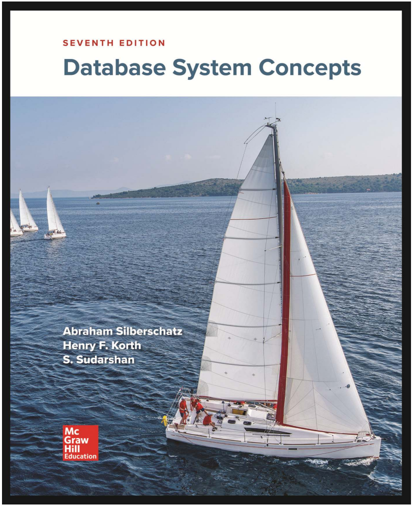
Database System Concepts, 7^{th} Edition
Chapter 7: Relational Database Design

Silberschatz, Korth and Sudarshan

October 21, 2025

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.

Plan

Features of Good Relational Designs

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

Figure 7.1 Database schema for the university example.

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep*, which represents the natural join on the relations *instructor* and *department*.

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Figure 7.2 The *in_dep* relation.

- There is repetition of information
- Need to use `null` values (if we add a new department with no instructors)

Decomposition

- The only way to avoid the repetition-of-information problem in the *in_dep* schema is to decompose it into two schemas –*instructor* and *department* schemas.
- Not all decompositions are good. Suppose we decompose:
  ```
  employee(ID, name, street, city, salary)
  ```
  into
  ```
  employee1 (ID, name)
  employee2 (name, street, city, salary)
  ```
  The problem arises when we have two employees with the same name.
- The next slide shows how we lose information –we cannot reconstruct the original employee relation– and so, this is a **lossy decomposition**.

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
⋮				

Figure 7.3 Loss of information via a bad decomposition.

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R. That is $R = R_1 \bigcup R_2$.
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \bigcup R_2$.
- Formally,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$:

$$R_1 = (A, B); R_2 = (B, C)$$

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

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

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B	C
α	1	A
β	2	B

Normalization Theory

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into set of relations $\{R_1, R_2, \cdots, R_n\}$ such that,
  - Each relation is in good form.
  - The decomposition is a lossless decomposition.
- Our theory is based on:
  1. functional dependencies.
  2. multivalued dependencies.

Plan

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID.
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.

Functional Dependencies (Cont.)

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation.
- A legal instance of a database is one where all the relation instances are legal instances.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a key.

Functional Dependencies Definition

- Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- 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,

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

- Example: Consider $r(A, B)$ with the following instance of r:

A	B
1	4
1	5
3	7

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

Closure of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
  - 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^+.

Plan

Keys and Functional Dependencies

- K is a superkey for relation schema R if and only if $K \to R$.
- K is a primary key for R if and only if:
  - $K \to R$, and
  - for no $\alpha \subset K, \alpha \to R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  `in_dep(ID, name, salary, dept_name, building, budget).`

  We expect these functional dependencies to hold:

$$dept\_name \to building$$
$$ID \to building$$

  but would not expect the following to hold:

$$dept\_name \to salary$$

Use of Functional Dependencies

- We use functional dependencies to:
  - 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.
  - To 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

$$name \rightarrow ID.$$

Trivial Functional Dependencies

- 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 $\beta \subseteq \alpha$.

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

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

$$R_1 \bigcap R_2 \rightarrow R_1$$

$$R_1 \bigcap R_2 \rightarrow R_2$$

- 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[1]

- $R = (A, B, C)$
$$F = \{A \rightarrow B \\ B \rightarrow C\}$$
- $R_1 = (A, B)$; $R_2 = (B, C)$
Lossless decomposition:
$R_1 \bigcap R_2 = \{B\}$ and $B \rightarrow BC$ holds.
- $R_1 = (A, B)$; $R_2 = (A, C)$
Lossless decomposition:
$R_1 \bigcap R_2 = \{A\}$ and $A \rightarrow AC$ holds.

