CMSC 424 – Database design
Lecture 9
Normalization

Mihai Pop
Administrative

• SQL assignment questions – Sharath

• Project – please pair up – submit pairs by Monday, March 4.

• For midterm – chapters 1-4, 6

• Anything you'd like me to go over now?
Accessing databases from software

• Embedded SQL (special commands within C, Java, etc. code)

SQL APIs
• ODBC
• JDBC
• Perl::DBI
• Ruby on Rails

Basic protocol
• connect to server
• run SQL commands – tuples returned as cursors/iterators (allows you iterate over each tuple in result table)
• disconnect from server

Read chapter 4!!! You'll need this for project.
SQL…last thoughts

• You learn best through practice

• Every database system is different (syntax, conventions, etc.)

• READ THE REFERENCE MANUALS!
Relational Database Design

Where did we come up with the *schema* that we used?

E.g. why not store the actor names with movies?
Or, store the author names with the papers?

Topics:

- Formal definition of what it means to be a “good” schema.
- How to achieve it.
Movies Database Schema

Movie(title, year, length, inColor, studioName, producerC#, starName)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

Changed to:

Movie(title, year, length, inColor, studioName, producerC#, starName)
<merged into above>
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
Example Relation

Movie(title, year, length, inColor, studioName, producerC#, starName)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>120</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>120</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>120</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>..</td>
<td>Studio_A</td>
<td>150</td>
<td>Naomi</td>
</tr>
<tr>
<td>King Kong</td>
<td>1940</td>
<td>..</td>
<td>Studio_B</td>
<td>20</td>
<td>Faye</td>
</tr>
</tbody>
</table>
What we're looking for in a schema

- Low/no redundancy
- Easy to understand structure
- Easy to write queries
- Efficient to answer queries
- Ease of maintaining integrity of the data

- Difficult to do this “by hand”

- Normalization – formal algorithms for creating a “reasonable” schema
Combine Schemas?

- Suppose we combine *borrow* and *loan* to get
  \[ \text{bor\_loan} = (\text{customer\_id}, \text{loan\_number}, \text{amount}) \]
- Result is possible repetition of information (L-100 in example below)
A Combined Schema Without Repetition

- Consider combining `loan_branch` and `loan`.
  - `loan_amt_br = (loan_number, amount, branch_name)`
- No repetition (as suggested by example below)
What About Smaller Schemas?

• Suppose we had started with bor_loan. How would we know to split up (decompose) it into borrower and loan?

• Write a rule “if there were a schema (loan_number, amount), then loan_number would be a candidate key”

• Denote as a functional dependency:

\[ \text{loan}_\text{number} \circ \text{amount} \]
**Functional Dependencies**

- set of attributes whose values uniquely determine the values of the remaining attributes e.g. a key defines an FD:
  
  e.g. in \( \text{EMP}(\text{eno}, \text{ename}, \text{sal}) \) key FDs: \( \text{eno} \rightarrow \text{ename} \)
  
  \( \text{DEPT}(\text{dno}, \text{dname}, \text{floor}) \) \( \text{eno} \rightarrow \text{sal} \)
  
  \( \text{WORKS-IN}(\text{eno}, \text{dno}, \text{hours}) \) other FDs: \( \{\text{eno}, \text{dno}\} \rightarrow \text{hours} \)
  
  for every pair of values of \( \text{eno}, \text{dno} \) there exists exactly one value for \( \text{hours} \)

- in general if \( \alpha \subseteq R \) and \( \beta \subseteq R \), then \( \alpha \rightarrow \beta \) holds in the extension \( r(R) \) of \( R \)
  
  iff for any pair \( t_1 \) and \( t_2 \) tuples of \( r(R) \) such that \( t_1(\alpha) = t_2(\alpha) \),
  
  then it is also true that \( t_1(\beta) = t_2(\beta) \) (uniqueness of \( \beta \) values)

- we can use the FDs as
  
  – constraints that we want to enforce (e.g. keys)
  
