Relational Database Design

CPS 216
Advanced Database Systems

Announcements (January 15)

- Review for Codd paper due tonight
  - Follow instructions on course Web site to write reviews and post on H2O
- Reading assignment for next week (Ailamaki et al., VLDB 2001) has been posted
  - Due next Wednesday night
  - Hunt for related/follow-up work too!
- Homework #1 assigned today
  - Look for an email regarding your DB2 account
  - Due February 3 (in 2 ½ weeks)
  - Start early!
- Course project will be assigned next week

Database (schema) design

- Understand the real-world domain being modeled
- Specify it using a database design model
  - Design models are especially convenient for schema design, but are not necessarily implemented by DBMS
  - Popular ones include
    - Entity/Relationship (E/R) model
    - Object Definition Language (ODL)
- Translate the design to the data model of DBMS
  - Relational, XML, object-oriented, etc.
- Apply database design theory to check the design
- Create DBMS schema
Entity-relationship (E/R) model

- Historically very popular
  - Primarily a design model; not implemented by any major DBMS nowadays
- Can think of as a “watered-down” object-oriented design model
- E/R diagrams represent designs

E/R example

- Entity: a “thing,” like a record or an object
- Entity set (rectangle): a collection of things of the same type, like a relation of tuples or a class of objects
- Relationship: an association among two or more entities
- Relationship set (diamond): a set of relationships of the same type; an association among two or more entity sets
- Attributes (ovals): properties of entities or relationships, like attributes of tuples or objects

ODL (Object Definition Language)

- Standardized by ODMG (Object Data Management Group)
  - Comes with a declarative query language OQL (Object Query Language)
  - Implemented by OODBMS (Object-Oriented DataBase Management Systems)
- Object oriented
- Based on C++ syntax
- Class declarations represent designs
ODL example

class Student {
    attribute integer SID;
    attribute string name;
    relationship Set<Course> enrolledIn inverse Course::students;
};
class Course {
    attribute string CID;
    attribute string title;
    relationship Set<Student> students inverse Student::enrolledIn;
};

- Easy to map them to C++ classes
  - ODL attributes correspond to attributes of objects; complex types are allowed
  - ODL relationships can be mapped to pointers to other objects (e.g., Set<Course> → set of pointers to objects of Course class)

Not covered in this lecture

- E/R and ODL design
- Translating E/R and ODL designs into relational designs
  - Reference book (GMUW) has all the details
- Next: relational design theory

Relational model: review

- A database is a collection of relations (or tables)
- Each relation has a list of attributes (or columns)
- Each attribute has a domain (or type)
- Each relation contains a set of tuples (or rows)
Keys

- A set of attributes $K$ is a key for a relation $R$ if
  - In no instance of $R$ will two different tuples agree on all attributes of $K$
    - That is, $K$ is a “tuple identifier”
  - No proper subset of $K$ satisfies the above condition
    - That is, $K$ is minimal
- Example: Student (SID, name, age, GPA)
  - SID is a key of Student
  - \{SID, name\} is not a key (not minimal)

Schema vs. data

<table>
<thead>
<tr>
<th>Student</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>name</td>
<td>age</td>
<td>GPA</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>1.1</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- Is name a key of Student?

More examples of keys

- Enroll (SID, CID)
- Address (street_address, city, state, zip)
- Course (CID, title, room, day_of_week, begin_time, end_time)
Usage of keys

- More constraints on data, fewer mistakes
- Look up a row by its key value
  - Many selection conditions are “key = value”
- “Pointers”
  - Example: \textit{Enroll} (\textit{SID}, \textit{CID})
    - \textit{SID} is a key of \textit{Student}
    - \textit{CID} is a key of \textit{Course}
    - An \textit{Enroll} tuple “links” a \textit{Student} tuple with a \textit{Course} tuple
  - Many join conditions are “key = key value stored in another table”

Motivation for a design theory

- Why is this design bad?
  - This design has redundancy, because the name of a student is recorded multiple times, once for each course the student is taking
- Why is redundancy bad?
- How about a systematic approach to detecting and removing redundancy in designs?
  - Dependencies, decompositions, and normal forms

Functional dependencies

- A functional dependency (FD) has the form $X \rightarrow Y$, where $X$ and $Y$ are sets of attributes in a relation $R$
- $X \rightarrow Y$ means that whenever two tuples in $R$ agree on all the attributes in $X$, they must also agree on all attributes of $Y$

\begin{center}
\begin{tabular}{ccc}
SID & Name & CID \\
142 & Bart & CPS214 \\
142 & Karl & CPS214 \\
857 & Lisa & CPS230 \\
... & ... & ... \\
\end{tabular}
\end{center}

- Must be $b$
- Could be anything
FD examples

Address (street_address, city, state, zip)

Keys redefined using FD’s

A set of attributes $K$ is a key for a relation $R$ if
- $K \rightarrow$ all (other) attributes of $R$
  - That is, $K$ is a "super key"
- No proper subset of $K$ satisfies the above condition
  - That is, $K$ is minimal