[1] Note: $B \rightarrow BC$ is a shorthand notation for $B \rightarrow \{B, C\}$.

Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly,
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low.
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Produced.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**.

Dependency Preservation Example

- Consider a schema:

  `dept_advisor(s_ID, i_ID, department_name)`

- With function dependencies:

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*.
- Any decomposition will not include all the attributes in:

$$s\_ID, dept\_name \rightarrow i\_ID$$

- Thus, the composition NOT be dependency preserving.

Plan

Boyce-Codd Normal Form

- 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.

Boyce-Codd Normal Form (Cont.)

- Example schema that is not in BCNF:

  `in_dep (ID, name, salary, dept_name, building, budget)`

  because :
  - $dept\_name \rightarrow building, budget$
    - holds on `in_dep`
    - but...
  - $dept\_name$ is not a superkey.
- When decompose `in_dept` into `instructor` and `department`:
  - `instructor` is in BCNF.
  - `department` is in BCNF.

Decomposing a Schema into BCNF

- Let R be a schema that is not in BCNF. Let $\alpha \rightarrow \beta$ be the Fuctional Dependency that causes a violation of BCNF.
- We decompose R into:
  - $(\alpha \bigcup \beta)$
  - $(R - (\beta - \alpha))$
- In our example of `in_dep`,
  - $\alpha = dept\_name$
  - $\beta = building, budget$

  and `in_dep` is replaced by
  - $(\alpha \bigcup \beta) = (dept\_name, building, budget)$
  - $(R - (\beta - \alpha)) = (ID, name, salary, dept\_name)$

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation.
- Consider a schema:

  `dept_advisor(s_ID, i_ID, department_name)`

- With function dependencies:

$$\begin{aligned} i\_ID \rightarrow &\quad dept\_name \\ s\_ID, dept\_name \rightarrow &\quad i\_ID \end{aligned}$$

- `dept_advisor` is not in BCNF:
  - *i_ID* is not a superkey.
- Any decomposition of `dept_advisor` will not include all the attributes in:

$$s\_ID, dept\_name \rightarrow \quad i\_ID$$

- Thus, the composition is NOT be dependency preserving.

Third Normal Form

- 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[2] for R.
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

[2] NOTE: each attribute may be in a different candidate key.

3NF Example

- Consider a schema:

  `dept_advisor(s_ID, i_ID, dept_name)`

- With function dependencies:

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

- Two candidate keys = $\{s\_ID, dept\_name\}$, $\{s\_ID, i\_ID\}$
- We have seen before that `dept_advisor` is not in BCNF.
- R, however, is in 3NF:
  - s_ID, $dept\_name$ is a superkey,
  - $i\_ID \rightarrow dept\_name$ and i_ID is NOT a superkey, but:
    - $dept\_name - i\_ID = dept\_name$ and
    - $dept\_name$ is contained in a candidate key.

Redundancy in 3NF

- Consider the schema R below, which is in 3NF:
  - $R = (J, K, L)$,
  - $F = \{JK \rightarrow L, L \rightarrow K\}$,
  - And an instance table:

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

- What is wrong with the table?

Redundancy in 3NF

- Consider the schema R below, which is in 3NF:
  - $R = (J, K, L)$,
  - $F = \{JK \rightarrow L, L \rightarrow K\}$,
  - And an instance table:

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

- What is wrong with the table?
  - Repetition of information,
  - Need to use `null` values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
  - We may have to use `null` values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in "good" form.
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, \cdots, R_n\}$ such that:
  - Each relation scheme is in good form,
  - The decomposition is a lossless decomposition,
  - Preferably, the decomposition should be dependency preserving.

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized...
- Consider a relation:

  `inst_info(ID, child_name, phone)`

  where an instructor may have more than one phone and can have multiple children.
- Instance of `inst_info`

ID	**child_name**	**phone**
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF.
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples:

  (99999, David, 981-992-3443)
  (99999, William, 981-992-3443)

Higher Normal Forms

- It is better to decompose `inst_info` into:
  - `inst_child`

ID	**child_name**
99999	David
99999	William

  - `inst_phone`

ID	**phone**
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later

Plan

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF.
- We then develop algorithms to test if a decomposition is dependency-preserving.

