CMSC 461, Database Management Systems
Spring 2018

Lecture 14 - Chapter 8 Relational Database Design Wrap-up

These slides are based on “Database System Concepts” 6th edition book and are a modified version of the slides which accompany the book (http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html), in addition to the 2009/2012 CMSC 461 slides by Dr. Kalpakis

Dr. Jennifer Sleeman

https://www.csee.umbc.edu/~jsleem1/courses/461/spr18

Logistics

- Homework #3 due today 3/26/2018
- Phase 3 of project due 3/28/2018

Lecture Outline

- ***An Example***
- Functional Dependencies Review
- BCNF Decomposition
- 3NF Decomposition
- Multivalued Decomposition
- Fourth Normal Form
- Database Design Process

An Example

```
CREATE TABLE film_actor_nn (
film_actor_id BIGINT NOT NULL PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
film_name varchar(100),
producer varchar(50));
```

An Example

```
insert into film_actor_nn values (1,'Michael', 'Douglas',
'The American President', 'Rob Reiner');
insert into film_actor_nn values (2,'Michael', 'Douglas',
'BettleJuice','Larry Wilson');
insert into film_actor_nn values (3,'Michael', 'Douglas',
'Fatal Attraction','Stanley R. Jaffe');
```

An Example

SELECT film_actor_id, first_name, last_name, COUNT(*) FROM film_actor_nn GROUP BY film_actor_id ORDER BY COUNT(*) DESC;

```
+---------------+-------------+------------+-----------+
| film_actor_id | first_name | last_name |  COUNT(*)|
+---------------+-------------+------------+-----------+
|             2 | Michael    | Douglas   |         1 |
|             3 | Michael    | Douglas   |         1 |
|             1 | Michael    | Douglas   |         1 |
+---------------+-------------+------------+-----------+
3 rows in set (0.00 sec)
```

An Example

```
CREATE TABLE film_actor_nn (
film_actor_id BIGINT NOT NULL PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
film_name varchar(100),
producer varchar(50));
```

An Example

CREATE TABLE actor (
actor_id BIGINT NOT NULL PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL);

CREATE TABLE film (film_id BIGINT NOT NULL PRIMARY KEY,   film_name VARCHAR(50) NOT NULL, producer VARCHAR(50) NOT NULL);

CREATE TABLE film_actor (actor_id BIGINT NOT NULL REFERENCES actor(actor_id),   film_id BIGINT NOT NULL references film(film_id),   PRIMARY KEY (actor_id, film_id));

An Example

```
insert into actor values (1, 'Michael', 'Douglas');
insert into actor values (2, 'Michael', 'Douglas');

insert into film values (100, 'The American
President','Rob Reiner');
insert into film values (200, 'BettleJuice','Larry Wilson');

insert into film_actor values (1,100);
insert into film_actor values (2,200);
```

An Example

```
SELECT actor_id, first_name, last_name, COUNT(*)
FROM actor JOIN film_actor USING (actor_id) GROUP
BY actor_id ORDER BY COUNT(*) DESC;

+----------+------------+-----------+----------+
| actor_id | first_name | last_name | COUNT(*) |
+----------+------------+-----------+----------+
|        1 | Michael    | Douglas   |        1 |
|        2 | Michael    | Douglas   |        1 |
+----------+------------+-----------+----------+
```

An Example

```
insert into film values (300, 'Fatal Attraction','Stanley R. Jaffe');
insert into film_actor values (1,300);
```

An Example

```
SELECT actor_id, first_name, last_name, COUNT(*)
FROM actor JOIN film_actor USING (actor_id) GROUP
BY actor_id ORDER BY COUNT(*) DESC;

+----------+------------+-----------+----------+
| actor_id | first_name | last_name | COUNT(*) |
+----------+------------+-----------+----------+
|        1 | Michael    | Douglas   |        2 |
|        2 | Michael    | Douglas   |        1 |
+----------+------------+-----------+----------+
```

Lecture Outline

- An Example
- ***Functional Dependencies Review***
- BCNF Decomposition
- 3NF Decomposition
- Multivalued Decomposition
- Fourth Normal Form
- Database Design Process

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:
    Given a schema r(A,B,C)
    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 F^+
- F^+ is a superset of *F*

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of a Set of Functional Dependencies

- We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (**reflexivity**)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (**augmentation**)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (**transitivity**)
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Based on and image from "Database System Concepts" book and slides, 6th edition

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

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of a set of Functional Dependencies

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

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

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of a set of Functional Dependencies 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 union rule, since* $CG \rightarrow H$ *and* $CG \rightarrow I$, *implies* $CG \rightarrow HI$

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** **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

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of Attribute Sets Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  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 $CG \subseteq AGBCH$)

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of Attribute Sets Uses

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 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 $\gamma \rightarrow S$.

