Reihe Informatik

1 /90

# On redundancy, anomalies and on the question "what do normal forms <u>really</u> do"

Mila E. Majster-Cederbaum

Fakultät für Mathematik und Informatik

der Universität Mannheim

Seminargebäude A5

6800 Mannheim 1

JANUAR 1990

## 1. INTRODUCTION

Since the very introduction of the relational model for data bases [2], informal and intuitive concepts like redundancy and anomaly have played an important role for the design theory of the relational model. In [3] Codd presents examples for anomalies (which he calls "undesirable insert, delete - and update dependencies") and thus yields a motivation for a data base design theory. This theory deals with the question: what constitutes a good data base design? The research in this area has concentrated on investigating the structure of sets of dependencies, defining various normal forms for relation schemes, on developing efficient algorithms for attaining these normal forms on investigations of general forms of dependencies, on complexity questions, on problems connected with the universal relation assumption and representation theory, see [7, 8] for a comprehensive list of references.

Only very few authors [1, 5, 6, 9] tried to tie up ends and investigated, if the original problems, i.e. redundancy and anomalies, have been solved by the theory. Is it true that the normal forms developed are good in the sense that if we use data base schemes that obey normal forms then we know that redundancy and anomalies cannot occur?

The first problem with even formulating such a statement is that there does not exist a general and formal definition of redundancy and anomalies. Even worse, when looking at examples or formalizations given for insert/delete anomalies by various authors we find that they seem to think of different things [1,3]. For special cases, some authors have attempted to capture the meaning of the anomalies in a formal way and related them to normal forms [1,5].

In this paper we first survey various examples for anomalies given in the literature [1,3,8]. We discuss the formalizations and relate them to each other and the examples. We give arguments that show that decomposition of a relation scheme can help in getting rid of deletion/insertion anomalies and can fail in getting rid of update anomalies in the decomposed case.

## 2. EXAMPLES OF ANOMALIES

In our attempt to understand the nature of anomalies we first review various examples that were given in [1, 3, 8] as explanation.

We start with two examples provided by Codd [3]. The first example is a relation scheme

T (S#, P#, SC)

where

S# = supplier number
P# = part number
SC = supplier city

together with the functional dependency

S -> SC

and the following relation for T

| S# | P# | SC |
|---|---|---|
| u | 1 | Poole |
| u | 2 | Poole |
| u | 3 | Poole |
| v | 1 | Feistritz |
| v | 3 | Feistritz |

**FIGURE 1**

Codd writes: "*Looking at a sample instantaneous tabulation of T (FIGURE 1), the undesirable properties of the T schema become immediately apparent. We observe for example that, if supplier u relocates his base of operations from Poole to Tolpuddle, more than one tuple has to be updated. Worse still, the number of tuples to be updated can and, usually will, change with time. It just happens to be three tuples at this instant. Now suppose supplier v ceases to supply parts 1 and 3, but may in the near future supply some other parts. Accordingly, we wish to retain the information that supplier v is located in Feistritz. Deletion of one of the two tupels does not cause the complete disappearance of the association of v with Feistritz, but deletion of both tupels does. This is an example of a deletion dependency which is a consequence of the relational schema itself. It is left to the reader to illustrate a corresponding insertion dependency using this example ....*"

The second example of Codd [3] is the schema

W(E#, JC , D#, M#, CT)

where

E# = employee serial number
JC = employee jobcode
D# = departmentnumber of employee
M# = serial number of department manage
CT = contract type (government or nongovernment)

together with the functional dependencies

D# -> CT, M#
E# -> M#, JC, D#, CT
M# -> CT, D# (E# primary key)

and the relation

| E# | JC | D# | M# | CT |
|---|---|---|---|---|
| 1 | a | x | 11 | g |
| 2 | c | x | 11 | g |
| 3 | a | y | 12 | n |
| 4 | b | x | 11 | g |
| 5 | b | y | 12 | n |
| 6 | c | y | 12 | n |
| 7 | a | z | 13 | n |
| 8 | c | z | 13 | n |

