Functional dependencies, decompositions, normal forms

Introduction to databases
CSCC43 Winter 2011
Ryan Johnson

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:
  - Students(utorid, name, email) vs. Students(utorid, name, email)
    - Students(utorid, name) vs. Emails(utorid, address)

- Consider also:
  - House(street, city, value, owner, propertyTax) vs. House(street, city, value, owner)
    - TaxRates(city, value, propertyTax)

Dependencies, constraints are domain-dependent
Part I: Functional dependencies

Functional dependencies

• Let X, Y be sets of attributes from relation R
  • X -> 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 -> h, where h is the hash of attributes in X

• Notational conventions
  – “a”, “b”, “c” – specific attributes
  – “A”, “B”, “C” – sets of (unnamed) attributes
  – abc -> def – same as {a,b,c} -> {d,e,f}

Most common to see *singletons* (X -> y or abc -> d)

Splitting FDs

• Attributes on right independent of each other
  – Consider a,b,c -> 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 -> def becomes abc -> d, abc -> e and abc -> f

• No safe way to split left side
  – abc -> def is NOT the same as ab -> def and c -> 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
  - $A \rightarrow A$ always holds
  - $abc \rightarrow a$ always holds

- FD with an attribute on both sides is “trivial”
  - Simplify by removing $L \cap R$ from $R$
    - $abc \rightarrow ad$ becomes $abc \rightarrow d$
  - Or, in singleton form, delete trivial FDs
    - $abc \rightarrow a$ and $abc \rightarrow d$ becomes just $abc \rightarrow d$

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 $\rightarrow$ email, city, country, surname
  - email $\rightarrow$ city, country, surname
  - city $\rightarrow$ country
  - surname $\rightarrow$ country

  Domain knowledge required to validate FDs

Keys and FDs

- Consider relation $R$ with attributes $A$
- Superkey
  - Any $S \subseteq A$ s.t. $S \rightarrow A$
  =>$ \rightarrow$ 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 $X \subset C$
  =>$ \rightarrow$ A superkey which contains no other superkeys
  =>$ \rightarrow$ Remove any attribute and you no longer have a key

- Primary key
  - The candidate key we use to identify the relation
  =>$ \rightarrow$ Always exists, only one allowed, doesn’t matter which $C$ we use

- Prime attribute
  - $\exists$ candidate key $C$ 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: relases 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  B \rightarrow A
  - What does this say about A and B?
- Example
  - street, city \rightarrow value  city, value \rightarrow tax
  - studentID \rightarrow email  email \rightarrow studentID

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 \( X_1 \rightarrow a_1, X_2 \rightarrow a_2, \) 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
  \( \Rightarrow A^+_F = \) 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: \( A \rightarrow B \forall B \in A^+_F \)

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

_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**

  \[
  \begin{array}{cccccc}
  a & b & c & d & e & f \\
  a & b & c & d & e & f \\
  a & b & c & d & e & f \\
  \end{array}
  \]

  \( ab \rightarrow cdef -- ab \) is a candidate key!
Constructing a minimal basis

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

Part II: Schema decomposition

FDs and redundancy

- Given relation \( R \) and FDs \( F \)
  - \( R \) often exhibits anomalies due to redundancy
  - \( F \) identifies many (not all) of the underlying problems

- Idea
  - Use \( F \) to identify “good” ways to split relations
  - Split \( R \) into 2+ smaller relations having less redundancy
  - Split \( F \) into subsets which apply to the new relations

Schema decomposition

- Given relation \( R \) and FDs \( F \)
  - Split \( R \) into \( R_i \) s.t. \( \forall i \ R_i \subset R \) (no new attributes)
  - Split \( F \) into \( F_i \) s.t. \( \forall i \ F \text{ entails } F_i \) (no new FDs)
  - \( F_i \) involves only attributes in \( R_i \)

- Caveat: entirely possible to lose information
  - \( F^* \) may entail FD \( X \) which is not in \( (U, F_j)^* \)
    \( \Rightarrow \) Decomposition lost some FDs
  - Possible to have \( R \subset \gg_i R_i \)
    \( \Rightarrow \) Decomposition lost some relationships

- Goal: minimize anomalies without losing info

*We’ll revisit information loss in a moment*
Splitting relations – example

- Consider the following relation:

<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>CSJC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSJC08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSJC43</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

- One possible decomposition
  - Students(email, name)
  - Courses(name, instructor)
  - Taking(studentEmail, courseName)

Gotcha: lossy join decomposition

- Consider a relation with one more tuple

<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>CSJC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSJC08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSJC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mary</td>
<td>mary@utsc</td>
<td>CSJC08</td>
<td>Rosenberg</td>
</tr>
</tbody>
</table>

- Students ≿ Taking ≿ Courses has bogus tuples!
  - Mary is not taking Bretscher’s section of D08
  - Xiao is not in Rosenberg’s section of D08

Why did this happen? How to prevent it?

