Principles of Database Systems (CS307)

Lecture 8: Normalization - A Deeper Look (Part 1)

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7^{th} Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

(Recall) Prerequisites

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are attributes

- $R = (A_1, A_2, \ldots, A_n)$ is a **relation schema**
  - Example on the right side:
    instructor = (*ID*, *name*, *dept_name*, *salary*)

- $r(R)$ denotes a <u>relation instance r defined over schema R</u>
  - Or to say, the entire table on the right side

- An element t of relation r is called a tuple
  - … and is represented by a <u>row</u> in a table

The relation schema ("R")

A_1	A_2	A_3	A_4
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

r(R)

A tuple

Relation Schema and Instance

- An analogy to programming languages:
  - Relation - Variables
  - Relation schema – Variable types
  - Relation instance – Value(s) stored in the variable

Database Schema

- Database schema is the logical structure of the database
  - It contains a set of relation schemas
  - … and a set of integrity constraints
- Database instance is a snapshot of the data in the database at a given instant in time

Keys

- Let K $\subseteq$ *R*
  - K is a **superkey** of *R* if values for K are sufficient to identify a unique tuple of each possible relation *r*(*R*)
    - E.g., {*ID*} and {*ID*, *name*} are both superkeys of instructor
    - If K is a superkey, any superset K' of K where K' $\subseteq$ *R* is a superkey as well
  - Superkey K is a **candidate key** if K is minimal, i.e., no proper subset of K is a superkey
    - E.g., {ID} is a candidate key for *instructor*
  - One of the candidate keys is selected to be the **primary key**
    - We mark the primary key with an underline:

      instructor = (*ID*, *name*, *dept_name*, *salary*)

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

instructor

Decomposition & Functional Dependency

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

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

instructor

dept_name	*building*	*budget*
Physics	Watson	70000
Finance	Painter	120000
History	Painter	50000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Biology	Watson	90000
Comp. Sci.	Taylor	100000
History	Painter	50000
Comp. Sci.	Taylor	100000
Music	Packard	80000
Physics	Watson	70000
Finance	Painter	120000

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

in_dep

Decomposition

- Avoid the repetition-of-information problem
  - Decompose *in_dep* into two schemas: *instructor* and *department*

- However, not all decompositions are good
  - E.g., decompose *employee*(<u>ID</u>, name, street, city, salary) into:
    - employee1(ID, name)
    - employee2(name, street, city, salary)

  The problem arises when we have <u>two employees with the same name</u>

A Lossy Decomposition

- (Continue) we cannot <u>reconstruct</u> the original employee relation with the join operation
  - We call it 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
⋮				

Two “ghost” records that do NOT exist in the original table

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R
  - That is, $R = R_1 \cup R_2$
  - The decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R = R_1 \cup R_2$
- Formally, $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
  - … and a decomposition is lossy if $r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

(!) proper subset

- Or to say, the two SQL queries on the right side generate identical results:

```sql
select * -- 1
from (select R1 from r)
    natural join
     (select R2 from r);

select * from R; -- 2
```

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 a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition

- Our theory is based on:
  - Functional dependencies
  - * Multivalued dependencies (self study)

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

- 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

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

- A functional dependency is a generalization of the notion of a key

Definition of Functional Dependencies

- Let R be a relation schema, and $\alpha \subseteq R$ *and* $\beta \subseteq R$,

  the functional dependency

$$\alpha \rightarrow \beta$$

  holds on R <u>if and only if</u> 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,
  - On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold

A	B
1	4
1	5
3	7

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^+
  - F^+ is a superset of *F*

Keys and Functional Dependencies

- Let’s see how can we <u>(re)define</u> the concept of “**keys**” under the language of functional dependencies

Keys and Functional Dependencies

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

(!) proper subset, again

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys

- E.g. Consider the schema: *inst_dept*(*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)

  - We expect these functional dependencies to hold:

$$dept\_name \rightarrow building, \; ID \rightarrow building$$

… but would not expect the following to hold:

$$dept\_name \rightarrow salary$$

Use of Functional Dependencies

- We use functional dependencies to
  - To test relations to see <u>if they are legal</u> under <u>a given set of functional dependencies</u>
    - If a relation *r* is legal under <u>a set *F* of functional dependencies</u>, we say that <u>*r* satisfies *F*</u>
  - 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*

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies
  - $A \rightarrow C$
  - $D \rightarrow B$

Can you find more?

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- 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.

