CS 4221: Database Design

Extending Classical Functional Dependencies for Physical Database Design

Ling Tok Wang
National University of Singapore

Topics

- Physical Database Design
- Normalization: Theory vs. Practice
- Strong Functional Dependency
  - Replicated 3NF
- Weak Functional Dependency
  - Relaxed 3NF
- Relax-replicated 3NF
- Preserving database integrity with relax-replicated 3NF

Ref: Tok Wang Ling, Cheng Hian Goh, Mong-Li Lee: Extending classical functional dependencies for physical database design. Information & Software Technology 38(9): 601-608 (1996)

Physical Database Design

- It is the process of transforming a logical database model into a physical database model of a database.
- Unlike a logical database design, a physical database design is optimized for data-access paths, performance requirements and other constraints of the target environment, i.e. hardware and software.
- Note that a database in good normal form (e.g. 3NF, BCNF, or 4NF) may not give good performance for some applications.
- We will introduce some extensions to functional dependency together with some theories for physical database design.

Normalization: theory vs. practice

Example 1. Consider the Supplier-Part database

Sup_info (sno, pno, sname, addr, pname, color, qty)

with FDs:

sno → sname, addr
pno → pname, color
sno, pno → qty

- The relation Sup_info is not in good normal form (in fact it is not in 2NF).
- According to normalization theory, we need to normalize it to the below 3NF and also BCNF relations:

Supplier (sno, sname, addr)
Part (pno, pname, color)
Supply (sno, pno, qty)

- Assume that the enterprise requires frequently reporting on the information held in the relation Supply together with sname and pname values, i.e. what we need is the relation

  Supply_View (<u>sno, pno</u>, sname, pname, qty)

- This effectively means computing the join on all the three 3NF relations to get Supply_View.

  A very expensive and time consuming operation!

- The above schema is not a good solution for this application.

- Any better solution?

Example 2. Consider a database whose intension is to capture information of employee in a company.

Emp (emp#, empname, phone#, post, ...)

Assumption: Every employee has only one phone except a few very senior managers.

- The database designer has 2 alternatives:

Solution 1. Create a new attribute alt_phone#

Emp (emp#, empname, phone#, alt_phone#, post, ...)

Problem: High storage cost. Also some senior managers may have more than 2 phones.

Solution 2. Treat phone# as a multivalued attribute,

i.e. emp# $\twoheadrightarrow$ phone#

So one more relation is needed, i.e.

Emp(emp#, empname, post, ...)

Emp_phone(emp#, phone#)

Solution 2 has two problems:

(1) need one extra relation, so extra storage cost

(2) To retrieve the phone#'s and other information such as name of employees, we need to join the 2 relations. An expensive and time consuming operation.

- Q: Any other better solutions?

  Yes!

We introduce the notions of **strong functional dependency** and **weak functional dependency**.

Strong Functional Dependency (SFD)

Defn: Let $X \rightarrow Y$ be a FD such that for each $z \in Y$, $X \rightarrow z$ is full FD. $X \rightarrow Y$ is a **SFD** if all the attributes in Y will not be updated, or if the updates need not be performed at real-time or on-line and such updates are very seldom.

We denote it as:

$$X \xrightarrow{S} Y$$

E.g. In example 1, as pname and sname are seldom changed, so we have 2 SFDs

$$pno \xrightarrow{S} pname$$

$$sno \xrightarrow{S} sname$$

Weak Functional Dependency (WFD)

Defn: Let X and Y be subsets of a relation R, and
$X \twoheadrightarrow Y$ in R.
If **most** of the X-values are associated with a Y-value in R, except for a handful of X-values which may be associated with more than one Y-value,

i.e. if we remove these handful of exception tuples from R, then $X \rightarrow Y$ holds in R.

We say Y is weakly dependent on X and denote this as

$$X \xrightarrow{w} Y$$

and it is a weak FD.

