Database Systems
Lecture10 – Normalization

SUNG KYUN KWAN
UNIVERSITY

Beomseok Nam (남범석)
bnam@skku.edu

Combine Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
  - *(No connection to relationship set inst_dept)*
- Result is possible repetition of information

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

A Combined Schema Without Repetition

- Consider combining relations
  - *sec_class(sec_id, building, room_number)* and
  - *section(course_id, sec_id, semester, year)*

  into one relation
  - *section(course_id, sec_id, semester, year, building, room_number)*
- No repetition in this case

What About Smaller Schemas?

- Suppose we started DB design with *inst_dept.*
- Q: Should we **decompose** it into *instructor* and *department*?
- A: Identify functional dependencies of attributes.
  - **functional dependency**:

    $dept\_name \rightarrow building, budget$
  - In *inst_dept*, *dept_name* is not a candidate key.
  - But building, budget depends on dept_name.
  - The building and budget of a department need to be repeated

    →This indicates the need to decompose *inst_dept*

What About Smaller Schemas?

- Not all decompositions are good.
  - Suppose we decompose

    employee(ID, name, street, city, salary) into

    employee1 (*ID*, *name*)

    employee2 (*name*, *street, city, salary*)
  - We cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**. (Next slide)

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
⋮				

Example of Lossless-Join Decomposition

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

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

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

First Normal Form

- Domain is **atomic** if its values are indivisible
  - Examples of non-atomic domains:
    - Set of names, composite attributes
  - ID like SWE-3003 can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes are atomic
- Non-atomic values result in redundant data
- We assume all relations are in first normal form

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- If a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Functional Dependencies

- 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,

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

- 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.)

- *Function dependency is a generalization of the notion of key.*
- 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 any $\alpha \subset K$, *no* $(K - \alpha) \rightarrow R$
- Consider the schema:

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

  We expect these functional dependencies to hold:

  $$dept\_name \rightarrow building$$

  and $$ID \rightarrow building$$

  but would not expect the following to hold:

  $$dept\_name \rightarrow salary$$

Functional Dependencies (Cont.)

- *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 $\beta \subseteq \alpha$

Closure of a Set of Functional Dependencies

- Functional dependencies can be derived using inference rules.
  - E.g.: 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*.

Properties of Functional Dependencies

- **Subset Property** (Trivial):
  If Y is a subset of X, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Pseudo-transitivity**:
  If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

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

- Example schema not in BCNF:
- instr_dept (ID, name, salary, dept_name, building, budget)
- because dept_name$\rightarrow$ building, budget
  holds on instr_dept, but dept_name is not a superkey

Decomposing a Schema into BCNF

- Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.
- We decompose R into:
  - (α U β)
  - (R - (β - α))
- In our example,
  - α = dept_name
  - β = building, budget

  and inst_dept is replaced by
  - (α U β) = (dept_name, building, budget)
  - (R - (β - α)) = (ID, name, salary, dept_name)

Decomposition Example

- HR(DPT_NO, MGR_NO, EMP_NO, EMP_NAME, PHONE)
- F = DPT_NO → MGR_NO
  DPT_NO → PHONE
  EMP_NO → EMP_NAME

- Pick BCNF violation: DPT_NO → MGR_NO

- Decomposed relations:
  - HR1(DPT_NO, MGR_NO)
  - HR2(DPT_NO, EMP_NO, EMP_NAME, PHONE)

Q: Are we done? No, HR2 needs to be decomposed

BCNF can decompose "too much"

- What if we have FDs $AB \longrightarrow C$ and $C \longrightarrow B$?
- Example
  - A = street, B = city, and C = zip
- The following table violates BCNF, so we must decompose into AC and BC as our database schema.

street	city	zip

An Unenforceable FD

- If we have AC and BC, we can not preserve the FD *A(street)B(city)* → *C (zip)*.

street	zip
545 Tech Sq. 230 First st.	02138 02139

city	zip
Cambridge Cambridge	02138 02139

- Because it is <u>not always possible to achieve both BCNF and dependency preservation</u>, we consider a weaker normal form, known as *third normal form.*

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)
  - <u>Allows some redundancy</u> (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

- 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.

- 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.

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

- Benefits of Normalization
  - Less storage space
  - Quicker updates
  - Less data inconsistency
  - Clearer data relationships
  - Easier to add data
  - Flexible Structure

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

Functional-Dependency Theory

- Formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.

Closure of a Set of Functional Dependencies

- We 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).

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

Closure of Functional Dependencies (Cont.)

- 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.

Dependency Preservation

- Let F_i be the set of dependencies in 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.

- There is always a lossless-join, dependency-preserving decomposition into 3NF.

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*
- Multi-valued dependencies (Read textbook)
- If a relation is in 4NF it is in BCNF

Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **5th normal form**)
- Rarely used, and not worth to discuss in this class.

Overall Database Design Process

- We have schema *R*
  - *i.e.,* we convert ad hoc E-R diagram to a set of tables.
  - *R* could be a single relation containing *all* attributes
  - Normalization breaks *R* into smaller relations

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

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* (a physical table containing all the tuples in the result of the query) defined as

  $$course \bowtie prereq$$

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