Relational Database Design (II)

Roadmap of This Lecture

- Algorithms for Functional Dependencies (cont'd)
- Decomposition Using Multi-valued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data

Boyce-Codd Normal Form (Reminder)

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for *R*

Example schema *not* in BCNF:

instr_dept (*<u>ID,</u> name, salary, <u>dept_name,</u> building, budget*)

because *dept_name* $\rightarrow$ *building, budget*
holds on *instr_dept,* but *dept_name* is not a superkey

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of *R*, that is, α is a superkey of *R*.
- **Simplified test**: To check if a relation schema *R* is in BCNF, it suffices to check only the dependencies in the given set *F* for violation of BCNF, rather than checking all dependencies in F^+.
  - If none of the dependencies in *F* causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.
- However, **simplified test using only *F* is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$
    - Decompose *R* into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
    - Neither of the dependencies in *F* contain only attributes from (A,C,D,E) so we might be misled into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
  - or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency

    $$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$

    can be shown to hold on R_i, and R_i violates BCNF.
  - We use above dependency to decompose R_i

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.

Example of BCNF Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

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

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\qquad L \rightarrow K\}$
  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

Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (Reminder)

- A relation schema *R* is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R.*

  (**NOTE***:* each attribute may be in a different candidate key)

3NF Example

- Relation *dept_advisor*:
  - *dept_advisor (s_ID, i_ID, dept_name)*
    F = {*s_ID, dept_name* $\rightarrow$ *i_ID, i_ID* $\rightarrow$ *dept_name*}
  - Two candidate keys: *s_ID, dept_name,* and *i_ID, s_ID*
  - *R* is in 3NF
    - *s_ID, dept_name* $\rightarrow$ *i_ID*
      - *s_ID, dept_name* is a superkey
    - *i_ID* $\rightarrow$ *dept_name*
      - *dept_name* is contained in a candidate key

Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetition of information (e.g., the relationship l_1, k_1)
  - *(i_ID, dept_name)*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).
  - *s_ID* may be null if there is no separate relation mapping instructors to departments

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R*
  - this test is rather more expensive, since it involves finding candidate keys (typically this amounts to computing α^+ for every set of attributes $\alpha \subseteq R$)
  - testing for 3NF has been shown to be NP-hard
  - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

