Relational Database Design

CPS 216
Advanced Database Systems

Announcements

• Homework #1 out today
  – Due next Thursday in class
• Sign up to present a research paper
  – Sign-up sheet available in my office (D327) during my office hours
    • First-come, first-serve
    • Participation is voluntary
    • Allows you to drop your lowest homework grade
  – In groups of 2-4

Relational model: a 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)

More examples of keys

• $Enroll (SID, CID)$
  – $(SID, CID)$
• $Address (street_address, city, state, zip)$
  – $(street_address, city, state)$
  – $(street_address, zip)$
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: Enroll (SID, CID)
    - SID is a key of Student
    - CID is a key of Course
    - Enroll “links” a Student row with a Course row
  - Many join conditions are “key = key value stored in another table”

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 of $X$, they must also agree on all attributes of $Y$

Enroll

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

Must be “b”

Could be anything

FD examples

Address (street_address, city, state, zip)

- $street\_address, city, state \rightarrow zip$
- $zip \rightarrow city, state$
- $zip, state \rightarrow zip$?
  - Trivial: LHS $\subseteq$ RHS
- $zip \rightarrow state, zip$?
  - Non-trivial, but not completely: LHS $\cap$ RHS $\neq \emptyset$
  - Completely non-trivial FD: LHS $\cap$ RHS $= \emptyset$

Keys redefined using FDs

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 FDs

Given a relation $R$ and set of FDs $F$

- Does another FD follow from $F$?
  - Are some of the FDs in $F$ redundant (because 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 FDs $F$ that holds in $R$, and a set of attributes $Z$ in $R$: The closure of $Z$ with respect to $F$ (denoted $Z^+$) is the set of all attributes functionally determined by $Z$
- Algorithm for computing the closure
  - Start with $Z$
  - If $X \rightarrow Y$ is in $F$ and $X$ is already in the closure, then also add $Y$ to the closure
  - Repeat until you can’t add anything more
A more complex example

* StudentGrade *(SID, name, email, CID, grade)*

- SID → name, email
- email → SID
- SID, CID → grade

Not a good design, and we will see why later

Example of computing closure

- \{ CID, email \}^+ = ?
- email → SID
  - Add SID; closure is now \{ CID, email, SID \}
- SID → name, email
  - Add name and email; closure is now \{ CID, email, SID, name, email \}
- SID, CID → grade
  - Add grade; closure is now all the attributes in *StudentGrade*

Using attribute closure

Given a relation *R* and set of FDs *F*

- Does another FD *X → Y* follow from *F* ?
  - Compute *X*^+ with respect to *F*
  - If *Y* ⊆ *X*^+, then *X → 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?)

Rules of FDs

- Armstrong’s axioms
  - Reflexivity: If *Y* ⊆ *X*, then *X → Y*
  - Augmentation: If *X → Y*, then *XZ → YZ* for any *Z*
  - Transitivity: If *X → Y* and *Y → Z*, then *X → Z*
- Rules derived from axioms
  - Splitting: If *X → YZ*, then *X → Y* and *X → Z*
  - Combining: If *X → Y* and *X → Z*, then *X → YZ*

Using rules of FDs

Given a relation *R* and set of FDs *F*

- Does another FD *X → Y* follow from *F* ?
  - Use the rules to come up with a proof
  - Example: *CID, email → grade?*
    - email → SID (given in *F* )
    - *CID, email → CID, SID* (augmentation)
    - SID, CID → grade (given in *F* )
    - *CID, email → grade* (transitivity)

Non-key FDs

- Consider a non-trivial FD *X → 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>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The fact that “a” is always associated with “b” is recorded in multiple rows: redundancy!
Problems with redundancy

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

- Wastes space
- Potential inconsistencies (update anomaly)

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 216</td>
<td>B</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 214</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS 216</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 216</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS 214</td>
<td>C</td>
</tr>
</tbody>
</table>

Decomposition

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 216</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS 214</td>
<td>B</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 216</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS 214</td>
<td>C</td>
</tr>
</tbody>
</table>

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

Unnecessary decomposition

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 216</td>
<td>B</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 214</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS 216</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 216</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS 214</td>
<td>C</td>
</tr>
</tbody>
</table>

- Fine: join returns the original relation
- Unnecessary: now SID is stored twice!

Bad decomposition

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 216</td>
<td>B</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS 214</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS 216</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 216</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS 130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS 214</td>
<td>C</td>
</tr>
</tbody>
</table>

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

Lossless join decomposition

- Suppose that R is decomposed into S and T
  
  \[ S = \Pi_{\text{atts}(S)}(R) \]
  
  \[ T = \Pi_{\text{atts}(T)}(R) \]

- It is a lossless join decomposition if, given constraints such as FDs, we can guarantee
  
  \[ R = S \sqcup T \]

Loss? But I got more rows!

- “Loss” refers not to the loss of tuples, but to the loss of information
  
  – Or, the ability to distinguish different original relations

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS 216</td>
<td>B</td>
</tr>
<tr>
<td>142</td>
<td>CPS 214</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>CPS 216</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>CPS 130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>CPS 214</td>
<td>C</td>
</tr>
</tbody>
</table>
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’s a lossless join decomposition!

BCNF decomposition algorithm

• Find a BNCF 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 \( X \cup Z \) (\( Z \) contains all attributes of \( R \) that are in neither \( X \) nor \( Y \))
• Repeat until all relations are in BNCF

BCNF decomposition example

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

BCNF violation: \( \text{SID} \rightarrow \text{name}, \text{email} \)

\text{Student} (\text{SID}, \text{name}, \text{email})

BCNF

\text{Grade} (\text{SID}, \text{CID}, \text{grade})

BCNF

Another example

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

BCNF violation: \( \text{email} \rightarrow \text{SID} \)

\text{StudentID} (\text{email}, \text{SID})

BCNF

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

BCNF violation: \( \text{email} \rightarrow \text{name} \)

\text{StudentName} (\text{email}, \text{name})

BCNF

\text{Grade} (\text{email}, \text{CID}, \text{grade})

BCNF

Why is BCNF decomposition lossless

• Given non-trivial \( X \rightarrow Y \) in \( R \) where \( X \) is not a super key of \( R \), need to prove:
  – Anything we project always comes back in the join:
    \[ R \subseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R) \]
  • Sure; and it doesn’t depend on the FD
  – Anything that comes back in the join must be in the original relation:
    \[ R \supseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R) \]
Yet another example

- Address (street_address, city, state, zip)
  - street_address, city, state → zip
  - zip → city, state
- Keys
  - {street_address, city, state}
  - {street_address, zip}
- BCNF?
  - Violation: zip → city, state

To decompose, or not to decompose

Address₁ (zip, city, state)
Address₂ (street_address, zip)

- FDs in Address₁
  - zip → city, state
- FDs in Address₂
  - None!
- Hey, where is street_address, city, state → zip?
  - Cannot check it without joining Address₁ and Address₂ back together

“Elegant” solution

- Define the problem away!
- R is in Third Normal Form (3NF) if for every non-trivial FD X → A, either
  - X is super key of R, or
  - A is a member of at least one key of R
- So Address is already in 3NF
- Tradeoff:
  - Can check all FDs in the decomposed relations
  - Might have some redundancy due to FDs

Recap

- Identifying tuples: keys
- Generalizing the key concept: FDs
- Non-key FDs: redundancy
- Avoiding redundancy: BCNF decomposition
- Preserving FDs: 3NF

What’s next

- Another kind of dependency and normal form
- A comprehensive design example
- SQL basics