Design Theory

Announcements

- HW 5 out tomorrow
- Midterm on Wednesday
  - Covers material through today’s lecture on design
    (Functional dependencies and closures)

Recap

- ER Diagrams
  - Conceptual modeling
  - Rules of thumb for converting diagram into schema

Goals for Today

- Figure out the fundamentals of what makes a good schema

Outline

- Background
  - Anomalies, i.e. things we want to avoid
  - Functional Dependencies (FDs)
  - Closures and formal definitions of keys
  - Normalization: BCNF Decomposition

Informal Design Guidelines

- Semantics of attributes should be self-evident
- Avoid redundant information in tuples
- Avoid NULL values in tuples
- Disallow the generation of “spurious” tuples
  - If certain tuples shouldn’t exist, don’t allow them
Database Design

Database Design is about (1) characterizing data and (2) organizing data

How to talk about properties we know or see in the data

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

The rules that are known to us, since we made them up or they correlate to things in the real world
How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
  - Dimension vs measure
  - Pattern analysis

The rules that are known to us since we made them up or they correlate to things in the real world.

Think About This

Make a simple directory that can:
- Hold information about name, SSN, phone, and city
- Associate people with the city they live in
- Associate people with any phone numbers they have

Anomalies:
- Redundancy → Slow Update
  - Change Fred’s city to Bellevue (two rows!)
- Deletion Anomalies
  - How to delete Joe’s phone without deleting Joe?
Think About This

Make a simple directory that can:
- Hold information about name, SSN, phone, and city
- Associate people with the city they live in
- Associate people with any phone numbers they have

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

Anomalies:
- **Redundancy** → Slow Update
- Change Fred’s city to Bellevue (two rows)
- **Deletion Anomalies**
- How to delete Joe’s phone without deleting Joe?

Anomalies:
- **Redundancy** → Slow Update
- Change Fred’s city to Bellevue (two rows)
- **Deletion Anomalies**
- How to delete Joe’s phone without deleting Joe?

Data Interrelationships

<table>
<thead>
<tr>
<th>Functional Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ( \rightarrow B ) holds in the relation R if:</td>
</tr>
<tr>
<td>( \forall t \in R, (t[A_1, A_2, ..., A_n] = t'[A_1, A_2, ..., A_n]) )</td>
</tr>
<tr>
<td>Informally, some attributes determine other attributes.</td>
</tr>
<tr>
<td>( A_1, A_2, ..., A_n ) is the antecedent.</td>
</tr>
<tr>
<td>( B_1, B_2, ..., B_m ) is the consequent.</td>
</tr>
</tbody>
</table>

Warning! Dependency does not imply causation!

Think About This

We can solve the anomalies by converting this

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

into this

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

How can we systematically avoid anomalies?

Functional Dependencies (FDs)

**Definition**

If two tuples agree on the attributes

\[ A_1, A_2, ..., A_n \]

then they must also agree on the attributes

\[ B_1, B_2, ..., B_m \]

Formally:

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

Example

An FD \( B \rightarrow A \) or \( \not{A \rightarrow B} \) 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>
<tr>
<td>didf</td>
<td>Shaffs</td>
<td>1010</td>
<td>Salesrep</td>
</tr>
</tbody>
</table>

EmpID \( \rightarrow \) Name, Phone, Position

Position \( \rightarrow \) Phone

but not Phone \( \rightarrow \) Position
### Example

<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>

But not Phone → Position

---

### 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>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Do all the FDs hold on this instance?

---

### 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>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Do all the FDs hold on this instance?

---

### Buzzwords

- FD **holds** or **does not hold** on an instance
- If we can be sure that every instance of R will be one in which a given FD is true, then we say that R **satisfies the FD**
- If we say that R satisfies an FD, we are **stating a constraint on R**
An Interesting Observation

If all these FDs are true:
- name → color
- category → department
- color, category → price

Then this FD also holds:
- name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.

