Introduction to Database Systems
CSE 414

Lecture 20: Design Theory

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
- Unit 3: Non-relational data
- Unit 4: RDBMS internals and query optimization
- Unit 5: Parallel query processing
  - Unit 6: DBMS usability, conceptual design
    - E/R diagrams
    - Schema normalization
- Unit 7: Transactions

Database Design Process

Conceptual Model:

Relational Model:

Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical Schema

Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object
- Attribute
- Relationship

Arrows in Multiway Relationships

Q: What does the arrow mean?

A: Any person buys a given product from at most one store
AND every store sells to every person at most one product

N-N Relationships to Relations

Orders(prod-ID, cust-ID, date)
Shipment(prod-ID, cust-ID, name, date)
Shipping-Co(name, address)
N-1 Relationships to Relations

Orders(prod-ID, cust-ID, date1, name, date2)
Shipping-Co(name, address)

Orders orders and Shipping-Co shipping are related.

Remember: no separate relations for many-one relationship.

Subclasses to Relations

Orders

Order

prod-ID cust-ID date name

Shipping-Co

ship-name

Other ways to convert are possible

Remember: no separate relations for many-one relationship

Subclasses to
Relations

Product

name category price

Sw.Product

name platforms

Gizmo unix

Ed.Product

Name Age Group

Gizmo toddler

Toy retired

Other ways to convert are possible

Modeling Union Types with Subclasses

Solution 2: better, more laborious

Owner

isa

isa

ownedBy

FurniturePiece

Person

Company

sw.Product

Name

Software Product

platforms

Gizmo

Educational Product

Age Group

Name

Gizmo

Toy

Weak Entity Sets

University

Team

affiliation

University

Team(sport, number, universityName)

University(name)

Other Constraints

Product

makes

Company

<100

Product

makes

Company

Q: What does this mean?

A: A Company entity cannot be connected by relationship to more than 99 Product entities
Constraints in SQL

Constraints in SQL:
- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions
- The more complex the constraint, the harder it is to check and to enforce

What happens when data changes?

- SQL has three policies for maintaining referential integrity:
  - **NO ACTION** reject violating modifications (default)
  - **CASCADE** after delete/update do delete/update
  - **SET NULL** set foreign-key field to NULL
  - **SET DEFAULT** set foreign-key field to default value
    - need to be declared with column, e.g.,
      ```
      CREATE TABLE Product (pid INT DEFAULT 42)
      ```

Relational Schema Design

<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>

One person may have multiple phones, but lives in only one city
Primary key is thus (SSN, PhoneNumber)
What is the problem with this schema?

Relation Decomposition

Break the relation into two:

Anomalies have gone:
- No more repeated data
- Easy to move Fred to “Bellevue” (how?)
- Easy to delete all Joe’s phone numbers (how?)
Relational Schema Design (or Logical Design)

How do we do this systematically?

• Start with some relational schema
• Find out its functional dependencies (FDs)
• Use FDs to normalize the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

A₁, A₂, ..., Aₙ → B₁, B₂, ..., Bₘ

then they must also agree on the attributes

B₁, B₂, ..., Bₘ

Formally:

A₁, A₂, ..., Aₙ → B₁, B₂, ..., Bₘ

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

Example

Position → Phone

<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
Do all the FDs hold on this instance?

<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>

What about this one?

<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>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

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**

Why bother with FDs?

**Anomalies:**
- **Redundancy** = repeat data
- **Update anomalies** = what if Fred moves to “Bellevue”?
- **Deletion anomalies** = what if Joe deletes his phone number?

<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>

An Interesting Observation

If all these FDs are true:

- name \(\rightarrow\) color
- category \(\rightarrow\) department
- color, category \(\rightarrow\) price

Then this FD also holds:

- name, category \(\rightarrow\) 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.