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
                    $i := i + 1$;
                    $R_i := \alpha\ \beta$
              end
if none of the schemas $R_j$, $1 \leq j \leq i$ contains a candidate key for $R$
  then begin
              $i := i + 1$;
              $R_i$ := any candidate key for $R$;
          end
/* Optionally, remove redundant relations */
repeat
   if any schema $R_j$ is contained in another schema $R_k$
      then begin /* delete $R_j$ */
          $R_j = R_k$;
          $i= i -1$;
          end
return ($R_1$, $R_2$, ..., $R_i$)
```

3NF Decomposition Algorithm (Cont.)

- Above algorithm ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join
  - Proof of correctness? (See textbook)

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (<u>*customer_id, employee_id*</u>, *branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* $\rightarrow$ *branch_name, type*
  2. *employee_id* $\rightarrow$ *branch_name*
  3. *customer_id, branch_name* $\rightarrow$ *employee_id*
- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id $\rightarrow$ *type*
    employee_id $\rightarrow$ *branch_name*
    customer_id, branch_name $\rightarrow$ *employee_id*

3NF Decompsition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

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
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.

Multivalued Dependencies

- Suppose we record names of children, and phone numbers for instructors:
  - *inst_child*(*ID*, *child_name*)
  - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
  - *inst_info*(*ID*, *child_name*, *phone_number*)
  - Example data:
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)
- This relation is in BCNF
  - Why?

Multivalued Dependencies (MVDs)

- Let *R* be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

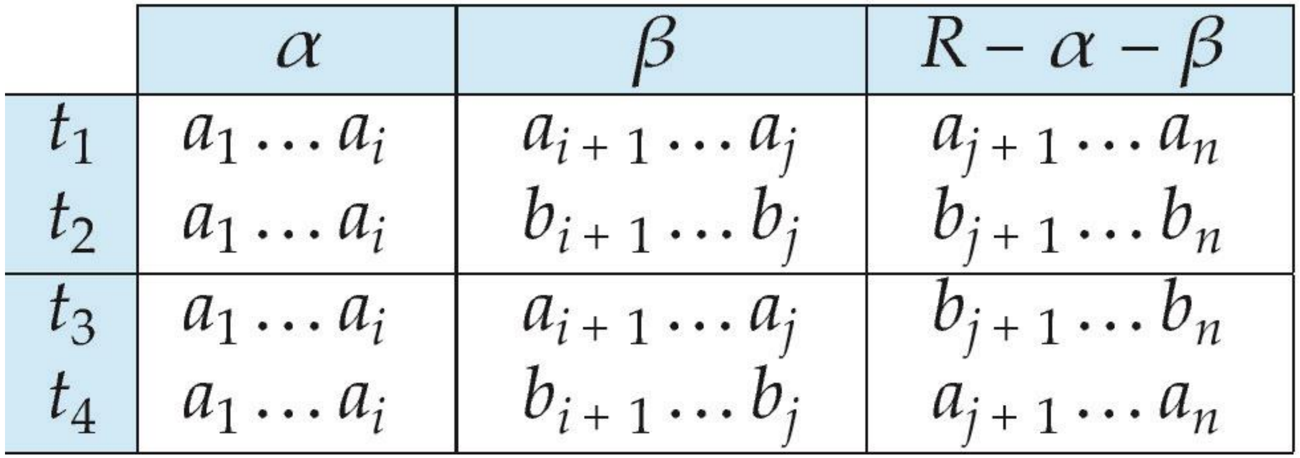
  $$\alpha \rightarrow\rightarrow \beta$$

  holds on *R* if in any legal relation *r(R),* for all pairs of 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:

  $$\begin{aligned} t_1[\alpha] = t_2[\alpha] &= t_3[\alpha] = t_4[\alpha] \\ t_3[\beta] &= t_1[\beta] \\ t_3[R - \beta] &= t_2[R - \beta] \\ t_4[\beta] &= t_2[\beta] \\ t_4[R - \beta] &= t_1[R - \beta] \end{aligned}$$

MVD (Cont.)

- Tabular representation of $\alpha \rightarrow\rightarrow \beta$



	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

- $\alpha \rightarrow\rightarrow \beta$ means relationship between α and β is independent of relationship between α and R - β.

Example

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

  $$Y, Z, W$$

- We say that $Y \rightarrow\rightarrow Z$ (Y **multidetermines** Z)
  if and only if for all possible relations $r(R)$

  $$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

  then

  $$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

- Note that since the behavior of Z and W are identical it follows that

  $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

Example (Cont.)

- In our example:

$$ID \rightarrow\!\!\!\rightarrow child\_name$$
$$ID \rightarrow\!\!\!\rightarrow phone\_number$$

- The above formal definition is supposed to formalize the notion that given a particular value of *Y* (*ID*) it has associated with it a set of values of *Z* *(child_name)* and a set of values of *W* *(phone_number)*, and these two sets are in some sense independent of each other.
- Note:
  - If $Y \rightarrow Z$ then $Y \rightarrow\!\!\!\rightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  2. To specify **constraints** on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r*′ that does satisfy the multivalued dependency by adding tuples to *r.*

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

  That is, every functional dependency is also a multivalued dependency
- The **closure** D^+ of *D* is the set of all functional and multivalued dependencies logically implied by *D*.
  - We can compute D^+ from *D*, using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules.

Inference rules for MVDs

1. **Reflexivity rule.** If *α is a set of attributes, and β ⊆ α, then α → β holds.*
2. **Augmentation rule.** If *α → β holds, and γ is a set of attributes, then γα → γβ* holds.
3. **Transitivity rule.** If *α → β holds, and β → γ holds, then α → γ holds***.**
4. **Complementation rule.** If *α →→ β holds, then α →→ R − β − α holds.*
5. **Multivalued augmentation rule.** If *α →→ β holds, and γ ⊆ R and δ ⊆ γ, then γα →→ δβ holds.*
6. **Multivalued transitivity rule.** If *α →→ β holds, and β →→ γ holds, then α →→ γ − β holds.*
7. **Replication rule.** If *α → β holds, then α →→ β.*
8. **Coalescence rule.** If *α →→ β holds, and γ ⊆ β, and there is a δ such that δ ⊆ R, and δ ∩ β = ∅, and δ → γ, then α → γ holds.*

Fourth Normal Form

- A relation schema *R* is in **4NF** with respect to a set *D* of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema *R*
- If a relation is in 4NF it is in BCNF

Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
  - All functional dependencies in D^+ that include only attributes of R_i
  - All multivalued dependencies of the form

    $\alpha \rightarrow\rightarrow (\beta \cap R_i)$

    where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ is in D^+

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 $\alpha \cap \beta = \phi$;
            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

Example

- $R = (A, B, C, G, H, I)$

  $D = \{ A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H \}$
- R is not in 4NF since $A \rightarrow\rightarrow B$ and A is not a super-key for R
- Decomposition

a) $R_1 = (A, B)$	(R_1 is in 4NF)
b) $R_2 = (A, C, G, H, I)$	(R_2 is not in 4NF, decompose into R_3 and R_4)
c) $R_3 = (C, G, H)$	(R_3 is in 4NF)
d) $R_4 = (A, C, G, I)$	(R_4 is not in 4NF, decompose into R_5 and R_6)
● $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI$	
gives $A \rightarrow\rightarrow HI$,	(MVD transitivity), and
● hence $A \rightarrow\rightarrow I$	(MVD restriction to R_4)
e) $R_5 = (A, I)$	(R_5 is in 4NF)
f) $R_6 = (A, C, G)$	(R_6 is in 4NF)

Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

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 → *building*
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a *relationship set* possible, but rare --- most relationships are binary

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 a materialized view defined as

  $course \bowtie prereq$
  - Benefits and drawbacks same as above, 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 make 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
    - Good for display only

Modeling Temporal Data

- **Temporal data** have an association time interval during which the data are *valid.*
- A **snapshot** is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like

  $ID \rightarrow street, city$

  not to hold, because the address varies over time
- A **temporal functional dependency** X → Y holds on schema *R* if the functional dependency X → Y holds on all snapshots for all legal instances r (*R*).

Modeling Temporal Data (Cont.)

- In practice, database designers may add start and end time attributes to relations
  - E.g., *course*(*course_id, course_title*) is replaced by

    course(*course_id, course_title, start, end*)
    - Constraint: no two tuples can have overlapping valid times
      - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g., student transcript should refer to course information at the time the course was taken

End