Database Design and Normalization

CPS352: Database Systems

Simon Miner
Gordon College
Last Revised: 9/27/12
Agenda

- Check-in
- Functional Dependencies (continued)
- Design Project E-R Diagram Presentations
- Database Normalization
- Homework 3
Check-in
Functional Dependencies
Database Design Goal

- Decide whether a particular relation $R$ is in “good” form.
  - Middle ground between the universal relation and relations which suffer from lossy join

- In the case that a relation $R$ is not in “good” form, decompose it into a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition

- Our theory is based on:
  - functional dependencies
  - database normal forms
  - multivalued dependencies
Functional Dependency (FD)

- When the value of a certain set of attributes uniquely determines the value for another set of attributes
  - Generalization of the notion of a key
  - A way to find “good” relations
  - $A \rightarrow B$ (read: $A$ determines $B$)

- Formal definition
  - For some relation scheme $R$ and attribute sets $A$ ($A \subseteq R$) and $B$ ($B \subseteq R$)
  - $A \rightarrow B$ if for any legal relation on $R$
    - If there are two tuples $t_1$ and $t_2$ such that $t_1(A) = t_2(A)$
    - It must be the case that $t_2(A) = t_2(B)$
Finding Functional Dependencies

- From keys of an entity
  - Primary and candidate keys

- From relationships between entities
  - One to one, one to many/many to one, and many to many relationships

- Implied functional dependencies
Implied Functional Dependencies

• Initial set of FDs *logically implies* other FDs
  • If $A \rightarrow B$ and $B \rightarrow C$, then $B \rightarrow C$

• Closure
  • If $F$ is the set of functional dependencies we develop from the logic of the underlying reality
  • Then $F^{+}$ (the *transitive closure* of $F$) is the set consisting of all the dependencies of $F$, plus all the dependencies they imply
Rules for Computing F+

- We can find $F^+$, the closure of $F$, by repeatedly applying Armstrong’s Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$  \hspace{1cm} (reflexivity)
  - Trivial dependency
    - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ \hspace{1cm} (augmentation)
    - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ \hspace{1cm} (transitivity)

- Additional rules (inferred from Armstrong’s Axioms)
  - If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta \gamma$ \hspace{1cm} (union)
  - If $\alpha \rightarrow \beta \gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ \hspace{1cm} (decomposition)
  - If $\alpha \rightarrow \beta$ and $\gamma \beta \rightarrow \delta$, then $\alpha \gamma \rightarrow \delta$ \hspace{1cm} (pseudotransitivity)
Applying the Axioms

- \( R = (A, B, C, G, H, I) \)
  \( \begin{align*}
    F &= \{ \ A \rightarrow B, \\
             & \quad A \rightarrow C, \\
             & \quad CG \rightarrow H, \\
             & \quad CG \rightarrow I, \\
             & \quad B \rightarrow H \}\end{align*} \)

- some members of \( F^+ \)
  - \( A \rightarrow H \)
    - by transitivity from \( A \rightarrow B \) and \( B \rightarrow H \)
  - \( AG \rightarrow I \)
    - by augmenting \( A \rightarrow C \) with \( G \), to get \( AG \rightarrow CG \)
      and then transitivity with \( CG \rightarrow I \)
  - \( CG \rightarrow HI \)
    - by augmenting \( CG \rightarrow I \) to infer \( CG \rightarrow CGI \),
      and augmenting of \( CG \rightarrow H \) to infer \( CGI \rightarrow HI \),
      and then transitivity
  - or by the union rule
Algorithm to Compute $F^+$

- To compute the closure of a set of functional dependencies $F$:

$$F^+ = F$$

repeat
  for each functional dependency $f$ in $F^+$
    apply reflexivity and augmentation rules on $f$
    add the resulting functional dependencies to $F^+$
  for each pair of functional dependencies $f_1$ and $f_2$ in $F^+$
    if $f_1$ and $f_2$ can be combined using transitivity
      then add the resulting functional dependency to $F^+$
  until $F^+$ does not change any further
Algorithm to Compute the Closure of Attribute Sets

• Given a set of attributes $\alpha$, define the closure of $\alpha$ under $F$ (denoted by $\alpha^+$) as the set of attributes that are functionally determined by $\alpha$ under $F$

