CS354: Database

Functional Dependency

- Let's consider the following *inst_dept* relation

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

What about smaller schemas?

- How would we know to split up (**decompose**) it into *instructor* and *department*?
- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose *inst_dept*
- However, not all decompositions are good.
- Suppose we decompose
  employee(ID, name, street, city, salary) into
  employee1 (*ID*, *name*)
  employee2 (*name*, *street*, *city*, *salary*)

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 $R = (A, B, C)$

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

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

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

$\Pi_{\mathrm{A,B}}\ (\mathrm{r}) \bowtie \Pi_{\mathrm{B,C}}\ (\mathrm{r})$

A	B	C
α	1	A
β	2	B

1^{ST} Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes

- *A relational schema R is in **first normal form** if the domains of all attributes of R are **atomic***

Goal- Devise a Theory for the Following

- 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-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

FUNCTIONAL DEPENDENCY

- 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*

FUNCTIONAL DEPENDENCY

- 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,

$$\forall t_1, t_2 \in r \ \ (t_1[\alpha] = t_2[\alpha] \ \ \Rightarrow \ \ t_1[\beta] = t_2[\beta])$$

Example

- Consider $r(\mathrm{A},B)$ with the following instance of r.

A	B
1	4
1	5
3	7

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

FUNCTIONAL DEPENDENCY

- 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 $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys
- 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:

  $$dept\_name \rightarrow salary$$

USE OF FUNCTIONAL DEPENDENCY

- We use functional dependencies 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*
  - 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

$$name \rightarrow ID$$

FUNCTIONAL DEPENDENCY (CONTD)

- *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 Dependency

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
  - For example: 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.

Closure of a Set of Functional Dependency

- We can 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

Procedure for computing F^+

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

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules 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
      then add the resulting functional dependency to F^+
until F^+ does not change any further

Closure of FDs

- Additional rules which can be inferred from Armstrong's axioms
  - **union :** If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **decomposition :** If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **pseudotransitivity :** If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

Functional Dependency Example

- *Flight* <*flight_no, c_arr, c_dept, pl_type*>
- *Seats_free* <*flight_no, date, seats_avl*>
- What are some possible valid FDs?
  - *flight_no* → *c_arr*
  - *flight_no* → *c_dept*
  - *flight_no* → *pl_type*
  - *flight_no, date* → *seats_avl*

FUNCTIONAL DEPENDENCY EXAMPLE

- *Stud_addr <name, address>*
- *Stud_grade <name, subject, grade>*
- Some possible FDs that hold are
  - *name* $\rightarrow$ *address*
  - *name, subject* $\rightarrow$ *grade*

- Which FDs hold here?

X	Y	Z	W
x_1	y_1	z_1	w_1
x_1	y_2	z_1	w_2
x_2	y_2	z_2	w_2
x_2	y_3	z_2	w_3
x_3	y_3	z_2	w_4

- $x \rightarrow y$
- $x \rightarrow z$ holds
- $x \rightarrow w$
- $y \rightarrow x$
- $y \rightarrow z$
- $y \rightarrow w$
- $z \rightarrow x$
- $z \rightarrow y$
- $z \rightarrow w$
- $w \rightarrow x$
- $w \rightarrow y$ holds
- $w \rightarrow z$

$xy \rightarrow z$ holds

$yz \rightarrow x$

FULL FUNCTIONAL DEPENDENCY

- When the functional dependency is 'minimal' in size (i.e., containing non redundant terms)
- FD $X \rightarrow A$ for which there is no proper subset Y of X such that $Y \rightarrow A$ (A is said to be **fully functionally dependent** on X)

CLOSURE OF ATTRIBUTE SETS

- The set of all attributes functionally determined by α under a set F of FDs
- It is denoted by α^+
- Let's consider the following example
  - $A \rightarrow BC$
  - $AC \rightarrow D$
  - $D \rightarrow B$
  - $AB \rightarrow D$
- So what is A^+, B^+, C^+, D^+
  - $A^+=\{A,B,C,D\}$, $B^+=\{B\}$, …

Cover of a set of FDs

Let f and g be two FDs on a relation scheme R.
Then f is a cover of g if $f^+=g^+$
This is also known as f is equivalent to g

f
A→BC
B →C
A →B
AB →C

g
A→BC
B →C
AB →C

Here $f^+=g^+$
So g covers f

MINIMAL COVER OR CANONICAL COVER

- A cover is said to be minimal if it has no redundant terms
- Denoted by F_c
- Example:

F	Fc
$A \rightarrow BC$	$A \rightarrow CD$
$AC \rightarrow D$	$D \rightarrow B$
$D \rightarrow B$	
$AB \rightarrow D$	

