Normal Forms I

Kai Brünnler

Bern University of Applied Sciences

Outline

Outline

Large Relation Schemas vs. Small Relation Schemas

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

The *faculty* table

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

(a) The *instructor* table

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

(b) The *department* table

Normal Forms

Features of a Good Database Schema

- We have contradicting requirements:
  - No redundancy – this means small relation schemas
  - Easily retrieve information – this means large relation schemas
- *Normal forms* characterize sweet spots in this spectrum

Strength of Normal Forms

Normal forms are classified by how much redundancy they permit:

- Weaker normal form: more redundancies possible
- Stronger normal form: fewer redundancies possible

Some normal forms, weak to strong:

1NF, 2NF, 3NF, BCNF, 4NF

Outline

Atomic Domains

Atomic Domains

- A domain is called *atomic* if its elements are considered to be indivisible units.
- Atomicity is actually a property of how the elements of the domain are used
- Non-atomic values are usually bad
  - they lead to encoding of information in application programs rather than in the database

Examples

- Atomic:
  - a last name like "Adams"
  - a street number like "10"
- Possibly non-atomic:
  - a name like "John Adams"
  - an address like "10 Downing Street"
  - a course ID like "CS-101"
  - a list of instructor IDs like "76766, 45565, 10101"

First Normal Form

First Normal Form

- A relational schema is in *First Normal Form (1NF)* if the domains of all its attributes are atomic

- 1NF is desirable
- But 1NF is rather weak (not good enough)
- All stronger normal forms implicitly require 1NF

Outline

Outline

How to Recognize Redundancy

No redundancy:

instructor(ID, name, dept_name, salary)
department(dept_name, building, budget)

Redundancy:

faculty(ID, name, salary, dept_name,
building, budget)

Can we recognize redundancy by just looking at the schema?

How to Recognize Redundancy

No redundancy:

instructor(ID, name, dept_name, salary)
department(dept_name, building, budget)

Redundancy:

faculty(ID, name, salary, dept_name,
building, budget)

Can we recognize redundancy by just looking at the schema?

The Cause of Redundancy

- A *dept_name* determines exactly one building
- In *faculty*, *dept_name* is not a superkey. So it can occur several times.
- But then its *building* also occurs several times.
- The cause of redundancy here is thus an attribute which:
  1. determines another attribute and
  2. is not a superkey

Outline

Functional Dependencies

Functional Dependency

If an attribute A determines another attribute B we call that a *functional dependency*, and write it as: $A \rightarrow B$.

Example

$$dept\_name \rightarrow building$$

Definition

Formally, a *functional dependency* is a pair of two non-empty sets of attributes α, β, written as:

$$\alpha \rightarrow \beta \quad ,$$

where α is called *premise* or *left-hand-side* and β is called *conclusion* or *right-hand-side*.

Functional Dependencies on Relations

Definition

A relation r *satisfies* the functional dependency $\alpha \rightarrow \beta$ if for any two tuples t_1 and t_2 of r we have that:

> if t_1 and t_2 have the same value for each attribute in α, then they have the same value for each attribute in β.

Example

This relation satisfies $B \rightarrow A$ but does not satisfy $A \rightarrow B$.

A	B
a_1	b_1
a_1	b_2
a_2	b_3

Trivial Functional Dependencies

Definition

A functional dependency $\alpha \rightarrow \beta$ is called *trivial* if $\beta \subseteq \alpha$.

Clearly, trivial functional dependencies are satisfied by all relations.

Examples

- *building* $\rightarrow$ *building*
- *building amount* $\rightarrow$ *building*

Single-Conclusion Functional Dependencies

Definition

A functional dependency $\alpha \rightarrow \beta$ is called *single-conclusion* if β is a singleton (contains exactly one attribute).

Each functional dependency can be expressed by a set of single-conclusion functional dependencies.

Example

$A \rightarrow BC$

is equivalent to

$\{A \rightarrow B, A \rightarrow C\}$

Which functional dependencies are satisfied by this relation?

