Mapping ER Diagrams to Relations

- **Regular Entity Type**
  - Create a relation R
  - Include simple attributes and simple components of composite attributes
  - Choose one of the key attributes as the primary key
  - Multi-valued attributes:
    - Don’t include in R
    - To be discussed later.

- **Weak Entity Type E**
  - Include all simple attributes and simple components of composite attributes.
  - Include the primary key of the relation corresponding to the owner entity type of E.

**Exercise**

```
Employee
| SSN |
Dependent
| Name | Sex |
Dependent_of
| Host |
```

- **Binary 1:1 relationship R**
  - Suppose S and T are the relations corresponding to the entity types participating R
  - Choose either S (or T), and include the primary of T (or S) as foreign key.
  - Include the simple attributes and the simple components of composite attributes of R.
  - Better to choose the one with total participation in R.
Exercise

Mapping ER Diagrams to Relations (Cont’d)

- Binary 1:N relationship R
  - Suppose S and T are the relations corresponding to the entity types participating R, and S is the N-side.
  - Choose either S, and include the primary key of T as foreign key in S.
  - Include simple attributes and simple components of composite attributes in S.

Exercise

Mapping ER Diagrams to Relations (Cont’d)

- Multi-valued attribute A
  - Suppose A is an attribute of the entity type corresponding relation S.
  - Create a relation R
  - Include an attribute corresponding to A and the primary key K of S as a foreign key.
  - The primary key of R is the combination of A and K.
Exercise

Mapping EER Diagrams to Relations
- Subclass-superclass relationships:
  - Commonly four options
  - Assume there are a super-class C and m subclasses \( \{S_1, S_2, \ldots, S_m\} \).
  - Assume the attributes of C are \( \{k, a_1, \ldots, a_n\} \), and k is the key attribute.

Mapping EER Diagrams to Relations (Cont’d)
- The n-ary relationship R
  - Create a new relation S
  - Include the primary keys of all the relations corresponding to the participating entity types in R. They form the primary key of S
  - Include the simple attributes and the simple components of composite attributes of R.

Exercise

Mapping EER Diagrams to Relations (Cont’d)
- Option 1
  - Create a relation L for C with all its attributes, and have k as the primary key.
  - For each subclass Si, create a relation Li with attributes k and all the attributes of Si. The primary key of Si is k.
  - Intuition: keep the attributes of the individual classes separately.

Mapping EER Diagrams to Relations (Cont’d)
- Option 2
  - For each subclass Si, create a relation Li with all the attributes of C and all the attributes of Si. The primary key of Si is k.
  - Intuition: replicate the attributes of the super-class in subclasses.
Mapping EER Diagrams to Relations (Cont’d)

Option 3
- Create a single relation with the attributes of the super-class and all the subclasses plus a type attribute.
- The type attribute is used to indicate the subclass to which each tuple belongs.
- Intuition:
  - store all classes together.
  - For disjoint specialization.

Option 4
- Create a single relation with the attributes of the super-class and all the subclasses plus m type attributes.
- The type attributes boolean attributes indicating whether the tuple belongs to the corresponding subclasses.
- Intuition:
  - Store all classes together.
  - For overlapping specialization.

Mapping EER Diagrams to Relations (Cont’d)

Multiple inheritance:
- all superclasses have the same key

Design Guidelines

- Have schemas that are easy to explain.
- Keep different entities and relationships apart where possible—at least in base relations
- Prevent anomalies in
  - insertion
  - deletion
  - modification

Exercise

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Bdate</th>
<th>DNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>111-22-3333</td>
<td>01/11/71</td>
<td>1</td>
</tr>
<tr>
<td>Tom</td>
<td>222-33-4444</td>
<td>02/14/68</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department</th>
<th>DNumbren</th>
<th>Mgr/SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>1</td>
<td>111-22-3333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Bdate</th>
<th>DNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>111-22-3333</td>
<td>01/11/71</td>
<td>1</td>
</tr>
<tr>
<td>Tom</td>
<td>222-33-4444</td>
<td>02/14/68</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department</th>
<th>DNumbren</th>
<th>Mgr/SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>1</td>
<td>111-22-3333</td>
</tr>
</tbody>
</table>

EMPLOYEE

111-22-3333

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Bdate</th>
<th>DNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>111-22-3333</td>
<td>01/11/71</td>
<td>1</td>
</tr>
<tr>
<td>Tom</td>
<td>222-33-4444</td>
<td>02/14/68</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department</th>
<th>DNumbren</th>
<th>Mgr/SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>1</td>
<td>111-22-3333</td>
</tr>
</tbody>
</table>
Design Guidelines (Cont’d)

- Avoid NULL values in base relations, although they may occur in views. NULLs should apply rarely and have well-defined meaning:
  - Not applicable
  - unknown
  - absent (known but absent)
- Prevent spurious tuples

Registration

<table>
<thead>
<tr>
<th>StudentID</th>
<th>Name</th>
<th>Course</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>CSC101</td>
<td>V 150</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>CSC101</td>
<td>V 150</td>
</tr>
</tbody>
</table>

Reasoning With FDs: 1

FDs can be inferred from other FDs.

