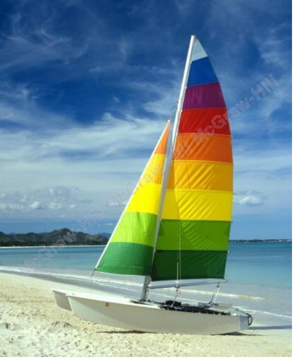


Topic 6: Functional Dependency and Normalization (Chapter 7)

Database System Concepts

(Modified for CS 4513)

Topic 6 Contents

- Integrity Constraints
- Functional Dependencies
- Relational Database Design: Features of a Good design
- Normalization: Decomposition using Functional Dependencies
- Database-Design Process

Integrity Constraints

- Domain constraints
  - Tested by the system whenever a new data item is inserted into the database
  - Comparisons must be made from compatible domains
  - Example:

Integrity Constraints (cont.)

- Referential Integrity
  - Ensures that a value that that appears in a relation for a given set of attributes also appear for a certain set of attributes in another relation
  - Is checked when database modification occurs.
  - Foreign key definition:
  - Example:

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.

Functional Dependencies (Cont.)

- Let R be a relation schema

$$\alpha \subseteq R \;\; and \;\; \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \;\; \Rightarrow \;\; t_1[\beta] = t_2[\beta]$$

- Notation $\alpha \rightarrow \beta$ **:** α *functionally determines* β, *or* β *is functionally dependent on* α
- Example: Consider r(A,B) with the following instance of r.

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Functional Dependencies (cont.)

- Another example:

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F.*
  - specify constraints on the set of legal relations
    - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy
    $name \rightarrow ID.$

Functional Dependencies (Cont.)

- Trivial FD*: A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example*:*
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
- Trivial FDs: automatically satisfied by all relations defined on R
  - Example*: Schema R (A, B, C)*
    - What are some trivial functional dependencies on R?
    - Answer:

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by $\mathbf{F^+}$.
- F^+ is a superset of *F*.

Closure of a Set of Functional Dependencies (Cont.)

- We can find F^{+}, the closure of F, by repeatedly applying **Armstrong's Axioms (rules of inference for FDs):**

  Given schema R and α, β, γ, and δ as subsets of R

  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity rule (trivial FD))**
  - if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$ **(augmentation rule)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity rule)**
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Closure of Functional Dependencies (Cont.)

- Additional inference rules:
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union rule**)
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition rule**)
  - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity rule**)

  The above rules can be inferred from Armstrong's axioms.

Example

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

Procedure for Computing 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

NOTE: We shall see an alternative procedure for this task later

How Keys are Related to Functional Dependencies?

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Example:

Functional Dependencies (Cont.)

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  inst_dept (*ID, name, salary, dept_name, building, budget*).

  We expect these functional dependencies to hold from the key constraint:

  ID, *dept_name* $\rightarrow$ name

  ID, *dept_name* $\rightarrow$ salary

  ID, *dept_name* $\rightarrow$ building

  ID, *dept_name* $\rightarrow$ budget

  ID, *dept_name* $\rightarrow$ ID

  ID, *dept_name* $\rightarrow$ dept_name

  but would not expect the following to hold from the key constraint unless it is specified:

  dept_name $\rightarrow$ *budget*

(meaning: each department has only one budget)

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*

- Algorithm to compute α^+, the closure of α under *F*

  result := α;
  while (changes to *result*) **do**
      for each $\beta \rightarrow \gamma$ **in** *F* **do**
        begin
          if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
        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. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^{+}, and check if α^{+} contains all attributes of *R*.
- Testing for candidate key:
  - To test if α is a candidate key, test if α is a superkey and minimal
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^{+}), just check if $\beta \subseteq \alpha^{+}$.
  - That is, we compute α^{+} by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$.

Relational Database Design

- Design Goal:
  - Generate a set of relations that allow data to be retrieved easily and allow data to be stored without unnecessary redundancy

- Properties of a bad design:
  - Unnecessary redundancy
  - Loss of data
  - Inability to represent some information

=> Design schemas that are in an appropriate normal form

Relational Database Design (Cont.)

- Normalization:
  - Process of decomposing a relation schema into smaller schemas
    - R => R_1, R_2,…, R_n
  - Objectives:
    - To reduce redundancy
    - To reduce database modification anomalies:
      - Insertion anomaly: inability to represent some information in the database
      - Deletion anomaly: deletion of some information causes loss of other information
      - Update anomaly: update one tuple requires updating many tuples

Desirable Properties of Decomposition (Cont.)

- 1) Lossless Join:
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

  $$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

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

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$

  $R_1 = (A, B) \quad R_2 = (B, C)$

A	*B*	*C*
α	1	A
β	2	B

r

A	*B*
α	*1*
β	*2*

$\Pi_{A,B}(r)$

B	*C*
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	*B*	*C*
α	1	A
β	2	B

Example of a Lossy Decomposition

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

name	*street*	*city*	*salary*
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Note: Additional tuples that were not in the original employee relation are called SPURIOUS TUPLES

Desirable Properties of Decomposition (Cont.)