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_2	c_2
a_2	b_3	c_3

Functional Dependencies on Schemas

Definition

- Given a relation schema, its *legal* relations are those relations on the schema which can actually occur in the real world.
- Let R be a relation schema and f a functional dependency. We say that R *satisfies* f (or f *holds* on R) if each legal relation $r(R)$ satisfies f.

Example

On the *department* schema

- $dept\_name \rightarrow building$ holds, but
- $building \rightarrow dept\_name$ does not hold.

However, a specific instance of *department* can (by chance) satisfy

- $building \rightarrow dept\_name$

Keys as Functional Dependencies

Keys as Functional Dependencies

The superkey and candidate key properties are special cases of functional dependencies.

- K is a superkey for relation schema R iff $K \rightarrow R$
- K is a candidate key for R iff
  - $K \rightarrow R$, and
  - for no $\alpha \subsetneq K$, $\alpha \rightarrow R$.

Functional Dependencies are more Expressive than Keys

Consider our schema:

faculty(ID, name, salary, dept_name, building, budget)

This functional dependency holds:

$$dept\_name \rightarrow building$$

but this can not be expressed in terms of keys.

Sets of Functional Dependencies

Most of the time we are not interested in a single functional dependency, but in a set of functional dependencies.

Definition

Given a relation r and a set of functional dependencies F, we say that r *satisfies* F iff r satisfies each functional dependency in F.

Definition

Given a relation schema R and a set of functional dependencies F, we say that R *satisfies* F (or F *holds* on R) if R satisfies each functional dependency in F.

Inferring Functional Dependencies

Some functional dependencies are implicit (hidden) in others.

Definition

Given a relational schema R, a set F of functional dependencies *implies* a functional dependency f if every relation r on R that satisfies F also satisfies f.

Example

$\{A \to B, B \to C\}$ implies $A \to C$

The Closure

We need to make the implicit functional dependencies explicit (reveal them).

Definition

The set of all functional dependencies implied by F is called the *closure* of F and is denoted by F^+.

Example

$$\{A \to B\}^+ = \{ \quad A \to A, A \to B, A \to AB, \; B \to B, \; AB \to A, AB \to B, AB \to AB \; \}$$

Equivalence of Functional Dependencies

Definition

Let F and G be sets of functional dependencies. We say that :

- F *implies* G iff F implies each functional dependency in G.
- F is *equivalent* to G iff F implies G and G implies F.

Proposition

Let F and G be sets of functional dependencies.

- F implies G iff $F^+ \supseteq G$.
- F is equivalent to G iff $F^+ = G^+$.

Outline

Boyce-Codd Normal Form

Definition

A relation schema R is in *Boyce-Codd Normal Form (BCNF)* for a set F of functional dependencies if for all non-trivial functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, α is a superkey for R.

A set of relation schemas is in BCNF for F if each member of it is in BCNF for F.

Example

- Let $F = \{$ *ID* $\rightarrow$ *name dept_name salary*, *dept_name* $\rightarrow$ *building budget* $\}$
- This schema is not in BCNF for F:

  faculty(ID, name, salary, dept_name, building, budget)
- This schema is in BCNF for F:

  instructor(ID, name, dept_name, salary)
  department(dept_name, building, budget)

The Banking Example

We want to store information about a bank's branches, their customers and their loans:

bank(branch-name, branch-city, assets, customer-name, loan-number, amount)

$F = \{$*branch-name* $\rightarrow$ *assets branch-city*, *loan-number* $\rightarrow$ *amount branch-name* $\}$

Questions

- Is the *bank* schema in BCNF?
- Why not?
- Can we get a schema which stores the same information but is in BCNF?

The Banking Example

branch_name	*branch_city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The Banking Schema

branch(branch-name, branch-city, assets)

borrower(customer-name, loan-number)

loan(loan-number, branch-name, amount)