Goal

- **Question**: The relational model is great, but how do I go about designing my database schema?

Outline

- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization

Database Design Process

1. Data Modeling
2. Refinement
3. SQL Tables
4. Files

- E/R Diagrams
- Relations
- Conceptual Schema
- Physical Schema
Conceptual Schema Design

- Conceptual Model

- Relational Model + Functional Dependencies (FDs)

- Normalization Eliminates Anomalies

Entity/Relationship Diagrams

- Attributes

- Entity Sets

- Relationship Sets

Example E/R Diagram

Resulting Relations

- One way to translate diagram into relations:

  PatientOf (pno, name, zip, dno, since)

  Doctor (dno, dname, specialty)
Entity/Relationship Model

- Typically, each entity has a key
- E/R relationships can include multiplicity
  - One-to-one, one-to-many, etc.
  - Indicated with arrows
- Can model multi-way relationships
- Can model subclasses
- And more...

Example with Inheritance

Example from Phil Bernstein’s SIGMOD’07 keynote talk

Converting Into Relations

- One way to translate our E/R diagram into relations:
  - HR (id, name)
  - Empl (id, dept) id is also a foreign key referencing HR
  - Client (id, name, credit_score, billing_addr)

- Today, we only talk about using E/R diagrams to help us design the conceptual schema of a database
- In general, apps may need to operate on a view of the data closer to E/R model (e.g., OO view of data) while DB contains relations
  - Need to translate between objects and relations
  - Object-Relational Mapping (ORM)
  - Hibernate, Microsoft ADO.NET Entity Framework, etc.

Outline

- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization
Problematic Designs

• Some DB designs lead to **redundancy**
  - Same information stored multiple times

• Problems:
  - **Redundant Storage**
  - **Update Anomalies**
  - **Insertion Anomalies**
  - **Deletion Anomalies**

Problem Examples

• What if we want to insert a patient without a doctor?
• What if we want to delete the last doctor for a patient?
• **Illegal as (pno,dno) is the primary key, cannot have nulls**

Solution: Decomposition

• Decomposition solves the problem, but need to be careful...

Lossy Decomposition

• **Decomposition can cause us to lose information!**
Schema Refinement Challenges

- How do we know that we should decompose a relation?
  - Functional dependencies
  - Normal forms
- How do we make sure decomposition does not lose any information?
  - Lossless-join decompositions
  - Dependency-preserving decompositions

Outline

- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization

Functional Dependency

- A functional dependency (FD) is an integrity constraint that generalizes the concept of a key
- An instance of relation R satisfies the FD: $X \rightarrow Y$
  - if for every pair of tuples $t_1$ and $t_2$
  - if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$
  - where $X, Y$ are two nonempty sets of attributes in R
- We say that $X$ determines $Y$
- FDs come from domain knowledge

FD Illustration

The FD $A_1, ..., A_m \rightarrow B_1, ..., B_n$ holds in $R$ if:

$$\forall t, t' \in R, (t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \implies t.B_1 = t'.B_1 \land ... \land t.B_n = t'.B_n$$

![FD Illustration Diagram](image-url)
**FD Example**

- An FD **holds**, or **does not hold** on an instance:

<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>E3542</td>
<td>Mike</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 → Name, Phone, Position
- Position → Phone
- but not Phone → Position

**FD Terminology**

- FDs are constraints
  - On some instances they hold
  - On others they do not
- If for every instance of R a given FD will hold, then we say that R satisfies the FD
  - If we say that R satisfies an FD, we are stating a constraint on R
- FDs come from domain knowledge

**An Interesting Observation**

- If all these FDs are true:
  - \( \text{name} \rightarrow \text{color} \)
  - \( \text{category} \rightarrow \text{department} \)
  - \( \text{color}, \text{category} \rightarrow \text{price} \)
- Then this FD also holds: \( \text{name}, \text{category} \rightarrow \text{price} \)
- Why ???

**How Is This All Useful?**

- Anomalies occur when certain “bad” FDs hold
- We know some of the FDs
- Need to find **all** FDs
- Then look for the bad ones
Closure of FDs

- Some FDs imply others
  - For example: Employee(ssn,position,salary)
  - FD1: ssn → position and FD2: position → salary
  - Imply FD3: ssn → salary
- Can compute closure of a set of FDs
  - Set $F^+$ of all FDs implied by a given set $F$ of FDs
- Armstrong’s Axioms: sound and complete
  - 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 $X \rightarrow Z$
- Convenient split/combine rule:
  If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

Example (cont’d)

- Starting from these FDs:
  1. $name \rightarrow color$
  2. $category \rightarrow department$
  3. $color$, $category \rightarrow price$

- Infer the following FDs:
<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>Reflexivity</td>
</tr>
<tr>
<td>5. $name$, $category \rightarrow color$</td>
<td>Transitivity on 4 and 1</td>
</tr>
<tr>
<td>6. $name$, $category \rightarrow category$</td>
<td>Reflexivity</td>
</tr>
<tr>
<td>7. $name$, $category \rightarrow color$, $category$</td>
<td>Split/Combine on 5 and 6</td>
</tr>
<tr>
<td>8. $name$, $category \rightarrow price$</td>
<td>Transitivity on 7 and 3</td>
</tr>
</tbody>
</table>

- TOO HARD! Let’s see an easier way.