• Algorithm to compute $\alpha^+$, the closure of $\alpha$ under $F$

\[
\text{result} := \alpha; \\
\text{while (changes to result) do} \\
\text{for each } \beta \rightarrow \gamma \text{ in } F \text{ do} \\
\quad \text{begin} \\
\quad \quad \text{if } \beta \subseteq \text{result} \text{ then } \text{result} := \text{result} \cup \gamma \\
\quad \text{end}
\]
## Example of Attribute Set Closure

- \( R = (A, B, C, G, H, I) \)
- \( F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\} \)

- \((AG)^+\)
  1. result = AG
  2. result = ABCG \((A \rightarrow C \text{ and } A \rightarrow B)\)
  3. result = ABCGH \((CG \rightarrow H \text{ and } CG \subseteq AGBC)\)
  4. result = ABCGHI \((CG \rightarrow I \text{ and } CG \subseteq AGBCH)\)

- Is AG a candidate key?
  1. Is AG a super key?
     1. Does AG \(\rightarrow R? \iff \text{Is } (AG)^+ \supseteq R\)
     2. Is any subset of AG a superkey?
        1. Does A \(\rightarrow R? \iff \text{Is } (A)^+ \supseteq R\)
        2. Does G \(\rightarrow R? \iff \text{Is } (G)^+ \supseteq R\)
Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: \( A \rightarrow C \) is redundant in: \( \{A \rightarrow B, \ B \rightarrow C, \ A \rightarrow C\} \)
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: \( \{A \rightarrow B, \ B \rightarrow C, \ A \rightarrow CD\} \) can be simplified to \( \{A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D\} \)
    - E.g.: on LHS: \( \{A \rightarrow B, \ B \rightarrow C, \ AC \rightarrow D\} \) can be simplified to \( \{A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D\} \)

- Intuitively, a canonical cover of \( F \) is a “minimal” set of functional dependencies equivalent to \( F \), having no redundant dependencies or redundant parts of dependencies
Definition of Canonical Cover

- **A canonical cover** for $F$ is a set of dependencies $F_c$ such that
  - $F$ logically implies all dependencies in $F_c$ and
  - $F_c$ logically implies all dependencies in $F$, and
  - No functional dependency in $F_c$ contains an extraneous attribute, and
  - Each left side of functional dependency in $F_c$ is unique.

- To compute a canonical cover for $F$:
  - **repeat**
    - Use the union rule to replace any dependencies in $F$
      - $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$
    - Find a functional dependency $\alpha \rightarrow \beta$ with an extraneous attribute either in $\alpha$ or in $\beta$
      - /* Note: test for extraneous attributes done using $F_c$, not $F$/
    - **until** $F$ does not change
  - Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied
Finding a Canonical Cover

• Another algorithm
  • Write F as a set of dependencies where each has a single attribute on the right hand side
  • Eliminate trivial dependencies
    • In which $\alpha \rightarrow \beta$ and $\beta \subseteq \alpha$
  • Eliminate redundant dependencies (implied by other dependencies)
  • Combine dependencies with the same left hand side

• For any given set of FDs, the canonical cover is not necessarily unique
Uses of Functional Dependencies

- Testing for lossless-join decomposition
- Testing for dependency preserving decompositions
- Defining keys
Testing for Lossless-Join Decomposition

- The closure of a set of FDs can be used to test if a decomposition is lossless-join.

- For the case of $R = (R_1, R_2)$, we require that for all possible relations $r$ on schema $R$
  
  $$ r = \Pi_{R_1}(r) \ \Pi_{R_2}(r) $$

- A decomposition of $R$ into $R_1$ and $R_2$ is lossless join if at least one of the following dependencies is in $F^+$:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

- Does the intersection of the decomposition satisfy at least one FD?
Testing for Dependency Preserving Decompositions

- The closure of a set of FDs allows us to test a new tuple being inserted into a table to see if it satisfies all relevant FDs without having to do a join
  - This is desirable because joins are expensive

- Let $F_i$ be the set of dependencies $F^+$ that include only attributes in $R_i$
  - A decomposition is **dependency preserving**, if
    \[ (F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+ \]
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