E.g. In Example 2, we have the below WFD

$$emp\# \xrightarrow{w} phone\#$$

Property: $X \xrightarrow{S} Y \Rightarrow X \longrightarrow Y \Rightarrow X \xrightarrow{W} Y$

Defn: [**Replicated 3NF**]

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ be a relational database schema, and $\mathcal{A}_j$ be the set of the attributes of R_j, for j = 1, 2, ..., n. A relation R_i in $\mathcal{R}$ is said to be in **replicated 3NF** if:

(1) For each $X \xrightarrow{s} Y$, $X \cup Y \subseteq \mathcal{A}_i$, where X is not a key of R_i,

Case (1) If X is not a role name of the key of R_i, then there exist a unique $R_j \in \mathcal{R}$, $j \neq i$, such that X is a key of R_j and $Y \subseteq \mathcal{A}_j$. R_j is said to be the primary instance of R_i w.r.t. the attributes in $X \cup Y$, or

Case (2) If X is a role name of the key of R_i, and Y is a role name of some attribute in R_i

(2) Let $\beta = \{B \mid X \xrightarrow{s} B, X \cup \{B\} \subseteq \mathcal{A}_i$,

X is not a key of R_i, B is a non-prime of $R_i\}$

The relation obtained from R_i after removing all attributes in β is in 3NF.

Example 3. Consider the database schema

Supplier (sno, sname, addr)
Part (pno, pname, color)
Supply (sno, pno, sname, pname, qty)

Clearly Supply relation is not in 3NF.
However, it is in replicated 3NF since

$$\text{sno} \xrightarrow{S} \text{sname}$$

$$\text{pno} \xrightarrow{S} \text{pname}$$

Supplier and Part relations are the primary instances of Supply w.r.t. {sno, sname} and {pno, pname} resp.
(i.e. case (1) and β = {sname, pname})

Observation: Supply relation contains redundant sname and pname information. However, there is no updating problem by sname and pname as we don’t change their vaules.

Example 4. Consider the relation

Emp_Mgr (emp#, ename, mgr#, mgrname, addr)

- emp# $\xrightarrow{s}$ ename
  mgr# $\xrightarrow{s}$ mgrname
- mgr# is a role name of emp#
  mgrname is a role name of ename
- Emp_Mgr is not in 3NF but it is in replicated 3NF by the condition in case (2) and β = {mgrname}.
- **Note:** Some mgrname's are duplicated. However, mgrname does not cause updating anomalies as managers do not change their name.

Properties:

(1) Replicated 3NF relations may contain redundant data. However, such redundancies can be controlled. **Q:** How?

(2) Replicated 3NF relations provide efficient retrieval for certain applications.

Defn: [**Relaxed 3NF**]

Let $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ be a relational database schema, and $\mathcal{A}_j$ be the set of the attributes of R_j, for j = 1, 2, ..., n.

A relation R_i in $\mathcal{R}$ is said to be in **relaxed 3NF** if whenever every weak FD $X \xrightarrow{w} Y$ which holds in R_i is replaced by its regular counterpart (i.e. $X \rightarrow Y$), R_i would have been in 3NF.

Example 5. Consider the relation Emp in Example 2

Emp (emp#, ename, phone#, post, ...)

We have

emp# → ename, post, ...

$\text{emp\#} \xrightarrow{w} \text{phone\#}$

Emp is not in 3NF, but it is in relaxed 3NF.

Q: Why?

- We can implement the WFD

  $emp\# \xrightarrow{w} phone\#$

  by treating phone# as if

  $emp\# \rightarrow phone\#$

  holds and accommodates exceptional cases (i.e. 2nd or 3rd, etc. phone of employees) in an **overflow relation** as follows:

  Emp (emp#, ename, phone#, post, ...)
  Emp_phone_overflow (emp#, overflow-phone#)

Question: How to maintenance the phone#'s of employees in the two relations?

Defn: [**Relax-Replicated 3NF**]

Let $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ be a relational database schema, and $\mathcal{A}_j$ be the set of the attributes of R_j, for j = 1, 2, ..., n.

A relation R_i in $\mathcal{R}$ is said to be in **relax-replicated 3NF** if whenever every weak FD $X \xrightarrow{w} Y$ which holds in R_i is replaced by its regular counterpart (i.e. $X \rightarrow Y$), R_i would have been in replicated 3NF.

Example 6. Consider the schema

Emp_mgr (emp#, ename, mgr#, mgrname, phone, addr)
Emp_phone_overflow (emp#, overflow-phone#)

where attribute mgr# and mgrname are role names of emp# and ename resp., and

$$\text{emp\#} \xrightarrow{s} \text{ename}$$

$$\text{mgr\#} \xrightarrow{s} \text{mgrname}$$

$$\text{emp\#} \xrightarrow{w} \text{phone\#}$$

The relation Emp_mgr is in relax-replicated 3NF.