**FIGURE 2**

Codd writes: *"Looking at a sample instantaneous tabulation of W (FIGURE 2) the undesirable properties of the W schema become immediately apparent. We observe, for example, that if the manager of department y should change, more than one tuple has to be updated. The actual number of tuples to be updated can and, usually will, change with time. A similar remark applies if department x is switched from government work (contract type g) to nongovernment work (contract type n). Deletion of the tuple for an employee has*

*two possible consequences:*
*deletion of the corresponding department information if his tuple is the sole one remaining just prior to deletion, and non-deletion of the department information otherwise (./.). If the data base does not permit any primary key to have an undefined value, then D# and CT information for a new department cannot be established in relation W before people are assigned to that department. If, on the other hand, the primary key E# could have an undefined value, and if a tuple were introduced with such a value for E# together with defined values for D# (a new department) and CT, then insertion of E# and IC values for the first employee in that department involves no new tuple whereas each subsequent assignment of an employee to that department does require a new tuple to be inserted. Conversion of W to third normal form consists of replacing W by two of its projections*

$$W_1 = \Pi_{E\#,\ JC,\ D\#}(W)$$

$$W_2 = \Pi_{D\#,\ M\#,\ CT}(W)$$

*We thus obtain the relations tabulated in FIGURE 3.*

| $W_1$ | ( E# | JC | D# ) |
|---|---|---|---|
| | 1 | a | x |
| | 2 | c | x |
| | 3 | a | y |
| | 4 | b | x |
| | 5 | b | y |
| | 6 | c | y |
| | 7 | a | z |
| | 8 | c | z |

| $W_2$ | (D# | M# | CT) |
|---|---|---|---|
| | x | 11 | g |
| | y | 12 | n |
| | z | 13 | n |

***FIGURE 3***

*Note how the undesirable insertion, update and deletion dependencies have disappeared with the removal of transitive dependencies. No essential information has been lost, since at any time the original relation W may be recovered by taking the natural join of $W_1$ and $W_2$ on D#."*

Let us now consult Ullman [8] about anomalies. We find there the example of a relation scheme

SUPPLIERS (SNAME, SADDRESS, ITEM, PRICE)

Ullman writes: "*We can see several problems with this scheme.*

1. *Redundancy. The address of the supplier is repeated once for each item supplied.*

2. *Potential inconsistency (update anomalies). As a consequence of the redundancy, we could update the address for a supplier in one tuple while leaving it fixed in another. Thus we would not have a unique address for each supplier as we feel intuitively we should.*

3. *Insertion anomalies. We cannot record an address for a supplier if that supplier does not currently supply at least one item. We might put null values in the ITEM and PRICE components of a tuple for that supplier but then when we enter an item for that supplier will we remember to delete the tuple with the nulls? Worse, ITEM and SNAME form a key for the relation and it might be award or impossible to look up tuples with null values in the key.*

4. *Deletion anomalies. The inverse to problem (3) is that should we delete all the items supplied by one supplier, we unintensionally lose track of its address...*

   *In this example all the above problems go away if we replace SUPPLIERS by the relation schemes*

*SA (SNAME, SADDRESS)*
*SIP (SNAME, ITEM, PRICE) .* "

It is typical that we do not find in [8] any statement that shows the general benefit of normal forms and decomposition.

Let us now consider what Bernstein and Goodman understand by anomalies, who were the first to attack the problem to prove that normal forms are good [1].
In their paper [1] introduce anomalies by referring to the above second example of Codd and Codd's comments on it.
It is very important at this point to note that in [1] Bernstein and Goodman only partially quote what Codd wrote: in our above quotation we have marked

the position up to which they quote Codd by a ./. sign, i.e. they quote Codd's comments on update and delete anomalies. Then Bernstein and Goodman continue in their paper with their own explanation about what an insert-anomaly is, which is different from Codd's.
They write: *"Inserting an employee tuple has the complementary problem: if the employee is the first member of a department, we must simultaneously insert new M# and CT information for that department; inserting the second, third, etc.) employee into that department has no such requirements".*

In a framework of a data base scheme (= a set of relation schemes, each of which has an associated set of functional dependencies they then give a formal definition, when a relation scheme with associated set of functional dependencies is free of anomalies (in their sense) and relate this concept to Boyce-Codd normal form.