EXTRANEOUS ATTRIBUTE

- An attribute of a FD is said to be extraneous if we can remove it without changing the closure of the set of FDs
- Formally,
- Consider a set F of FDs and $\alpha \rightarrow \beta$ in F
  - Attribute A is extraneous in α if A ε α, and F logically implies (F-{$\alpha \rightarrow \beta$}) U{(α –A) $\rightarrow \beta$}
  - Attribute A is extraneous in β if A ε β, and the set of functional dependencies (F-{$\alpha \rightarrow \beta$}) U{$\alpha \rightarrow$ (β - A)} logically implies F

Example

- Find out the extraneous attribute in following FDs
- Case 1:- F:{AB →C and A →C}
  - B is extraneous in AB →C
- Case 2:- F:{AB →CD and A →C}
  - C is extraneous in AB →CD

NORMAL FORMS

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)
  - Also known as Project Join Normal Form (PJNF)

Included in the definition of relation

Defined in terms of FDs

Defined using MVDs

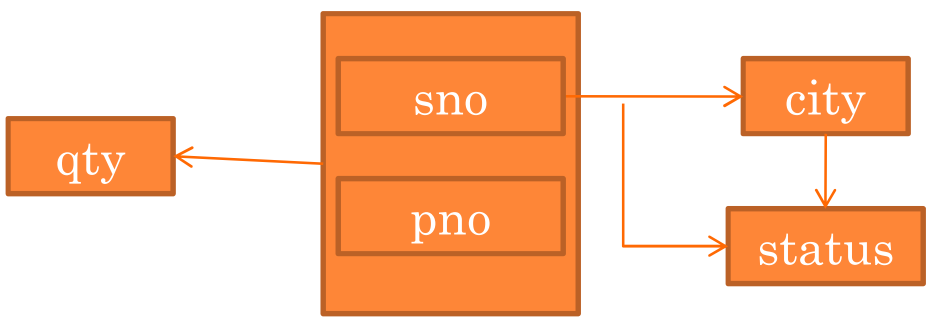
Defined using join dependency

2ND NORMAL FORM

- Prime attribute: an attribute that is part of any candidate key
- 2NF: A relation schema R is in 2NF if every non-prime attribute A in R is not partially dependent on any candidate key of R

Example

- Let's consider the following supplier-parts database system
- *first <sno, status, city, pno, qty>*
- Here the only possible candidate key is (*sno, pno*)
- FDs for relation *first*



- Instance of relation *first*

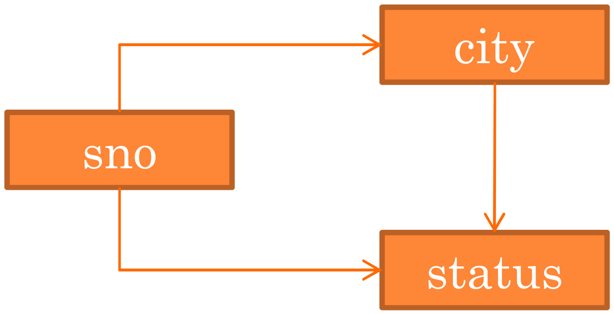
sno	status	city	pno	qty
s1	20	mumbai	p1	300
s1	20	mumbai	p2	200
s1	20	mumbai	p3	400
s1	20	mumbai	p4	200
s1	20	mumbai	p5	100
s1	20	mumbai	p6	700
s2	10	chennai	p1	200
s2	10	chennai	p2	120
s3	10	chennai	p2	340
s4	20	mumbai	p2	230
s4	20	mumbai	p4	432
s4	20	mumbai	p5	120

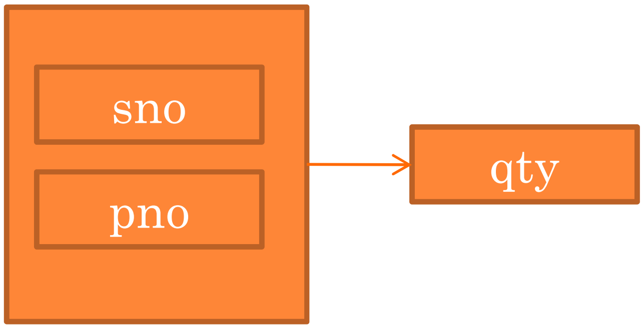
ANOMALIES

- Insert:
  - Insertion not possible until a supplier supplied some items
  - Ex. *s5* located in *Delhi* in cannot be inserted
- Delete:
  - May loose some additional information
  - Ex. if *s3*, *p2* is deleted then we loose the information that *s3* is located in *Chennai*
- Update:
  - Same city value appears in many places
  - Ex. if *s1* moves from *Mumbai* to *Ahmedabad* then update is to be done in many places