Reasoning with FD’s

Given a relation $R$ and a set of FD’s $\mathcal{F}$
- Does another FD follow from $\mathcal{F}$?
  - Are some of the FD’s in $\mathcal{F}$ redundant (i.e., they follow from the others)?
- Is $K$ a key of $R$?
  - What are all the keys of $R$?
Attribute closure

- Given \( R \), a set of FD's \( \mathcal{F} \) that hold in \( R \), and a set of attributes \( Z \) in \( R \):
  - The closure of \( Z \) (denoted \( Z^+ \)) with respect to \( \mathcal{F} \) is the set of all attributes functionally determined by \( Z \)
- Algorithm for computing the closure
  - Start with closure = \( Z \)
  - If \( X \to Y \) is in \( \mathcal{F} \) and \( X \) is already in the closure, then also add \( Y \) to the closure
  - Repeat until no more attributes can be added

A more complex example

\( \text{StudentGrade (SID, name, email, CID, grade)} \)

- Not a good design, and we will see why later

Example of computing closure

- \( \mathcal{F} \) includes:
  - \( \text{SID} \to \text{name, email} \)
  - \( \text{email} \to \text{SID} \)
  - \( \text{SID, CID} \to \text{grade} \)
- \( \{ \text{CID, email} \}^+ = ? \)
  - \( \text{email} \to \text{SID} \)
    - Add \( \text{SID} \), closure is now \( \{ \text{CID, email, SID} \} \)
  - \( \text{SID} \to \text{name, email} \)
    - Add \( \text{name, email} \), closure is now \( \{ \text{CID, email, SID, name} \} \)
  - \( \text{SID, CID} \to \text{grade} \)
    - Add \( \text{grade} \), closure is now all the attributes in \( \text{StudentGrade} \)
Using attribute closure

Given a relation \( R \) and set of FD’s \( F \)

- Does another FD \( X \rightarrow Y \) follow from \( F \)?
  - Compute \( X^+ \) with respect to \( F \)
  - If \( Y \subseteq X^+ \), then \( X \rightarrow Y \) follow from \( F \)
- Is \( K \) a key of \( R \)?
  - Compute \( K^+ \) with respect to \( F \)
  - If \( K^+ \) contains all the attributes of \( R \), \( K \) is a super key
  - Still need to verify that \( K \) is minimal (how?)

Useful rules of FD’s

- Armstrong’s axioms
  - 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 \( X \rightarrow Z \)
- Rules derived from axioms
  - Splitting: If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)
  - Combining: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)

Non-key FD’s

- Consider a non-trivial FD \( X \rightarrow Y \) where \( X \) is not a super key
  - Since \( X \) is not a super key, there are some attributes (say \( Z \)) that are not functionally determined by \( X \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The fact that \( a \) is always associated with \( b \) is recorded in multiple rows: redundancy!
Example of redundancy

- **StudentGrade** (SID, name, email, CID, grade)
- SID \(\rightarrow\) name, email

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS216</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS214</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS216</td>
<td>B+</td>
</tr>
<tr>
<td></td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS216</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS214</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Decomposition

- Eliminates redundancy
- To get back to the original relation:

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS216</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS214</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS216</td>
<td>B+</td>
</tr>
<tr>
<td></td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS216</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS214</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Unnecessary decomposition

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed, and now SID is stored twice!
Bad decomposition

- Association between CID and grade is lost
- Join returns more rows than the original relation

Questions about decomposition

- When to decompose
- How to come up with a correct decomposition

An answer: BCNF

- A relation R is in Boyce-Codd Normal Form if
  - For every non-trivial FD $X \rightarrow Y$ in $R$, $X$ is a super key
  - That is, all FDs follow from “key $\rightarrow$ other attributes”

- When to decompose
  - As long as some relation is not in BCNF
- How to come up with a correct decomposition
  - Always decompose on a BCNF violation
  - Then it is guaranteed to be a correct decomposition!
BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD $X \rightarrow Y$ in $R$ where $X$ is not a super key of $R$
- Decompose $R$ into $R_1$ and $R_2$, where
  - $R_1$ has attributes $X \cup Y$
  - $R_2$ has attributes $X \cup Z$, where $Z$ contains all attributes of $R$ that are in neither $X$ nor $Y$
- Repeat until all relations are in BCNF

BCNF decomposition example

- StudentGrade ($SID$, name, email, CID, grade)
  - BCNF violation: $SID \rightarrow$ name, email
- Student ($SID$, name, email)
  - BCNF
- Grade ($SID$, CID, grade)
  - BCNF

Another example

- StudentGrade ($SID$, name, email, CID, grade)
  - BCNF violation: email $\rightarrow$ SID
- StudentID (email, SID)
  - BCNF
- StudentGrade' (email, name, CID, grade)
  - BCNF violation: email $\rightarrow$ name
- StudentName (email, name)
  - BCNF
- Grade (email, CID, grade)
  - BCNF
Recap

- Functional dependencies: generalization of keys
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method of removing redundancies due to FD's
- BCNF: schema in this normal form has no redundancy due to FD's

- Not covered in this lecture: many other types of dependencies (e.g., MVD) and normal forms (e.g., 4NF)
  - GMUW has all the details
  - Relational design theory was a big research area in the 1970's, but there is not much going on now