- 2) Dependency Preserving:

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.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$ in F^+, i.e., $R_1 \cap R_2 \rightarrow R_2$ in F^+
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$ in F^+, i.e., $R_1 \cap R_2 \rightarrow R_1$ in F^+
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Normal Forms (NF)

- First Normal Form (1NF):
  - A schema R is in 1NF if every attribute in R is atomic (only single value, not divisible, no composite value)
  - Example:
    - Student (name, gpa, degree)
    - name: cannot be divided into first name and last name
    - degree: one degree only, cannot be divided into multiple degrees

Normal Forms (NF) (cont.)

- Second Normal Form (2NF): given a relation schema R and a set of functional dependencies F defined on R, R is in 2NF if
  - R is 1NF and
  - Every nonprime attribute in R is fully dependent on every candidate key of R
- Nonprime attribute in R: is not a subset of any candidate key of R
- Fully dependent:
  - Given $X \rightarrow Y \in F^+$
  - If $Z \subseteq X$ and $Z \rightarrow Y \in F^+$ then Y is partially dependent on X
  - If no such Z exists, then Y is fully dependent on X
  - Example:

Normal Forms (NF) (cont.)

- Example: Is the following schema student_class in 2NF, assuming (studentid, classid) is the only candidate key of the schema?

student_class (name, studentid, gpa, classid, grade)

name	studentid	gpa	classid	grade
Harris	1234	3.4	Physics_1A	A
Johnson	2346	3.1	Physics_1A	B
Sampson	1236	2.8	Chem_2B	A
Harris	1234	3.4	Chem_2B	A

Answer:

Normal Forms (NF) (cont.)

- If student_class is not in 2NF, describe the database modification anomalies and decompose it into 2NF schemas
- Answer:

Normal Forms (NF) (cont.)

- Third Normal Form (3NF): given a relation schema R and a set of functional dependencies F defined on R, R is in 3NF if
  - R is in 1NF and
  - For each X -> A in F^+ where X is a set of attributes in R and A is a single attribute in R then
    - Either X -> A is trivial FD or
    - X is a superkey of R or
    - A is a prime attribute of R
- Note: a prime attribute of R is a subset of a candidate key of R

Normal Forms (NF) (Cont.)

- Example: Is the following schema class_instructor in 3NF, assuming that classid is the only candidate key of class_instructor?

class_instructor (<u>classid</u> instid office)

classid	instid	office
Physics_1A	Smith	M11
Music_1	Harris	M22
Chem_2B	Parker	C12
Music_5	Harris	M22

- Answer:

Normal Forms (NF) (Cont.)

- If class_instructor is not in 3NF, describe the database modification anomalies and decompose it into 3NF schemas
- Answer:

Normal Forms (NF) (Cont.)

- Boyce-Codd Normal Form (BCNF): given a relation schema R and a set of functional dependencies F defined on R, R is in BCNF if
  - R is in 1NF and
  - For each X -> A in F^+ where X is a subset of attributes in R and A is a single attribute in R then
    - Either X -> A is trivial FD or
    - X is a superkey of R

Normal Forms (NF) (cont.)

- Example: given the following relational schema and rules, is the schema is in BCNF? If not, decompose it into BCNF schemas

  student_sport (student, sport, coach)

  Rules:

  1) Each student may participate in one or more sports;
  2) For each sport in which a student participates, he/she has a different coach;
  3) Each sport may have several coaches
  4) Each coach works with only one sport

Normal Forms (NF) (cont.)

- Answer:

BCNF Decomposition Algorithm

Given relational schema R and set of functional dependencies F defined on R

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 functional dependency that
                holds on R_i and violates BCNF
              result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
            end
      else *done* := **true;**

Note: each R_i in the final result is in BCNF, and decomposition is lossless-join; (the same algorithm is for 3NF decomposition when replacing "BCNF" with "3NF"). The algorithm does not guarantee dependency-preservation, but guarantees lossless join decomposition

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?
- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**).
  - Normalization breaks *R* into smaller relations.
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department_name* and *building*, and a functional dependency *department_name* $\rightarrow$ *building*
  - Good design would have made department an entity

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use the normalized schema, but additionally store a materialized view defined as the join of *course* and *prereq*:

  $$course \bowtie prereq$$

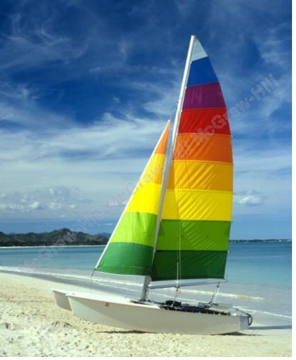
  - Materialized views: a view whose results is stored in the database and brought up to date (by the database system) when the relations used in the view are updated
  - Benefits and drawbacks are the same as in Alternative 1, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design to be avoided:

  Instead of *earnings* (*company_id, year, amount*), use

  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but makes querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools



End of Topic 6