>
<td>11790</td>
</tr>
</tbody>
</table>

Redundancy and Other Problems

- Set valued attributes in the E-R diagram result in multiple rows in corresponding table
- Example: Person ($SSN$, $Name$, $Address$, $Hobbies$)
  - A person entity with multiple hobbies yields multiple rows in table Person
    - Hence, the association between Name and Address for the same person is stored redundantly
  - $SSN$ is key of entity set, but ($SSN$, $Hobby$) is key of corresponding relation
    - The relation Person can’t describe people without hobbies
Example

ER Model

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>{biking, hiking}</td>
</tr>
</tbody>
</table>

Relational Model

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>Joe</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Redundancy</td>
</tr>
</tbody>
</table>

Anomalies

- Redundancy leads to anomalies:
  - **Update anomaly**: A change in *Address* must be made in several places
  - **Deletion anomaly**: Suppose a person gives up all hobbies. Do we:
    - Set *Hobby* attribute to null? **No**, since *Hobby* is part of key
    - Delete the entire row? **No**, since we lose other information in the row
  - **Insertion anomaly**: *Hobby* value must be supplied for any inserted row since *Hobby* is part of key
Decomposition

- **Solution**: use two relations to store Person information
  - Person1 (SSN, Name, Address)
  - Hobbies (SSN, Hobby)
- The decomposition is more general: people with/without hobbies can now be described
- No update anomalies:
  - Name and address stored once
  - A hobby can be separately supplied or deleted

Normalization Theory

- Result of E-R analysis need further refinement
- Appropriate decomposition can solve problems
- The underlying theory is referred to as *normalization theory* and is based on *functional dependencies* (and other kinds, like *multivalued dependencies*)
Functional Dependencies

- Definition: A **functional dependency** (FD) on a relation schema R is a constraint $X \rightarrow Y$, where $X$ and $Y$ are subsets of attributes of R.

- Definition: An FD $X \rightarrow Y$ is satisfied in an instance $r$ of R, if for every pair of tuples, $t$ and $s$: if $t$ and $s$ agree on all attributes in $X$ then they must agree on all attributes in $Y$
  - Key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:
    - SSN $\rightarrow$ SSN, Name, Address

---

Functional Dependencies

- **Address $\rightarrow$ ZipCode**
  - Stony Brook’s ZIP is 11733

- **ArtistName $\rightarrow$ BirthYear**
  - Picasso was born in 1881

- **Autobrand $\rightarrow$ Manufacturer, Engine type**
  - Pontiac is built by General Motors with gasoline engine

- **Author, Title $\rightarrow$ PublDate**
  - Shakespeare’s Hamlet published in 1600
Functional Dependency - Example

- Consider a brokerage firm that allows multiple clients to share an account, but each account is managed from a single office and a client can have no more than one account in an office
- HasAccount (AcctNum, ClientId, OfficeId)
  - FDs:
    \[ \text{ClientId}, \text{OfficeId} \rightarrow \text{AcctNum} \]
    \[ \text{AcctNum} \rightarrow \text{OfficeId} \]
  - keys:
    \( (\text{ClientId}, \text{OfficeId}) \)
    \( (\text{AcctNum}, \text{ClientId}) \)
  - Thus, attribute values need not depend only on key values

Entailment, Closure, Equivalence

- **Definition**: If \( F \) is a set of FDs on schema \( R \) and \( f \) is another FD on \( R \), then \( F \) entails \( f \) if every instance \( r \) of \( R \) that satisfies every FD in \( F \) also satisfies \( f \)
  - Ex: \( F = \{ A \rightarrow B, B \rightarrow C \} \) and \( f \) is \( A \rightarrow C \)
    - If Town \( \rightarrow \) Zip and Zip \( \rightarrow \) AreaCode then Town \( \rightarrow \) AreaCode

- **Definition**: The closure of \( F \), denoted \( F^+ \), is the set of all FDs entailed by \( F \)

- **Definition**: \( F \) and \( G \) are equivalent if \( F \) entails \( G \) and \( G \) entails \( F \)
Entailment (cont’d)

- Satisfaction, entailment, and equivalence are **semantic** concepts – defined in terms of the actual relations in the “real world.”
  - They define *what these notions are*, not how to compute them.

- How to check if \( F \) entails \( f \) or if \( F \) and \( G \) are equivalent?
  - Apply the respective definitions for all possible relations?
    - *Bad idea*: might be infinite number for infinite domains
    - Even for finite domains, we have to look at relations of all arities
  - **Solution**: find algorithmic, **syntactic** ways to compute these notions
    - *Important*: The syntactic solution must be “correct” with respect to the semantic definitions
    - Correctness has two aspects: **soundness** and **completeness** – see later.