- The closure of a dependency preserving decomposition equals the closure of the original set

- Can all FDs be tested (either directly or by implication) without doing a join?
Keys and Functional Dependencies

- Given a relation scheme R with attribute set $K \subseteq R$
  - $K$ is a superkey if $K \rightarrow R$
  - $K$ is a candidate key if there is no subset $L$ of $K$ such that $L \rightarrow R$
    - A superkey with one attribute is always a candidate key
    - Primary key is the candidate key $K$ chosen by the designer
- Every relation must have a superkey (possibly the entire set of attributes)
- *Key attribute* – an attribute that is or is part of a candidate key
Design Project Presentations

Part 2: E-R Diagrams
Database Normalization
Database Design Goals (Updated)

• Goals
  • Avoid redundancies and the resulting from insert, update, and delete anomalies by decomposing schemes as needed
  • Ensure that all decompositions are lossless-join
  • Ensure that all decompositions are dependency preserving

• Sometimes you cannot have all three
  • Allow for redundancy to preserve dependencies
  • Or give up dependency preservation to eliminate redundancy
  • Never give up lossless-join as doing so would remove the ability to connect tuples in different relations

• Database normal forms help eliminate redundancy and anomalies
  • Specify a set of decomposition rules to convert a database that is not in a given normal form into one that is
First Normal Form (1NF)

- A relation scheme $R$ is in 1NF if the domains of all attributes in $R$ are atomic
  - Single and non-composite
  - Guarantees that each non-key attribute in $R$ is functionally dependent on the primary key
Second Normal Form (2NF)

- A 1NF relationship scheme \( R \) is in 2NF if each non-key attribute is fully functionally dependent on each candidate key.
  - Functionally dependent on the whole key, not just part of it
    - This restriction does not apply to key attributes
  - Avoids redundancy of information which is dependent on part of the primary key
- Any non-2NF scheme can be decomposed into 2NF schemes by factoring out
  - The non-key attributes dependent on a portion of a candidate key
  - The portion of the candidate key these attributes depend on
- Any 1NF scheme without a composite primary is in 2NF
Third Normal Form (3NF)

- A 2NF relation scheme R is in 3NF if no non-key attribute of R is transitively dependent on a candidate key through some other non-key attribute(s)
  - This restriction does not apply to key attributes
  - Transitive dependencies on a candidate key lead to insert, update, and delete anomalies

- Any non-3NF scheme can be decomposed into 3NF schemes by factoring out
  - The transitively dependent attributes
  - The “transitional” attributes which connect these to the candidate key

- Any non-3NF relation can be decomposed into 3NF in a lossless-join and dependency preserving manner
3NF Decomposition Algorithm

Let $F_c$ be a canonical cover for $F$;

$i := 0$;

for each functional dependency $\alpha \rightarrow \beta$ in $F_c$ do

if none of the schemas $R_j, 1 \leq j \leq i$ contains $\alpha \beta$

then begin

\begin{align*}
  i &:= i + 1; \\
  R_i &:= \alpha \beta
\end{align*}

end

if none of the schemas $R_j, 1 \leq j \leq i$ contains a candidate key for $R$

then begin

\begin{align*}
  i &:= i + 1; \\
  R_i &:= \text{any candidate key for } R;
\end{align*}

end

/* Optionally, remove redundant relations */

repeat

if any schema $R_j$ is contained in another schema $R_k$

then /* delete $R_j$ */

\begin{align*}
  R_j &:= R_k; \\
  i &:= i - 1;
\end{align*}

return $(R_1, R_2, \ldots, R_i)$
Boyce-Codd Normal Form (BCNF)

- 3NF did not take multiple candidate keys into account
  - BCNF developed to address this

- A normalized relation is in BCNF if every FD satisfied by R is of the form $A \rightarrow B$, where $A$ is a superkey
  - BCNF is a stronger 3NF
  - Every BCNF schema is also in 3NF
  - Not every 3NF schema is in BCNF

- Some 3NF schemas cannot be decomposed into BCNF in a lossless-join and dependency preserving manner

