Database Design Theory

• Guides systematic improvements to database schemas
• General idea:
  – Express constraints on the data
  – Use these to decompose the relations
• Ultimately, get a schema that is in a “normal form” that guarantees certain desirable properties
• “Normal” in the sense of conforming to a standard
• The process of converting a schema to a normal form is called normalization

Goal #1: redundancy, redundancy

• Consider this schema

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

• What if...
  – Xiao changes email addresses? (update anomaly)
  – Xiao drops CSCD08? (deletion anomaly)
  – UTSC creates a new course, CSCC44 (insertion anomaly)

Multiple relations => exponentially worse

Goal #2: expressing constraints

• Consider the following sets of schemas:

<table>
<thead>
<tr>
<th>Students(utorid, name, email) vs. Students(utorid, name) Emails(utorid, address)</th>
</tr>
</thead>
</table>

• Consider also:

<table>
<thead>
<tr>
<th>House(street, city, value, owner, propertyTax) vs. House(street, city, value, owner) TaxRates(city, value, propertyTax)</th>
</tr>
</thead>
</table>

Dependencies, constraints are domain-dependent
Part I: Theory of functional dependencies

Functional dependencies

• Let $X, Y$ be sets of attributes from relation $R$  
  
  • $X \rightarrow Y$ is an assertion about tuples in $R$  
    – Any tuples which agree in all attributes of $X$ must also agree in all attributes of $Y$  
  
  • “$X$ functionally determines $Y$”  
    – Or, “The values of attributes $Y$ are a function of those in $X$”  
    – Not necessarily an easy function to compute, mind you  
  
  => Consider $X \rightarrow h$, where $h$ is the hash of attributes in $X$

Notational conventions

• “$a$, “$b$, “$c$” – specific attributes  
• $abc \rightarrow def$ – same as $(a,b,c) \rightarrow (d,e,f)$

Most common to see singletons ($X \rightarrow y$ or $abc \rightarrow d$)

Splitting FDs

• Attributes on right independent of each other  
  – Consider $a,b,c \rightarrow d,e,f$  
  – “Attributes $a$, $b$, and $c$ functionally determine $d$, $e$, and $f$”  
  => No mention of $d$ relating to $e$ or $f$ directly

• Useful to split right side of FD  
  – $abc \rightarrow def$ becomes $abc \rightarrow d$, $abc \rightarrow e$ and $abc \rightarrow f$

• No safe way to split left side  
  – $abc \rightarrow def$ is NOT the same as $ab \rightarrow def$ and $c \rightarrow def$

Splitting FDs – example

• Consider the relation  
  – EmailAddress(user, domain, firstName, lastName)  
  – user,domain -> firstName, lastName

• The following hold  
  – user,domain -> firstName
  – user,domain -> lastName

• The following do NOT hold!  
  – user -> firstName,lastName
  – domain -> firstName,lastName

Gotcha: “doesn’t hold” = “not all tuples” != “all tuples not”
Trivial FDs

- Not all functional dependencies are useful
  - A -> A always holds
  - abc -> a also always holds
- FD with an attribute on both sides is “trivial”
  - Simplify by removing \( L \cap R \) from R
  - abc -> ad becomes abc -> d
  - Or, in singleton form, delete trivial FDs
    - abc -> a and abc -> d becomes just abc -> d

Identifying functional dependencies

- FDs are domain knowledge
  - Intrinsic features of the data you’re dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- DBMS cannot safely “optimize” FDs either
  - It has only a finite sample of the data
  - An FD constrains the entire domain

Coincidence or FD?

<table>
<thead>
<tr>
<th>ID</th>
<th>Email</th>
<th>City</th>
<th>Country</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td><a href="mailto:tom@gmail.com">tom@gmail.com</a></td>
<td>Toronto</td>
<td>Canada</td>
<td>Fairgrieve</td>
</tr>
<tr>
<td>8624</td>
<td><a href="mailto:mar@bell.com">mar@bell.com</a></td>
<td>London</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>9141</td>
<td><a href="mailto:scotty@gmail.com">scotty@gmail.com</a></td>
<td>Winnipeg</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>1204</td>
<td><a href="mailto:birds@gmail.com">birds@gmail.com</a></td>
<td>Aachen</td>
<td>Germany</td>
<td>Lakemeyer</td>
</tr>
</tbody>
</table>

- What if we try to infer FDs from the data?
  - ID -> email, city, country, surname
  - email -> city, country, surname
  - city -> country
  - surname -> country