Closure of a Set of Functional Dependencies

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

Closure of a Set of Functional Dependencies

- We can compute F^+, the closure of F, by repeatedly applying **Armstrong's Axioms**:
  - **Reflexive rule**: if $\beta \subseteq \alpha$, then $\alpha \to \beta$,
  - **Augmentation rule**: if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$,
  - **Transitivity rule**: if $\alpha \to \beta$, and $\beta \to \gamma$, then $\alpha \to \gamma$.
- These rules are:
  - **sound** – generate only functional dependencies that actually hold, and
  - **complete** – generate all functional dependencies that hold.

Example of F^+

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

$$\begin{aligned} F = \{A &\rightarrow B \\ A &\rightarrow C \\ CG &\rightarrow H \\ CG &\rightarrow I \\ B &\rightarrow H\} \end{aligned}$$

- 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.

Closure of a Set of Functional Dependencies (Cont.)

- Additional rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
  - **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
- The above rules can be inferred from Armstrong's axioms.

Procedure for Computing F^+

- To compute[3] the closure of a set of functional dependencies F:

$F^+ = F$
apply the reflexivity rule /* Generates all trivial dependencies */
repeat
    for each functional dependency f in F^+
        apply the augmentation rule 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
            add the resulting functional dependency to F^+
until F^+ does not change any further

Figure 7.7 A procedure to compute F^+.

[3] NOTE: We shall see an alternative procedure for this task later.

Plan

Multivalued Dependencies (MVDs)

- 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?**

MVDs

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

$$\alpha \twoheadrightarrow \beta$$

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:

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

MVDs (Cont.)

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

Figure 7.13 Tabular representation of $\alpha \twoheadrightarrow \beta$.

MVDs (Cont.)

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets: Y, Z, W.
- We say that $Y \twoheadrightarrow Z$ (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

$$Y \twoheadrightarrow Z \text{ if } Y \twoheadrightarrow W$$

Example: MVDs

- In our example:
  $ID \twoheadrightarrow child\_name$
  $ID \twoheadrightarrow phone\_number$
- 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 $Y \twoheadrightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

Use of MVDs

- 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 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:

  $$\text{If } \alpha \to \beta, \text{ then } \alpha \twoheadrightarrow \beta$$

  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 (Section 28.1.1).

Plan

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 $\alpha \bigcup \beta = R$).
  - α is a superkey for schema R.
- If a relation is in 4NF it is in BCNF.

Restriction of MVDs

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:

$$\alpha \twoheadrightarrow (\beta \bigcap R_i)$$

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

4NF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute D^+; Given schema R_i, let D_i denote the restriction of D^+ to R_i
while (**not** *done*) **do**
  if (there is a schema R_i in *result* that is not in 4NF w.r.t. D_i)
    then begin
      let $\alpha \twoheadrightarrow \beta$ 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;

Figure 7.16 4NF decomposition algorithm.

Example: 4NF Decomposition Algorithm

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

$$\begin{aligned} F = \{A &\twoheadrightarrow B \\ B &\twoheadrightarrow HI \\ CG &\twoheadrightarrow H\} \end{aligned}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
    1. $R_1 = (A, B)$ [R_1 is in 4NF]

Example: 4NF Decomposition Algorithm

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

$$\begin{aligned} F = \{A &\twoheadrightarrow B \\ B &\twoheadrightarrow HI \\ CG &\twoheadrightarrow H\} \end{aligned}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]

Example: 4NF Decomposition Algorithm

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

$$F = \{A \twoheadrightarrow B \\ B \twoheadrightarrow HI \\ CG \twoheadrightarrow H\}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]
  3. $R_3 = (C, G, H)$ [R_3 is in 4NF]

