Outline

• ER diagrams (Ch 2)
• ER diagrams to relational tables (Ch 3.5)
• Schema Refinement, Normalization (Ch 19)
Relational Schema Design

Conceptual Model:

Relational Model: plus Integrity Constraints

Normalization: Eliminates anomalies
Entity / Relationship Diagrams

Attributes

Entity sets

Relationships

Product

address

buys
Keys in E/R Diagrams

- Every entity set must have a key
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many
What does this say?
Multi-way Relationships

- Product
- Purchase
- Person
- Store
Arrows in Multiway Relationships

Q: what does the arrow mean?

A: if I know the store, person, invoice, I know the movie too
Arrows in Multiway Relationships

Q: what do these arrow mean?

A: store, person, invoice determines movie and store, invoice, movie determines person
Roles in Relationships

What if we need an entity set twice in one relationship?

Product

Purchase

Store

Person

salesperson

buyer
Attributes on Relationships

- **Product**
- **Person**
- **Store**
- **Purchase**

Additional attribute: **date**
Design Principles

What’s wrong?

Product \rightarrow Purchase \rightarrow Person

Country \rightarrow President \rightarrow Person

Moral: be faithful!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Design Principles: What’s Wrong?

Moral: don’t complicate life unnecessarily.
From E/R Diagrams to Relational Schema

• Entity set $\rightarrow$ relation
• Relationship $\rightarrow$ relation
Entity Set to Relation

**Product**

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>$19.99</td>
</tr>
</tbody>
</table>
Relationships to Relations

Makes(product-name, product-category, company-name, year)

(watch out for attribute name conflicts)
No need for **Makes**. Modify **Product**:

**Product**(name, category, price, startYear, companyName)
Multi-way Relationships to Relations

Product
- name
- price

Purchase

Person
- ssn
- name

Store
- name
- address

Purchase(prodName, stName, ssn)
Subclasses to Relations

Notice: subclass = subset
Alternative: disjoint classes (Java, C++)
Referential Integrity Constraints

Each product made by at most one company.
(Some products made by no company)

Each product made by *exactly* one company.
Other Constraints

What does this mean?
Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.

University(name)
Team(\textit{universityName}, number, sport)
Relational Schema Design

Conceptual Model:

Relational Model: plus Integrity Constraints

Normalization: Eliminates anomalies
Schema Refinement
Integrity Constraints

• Limitation on feasible data instances
  – Functional dependency
  – Key Constraint
  – Foreign key constraint

• An relation instance can violate an IC

• But satisfaction of an IC can never be determined by inspecting a relation instance.
Evils of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:

  - **Redundant storage**: data is repeated
  - **Update anomalies**: need to change in several places
  - **Insertion anomalies**: may not be able to add data we want to
  - **Deletion anomalies**: may lose data when we don’t want to
Schema Refinement

• 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?
First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat
More Normal Forms

• Based on Functional Dependencies
  – 2nd Normal Form (obsolete)
  – 3rd Normal Form
  – Boyce Codd Normal Form (BCNF)

• Based on Multivalued Dependencies
  – 4th Normal Form

• Based on Join Dependencies
  – 5th Normal Form
Functional Dependencies

• A kind of integrity constraint
  – (hence, part of the schema)
• Finding them is part of the database design

Recall that a function returns a single unique value when evaluated on any input.
Functional Dependencies

Table $R(\ldots, A_1, A_2, \ldots, A_n, \ldots, B_1, B_2, \ldots, B_m, \ldots)$

Functional Dependency:

$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Meaning:

If two tuples agree on the attributes $A_1, A_2, \ldots, A_n$, then they must also agree on the attributes $B_1, B_2, \ldots, B_m$.
Functional Dependencies

**Definition:** $A_1, ..., A_n \rightarrow B_1, ..., B_m$ holds in $R$ if:

$$\forall t, t' \in R, (t.A_1 = t'.A_1 \land ... \land t.A_n = t'.A_n \Rightarrow t.B_1 = t'.B_1 \land ... \land t.B_m = t'.B_m)$$

<table>
<thead>
<tr>
<th>R</th>
<th>$A_1$</th>
<th>$...$</th>
<th>$A_n$</th>
<th>$B_1$</th>
<th>$...$</th>
<th>$B_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if $t$, $t'$ agree here

then $t$, $t'$ agree here
Examples

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E1847</td>
<td>John</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

- **EmpID** $\rightarrow$ Name, Phone, Position
- **Position** $\rightarrow$ Phone
- **but** Phone $\nRightarrow$ Position
Example