DECOMPOSITION

- The relation *first* must be decomposed in such a way so that the decomposed relations satisfy 2NF
- *second* <*sno, status, city*> and
- *sp* <*sno, pno, qty*>
- FDs for the above relations





EXAMPLE OF 2NF RELATIONS

sp

sno	pno	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	700
s2	p1	200
s2	p2	120
s3	p2	340
s4	p2	230
s4	p4	432
s4	p5	120

second

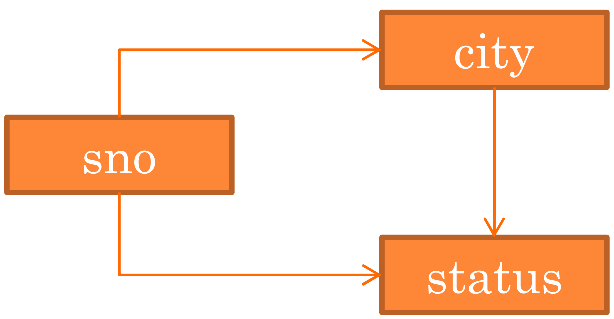
sno	status	city
s1	20	mumbai
s2	10	chennai
s3	10	chennai
s4	20	mumbai
s5	30	delhi

- Thus in *r*(A,B,C,D) if (A,B) is a candidate key and A →D holds
- Then by 2NF *r* can be replaced by *r1* and *r2* as follows
  - *r1*(A,D) candidate key {A}
  - *r2*(A,B,C) candidate key {A,B} and foreign key A references *r1*(A)

3NF

- A relation schema R is in 3NF if, whenever a non-trivial functional dependency X→A holds in R, either
  - X is a superkey of R or
  - A is a prime attribute of R

- Addresses two type of cases-
  - A proper subset of a key of R functionally determines a non-prime attribute
  - A non-prime attribute determines another non-prime attribute. This is same as addressing the transitive dependency
- Now consider relation *second*



ANOMALIES

- Insert:
  - A particular city has a particular status
  - Ex: any supplier in city Kanpur has 10 status
  - Cannot be inserted until there is actually a supplier located in that city
- Delete:
  - If we delete S5 then we lose information that Delhi has status 30
- Update:
  - The status of a given city appears in many places
  - So updating the status value may be problematic

- Now if we decompose the relation *second* into two relations such that they satisfy 3NF
- sc <sno, city>
- cs <city, status>
- The FDs of the above relations are

- Thus if *r*(A,B,C) and A is a candidate key and B → C holds
- Then by 3NF *r* can be replaced by
  - *r1* (B,C) and B is a candidate key
  - *r2*(A,B) and A is a candidate key and foreign key B references *r1(A)*

Properties of decomposition

- Decomposition1: Relation *second* is decomposed into
  - sc <sno, city>
  - cs <city,status>
- Decomposition2: Relation *second* is decomposed into
  - sc <sno,city>
  - ss <sno,status>
- **Which of the above decomposition is lossless and dependency preserving?**

Desirable properties of decomposition

- Lossless join
  - When decomposing a relation into number of smaller ones then it is crucial that the decomposition be lossless
- Dependency preservation
  - The system must not create relation that does not satisfy all the given functional dependencies

LOSSLESS JOIN

- Let R be a relation schema and F be a set of functional dependencies
- Let R_1 and R_2 form a decomposition of R
- The decomposition will be **lossless** if atleast one of the following functional dependencies is in F^+

  $R_1 \cap R_2 \rightarrow R_1$

  $R_1 \cap R_2 \rightarrow R_2$

  In other words, $R_1 \cap R_2$ forms a super key of either R_1 or R_2

Dependency preservation

- Create legal relations preserving the dependencies
- Let F be a set of functional dependencies on a schema R and let $R_1, R_2, \ldots, R_n$ be a decomposition of R
- The restriction of F to R_i is the set of all functional dependencies in F^+ that include only attributes of R_i
- The set of restrictions $F_1, F_2, \ldots, F_n$ is the set of dependencies that can be checked efficiently
- Now we check whether testing only the restrictions is sufficient?

- Let $F' = F_1 \cup F_2 \cup \ldots \cup F_n$
- F' is the set of all functional dependencies on schema R but in general $F' \neq F$
- But if $F'^+ = F^+$ is satisfied then we say that it is a dependency preserving decomposition

DEPENDENCY PRESERVING: EXAMPLE

- Example:
  - Suppose $F=\{A \rightarrow B, B \rightarrow C\}$ and the original relation is r<A,B,C>
  - And the decompositions are r_1<A,B> and r_2<A,C>
- Is it dependency preserving?