Closure of a Set of Attributes

- Given a set of attributes $A_1$, ..., $A_n$
- The closure $\{A_1, ..., A_n\}^+ = \{B\}$ such that $A_1$, ..., $A_n \rightarrow B$

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

- Closures:
  - $name^+ = \{name, color\}$
  - $\{name, category\}^+ = \{name, category, color, department, price\}$
  - $color^+ = \{color\}$
Closure Algorithm For Attributes

To find closure \( \{A_1, ..., A_n\}^+ \):
1. Start with \( X = \{A_1, ..., A_n\} \)
2. Repeat until \( X \) doesn’t change:
   3. if \( B_1, ..., B_n \rightarrow C \) is a FD and \( B_1, ..., B_n \) are all in \( X \)
   4. then add \( C \) to \( X \)

Can use this algorithm to find keys
- Compute \( X^+ \) for all sets \( X \)
- If \( X^+ = \) all attributes, then \( X \) is a superkey
- Minimal superkeys are keys

Closure For Attributes Example

- Example:
  - category → department
  - name → color
  - color, category → price
- Closures:
  - \( \text{name}^+ = \{\text{name}, \text{color}\} \)
  - \( \{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\} \)
  - \( \text{color}^+ = \{\text{color}\} \)

Another Example

- \( R(A, B, C, D, E, F) \)
  - \( A, B \rightarrow C \)
  - \( A, D \rightarrow E \)
  - \( B \rightarrow D \)
  - \( A, F \rightarrow B \)
- Compute \( \{A, B\}^+ \)
  - \( X = \{A, B, C, D, E\} \)
- Compute \( \{A, F\}^+ \)
  - \( X = \{A, F, B, C, D, E\} \)

Using Closure To Infer ALL FDs

- Example: \( A, B \rightarrow C \)
  - \( A, D \rightarrow B \)
  - \( B \rightarrow D \)
1. Step 1: Compute \( X^+ \), for every \( X \):
   - \( A^+ = A \)
   - \( B^+ = BD \)
   - \( C^+ = C \)
   - \( D^+ = D \)
   - \( AB^+ = ABCD \)
   - \( AC^+ = AC \)
   - \( AD^+ = ABCD \)
   - \( BC^+ = BCD \)
   - \( BD^+ = BD \)
   - \( CD^+ = CD \)
   - \( ABC^+ = ABD^+ = ACD^+ = ABCD \)
   - \( BCD^+ = BCD \)
   - \( ABCD^+ = ABCD \)
2. Step 2: Enumerate all FDs \( X \rightarrow Y \), s.t. \( Y \subseteq X^+ \) and \( X \cap Y = \emptyset \):
   - \( AB \rightarrow CD \)
   - \( AD \rightarrow BC \)
   - \( BD \rightarrow D \)
   - \( ABD \rightarrow C \)
   - \( ACD \rightarrow B \)
Decomposition Problems

- FDs will help us identify possible redundancy
  - Identify redundancy and split relations to avoid it

- Can we get the data back correctly?
  - Lossless-join decomposition

- Can we recover the FDs on the ‘big’ table from the FDs on the small tables?
  - Dependency-preserving decomposition
  - So that we can enforce all FDs without performing joins

Outline

- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization

Normal Forms

- Based on Functional Dependencies
  - 3rd Normal Form
  - Boyce Codd Normal Form (BCNF)

- Based on Multi-valued Dependencies
  - 4th Normal Form

- Based on Join Dependencies
  - 5th Normal Form

BCNF

- 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

- BCNF ensures that no redundancy can be detected using FD information alone
**Example**

<table>
<thead>
<tr>
<th>pno</th>
<th>name</th>
<th>zip</th>
<th>dno</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>3</td>
<td>2003</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98112</td>
<td>1</td>
<td>2002</td>
</tr>
<tr>
<td>3</td>
<td>p1</td>
<td>98143</td>
<td>1</td>
<td>1985</td>
</tr>
</tbody>
</table>

- \{pno, dno\} is a key, but \textbf{pno $\rightarrow$ name, zip}
- BCNF violation, so we decompose

**Decomposition in General**

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

- \(R_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m\)
- \(R_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p\)

- \textbf{Theorem} If \(A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m\), then the decomposition is lossless
- Note: don’t need necessarily \(A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p\)

**BCNF Decomposition Algorithm**

Repeat
- choose \(A_{1'}, \ldots, A_{m'} \rightarrow B_{1'}, \ldots, B_n\) that violates BCNF 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
- \textbf{Lossless-join decomposition}: Attributes common to \(R_1\) and \(R_2\) must contain a key for either \(R_1\) or \(R_2\)

**BCNF and Dependencies**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
</table>

- FDs: \textbf{Unit $\rightarrow$ Company}
  \textbf{Company, Product $\rightarrow$ Unit}
- So there is a BCNF violation, and we decompose
BCNF and Dependencies

- FDs: Unit → Company, Product → Unit
- So there is a BCNF violation, and we decompose
- In BCNF we lose the FD: Company, Product → Unit

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 $\{A_1, A_2, ..., A_n\}$ is a superkey for R, or B is part of a key

3NF Discussion

- 3NF decomposition vs. 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

Summary

- Database design is not trivial
  - Use E/R models
  - Translate E/R models into relations
  - Normalize to eliminate anomalies
- Normalization tradeoffs
  - BCNF: no anomalies, but may lose some FDs
  - 3NF: keeps all FDs, but may have anomalies
  - Too many small tables affect performance
This Time

- Design Theory for Relational Databases
  - Chapter 3: 3.1 – 3.5
- High-Level Database Models
  - Chapter 4: 4.1 – 4.6