Product(name, category, color, department, price)

Consider these FDs:

name $\rightarrow$ color

category $\rightarrow$ department

color, category $\rightarrow$ price
Example

FD’s are constraints:
- On some instances they hold
- On others they don’t

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Does this instance satisfy all the FDs?
### Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supply</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
Reasoning about FD’s

If some FDs are satisfied, then others are satisfied too

If all these FDs are true:

- name → color
- category → department
- color, category → price

Then this FD also holds:

- name, category → price

Why ?? We say that the new FD is implied
Closure of a Set of FDs

Definition. Given a set F of functional dependencies, the closure, $F^+$, denotes all FDs implied by F.

How can we compute $F^+$?
Computing the Closure $F^+$

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 $X \rightarrow Z$

**Theorem.** Armstrong axioms are *sound* and *complete* for computing $F^+$

What do *sound* and *complete* mean?
Convenient rule

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]
Example (continued)

From:
1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

To:
name, category $\rightarrow$ price

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category $\rightarrow$ name</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>5. name, category $\rightarrow$ color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category $\rightarrow$ category</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>7. name, category $\rightarrow$ color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category $\rightarrow$ price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>
Problem: Compute $F^+$

Given $F$ compute its closure $F^+$.

How to proceed?

- Apply Armstrong’s Axioms repeatedly
- Better: use the *Closure Algorithm* for a set of attributes (next)
Closure of a set of Attributes

**Given** a set of attributes $A_1, \ldots, A_n$

The closure, $\{A_1, \ldots, A_n\}^+$, with respect to $F$, is the set of attributes $B$ s.t. $A_1, \ldots, A_n \rightarrow B$

Example: $F=$

- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

Closures, with respect to $F$:
- $name^+ = \{name, color\}$
- $\{name, category\}^+ = \{name, category, color, department, price\}$
- $color^+ = \{color\}$
Closure Algorithm (for Attributes)

Start with $X=\{A_1, \ldots, A_n\}$.

Repeat until $X$ doesn’t change do:

- if $B_1, \ldots, B_n \rightarrow C$ is a FD and $B_1, \ldots, B_n$ are all in $X$
  - then add $C$ to $X$.

Example:

- $\text{name} \rightarrow \text{color}$
- $\text{category} \rightarrow \text{department}$
- $\text{color, category} \rightarrow \text{price}$

\[ \{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\} \]
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[ F= \begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array} \]

Compute \( \{A,B\}^+ \)  \[ X = \{A, B, C, D, E\} \]

Compute \( \{A, F\}^+ \)  \[ X = \{A, F, B, D, C, E\} \]
Closure Algorithm (for FDs)

Example:

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow B \\
B & \rightarrow D
\end{align*}
\]

Step 1: Compute \(X^+\), for every \(X\):

\[
\begin{align*}
A^+ &= A, \quad B^+ = BD, \quad C^+ = C, \quad D^+ = D \\
AB^+ &= ABCD, \quad AC^+ = AC, \quad AD^+ = ABCD \\
ABC^+ &= ABD^+ = ACD^+ = ABCD \text{ (no need to compute– why?)} \\
BCD^+ &= BCD, \quad ABCD^+ = ABCD
\end{align*}
\]

Step 2: Enumerate all FD’s \(X \rightarrow Y\), s.t. \(Y \subseteq X^+\) and \(X \cap Y = \emptyset\):

\[
\begin{align*}
AB \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B
\end{align*}
\]
Keys

• A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. $A_1, \ldots, A_n \rightarrow B$ for all attributes $B$

• A **key** is a minimal superkey
Computing Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a superkey
- Consider only the minimal superkeys

Note: there can be exponentially many keys!

- Example: $R(A,B,C)$, $AB \rightarrow C$, $BC \rightarrow A$
  Keys: $AB$ and $BC$
Examples of Keys

- **Product(name, price, category, color)**
  - name, category $\rightarrow$ price
  - category $\rightarrow$ color

  Key:  \{name, category\}  
  Superkeys: supersets

- **Enrollment(student, address, course, room, time)**
  - student $\rightarrow$ address
  - room, time $\rightarrow$ course
  - student, course $\rightarrow$ room, time

  Keys are:  [in class]
Data Anomalies

When a database is poorly designed we get anomalies:

**Redundancy**: data is repeated

**Update anomalies**: need to change in several places

**Delete anomalies**: may lose data when we don’t want to

**Schema refinement** means removing the data anomalies.
Data Anomalies

Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN $\rightarrow$ Name, City but not SSN $\rightarrow$ PhoneNumber