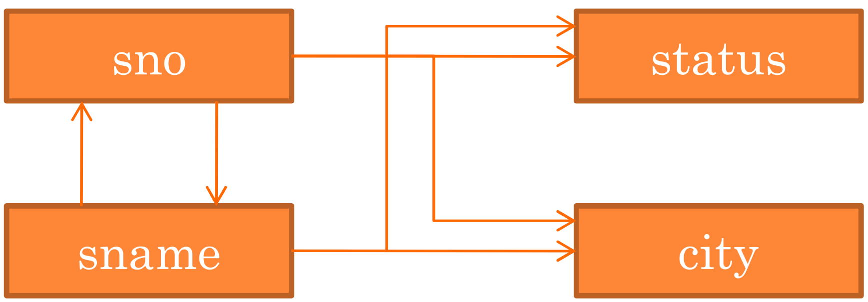
BOYCE/CODD NORMAL FORM (BCNF)

- A relation schema R is in BCNF, if whenever a non-trivial functional dependency X→A holds in R, then X is a superkey of R.
- BCNF is strictly stronger than 3NF definition. Thus, every relation in BCNF also satisfies 3NF but every relation in 3NF does not necessarily satisfy BCNF

- Let's check whether the following relations are in BCNF
- Relation *first <sno, status, city, pno, qty>*
  - The left hand sides of FDs– {sno}, {city}, {sno, pno}
  - Only the last one is a superkey
  - So not in BCNF
- Relation *second <sno, status, city>*
  - The left hand sides of FDs - {sno}, {city}
  - Only sno is a superkey
  - So not in BCNF
- Relation *sp <sno, pno, qty>*
  - The left hand sides of FDs -{sno,pno}
  - That is also superkey
  - It is in BCNF

Some more example

- Now let us consider relation *suppliers <sno, sname, status, city>*
- Potential candidate keys: *sno* and *sname*
- So FDs of this relation



So here *suppliers* is in BCNF

ANOTHER EXAMPLE

- Now consider the relation *ssp <sno, sname, pno, qty>*
- Assume the candidate keys are {sno, pno} and {sname, pno}
- So here the candidate keys overlap
- But is it BCNF?
  - F: sno,pno → qty, sno → sname, sname → sno, sname, pno → qty
  - As {sno} and {sname} are not super keys, so it is not in BCNF

DECOMPOSTION

- So a possible decomposition will be
  - ss <sno, sname>
  - sp<sno,pno,qty>
- And another valid decomposition
  - ss <sno, sname>
  - sp<sname,pno,qty>

Are they lossless and dependency preserving?

ANOTHER EXAMPLE

- Let's consider a relation **sjt <s,j,t>**
- Here attributes ***s: student***, ***j: subject*** and ***t: teacher***
- The meaning of each tuple
  - **"student *s* is taught subject *j* by teacher *t*"**
- Now the following constraints apply
  1. **For each subject, each student of the subject is taught by only one teacher**
  2. **Each teacher teaches only one subject**
  3. **However, each subject is taught by several teachers**

FDs

$\{s,j\} \rightarrow t$

$t \rightarrow j$

$j \rightarrow t$ **does not hold**

What is a possible instance of relation *sjt*?

sjt

s	j	t
Alice	Maths	Prof. Ross
Alice	Physics	Prof. Andrew
Bob	Maths	Prof. Ross
Bob	Physics	Prof. Pal

- So what are the candidate keys?
  - {s,j} and {s,t}
- But the relation is not in BCNF as in $t \rightarrow j$ t is not a super key
- So how do we decompose *sjt*?
- Relation *sjt* can be decomposed into
  - st<s,t>
  - tj<t,j>

st

s	t
Alice	Prof. Ross
Alice	Prof. Andrew
Bob	Prof. Ross
Bob	Prof. Pal

tj

t	j
Prof. Ross	Maths
Prof. Andrew	Physics
Prof. Pal	Physics

There is a problem with this decomposition.
The decomposition is not independent.
Because of FD $\{s,j\} \rightarrow t$

- So the main problem with the last decomposition is that the relations *cannot be independently updated*
- When a relation cannot be decomposed into independent components then it is said to be ***atomic***
- So sometime there may be conflicts between
  - BCNF components
  - Decomposing into independent components
- Thus it may not always possible to satisfy both of them at the same time

Conclusion

- The normalized relations will help achieving the following objectives
  - Less redundancy
  - Avoid inability to represent some information
  - Easier maintenance
- However, for some time critical operations the designer may choose to use non-normalized relations
  - So a normalized relation may be ***denormalized***
  - This may introduce redundancy in some cases but may improve the performance in some specific applications