Yet another view of insert/delete anomalies is given by Fagin [5]. Let us consider a relation scheme ABC with the functional dependency B -> C and the condition that the values for A, B, C are numbers in {1.100}. A relation for this scheme is e.g.

| r = | A | B | C |
|---|---|---|---|
| | 1 | 3 | 99 |
| | 2 | 3 | 99 |
| | 2 | 4 | 99 |

In the sense of Fagin this scheme has an insert anomaly because there is a tuple t, e.g. t = (5, 3, 97), satisfying the range conditions such that r ∪ {t} is no longer a "valid" relation for the scheme (because r ∪ {t} violates B -> C).

## 3. DISCUSSION OF EXAMPLES AND FORMALIZATIONS

We want now to investigate the various concepts and their relevance for database design theory. For that purpose we separate the issue of update (= replacement, = change) anomalies from those of insert/delete anomalies and treat the latter first.

### 3.1 DELETE ANOMALIES

In Codd's first example (FIG. 1) a delete anomaly arises because the address of a supplier gets lost when this supplier momentarily does no longer supply any parts and does not get lost otherwise. In Codd's second example ( FIG. 2) "information" on a department may or may not disappear depending on the fact if the employee tuple to be deleted is or is not the last containing this information. So, in both examples a "delete anomaly" arises because there is some information that is considered to be relevant, i.e. the "information" on the address of a supplier in the first case and the manager and contract type "information" in the second case.

Now let us consider a slightly modified situation. We consider again the relation scheme W (E#, JC, D#, M#, CT) but admit that a department has more than one manager and contract type, i.e. we drop D# -> M#, CT while maintaining the remaining dependencies. We now look at the sample relation

| E# | JC | D# | M# | CT |
|---|---|---|---|---|
| 1 | a | x | 11 | g |
| 2 | c | x | 11 | g |
| 3 | a | y | 12 | n |
| 4 | b | x | 14 | n |
| 5 | b | x | 15 | g |
| 6 | c | y | 12 | n |
| 7 | a | z | 13 | w |
| 8 | c | z | 13 | w |

**FIGURE 4**

After the removal of the tuple for the employee with E# = 1 (or E# = 2) we can still obtain the "information" that department x has the three managers 11, 14, 15. After the removal of the tuple with E# = 4 (or E# = 6) this "information" is lost. Now, if we consider this "information" as relevant, we have a delete anomaly.

The same arguments can be made for Ullman's example. Ullman's example describes the same type of delete-anomaly as Codd. Ullmann states that a delete anomaly for the scheme

SUPPLIERS (SNAME, SADDRESS, ITEM, PRICE)

arises, because "when we delete all the items by one supplier then we unintensionally lose track of his address". So again, we have some "information" that is considered to be relevant, namely the address, that may or may not get lost. Now, let us modify Ullman's example by taking as functional dependency

SNAME, ITEM -> PRICE

and hence SNAME, SADDRESS, ITEM form a key. By this we model that a company may locate at different places. In this modified example a delete anomaly arises as follows: when location x of company y currently does not supply any items then the information about this location is lost. Take e.g.

| SNAME | SADRESS | ITEM | PRICE |
|---|---|---|---|
| SIEMENS | München | a | 100 |
| SIEMENS | München | b | 200 |
| SIEMENS | Erlangen | c | 100 |
| IBM | München | a | 90 |

**FIGURE 5**

Here, if SIEMENS Erlangen stops supplying c, we lose the information about the location Erlangen alltogether. It should be clear that an attempt to get rid of the delete anomaly in our modified example by decomposing into BCNF must fail.

From the above example we argue that deletion anomalies arise because there is "information", considered to be relevant, that may unintensionally get lost. This "information" may or may not obey functional dependencies. If it does not, decomposition into BNCF will not help getting rid of the problem. The reason, why some "information" is considered to be relevant and some is not, depends on what operations will be performed on the data base. If, e.g. we will never ask for the address of a supplier that currently does not supply anything then we do not care that such address information may get lost.