Anomalies:

• Redundancy = repeat data
• Update anomalies = Fred moves to “Bellevue”
• Deletion anomalies = Joe deletes his phone number: what is his city?
Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:
- No more repeated data
- Easy to move Fred to “Bellevue” (how ?)
- Easy to delete all Joe’s phone number (how ?)
Decompositions in General

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

- \( R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \)
- \( R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \)

\( R_1 = \) projection of \( R \) on \( A_1, \ldots, A_n, B_1, \ldots, B_m \)

\( R_2 = \) projection of \( R \) on \( A_1, \ldots, A_n, C_1, \ldots, C_p \)
Problems With Decomposition

• Can we get the data back correctly?
  – Lossless decomposition
  – Discuss next

• Can we recover the FD’s on the ‘big’ table from the FD’s on the small tables?
  – Dependency-preserving decomposition
  – Please read 19.5.2
Lossless Decomposition

• Sometimes it is correct:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Lossy Decomposition

• Sometimes it is not:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

What's wrong??
Decompositions in General

Theorem
If \( A_1, ..., A_n \rightarrow B_1, ..., B_m \)
Then the decomposition is lossless

Note: don’t need necessarily \( A_1, ..., A_n \rightarrow C_1, ..., C_p \)

Example: name \( \rightarrow \) price, hence the first decomposition is lossless
Normal Forms

• Decomposition into Boyce Codd Normal Form (BCNF)
  – Lossless

• Decomposition into 3rd Normal Form
  – Lossless
  – Dependency preserving
Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if:

If $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency in R,
then $\{A_1, \ldots, A_n\}$ is a superkey for R

Equivalently: for any set of attributes X,
either $X^+ = X$
or $X^+ = \text{all attributes}$
BCNF Decomposition Algorithm

**Repeat**
choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates the BNCF condition
split $R$ into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, \text{[rest]})$
continue with both $R_1$ and $R_2$

**Until** no more violations

Heuristics:
choose $B_1, \ldots, B_n$
“as large as possible”

Note: need to compute the FDs on $R_1, R_2$ (how?)
BCNF Example

FD: $\text{SSN} \rightarrow \text{Name, City}$
Key: $\{\text{SSN, PhoneNumber}\}$
Is it in BCNF?

Another way: $\text{SSN}^+ = \{\text{SSN, Name, City}\}$ but no PhoneNumber
BCNF Example

Let’s check anomalies:
- Redundancy?
- Update?
- Delete?

### Name, SSN, City

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

### SSN, PhoneNumber

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-1234</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-6543</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-2121</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-1234</td>
</tr>
</tbody>
</table>
Example

• $R(A,B,C,D)$ $A \rightarrow B$, $B \rightarrow C$

• Key: $AD$

• Violations of BCNF:
  $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow BC$, $B \rightarrow C$

• Pick $A \rightarrow BC$ first: split into $R_1(A,B,C)$ $R_2(A,D)$

• In $R_1$ : $B \rightarrow C$; split into $R_{11}(A,B)$, $R_{12}(B,C)$
Example (cont’d)

- $R(A,B,C,D)$ $A \rightarrow B$, $B \rightarrow C$

- Order matters!
- Pick $A \rightarrow C$ first: $R_1(A,C)$, $R_2(A,B,D)$
- In $R_2$: $A \rightarrow B$; decompose into $R_{21}(A,B)$, $R_{22}(A,D)$

- Final answer: $R_1(A,C)$, $R_{21}(A,B)$, $R_{22}(A,D)$

- Which one is better?
BCNF and Dependencies

FD’s: Unit $\rightarrow$ Company; Company, Product $\rightarrow$ Unit
So, there is a BCNF violation, and we decompose.

In BCNF we lose the FD: Company, Product $\rightarrow$ Unit
Solution: 3rd Normal Form (3NF)

A simple condition for removing anomalies from relations:

A relation R is in 3rd normal form if:

Whenever there is a nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for R, then \{ $A_1, A_2, ..., A_n$ \} a super-key for R, or B is part of a key.

Please read in the book !!!
3NF Discussion

• 3NF decomposition v.s. BCNF decomposition:
  – Use same decomposition steps, for a while
  – 3NF may stop decomposing, while BCNF continues

• Tradeoffs
  – BCNF = no anomalies, but may lose some FDs
  – 3NF = keeps all FDs, but may have some anomalies