Based on and image from "Database System Concepts" book and slides, 6th edition

Closure of Attribute Sets Examples

- R = *(name,color,category,department,price)*
- F = {name $\rightarrow$ color
  category $\rightarrow$ department
  color,category $\rightarrow$ price}

You find:

- $name^+$
- $\{name, category\}^+$
- $\{color\}^+$

Based on and image from "Database System Concepts" book and slides, 6th edition

Lossless-join Decomposition

- 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^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

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 $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preservation

- 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.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to F)
  - *result* = α
    while (changes to *result*) do
    for each R_i in the decomposition
    $t = (result \cap R_i)^+ \cap R_i$
    $result = result \cup t$
  - If *result* contains all attributes in β, then the functional dependency
    $\alpha \rightarrow \beta$ is preserved.

Testing for Dependency Preservation

- We apply the test on all dependencies in *F* to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

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, it is a superkey of *R*.

Testing for BCNF

- **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.

Testing for BCNF

- 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 mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

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) ∪ ($R_i - \beta$) ∪ (α, β);
end
else *done* := **true;**

Note: each R_i is in BCNF, and decomposition is lossless-join.

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*

Example of BCNF Decomposition

- 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

- *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.

Based on "Database System Concepts" book and slides, 6th edition

BCNF and Dependency Preservation

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

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

Lecture Outline

- An Example
- Functional Dependencies Review
- BCNF Decomposition
- ***3NF Decomposition***
- Multivalued Decomposition
- Fourth Normal Form
- Database Design Process

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.

3NF Example

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

Based on “Database System Concepts” book and slides, 6th edition

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).
  - - (*i_ID, dept_name)* 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 β is contained in a candidate key of *R*
  - this test is rather more expensive, since it involve finding candidate keys
  - 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 α → β in F_c do
 if none of the schemas R_j, 1 ≤ j ≤ i contains α β
      then begin
              i := i + 1;
              R_i := α β
          end
if none of the schemas R_j, 1 ≤ j ≤ 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 /* delete R_j */
       R_j = R;;
       i=i-1;
return (R_1, R_2, ..., R_i)
```

3NF Decomposition Algorithm

- Above algorithm ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join
  - For proof of correctness see original slides that accompany book

3NF Decomposition: An Example

- Relation schema:
  cust_banker_branch = (*customer_id, employee_id*, *branch_name, type*)
- The functional dependencies for this relation schema are:
  - *customer_id, employee_id* → *branch_name, type*
  - *employee_id* → *branch_name*
  - *customer_id, branch_name* → *employee_id*

3NF Decomposition: An Example

- 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 → *type*

    employee_id → *branch_name*

    customer_id, branch_name → *employee_id*

3NF Decomposition Example

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

  (*customer_id, employee_id, type*)

  (*<u>employee_id</u>, 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

3NF Decomposition Example

- 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

Design Goals

- 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.

Lecture Outline

- An Example
- BCNF Decomposition
- 3NF Decomposition
- ***Multivalued Decomposition***
- Fourth Normal Form
- Database Design Process

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

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 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:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

Multivalued Dependencies (MVDs)

- 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	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Example

- In our example:
  ID →→ *child_name*
  ID →→ *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.

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

Theory of MVDs

- 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

Theory of MVDs

*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,

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

1	4
1	5
3	7

- 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:

  $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$

  $t_3[\beta] = t_1[\beta]$

  $t_3[R - \beta] = t_2[R - \beta]$

  $t_4[\beta] = t_2[\beta]$

  $t_4[R - \beta] = t_1[R - \beta]$

Lecture Outline

- An Example
- Review for Midterm
- BCNF Decomposition
- 3NF Decomposition
- Multivalued Decomposition
- ***Fourth Normal Form***
- Database Design Process

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 $\alpha \rightarrow\rightarrow \beta$, 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 $\alpha \cup \beta = R$)
  - 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 $\mathbf{R_i}$ 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 $\alpha \cap \beta = \varphi$;
result := (*result* - R_i) ∪ (R_i - β) ∪ (α, β);
end
else *done*:= true;
Note: each R_i is in 4NF, and decomposition is lossless-join

Example

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

$F = \{ A \rightarrow\rightarrow B$

$B \rightarrow\rightarrow HI$

$CG \rightarrow\rightarrow H \}$

R is not in 4NF since $A \rightarrow\rightarrow B$ and A is not a superkey for R

Decomposition

$R_1 = (A, B)$ (R_1 is in 4NF)

$R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF, decompose into R_3 and R_4)

$R_3 = (C, G, H)$ (R_3 is in 4NF)

$R_4 = (A, C, G, I)$ (R_4 is not in 4NF, decompose into R_5 and R_6)

- $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI \Rightarrow A \rightarrow\rightarrow HI$, (MVD transitivity), and
- and hence $A \rightarrow\rightarrow I$ *(MVD restriction to R_4)*

$R_5 = (A, I)$ (R_5 is in 4NF)

$R_6 = (A, C, G)$ (R_6 is in 4NF)

Lecture Outline

- An Example
- Functional Dependencies Review
- BCNF Decomposition
- 3NF Decomposition
- Multivalued Decomposition
- Fourth Normal Form
- ***Database Design Process***

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.

ER Model and 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

ER Model and Normalization

- 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*

Denormalization for Performance

- 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

Denormalization for Performance

- 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

Other Design Issues

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

Other Design Issues

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