- Let F be a set of FDs
- X → Y is inferred from F if X → Y holds in every relation r that satisfies F
  - (notation: F |= X → Y)
- F+ is the set of all FDs that can be inferred from F
  - Called the closure of F.

Functional Dependencies

- A constraint
- R is treated as a set of attributes below
  - For subsets X and Y of R, X → Y means that
    - For all relations r of R, (for all t1, t2: t1, t2 in r
      ⇒ (t1[X] = t2[X] ⇒ t1[Y] = t2[Y]))
- FDs depend on R and its meaning, not on r.

Reasoning With FDs: 2

FDs can be inferred based on

- Their formal definition
- Armstrong’s rules:
  - Reflexivity: If X contains Y, then X → Y
  - Augmentation: X → Y | = XZ → YZ
  - Transitivity: {X → Y, Y → Z} | = X → Z
- which are provably complete.
Reasoning With FDs: 3

Additional Inference Rules:
- Decomposition rule
  \[ X \rightarrow YZ \Rightarrow X \rightarrow Y \]
- Union rule
  \[ X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ \]
- Pseudo-transitive rule
  \[ X \rightarrow Y, WY \rightarrow Z \Rightarrow WX \rightarrow Z \]

Minimal Cover (Cont’d)

- A minimal cover of a set F of FDs is a minimal set of FDs that is equivalent to F.
- Always exist.
- Algorithm 14.2 in textbook
  - Step 1.
  - Step 2. Change the FDs to those with one attribute on the right-hand side.
  - Step 3. Try to remove attributes from the left-hand sides of FDs.
  - Step 4. Try to remove redundant FDs.

Find Additional FDs

- Given a set F of FDs,
  - For each set of attributes X that appears as a left-hand side of an FD in F,
  - Determine the set \( X^+ \) of attributes that are functionally determined by X based on F.
  - \( X^+ \): the closure of X under F.
  - Algorithm to compute \( X^+ \) under F:
    - \( X^0 \leftarrow X \)
    - Repeat
      - Old \( X^+ \leftarrow X^+ \)
      - For each FD \( Y \rightarrow Z \) in F do
        - If \( X^+ \supseteq Y \) then \( X^+ = X^+ \cup Z \)
      - Until \( X^+ = \text{old}X^+ \)

Minimal Cover

- Equivalence of sets of FDs
  - Two sets of functional dependencies E and F are equivalent if \( E^+ = F^+ \).
  - A set F of FDs is minimal iff
    - Every FD in F has a single attribute for the right-hand side.
    - We cannot replace any \( X \rightarrow A \) with \( Y \rightarrow A \), where \( Y \subseteq X \), and still have a set of FDs equivalent to F.
    - We cannot remove any FD from F and still have a set of FDs equivalent to F.

Normalization

- A process of cleaning up a schema by decomposing the relations in it
  - to remove various anomalies
  - but additional considerations apply
  - A schema is in some normal form if it satisfies the specified mathematical properties and thereby avoids some potential anomalies

Normalization (Cont’d)

- Algorithm 14.2 in textbook
  - Step 1.
  - Step 2. Change the FDs to those with one attribute on the right-hand side.
  - Step 3. Try to remove attributes from the left-hand sides of FDs.
  - Step 4. Try to remove redundant FDs.

Find Additional FDs

- Given a set F of FDs,
  - For each set of attributes X that appears as a left-hand side of an FD in F,
  - Determine the set \( X^+ \) of attributes that are functionally determined by X based on F.
  - \( X^+ \): the closure of X under F.
  - Algorithm to compute \( X^+ \) under F:
    - \( X^0 \leftarrow X \)
    - Repeat
      - Old \( X^+ \leftarrow X^+ \)
      - For each FD \( Y \rightarrow Z \) in F do
        - If \( X^+ \supseteq Y \) then \( X^+ = X^+ \cup Z \)
      - Until \( X^+ = \text{old}X^+ \)

Minimal Cover

- Equivalence of sets of FDs
  - Two sets of functional dependencies E and F are equivalent if \( E^+ = F^+ \).
  - A set F of FDs is minimal iff
    - Every FD in F has a single attribute for the right-hand side.
    - We cannot replace any \( X \rightarrow A \) with \( Y \rightarrow A \), where \( Y \subseteq X \), and still have a set of FDs equivalent to F.
    - We cannot remove any FD from F and still have a set of FDs equivalent to F.