- BCNF does not build on other normal forms
BCNF Decomposition Algorithm

result := \{R\};
done := false;
compute \(F^+\);
while (not done) do
  if (there is a schema \(R_i\) in result that is not in BCNF)
    then begin
      let \(\alpha \rightarrow \beta\) be a nontrivial functional dependency that holds on \(R_i\) such that \(\alpha \rightarrow R_i\) is not in \(F^+\),
      and \(\alpha \cap \beta = \emptyset\);
      \(result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)\);
    end
  else done := true;

Note: each \(R_i\) is in BCNF, and decomposition is lossless-join.
Multivalued Dependencies (MVDs)

- A set of attributes $A$ multi-determines a set of attributes $B$ if
  - In any relation including attributes $A$ and $B$
  - For any given value of $A$ there is a (non-empty) set of values for $B$
  - Such that we expect to see all of those $B$ values (and no others) associated with the given $A$
  - Along with remaining attribute values
  - The number of $B$ values associated with a given $A$ value may vary between $A$ values.
Formal Definition of Multivalued Dependency

• Let \( R \) be a relation schema and let \( \alpha \subseteq R \) and \( \beta \subseteq R \). The **multivalued dependency**

\[
\alpha \rightarrow \rightarrow \beta
\]

holds on \( R \) if in any legal relation \( r(R) \), for all pairs for tuples \( t_1 \) and \( t_2 \) in \( r \) such that \( t_1[\alpha] = t_2[\alpha] \), there exist tuples \( t_3 \) and \( t_4 \) in \( r \) such that:

\[
\begin{align*}
    t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\
    t_3[\beta] &= t_1[\beta] \\
    t_3[R - \beta] &= t_2[R - \beta] \\
    t_4[\beta] &= t_2[\beta] \\
    t_4[R - \beta] &= t_1[R - \beta]
\end{align*}
\]
MVDs and E-R Diagrams

- MVDs correspond to multi-valued attributes

A → B
A ➔ C
Properties of MVDs

- MVDs require the addition of certain tuples
  - Example: copies of a book with multiple authors
  - Opposite to FDs which prohibit certain tuples

- If $A \rightarrow B$, then $A \rightarrow\rightarrow B$
  - FDs are a special case of MVDs

- An MVD is trivial if either of the following is true
  - Its right-hand side is a subset of its left-hand side (just like FDs)
  - The union of its left- and right-hand sides is the entire scheme

- The closure $D^+$ of $D$ is the set of all FDs and MVDs implied by $D$
  - $D^+$ can be computed from the formal definitions of FD and MVD
  - Additional rules of inference (see Appendix C of *Database Systems Concepts*)
Fourth Normal Form (4NF)

- A relation schema $R$ is in **4NF** for all MVDs in $D^+$ of the form $\alpha \rightarrow \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - $\alpha$ is a superkey for schema $R$ (in which case it is an FD)
- If a relation is in 4NF it is in BCNF
- 4NF avoids redundancies introduced by MVDs
**4NF Decomposition Algorithm**

`result: = \{R\};`

`done := false;`

`compute D^+;`

Let $D_i$ denote the restriction of $D^+$ to $R_i$

**while (not done)**

**if** (there is a schema $R_i$ in `result` that is not in 4NF) **then**

**begin**

let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds

on $R_i$ such that $\alpha \rightarrow R_i$ is not in $D_i$, and $\alpha \cap \beta = \emptyset$;

`result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);`

**end**

**else** `done := true;`

Note: each $R_i$ is in 4NF, and decomposition is lossless-join
Database Design Guidelines

- Use the highest normal form possible
  - 4NF unless it is not dependency preserving
  - BCNF unless (in rare cases) it is not dependency preserving
  - 3NF otherwise – never need to compromise beyond this
  - Lower normal forms may be useful for efficiency purposes

- Use good keys
  - Every attribute should depend on the key, the whole key, and nothing but the key (BCNF)
  - Avoid composite keys (automatic 2NF)
    - Generate a unique single-attribute key if needed

- Factor out transitive dependencies (“sub-relations”) into their own schemes (3NF)

- Isolate MVDs to their own schema (4NF)
Approaches to Database Design

- Start with a universal relation and decompose it
  - The approach taken in this lecture
- Start with an E-R diagram
  - Modify it while you normalize it
  - Normalize it when converting it to a relational schema
Homework 3