- Example: we see that *room_number* $\rightarrow$ *capacity* is satisfied.
  - However, in real world, two classrooms in different buildings can have the same room number but with different room capacity
  - We prefer {*building*, *room_number*} $\rightarrow$ *capacity*

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Trivial Functional Dependencies

- A functional dependency is **trivial** if it is satisfied by all relations

- 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

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

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

In other words, if $R_1 \cap R_2$ forms a superkey for either R_1 or R_2, the decomposition of R is a lossless decomposition

Lossless Decomposition

- Example:
  - *in_dep* (*<u>ID</u>*, *name*, *salary*, *<u>dept_name</u>*, *building*, *budget*)
  - ... and the decomposed schemas, *instructor* and *department*:
    - *instructor*(*<u>ID</u>*, *name*, *dept_name*, *salary*)
    - *department*(*<u>dept_name</u>*, *building*, *budget*)

$instructor \cap department = dept\_name$

$dept\_name \rightarrow dept\_name, building, budget$

(… which means the decomposition is lossless)

Lossless Decomposition

- Note: the above functional dependencies are a <u>sufficient condition</u> for lossless join decomposition
  - The dependencies are a <u>necessary condition</u> only if <u>all constraints are functional dependencies</u>

*(There are other types of constraints, e.g., **multivalued dependencies**, that can ensure that a decomposition is lossless even if no functional dependencies are present)*

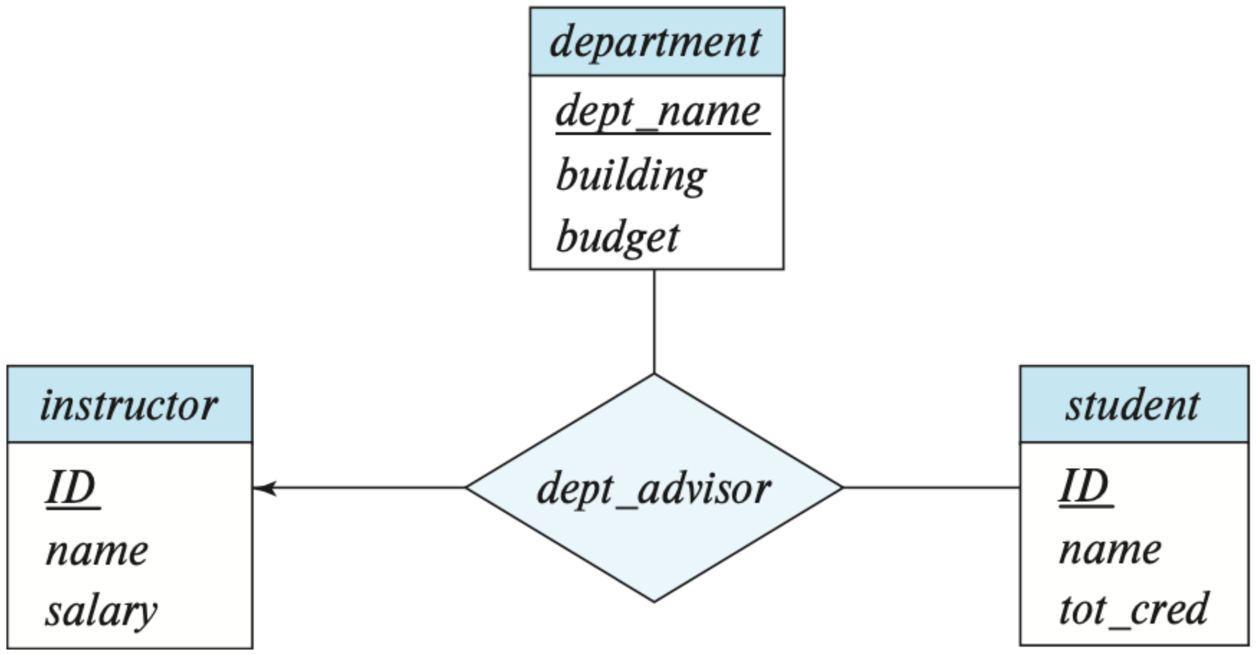
Dependency Preservation

- 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 a functional dependency in the original relation *R* does not exist in any of the decomposed relations, we say it is not dependency-preserving
  - In the dependency preservation, at least one decomposed table must satisfy every dependency

Dependency Preservation

- Consider a new E-R design for relationships between students, instructors, and departments
  - An instructor can only be associated with one department
  - A student can have multiple advisors but not more than one from a given department
    - Think about double-major students



Dependency Preservation

- Consider a schema
  - *dept_advisor*(<u>*s_ID*</u>, <u>*i_ID*</u>, <u>*dept_name*</u>)
  - … with function dependencies: (1) $i\_ID \rightarrow dept\_name$ (2) $s\_ID, dept\_name \rightarrow i\_ID$

In the above design, we are forced to <u>repeat the department name</u> once for each time an instructor participates in a *dept_advisor* relationship.