November 1, 2019

Fundamentals of FDs

Armstrong’s Axioms

- **Axiom of Reflexivity (Trivial FD)**
  
  If \( B \subseteq A \) then \( A \rightarrow B \)

  \[ \text{ex.} \quad \text{name} \subseteq \text{name}, \text{job} \quad \text{so} \quad \text{name, job} \rightarrow \{ \text{name} \} \]

- **Axiom of Augmentation**
  
  If \( A \rightarrow B \) then \( \forall C \subseteq BC \rightarrow AC \rightarrow BC \)

  \[ \text{ex.} \quad \{(ID) \rightarrow \{\text{name}\}\} \subseteq \{(ID, job) \rightarrow \{\text{name, job}\}\} \]

- **Axiom of Transitivity**
  
  If \( A \rightarrow B \) and \( B \rightarrow C \) then \( A \rightarrow C \)

  \[ \text{ex.} \quad \{(ID) \rightarrow \{\text{name}\}\} \subseteq \{(ID, job) \rightarrow \{\text{name, job}\}\} \]

  \[ \text{We’ll use transitivity all the time} \]

  \[ \text{so} \quad \{(ID, job) \rightarrow \{\text{name, job}\}\} \]

Interesting Secondary Rules

- **Pseudo Transitivity**
  
  If \( A \rightarrow BC \) and \( C \rightarrow D \) then \( A \rightarrow BD \)

- **Extensivity**
  
  If \( A \rightarrow B \) then \( A \rightarrow AB \)
Can I do this to FDs?

I only know ID → \{name\}
So (ID, hair color) → \{name\}

Yes!

Adding more attributes to the antecedent can never remove attributes in the consequent.

What about this?

I only know ID → \{name\}
So ID → \{name, hair color\}

No!

No way to use the axioms to introduce hair color to the consequent without also introducing it to the antecedent.
Finding Keys

All this talk about FDs sounds awfully similar to keys...

Closure

The Closure of a set \( \{ A_1, \ldots, A_n \} \), written as \( \{ A_1, \ldots, A_n \}^* \), is the set of attributes \( A \) such that \( A \subseteq A_1, \ldots, A_n \rightarrow B \).

A closure finds everything a set of attributes determines.

Announcements

- Midterm on Wednesday
  - Covers material through FDs and closures (slide 45 of 10/25 lecture)
  - Calculator not needed
  - 1 page of notes, front and back

Closure Algorithm

Find the closure of \( \{ A_1, \ldots, A_n \} \)

\[ X = \{ A_1, \ldots, A_n \} \]

Repeat until \( X \) does not change:

If \( B_1, \ldots, B_n \rightarrow C \) is a FD and \( B_1, \ldots, B_n \in X \)

then \( X = X \cup C \)

In practice:

Repeated use of transitivity

If a FD applies, add the consequent to the answer
Closure of a set of Attributes

Given a set of attributes \(A_1, \ldots, A_n\).

The closure is the set of attributes \(B\), notated \(\{A_1, \ldots, A_n\}^+\), s.t. \(A_1, \ldots, A_n \Rightarrow B\).

Example:

Closures:

- \(\text{name}^+ = \{\text{name}, \text{color}\}\)
- \(\text{color}^+ = \{\text{color}\}\)

\[\begin{align*}
1. & \text{name} \Rightarrow \text{color} \\
2. & \text{category} \Rightarrow \text{department} \\
3. & \text{color, category} \Rightarrow \text{price}
\end{align*}\]

Closure Algorithm

\(X = \{A_1, \ldots, A_n\}\).

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:

\(\{\text{name, category}\}^+ = \{\text{name, category, color}\}\)

The "trivial" FD
Closure Algorithm

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:
\( (\text{name, category})^* = \{\text{name, category, color, department, price}\} \)
Hence: \( \text{name, category} \rightarrow \text{color, department, price} \)

Usefulness of Keys in Design

What intuitions do we get from data interrelationships?
• FDs that are not superkeys hint at redundancy
  • If a FD antecedent is \textit{not} a superkey, we can remove redundant information, i.e. the FD consequent

Usefulness of Keys in Design

What intuitions do we get from data interrelationships?
• FDs that are not superkeys hint at redundancy
  • If a FD antecedent is \textit{not} a superkey, we can remove redundant information, i.e. the FD consequent

Rephrased
• \( A \rightarrow B \) is fine if \( A \) is a superkey
• Otherwise, we can extract \( B \)

Usefulness of Keys in Design

What intuitions do we get from data interrelationships?
• FDs that are not superkeys hint at redundancy
  • If a FD antecedent is \textit{not} a superkey, we can remove redundant information, i.e. the FD consequent

Rephrased
• \( A \rightarrow B \) is fine if \( A \) is a superkey
• Otherwise, we can extract \( B \)

SSN is not a superkey!
Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid \(\rightarrow\) name
- rid \(\rightarrow\) rating
- rating \(\rightarrow\) popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>Respectable</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco's Tacos</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cnergy</td>
<td>5</td>
<td>Poppin</td>
</tr>
</tbody>
</table>