  – for checking if the FDs are satisfied in the database

<table>
<thead>
<tr>
<th>( R(A, B, C, D) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
</tr>
<tr>
<td>A \rightarrow B</td>
</tr>
<tr>
<td>satisfied? no</td>
</tr>
<tr>
<td>1 2 1 2</td>
</tr>
<tr>
<td>A \rightarrow C</td>
</tr>
<tr>
<td>-&quot;- yes</td>
</tr>
<tr>
<td>2 2 2 2</td>
</tr>
<tr>
<td>C \rightarrow A</td>
</tr>
<tr>
<td>-&quot;- no</td>
</tr>
<tr>
<td>2 3 2 3</td>
</tr>
<tr>
<td>AB \rightarrow &gt; D</td>
</tr>
<tr>
<td>-&quot;- yes</td>
</tr>
<tr>
<td>3 3 2 4</td>
</tr>
</tbody>
</table>
FDs continued

- trivial dependencies: \( \alpha \rightarrow \alpha \)
  \( \alpha \rightarrow \beta \) if \( \beta \subseteq \alpha \)

- closure
  - need all FDs
  - some logically implied by others e.g. if \( A \rightarrow B \) & \( B \rightarrow C \) then \( A \rightarrow C \) is implied

- given \( F = \) set of FDs, find \( F^+ \) (the closure) of all logically implied by \( F \)

Amstrong’s axioms

- reflexivity: if \( \beta \subseteq \alpha \) then \( \alpha \rightarrow \beta \) (trivial FD)
- augmentation: if \( \alpha \rightarrow \beta \) then \( \gamma \alpha \rightarrow \gamma \beta \)
- transitivity: if \( \alpha \rightarrow \beta \) & \( \beta \rightarrow \gamma \) then \( \alpha \rightarrow \gamma \)
More FD Rules

- union rule: \[ \text{if } \alpha \rightarrow \beta \ \& \ \alpha \rightarrow \gamma \ \text{then } \alpha \rightarrow \beta \gamma \]
- decomposition rule: \[ \text{if } \alpha \rightarrow \beta \gamma \ \text{then } \alpha \rightarrow \beta \ \& \ \alpha \rightarrow \gamma \]
- pseudotransitivity rule: \[ \text{if } \alpha \rightarrow \beta \ \& \ \gamma \beta \rightarrow \delta \ \text{then } \alpha \gamma \rightarrow \delta \]

Example: \[ R(A,B,C,G,H,I) \]
\[ F= \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \} \]
\[ F+= \{ A \rightarrow H \quad /* A \rightarrow B \rightarrow H \quad \text{transitivity} \]
\[ \quad CG \rightarrow HI \quad /* CG \rightarrow H, CG \rightarrow I \ \text{union rule} \]
\[ \quad AG \rightarrow I \quad /* A \rightarrow C \ \text{augmentation} AG \rightarrow CG \rightarrow I \]
\[ \quad AG \rightarrow H \} \quad /* \]
\[ \quad CG \rightarrow H \]

- there is a non-trivial (exponential) algorithm for computing \( F^+ \)
Closure of Attribute Sets

• useful to find if a set of attributes is a superkey
• the closure $\alpha^+$ of a set of attributes $\alpha$ under F is the set of all attributes that are functionally determined by $\alpha$
• there is an algorithm that computes the closure

Example:

<table>
<thead>
<tr>
<th>Algorithm to Compute $(AG)^+$</th>
<th>start with</th>
<th>result = (AG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B$</td>
<td>expands</td>
<td>result = (AGB)</td>
</tr>
<tr>
<td>$A \rightarrow C$</td>
<td>expands</td>
<td>result = (AGBC)</td>
</tr>
<tr>
<td>$CG \rightarrow H$</td>
<td>&quot;-&quot;</td>
<td>result = (AGBCH)</td>
</tr>
<tr>
<td>$CG \rightarrow I$</td>
<td>&quot;-&quot;</td>
<td>result = (AGBCHI)</td>
</tr>
<tr>
<td>$B \rightarrow H$</td>
<td>no more expansion</td>
<td></td>
</tr>
</tbody>
</table>

Note that since G is not on any right hand side, no subset of the attributes can be a superkey unless it contains G for there is no FD to generate it.