Dependency Preservation

- Consider a schema
  - *dept_advisor*(<u>*s_ID*</u>, <u>*i_ID*</u>, <u>*dept_name*</u>)
  - … with function dependencies: (1) $i\_ID \rightarrow dept\_name$ (2) $s\_ID, dept\_name \rightarrow i\_ID$

In the above design, we are forced to <u>repeat the department name</u> once for each time an instructor participates in a *dept_advisor* relationship.

- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition <u>will not include all the attributes</u> in

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

  - Thus, the decomposition will NOT be dependency-preserving

Dependency Preservation

- Problem when not meeting dependency preservation
  - Every time the database wants to check the integrity of the functional dependency

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

    the decomposed tables must be joined

  - … where the computational cost could be very high with join operations

BCNF and 3NF

Normal Forms: Revisited

- Boyce-Codd Normal Form (BCNF)
- 3NF
- Higher-order normal forms

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

- * A database design is in BCNF if each member of the set of relation schemas that constitutes the design is in BCNF

Boyce-Codd Normal Form

- Example schema that is **not** in BCNF:
  - *in_dep (<u>ID</u>, name, salary, <u>dept_name</u>, building, budget)*

  Because,

  $$dept\_name \rightarrow building, budget$$

  - holds in *in_dep*, however, *dept_name* is not a superkey
    - … where {ID, dept_name} is

  - 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 R that is not in BCNF
- Let $\alpha \rightarrow \beta$ be the functional dependency that causes a violation of BCNF
  - We decompose R into:
    - $(\alpha \cup \beta)$
    - $(R - (\beta - \alpha))$

- Example: *in_dep* (*<u>ID</u>*, *name*, *salary*, *<u>dept_name</u>*, *building*, *budget*)
  - α = *dept_name*, β = *building*, *budget*
  - Thus, *in_dep* is replaced by:
    - $(\alpha \cup \beta)$ = (*dept_name*, *building*, *budget*)
    - $(R - (\beta - \alpha))$ = (*ID*, *name*, *dept_name*, *salary*)

BCNF and Dependency Preservation

- It is not always possible to achieve both <u>BCNF</u> and <u>dependency preservation</u>
- Consider the schema (that we have visited before)
  - *dept_advisor*(<u>*s_ID*</u>, <u>*i_ID*</u>, <u>*dept_name*</u>)
  - … with function dependencies: (1) $i\_ID \rightarrow dept\_name$ (2) $s\_ID, dept\_name \rightarrow i\_ID$

  - *dept_advisor* is not in BCNF since for $i\_ID \rightarrow dept\_name$, i_ID is not a superkey
    - (where $\{s\_ID, i\_ID, dept\_name\}$ is)

- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition <u>will **not** include all the attributes</u> in

    $$s\_ID, dept\_name \rightarrow i\_ID$$
  - Thus, the decomposition will NOT be dependency-preserving

Third Normal Form (3NF)

- 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., $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

- Notes
  - Each attribute A may be in a different candidate key
  - If a relation is in BCNF, it is in 3NF (... since in BCNF, one of the first two conditions above must hold)
  - The third condition above is a <u>minimal relaxation of BCNF</u> to ensure dependency preservation

3NF Example

- Consider the schema (that we have visited before)
  - *dept_advisor(s_ID, i_ID, department_name)*
  - … with function dependencies: (1) *i_ID* $\rightarrow$ *dept_name* (2) *s_ID, dept_name* $\rightarrow$ *i_ID*

  - We have two candidate keys: {*s_ID, dept_name*} and {*s_ID, i_ID*}

- *dept_advisor* is not in BCNF, but it can be in 3NF
  - {s_ID, dept_name} is a superkey
  - *i_ID* $\rightarrow$ *dept_name* and *i_ID* is NOT a superkey (which violates BCNF), but:
    - α is *i_ID*, β is *dept_name*
    - {*dept_name*} – {*i_ID*} = {*dept_name*}
    - *dept_name* is contained in a candidate key (-> {*s_ID, dept_name*})

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

- Problems in this table:
  - Repetition of information
    - Row 1-3: L and K
  - Need to use nulls
    - Row 4: Represent the relationship l_2, k_2 with 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 nulls to represent some of the possible meaningful relationships among data items
  - There is a problem of potential 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, need to decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that:
    - Each relation scheme is in good form
    - The decomposition is a lossless decomposition
    - *Preferably*, the decomposition should be dependency preserving