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:

1. name \(\rightarrow\) color
2. category \(\rightarrow\) department
3. color, category \(\rightarrow\) price

Closures:

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

Example

In class:

\[ \begin{align*}
R(A,B,C,D,E,F) & \\
\text{Compute } (A,B)^+ & X = \{ A, B, C, D, E \} \\
\text{Compute } (A,F)^+ & X = \{ A, F, C, D, E \}
\end{align*} \]

Example

In class:

\[ \begin{align*}
R(A,B,C,D,E,F) & \\
\text{Compute } (A,B)^+ & X = \{ A, B, C, D, E \} \\
\text{Compute } (A,F)^+ & X = \{ A, F, B, C, D, E \}
\end{align*} \]

Example

In class:

\[ \begin{align*}
R(A,B,C,D,E,F) & \\
\text{Compute } (A,B)^+ & X = \{ A, B, C, D, E \} \\
\text{Compute } (A,F)^+ & X = \{ A, F, B, C, D, E \}
\end{align*} \]

Example

In class:

\[ \begin{align*}
R(A,B,C,D,E,F) & \\
\text{Compute } (A,B)^+ & X = \{ A, B, C, D, E \} \\
\text{Compute } (A,F)^+ & X = \{ A, F, B, C, D, E \}
\end{align*} \]

Practice at Home

Find all FD's implied by:

\[ \begin{align*}
\text{Step 1: Compute } X^+ \text{ for every } X: \\
A^* = A, & B^* = BD, & C^* = C, & D^* = D \\
AB^* = ABCD, & AC^* = AC, & AD^* = ABCD, & \\
& BC^* = BCD, & BD^* = BD, & CD^* = CD \\
ABC^* = ABD^* = ACD^* = ABCD (no need to compute-- why ?) & BCD^* = BCD, & ABCD^* = ABCD
\end{align*} \]

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

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

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute $B$, we have $A_1, ..., A_n \rightarrow B$

- A **key** is a minimal superkey
  - A superkey and for which no subset is a superkey

**Computing (Super)Keys**

- For all sets $X$, compute $X^+$
- If $X^+ = \{\text{all attributes}\}$, then $X$ is a superkey
- Try reducing to the minimal $X$’s to get the key

**Example**

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

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

What is the key?

**Example**

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

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

What is the key?

($name$, $category$) $+$ $=$ $\{name$, $category$, $price$, $color\}$

Hence $(name$, $category$) is a key

**Key or Keys?**

Can we have more than one key?

Given $R(A,B,C)$ define FD’s s.t. there are two or more distinct keys

$$A \rightarrow B$$
$$B \rightarrow C$$
$$C \rightarrow A$$

or

$$AB \rightarrow C$$
$$BC \rightarrow A$$

or

$$A \rightarrow BC$$
$$B \rightarrow AC$$

What are the keys here?
Eliminating Anomalies

Main idea:

• \( X \to A \) is OK if \( X \) is a (super)key

• \( X \to A \) is not OK otherwise
  – Need to decompose the table, but how?

Boyce-Codd Normal Form

Dr. Raymond F. Boyce

There are no "bad" FDs:

Definition. A relation \( R \) is in BCNF if:
Whenever \( X \to B \) is a non-trivial dependency, then \( X \) is a superkey.

Equivalently:

Definition. A relation \( R \) is in BCNF if:
\( \forall X, \text{ either } X^* = X \text{ or } X^* = \text{[all attributes]} \)

BCNF Decomposition Algorithm

Normalize(\( R \))

find \( X \) s.t.: \( X \neq X^* \) and \( X^* \neq \text{[all attributes]} \)
if (not found) then "R is in BCNF"

let \( Y = X^* - X \); \( Z = \text{[all attributes]} - X^* \)

decompose \( R \) into \( R1(X \cup Y) \) and \( R2(X \cup Z) \)
 Normalize(\( R1 \)); Normalize(\( R2 \)).

Example

<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>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

The only key is: \{SSN, PhoneNumber\}

Hence \( SSN \to Name, City \) is a "bad" dependency

In other words:

\( SSN^+ = SSN, Name, City \) and is neither \( SSN \) nor \text{All Attributes}
Example BCNF Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-8789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

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

SSN → Name, City

<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>