Example: 4NF Decomposition Algorithm

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

$$\begin{aligned} F = \{A &\twoheadrightarrow B \\ B &\twoheadrightarrow HI \\ CG &\twoheadrightarrow H\} \end{aligned}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]
  3. $R_3 = (C, G, H)$ [R_3 is in 4NF]
  4. $R_4 = (A, C, G, I)$ [R_4 is not in 4NF, decompose into R_5 and R_6]

Example: 4NF Decomposition Algorithm

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

$$\begin{aligned} F = \{A &\twoheadrightarrow B \\ B &\twoheadrightarrow HI \\ CG &\twoheadrightarrow H\} \end{aligned}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]
  3. $R_3 = (C, G, H)$ [R_3 is in 4NF]
  4. $R_4 = (A, C, G, I)$ [R_4 is not in 4NF, decompose into R_5 and R_6]
     - $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI \longrightarrow A \twoheadrightarrow HI$, (MVD transitivity),
     - and hence $A \twoheadrightarrow I$ (MVD restriction to R_4).

Example: 4NF Decomposition Algorithm

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

$$F = \{A \twoheadrightarrow B \\ B \twoheadrightarrow HI \\ CG \twoheadrightarrow H\}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]
  3. $R_3 = (C, G, H)$ [R_3 is in 4NF]
  4. $R_4 = (A, C, G, I)$ [R_4 is not in 4NF, decompose into R_5 and R_6]
     - $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI \longrightarrow A \twoheadrightarrow HI$, (MVD transitivity),
     - and hence $A \twoheadrightarrow I$ (MVD restriction to R_4).
  5. $R_5 = (A, I)$ [R_5 is in 4NF]

Example: 4NF Decomposition Algorithm

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

$$F = \{A \twoheadrightarrow B \\ B \twoheadrightarrow HI \\ CG \twoheadrightarrow H\}$$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R.
- Decomposition
  1. $R_1 = (A, B)$ [R_1 is in 4NF]
  2. $R_2 = (A, C, G, H, I)$ [R_2 is not in 4NF, decompose into R_3 and R_4]
  3. $R_3 = (C, G, H)$ [R_3 is in 4NF]
  4. $R_4 = (A, C, G, I)$ [R_4 is not in 4NF, decompose into R_5 and R_6]
     - $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI \longrightarrow A \twoheadrightarrow HI$, (MVD transitivity),
     - and hence $A \twoheadrightarrow I$ (MVD restriction to R_4).
  5. $R_5 = (A, I)$ [R_5 is in 4NF]
  6. $R_6 = (A, C, G)$ [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 (DKNF)**.
- 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.

Plan

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.

E-R 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 $building$,
    - functional dependency:
      $department\_name \rightarrow building$,
    - 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:

$$course \bowtie prereq$$

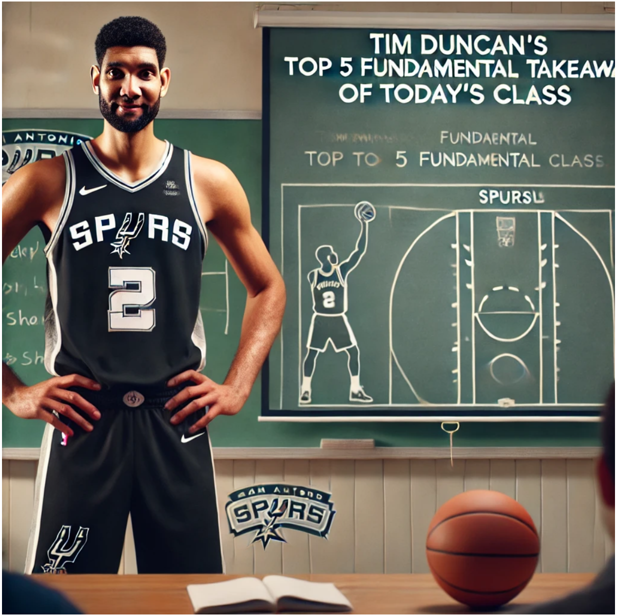
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors.

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

End of Chapter 7.