Note:

We can similarly define relax-replicated improved 3NF, relax-replicated BCNF, relax-replicated 4NF, etc.

Preserving database integrity with relax-replicated 3NF

Example 7. [Preserving integrity of replicated 3NF]

Consider schema

Supplier (sno, sname, addr)
Part (pno, pname, color)
Supply (sno, pno, sname, pname, qty)

where

$pno \xrightarrow{s} pname$

$sno \xrightarrow{s} sname$

Supply relation is in replicated 3NF.

In order to preserve the integrity of this database, we need to enforce the below inclusion dependencies:

Supply [sno, sname] $\subseteq$ Supplier [sno, sname]
Supply [pno, pname] $\subseteq$ Part [pno, pname]

Q: How?

E.g. Insert into supply values (“s1”, p1”, “acme”, “screw”, 10)

The insertion operation might be rewritten to the following:

```
S := select *
     from Supplier
     where Supplier.sno = “s1” and
           Supplier.sname = “acme”;

P:= select *
     from part
     where Part.pno = “p1” and
           Part.pname = “screw”;

if  S = NULL  or  P = NULL
   then     reject transaction
   else     insert into Supply
            value (“s1”, “p1”, “acme”, “screw”, 10);
```

To test whether there is a part “p1” with name “srew” in the relation Part.

Q: How about **update** and **delete** operations on relation Supply?

Example 8. [preserving integrity of relaxed 3NF]

Consider the schema in Example 5 again.

Emp (emp#, ename, phone#, post, ...)
Emp_phone_overflow (emp#, overflow-phone#)

- Find **all** the phone numbers of Smith.

```
select phone#
from Emp
where ename = "Smith"
union
select overflow-phone#
from Emp_phone_overflow, Emp
where Emp.ename = "Smith" and
          Emp.emp# = Emp_phone_overflow.emp#;
```

> Note that there are very few tuples in relation Emp_phone_overflow, only very few employees have more than one phone.

- In fact, we could have an **interface** and users only see the relation Emp. In this case, the users could query the relation directly say, with

```
select phone#
from Emp
where ename = "Smith";
```

In this case, users don't need to know WFD and its implementation.

- The **insertion** operation. E.g.

```
        insert into Emp values (Eno, Ename, Ephone, ...)
can be transformed to
      E :=  select *
            from Emp
            where emp# = Eno;
      if  E = NULL   then        // a new employee
         insert into Emp values (Eno, Ename, Ephone, ...)
      else
         if  E.phone# = NULL then       // employee has no phone yet
             update Emp set phone# = Ephone
         else         // employee has one or more than one phone
             if  E.phone# = Ephone then reject transaction  // same phone #  value
             else
                 insert into Emp_phone_overflow values (Eno, Ephone)
                     // also need to check duplicate phone # value in Emp_phone_overflow
```

- The **deletion** operation

E.g. Delete a phone# with value Ephone of an emplyoee with E# value Eno. The query is written as below:

update Emp set phone# = NULL
    where emp# = Eno and phone# = Ephone

The above query is transformed to a query on the two tables Emp and Emp_phone_overflow。

- If the phone # to be deleted is in Emp table, then delete it and move a phone# in the overflow table to Emp if any, and exit.
- If the phone # to be deleted is not in Emp table, then check whether it is in the overflow table. If yes, delete it, else error.

```
select phone#
from Emp
where emp# = Eno;
if phone# = Ephone then   // the phone# (i.e. Ephone) to be deleted is in Emp relation
   S :=    select overflow-phone#   // check whether this employee has other phones
           from Emp_phone_overflow
           where emp# = Eno;
   if  S = NULL then      // this employee has no other phone
           update Emp set phone# = NULL   // this employee now has no phone
           where emp# = Eno
   else    // this employee has other phones, move a phone# in the overflow relation
           to Emp
           p := any arbitary phone# value in S
           delete from Emp_phone_overflow
                   where emp# = Eno and overflow-phone# = p;
           update Emp set phone# = p
           where emp# = Eno
else   // the phone# to be deleted is not in Emp, delete it in Emp_phone_overflow relation
   delete from Emp_phone_overflow
   where emp# = Eno and overflow-phone# = Ephone
```

Q: Efficient?   How about update a phone#?