Armstrong’s Axioms for FDs

- This is the **syntactic** way of computing/testing the various properties of FDs

- **Reflexivity**: If \( Y \subseteq X \) then \( X \to Y \) (trivial FD)
  - *Name, Address* \( \to *Name*

- **Augmentation**: If \( X \to Y \) then \( XZ \to YZ \)
  - If *Town* \( \to *Zip* then *Town, Name* \( \to *Zip, Name*

- **Transitivity**: If \( X \to Y \) and \( Y \to Z \) then \( X \to Z \)

Soundness

- Axioms are sound: If an FD $f: X \rightarrow Y$ can be derived from a set of FDs $F$ using the axioms, then $f$ holds in every relation that satisfies every FD in $F$.
- Example: Given $X \rightarrow Y$ and $X \rightarrow Z$ then

\[
\begin{align*}
X \rightarrow XY & \quad \text{Augmentation by } X \\
YX \rightarrow YZ & \quad \text{Augmentation by } Y \\
X \rightarrow YZ & \quad \text{Transitivity}
\end{align*}
\]

- Thus, $X \rightarrow YZ$ is satisfied in every relation where both $X \rightarrow Y$ and $X \rightarrow Z$ are satisfied
- Therefore, we have derived the union rule for FDs: we can take the union of the RHSs of FDs that have the same LHS

Completeness

- Axioms are complete: If $F$ entails $f$, then $f$ can be derived from $F$ using the axioms
- A consequence of completeness is the following (naïve) algorithm to determining if $F$ entails $f$:

  - Algorithm: Use the axioms in all possible ways to generate $F^+$ (the set of possible FD’s is finite so this can be done) and see if $f$ is in $F^+$
Correctness

- The notions of soundness and completeness link the syntax (Armstrong’s axioms) with semantics (the definitions in terms of relational instances).

- This is a precise way of saying that the algorithm for entailment based on the axioms is “correct” with respect to the definitions.

Generating $F^*$

Thus, $AB \rightarrow BD$, $AB \rightarrow BCD$, $AB \rightarrow BCDE$, and $AB \rightarrow CDE$ are all elements of $F^*$.
Attribute Closure

- Calculating *attribute closure* leads to a more efficient way of checking entailment

- The *attribute closure* of a set of attributes, $X$, with respect to a set of functional dependencies, $F$, (denoted $X^+_F$) is the set of all attributes, $A$, such that $X \rightarrow A$
  - $X^+_F$ is not necessarily the same as $X^+_F$ if $F_1 \neq F_2$

- *Attribute closure and entailment*:
  - Algorithm: Given a set of FDs, $F$, then $X \rightarrow Y$ if and only if $X^+_F \supseteq Y$

Example - Computing Attribute Closure

<table>
<thead>
<tr>
<th>$F$: $AB \rightarrow C$</th>
<th>$X$</th>
<th>$X^+_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow D$</td>
<td>$A$</td>
<td>${A, D, E}$</td>
</tr>
<tr>
<td>$D \rightarrow E$</td>
<td>$AB$</td>
<td>${A, B, C, D, E}$</td>
</tr>
<tr>
<td>$AC \rightarrow B$</td>
<td>$B$</td>
<td>${B}$</td>
</tr>
<tr>
<td></td>
<td>$D$</td>
<td>${D, E}$</td>
</tr>
</tbody>
</table>

Is $AB \rightarrow E$ entailed by $F$? Yes
Is $D \rightarrow C$ entailed by $F$? No

*Result: $X^+_F$ allows us to determine FDs of the form $X \rightarrow Y$ entailed by $F$*
Computation of Attribute Closure $X^+_F$

```
closure := X; // since $X \subseteq X^+_F$
repeat
    old := closure;
    if there is an FD $Z \rightarrow V$ in $F$ such that
        $Z \subseteq closure$ and $V \notin closure$
    then closure := closure $\cup$ V
until old = closure

- If $T \subseteq closure$ then $X \rightarrow T$ is entailed by $F$
```

Example: Computation of Attribute Closure

- **Problem:** Compute the attribute closure of $AB$ with respect to the set of FDs:
  
  $AB \rightarrow C$ (a)
  $A \rightarrow D$ (b)
  $D \rightarrow E$ (c)
  $AC \rightarrow B$ (d)

- **Solution:**
  
  Initially $closure = \{AB\}$
  Using (a) $closure = \{ABC\}$
  Using (b) $closure = \{ABCD\}$
  Using (c) $closure = \{ABCDE\}$