TDT5FTOTTC


TIM DUNCAN'S
TOP 5 FUNDAMENTAL TAKEAW
OF TODAY'S CLASS
FUNDAENTAL
TOP TO 5 FUNDAMENTAL CLASS
SPURS
2

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5. **The Goal of Good Relational Design is Normalization**: The core objective is to decompose poorly structured relations into a set of smaller, non-redundant, and consistent relations.

Top 5 Fundamental Takeaways

5 **The Goal of Good Relational Design is Normalization**: The core objective is to decompose poorly structured relations into a set of smaller, non-redundant, and consistent relations.

4 **FDs are Key Constraints**: FDs ($\alpha \rightarrow \beta$) generalize keys by specifying that a set of attributes (α) uniquely determines the value of another set (β).

Top 5 Fundamental Takeaways

5 **The Goal of Good Relational Design is Normalization**: The core objective is to decompose poorly structured relations into a set of smaller, non-redundant, and consistent relations.

4 **FDs are Key Constraints**: FDs ($\alpha \rightarrow \beta$) generalize keys by specifying that a set of attributes (α) uniquely determines the value of another set (β).

3 **Lossless Decomposition is Essential for Preserving Information**: A decomposition is lossless only if the natural join of the smaller resulting relations perfectly reconstructs the original relation, with no loss or spurious generation of tuples.

Top 5 Fundamental Takeaways

5 **The Goal of Good Relational Design is Normalization**: The core objective is to decompose poorly structured relations into a set of smaller, non-redundant, and consistent relations.

4 **FDs are Key Constraints**: FDs ($\alpha \rightarrow \beta$) generalize keys by specifying that a set of attributes (α) uniquely determines the value of another set (β).

3 **Lossless Decomposition is Essential for Preserving Information**: A decomposition is lossless only if the natural join of the smaller resulting relations perfectly reconstructs the original relation, with no loss or spurious generation of tuples.

2 **BCNF and 3NF are the Primary Normal Forms for FDs**: BCNF eliminates all non-trivial dependencies where the determinant is not a superkey, while 3NF is a slightly weaker, more achievable form that always guarantees a lossless and dependency-preserving decomposition.

Top 5 Fundamental Takeaways

5 **The Goal of Good Relational Design is Normalization**: The core objective is to decompose poorly structured relations into a set of smaller, non-redundant, and consistent relations.

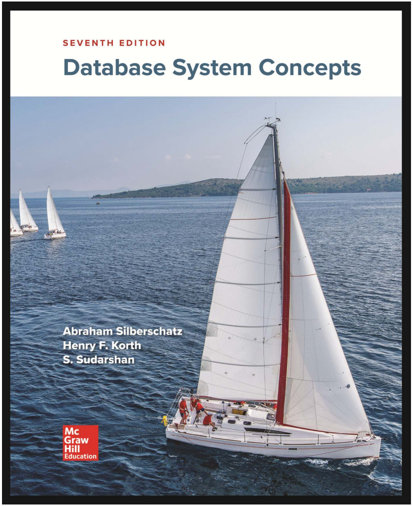
4 **FDs are Key Constraints**: FDs ($\alpha \rightarrow \beta$) generalize keys by specifying that a set of attributes (α) uniquely determines the value of another set (β).

3 **Lossless Decomposition is Essential for Preserving Information**: A decomposition is lossless only if the natural join of the smaller resulting relations perfectly reconstructs the original relation, with no loss or spurious generation of tuples.

2 **BCNF and 3NF are the Primary Normal Forms for FDs**: BCNF eliminates all non-trivial dependencies where the determinant is not a superkey, while 3NF is a slightly weaker, more achievable form that always guarantees a lossless and dependency-preserving decomposition.

1 **MVDs Lead to 4NF to Address Further Redundancy**: Even after BCNF, problems with independent, multivalued facts necessitate the use of MVDs ($\alpha \twoheadrightarrow \beta$), which drive decomposition into 4NF to fully eliminate data redundancy.

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.