Find Additional FDs

- Given a set F of FDs,
  - For each set of attributes X that appears as a left-hand side of an FD in F,
  - Determine the set \( X^+ \) of attributes that are functionally determined by X based on F.
  - \( X^+ \): the closure of X under F.
  - Algorithm to compute \( X^+ \) under F:
    - \( X^0 \leftarrow X \)
    - Repeat
      - Old \( X^+ \leftarrow X^+ \)
      - For each FD \( Y \rightarrow Z \) in F do
        - If \( X^+ \supseteq Y \) then \( X^+ = X^+ \cup Z \)
      - Until \( X^+ = \text{old}X^+ \)

Minimal Cover

- Equivalence of sets of FDs
  - Two sets of functional dependencies E and F are equivalent if \( E^+ = F^+ \).
  - A set F of FDs is minimal iff
    - Every FD in F has a single attribute for the right-hand side.
    - We cannot replace any \( X \rightarrow A \) with \( Y \rightarrow A \), where \( Y \subseteq X \), and still have a set of FDs equivalent to F.
    - We cannot remove any FD from F and still have a set of FDs equivalent to F.

Find Additional FDs

- Given a set F of FDs,
  - For each set of attributes X that appears as a left-hand side of an FD in F,
  - Determine the set \( X^+ \) of attributes that are functionally determined by X based on F.
  - \( X^+ \): the closure of X under F.
  - Algorithm to compute \( X^+ \) under F:
    - \( X^0 \leftarrow X \)
    - Repeat
      - Old \( X^+ \leftarrow X^+ \)
      - For each FD \( Y \rightarrow Z \) in F do
        - If \( X^+ \supseteq Y \) then \( X^+ = X^+ \cup Z \)
      - Until \( X^+ = \text{old}X^+ \)

Find Additional FDs

- Given a set F of FDs,
  - For each set of attributes X that appears as a left-hand side of an FD in F,
  - Determine the set \( X^+ \) of attributes that are functionally determined by X based on F.
  - \( X^+ \): the closure of X under F.
  - Algorithm to compute \( X^+ \) under F:
    - \( X^0 \leftarrow X \)
    - Repeat
      - Old \( X^+ \leftarrow X^+ \)
      - For each FD \( Y \rightarrow Z \) in F do
        - If \( X^+ \supseteq Y \) then \( X^+ = X^+ \cup Z \)
      - Until \( X^+ = \text{old}X^+ \)

Keys

- S is a superkey of \( R = \{ A_1, ..., A_n \} \) iff
  - \( R \) contains \( S \)
  - (forall \( t_1, t_2: t_1[S] = t_2[S] \Rightarrow ?) \)
- K is a (candidate) key or identifier of \( R \) iff
  - K is a superkey
  - (forall \( L: K \) contains \( L \Rightarrow ?) \)
- The primary key is one of the candidate keys.
Achtung!
Prime attribute
■ member of any key
■ not just the primary key

Full FD
■ X → Y is a full FD if
  ● (forall W: (X contains W & W → Y) ⇒ X = W)
■ X → Y is a partial FD, otherwise

1NF
■ Attributes must be atomic:
  ● they can be chars, ints, strings
  ● they can’t be
    ■ tuples
    ■ sets
    ■ relations
    ■ composite
    ■ multivalued

2NF
■ R is in 2NF if every nonprime attribute is fully functionally dependent on every key of R
■ Note that every attribute must be functionally dependent on every key (by definition of a key)

Obtaining 1NF
1NF is obtained by
■ Splitting composite attributes
■ splitting the relation and propagating the primary key to remove multivalued attributes

Obtaining 2NF
■ If a nonprime attribute is dependent only on a proper part of a key, then we take the given attribute as well as the key attributes that determine it and move them all to a new relation
■ We can bundle all attributes determined by the same subset of the key as a unit
Exercise

3NF: Alternative Definition
- R is in 3NF if every nonprime attribute of R
  - fully functionally dependent on every key of R, and
  - nontransitively dependent on every key of R.

Obtaining 3NF
- Split off the attributes in the FD that causes trouble and move them, so there are two relations for each such FD
- The determinant of the FD remains in the original relation

Transitive Dependency
- X $\rightarrow$ Y is a transitive dependency if
  - (exists Z: Z is not contained in any key of R & X $\rightarrow$ Z and Z $\rightarrow$ Y)

Exercise
**BCNF**

R is in Boyce-Codd Normal Form iff
- if $X \rightarrow A$ then
  - $X$ is a superkey of $R$
- more restrictive than 3NF
  - preferable—has fewer anomalies

---

**Universal Relation: 1**

- A universal relation is a single giant relation containing the entire database
  - has all attributes (renamed to be unique)
  - has enough tuples with NULL values as appropriate
  - not used in practice
  - only a theoretical construct!

---

**Obtaining BCNF**

- As usual, split the schema to move the attributes of the troublesome FD to another relation, leaving its determinant in the original so they remain connected
  - not always attainable

---

**Universal Relation: 2**

- One way to think about DB design is to imagine that we begin with the universal relation and normalize the schema to whatever level we like
  - theoretically interesting
  - partially usable
  - should not be the only tool in one’s DB design

---

**Exercise**

```
Student   Course   Instructor
---------   -------   --------
        |        |          
        |        |
```