Fine because rid is a superkey

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid \(\rightarrow\) name
- rid \(\rightarrow\) rating
- rating \(\rightarrow\) popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>Respectable</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco's Tacos</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cnergy</td>
<td>5</td>
<td>Poppin</td>
</tr>
</tbody>
</table>

Fine because rid is a superkey

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid \(\rightarrow\) name
- rid \(\rightarrow\) rating
- rating \(\rightarrow\) popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>Respectable</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco's Tacos</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cnergy</td>
<td>5</td>
<td>Poppin</td>
</tr>
</tbody>
</table>

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid \(\rightarrow\) name
- rid \(\rightarrow\) rating
- rating \(\rightarrow\) popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>Respectable</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco's Tacos</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cnergy</td>
<td>5</td>
<td>Poppin</td>
</tr>
</tbody>
</table>

Fine because rid is a superkey

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid \(\rightarrow\) name
- rid \(\rightarrow\) rating
- rating \(\rightarrow\) popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>Respectable</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco's Tacos</td>
<td>4</td>
<td>Poppin</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cnergy</td>
<td>5</td>
<td>Poppin</td>
</tr>
</tbody>
</table>

Fine because rid is a superkey

Database Design

Database Design is about:
(1) characterizing data and (2) organizing data

How to talk about properties we know or see in the data

Redundancy!

Database Design

Database Design is about:
(1) characterizing data and (2) organizing data

How to organize data to promote ease of use and efficiency
Normal Forms

1NF: Normal Form if all attribute values are atomic.
2NF: Normal Form if all non-trivial dependencies, X \rightarrow A, X is a superkey.
3NF: Normal Form if all non-trivial dependencies, X \rightarrow A, X is a superkey.
4NF: Normal Form if all multi-valued dependencies are preserved.
5NF: Normal Form if all join dependencies are preserved.

BCNF: Boyce-Codd Normal Form (BCNF)
A relation R is in BCNF if for every non-trivial dependency, X \rightarrow A, X is a superkey.

We often call these “bad FDs” because they prevent the relation from being in BCNF.

Decomposition

Extracting” attributes can be done with decomposition (split the schema into smaller parts).

For this class, decomposition means the following:

Remove all the bad FDs, then the relation is in BCNF.
For this class, decomposition means the following:

- “Extracting” attributes can be done with decomposition (split the schema into smaller parts)
- For this class, decomposition means the following:

\[
R(A_1, A_2, B_1, B_2, C_1, C_2) \subseteq R_1(A_1, A_2, B_1, B_2) \cup R_2(A_1, A_2, C_1, C_2)
\]

Some common attributes are present so we can retain data.

### BCNF Decomposition Example

Restaurants(rid, name, rating, popularity, recommended)

(1) rating \(\rightarrow\) rating, popularity, recommended

(“bad” FD)
BCNF Decomposition Example

Restaurants(rid, name, rating, popularity, recommended)

1. rating → rating, popularity, recommended ("bad" FD)
2. R1 = rating, popularity, recommended
3. R2 = rid, name, rating

Finished? NO! (popularity (3) rating (2) recommended (1))

R2 =
R1 = rating, popularity, recommended

BCNF Decomposition Example

Restaurants(rid, name, rating, popularity, recommended)

1. rating → rating, popularity, recommended ("bad" FD)
2. R1 = rating, popularity, recommended
3. R2 = rid, name, rating

Finished? NO! (popularity (3) rating (2) recommended (1))

BCNF Decomposition Example

Restaurants(rid, name, rating, popularity, recommended)

1. rating → rating, popularity, recommended ("bad" FD)
2. R1 = rating, popularity, recommended
3. R2 = rid, name, rating

Finished? NO! (popularity (3) rating (2) recommended (1))

We decompose R1 into R3, R3

Losslessness

Definition

Lossless Decomposition is a reversible decomposition, i.e., rejoining all decomposed relations will always result exactly with the original data.

This is the opposite of a Lossy Decomposition, an irreversible decomposition, where rejoining all decomposed relations may result something other than the original data, specifically with extra tuples.

This concept might be familiar if you have ever encountered lossless data compression (e.g. Huffman encoding or PNG) or lossy data compression (e.g. JPEG).
Losslessness

Is BCNF decomposition lossless?

Yes!

In our example:

R2 = ride, name, rating
R4 = rating, popularity
R4 = popularity, recommended

...gives us original R