Ensuring lossless joins

- If we decompose R into S and T
- Either (S ∩ T) → S or (S ∩ T) → T must be in F⁺
- In our example:
  - ({email, course} ∩ {course, instructor}) = {course}
  - course →/→ instructor (one-many relationship)

Projecting FDs

- Once we’ve split a relation we have to refactor our FDs to match
  - Each FDs must only mention attributes from one relation
- Similar to geometric projection
  - Many possible projections (depends on how we slice it)
  - Keep only the ones we need (minimal basis)
FD projection algorithm

- Start with $F_i = \emptyset$
- For each subset $X$ of $R_i$
  - Compute $X^+$
  - For each attribute $a$ in $X^+$
    - If $a$ is in $R_i$
      - add $X \rightarrow a$ to $F_i$
- Compute the minimal basis of $F_i$
- Projection is expensive
  - Suppose $R_i$ has $n$ attributes
  - How many subsets of $R_i$ are there?
  - How many times do we consider each attribute?

Making projection more efficient

- Ignore trivial dependencies
  - No need to add $X \rightarrow A$ if $A$ is in $X$ itself
- Ignore trivial subsets
  - The empty set or of the set of all attributes (both are subsets of $X$)
- Ignore supersets of $X$ if $X^+ = R$
  - They can only give use “weaker” FDs (with more on the LHS)

Example: Projecting FD’s

- $ABC$ with FD’s $A \rightarrow B$ and $B \rightarrow C$. Project onto $AC$.
  - $A^+ = ABC$; yields $A \rightarrow B, A \rightarrow C$.
    - We do not need to compute $AB^+$ or $AC^+$.
  - $B^+ = BC$; yields $B \rightarrow C$.
  - $C^+ = C$; yields nothing.
  - $BC^+ = BC$; yields nothing.

Example -- Continued

- Resulting FD’s: $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$.
- Projection onto $AC$: $A \rightarrow C$.
  - Only FD that involves a subset of $\{A, C\}$.
- Projection on $BC$: $B \rightarrow C$.
  - Only FD that involves subset of $\{B, C\}$. 
Part III: Normal forms

Motivation for normal forms

• Identify a “good” schema
  – For some definition of “good”
  – Avoid anomalies, redundancy, etc.

• Several known normal forms
  – 1st
  – 2nd
  – 3rd
  – Boyce-Codd
  – … and several more we won’t discuss...

BCNF $\subseteq$ 3NF $\subseteq$ 2NF $\subseteq$ 1NF (focus on 3NF/BCNF)

1st normal form (1NF)

• No multi-valued attributes allowed
  – Imagine storing a list/set of things in an attribute
  => Not really even expressible in RA

• Counterexample
  – Course(name, instructor, [student,email]*)
  – Redundancy in non-list attributes

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

2nd normal form (2NF)

• Non-prime attributes depend on candidate keys
  – Consider non-prime attribute ‘a’
  – Then $\exists$ FD X s.t. X $\rightarrow$ a and X is a candidate key

• Counterexample
  – Movies(title, year, star, studioName, studioAddress, salary)
  – FD: title, year $\rightarrow$ studioName, studioAddress

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Star</th>
<th>StudioName</th>
<th>StudioAddr</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Hamill</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Fisher</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Ford</td>
<td>Paramount</td>
<td>Cloud 9</td>
<td>$2,000,000</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
3rd normal form (3NF)

- Non-prime attr. depend only on candidate keys
  - Consider FD \(X \rightarrow a\)
  - Either \(a \in X\) OR \(X\) is a superkey OR \(a\) is prime
  => No transitive dependencies allowed

Counterexample:
- \(\text{studioName} \rightarrow \text{studioAddr}\)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>StudioName</th>
<th>StudioAddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Paramount</td>
<td>Cloud 9</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
</tbody>
</table>

3NF, dependencies, and join loss

- Theorem: always possible to convert a schema to join-lossless, dependency-preserving 3NF
- Caveat: still possible to create schemas in 3NF for which these properties do not hold
- Lost dependencies
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)
  => Unable to enforce studioName \(\rightarrow\) studioAddress
- Lossy joins
  - Movies(title, year, star)
  - StarSalary(star, salary)
  => Movies< StarSalary yields bogus tuples

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FD have superkey LHS
- Counterexample
  - CanadianAddress(street, city, province, postalCode)
  - Candidate keys: \{street, postalCode\}, \{street, city, province\}
  - FD: postalCode \(\rightarrow\) city, province
  - Satisfies 3NF: city, province both non-prime
  - Violates BCNF: postalCode is not a superkey
  => Possible anomalies involving postalCode

Do we care? How often do postal codes change?

Limits of decomposition

- Pick two...
  - Lossless
  - Dependency join
  - Anomaly-free
- 3NF
  - Always allows join lossless and dependency preserving
  - May allow some anomalies
- BCNF
  - Always excludes anomalies
  - May give up one of join lossless or dependency preserving

Use domain knowledge to choose 3NF vs. BCNF