Domain knowledge required to validate FDs

Keys and FDs

- Consider relation R with attributes A
  - Superkey
    - Any \( S \subseteq A \) s.t. \( S \rightarrow A \)
    - \( S \) is any subset of \( A \) which determines all remaining attributes in \( A \)
  - Candidate key
    - \( C \subseteq A \) s.t. \( C \rightarrow A \) and \( X \rightarrow A \) does not hold for any \( X \subseteq C \)
    - A superkey which contains no other superkeys
  - Primary key
    - The candidate key we use to identify the relation
    - Always exists, only one allowed, doesn’t matter which \( C \) we use
  - Prime attribute
    - \( \exists \) candidate key \( C \subseteq A \) s.t. \( x \in C \)
Candidate keys vs. superkeys

- Consider these relations
  - Students(ID, surname, name, email, address, major)
  - Houses(street, city, value, owner, tax)
- What are the candidate keys?
  - Students: ID, what else?
  - Houses: ?
- What other superkeys exist?
  - Students: ID, surname ID, name ID, name, surname ...
  - Houses: ?
- Prime attributes?
  - Students: ?
  - Houses: ?

FD: relaxes the concept of a “key”

- Functional dependency: \( X \rightarrow Y \)
- Superkey: \( X \rightarrow R \)
- A superkey must include all remaining attributes of the relation on the RHS
- An FD can involve just a subset of them
- Example:
  - Houses(street, city, value, owner, tax)
  - street, city \( \rightarrow \) value, owner, tax (both FD and key)
  - city, value \( \rightarrow \) tax (FD only)

Cyclic functional dependencies?

- Attributes on right side of one FD may appear on left side of another!
  - Simplest example: \( A \rightarrow B \quad B \rightarrow A \)
  - What does this say about A and B?
- Example
  - street, city \( \rightarrow \) value
  - studentID \( \rightarrow \) email

Geometric view of FDs

- Let \( D \) be the domain of tuples in \( R \)
  - Every possible tuple is a point in \( D \)
- FD \( X \) on \( R \) restricts tuples in \( R \) to a subset of \( D \)
  - Points in \( D \) which violate \( X \) cannot be in \( R \)
- Example: \( D(x,y,z) \)
  - \( xy \rightarrow z \)
  - \( z \rightarrow x, y \)
  - \( x = y = \text{abs}(z)/2 \)
Inferring functional dependencies

- **Problem**
  - Given FDs $X_i \rightarrow a_i$, $X_j \rightarrow a_j$, etc.
  - Does some FD $Y \rightarrow B$ (not given) also hold?

- **Consider the dependencies**
  
  $A \rightarrow B \quad B \rightarrow C$
  
  Intuitively, $A \rightarrow C$ also holds
  
  The given FDs entail (imply) it

**How to prove it in the general case?**

Closure test for FDs

- **Given attribute set $A$ and FD set $F$**
  
  - Denote $A^+_F$ as the closure of $A$ relative to $F$
  
  $\Rightarrow A^+_F = \text{set of all FDs given or implied by } A$

- **Computing the [transitive] closure of $A$**
  
  - Start: $A^+_F = A$, $F' = F$
  
  - While $\exists X \in F$ s.t. $LHS(X) \subseteq A^+_F$:
    
    $A^+_F = A^+_F \cup RHS(X)$
    
    $F' = F' - X$

  - At end: $A \rightarrow B \; \forall B \in A^+_F$

Closure test – example

- **Consider $R(a,b,c,d,e,f)$**
  
  with FDs $ab \rightarrow c$, $ac \rightarrow d$, $c \rightarrow e$, $ade \rightarrow f$

- **Find $A^+$ if $A = ab$**

  $ab -> cdef -- ab$ is a candidate key!

Discarding redundant FDs

- **Minimal basis:** opposite extreme from closure

- **Given a set of FDs $F$, want to minimize $F'$ s.t.**
  
  - $F' \subseteq F$
  
  - $F'$ entails $X \forall X \in F$

- **Properties of a minimal basis**
  
  - RHS is always singleton
  
  - Removing any FD from $F'$ loses information
  
  - Removing any attribute from any $X \in F$ loses information
Constructing a minimal basis

- **Straightforward but time-consuming**
  1. Split all RHS into singletons
  2. \( \forall X \in F', \) test whether \((F'-X)^+\) is still equivalent to \(F^+\)
     => Might make \(F'\) too small
  3. \( \forall i \in \text{LHS}(X), \forall X \in F', \) let \(\text{LHS}(X')=\text{LHS}(X)-i\)
     Test whether \((F'-X+X')^+\) is still equivalent to \(F^+\)
     => Might make \(F'\) too big
  4. Repeat (2) and (3) until neither succeeds