Of course our above arguments concerning delete anomalies and BCNF are informal, so let us have a look at formal definitions for delete anomalies and the related results as found in [5].

In [5] Fagin defines a relation scheme as a set of attributes together with a set of constraints. A relation is a valid instance of a relation scheme if it has the same attributes and obeys all constraints. Fagin considers classical constraints as e.g. functional, multivalued and join dependencies and in addition domain dependencies (DD) and key dependencies (KD). Fagin then defines that a relation scheme has a deletion anomaly (which we call F= deletion anomaly) iff there is a valid instance (of the scheme) that contains a tuple the removal of which yields an instance that is no longer valid (Fagin then develops a normal form that is shown to be equivalent to the nonexistence of F-insert/delete anomalies).

It is easy to see that a F-deletion anomaly is something different than the delete anomalies presented by Codd and Ullman. The deletion of a tuple in Codd's example does not yield a nonvalid instance, it may or may not cause a loss of information.

So, Fagin's formal model, together with his normal form and the associated results, does not help in getting results for the connection between normal forms and Codd-type deletion anomalies.

## 3.2 INSERT ANOMALIES

In Codd's second example an insert anomaly arises because

i) we cannot store D# and CT information for a new department before people are assigned to that department in case that primary keys are not allowed to have undefined values

or

ii) if primary keys may have undefined values, then other irregular situations occur, see section 2) above.

In Ullman's example an insert anomaly arises because

i) we cannot record an address for a supplier if that supplier does not currently supply at least some item (if null values are not admitted)

or

ii) if null values for ITEM and PRICE are admitted then other irregular situations occur (see section 2).

So, basicly Codd and Ullman describe the same problem, namely that there is some "information" that is considered to be important (D#-CT, SNAME-SADDRESS) that sometimes cannot be represented in the chosen relation scheme, if null values are not admitted. And if null values are admitted, the information could be represented, but then other problems arise.

As in the case of delete anomaly we argue that this missing facility to store information is not restricted to the case of functional dependencies. Let us e.g. for the scheme W (E#, JC, D#, M#, CT) drop the functional dependencies D# -> CT, M#, i.e. a department may have more than one manager and work under different contract types. We still request M# -> D#, CT and E# -> JC, D#, M#, CT. In this modified example that we cannot adequately represent under which contract types a department may work. E.g. the situation that one group of employees works under manager 11 under contract type g, while another could also work under some other manager under contract type n is not representable without null values. With nulls we might get something like

| E# | JC | D# | M# | CT |
|---|---|---|---|---|
| 1 | a | x | 11 | g |
| 2 | c | x | 11 | g |
| ⊥ | ⊥ | x | ⊥ | n |

**FIGURE 6**

where the same problems with nulls arise as Codd and Ullman mention.

So again, we argue that insert anomalies are not restricted to the case of functional dependencies (and hence decomposition into BCNF does not help). For an insert anomaly it is important that certain information is considered to relevant, i.e. in the above example under which contract type(s) a department may work. If we are only interested in the contract type(s) under which a department currently works, we do not get into problems with the above scheme.

Let us now see, what formal treatments we find in the literature for the case of insert anomalies.

For Fagin [5], who works in the framework of a single relation scheme with constaints (FD, MVD, JOIN-dependencies, DD, KD) a relation scheme has an insertion anomaly (which we call F-insertion anomaly) if there is a valid instance of the scheme and a compatible tuple such that adding the tuple to the instance yields a non-valid instance. (A tuple is compatible with an instance if it has the right attributes, its components satisfy the

domain dependencies and if -for every key- it does not have the same value as any other tuple of the instance).

This notion of insert-anomaly is different from that of the examples of Codd and Ullman. The problem in the examples of Codd and Ullman is that certain information cannot be adequately stored and not that the addition of tuples yields non-valid instances.

Let us now turn to the approach of Bernstein and Goodman [1]. Even though [1] seem to quote Codd, we already found that, while working with the same sample relation scheme, Bernstein and Goodman's explanation of insert anomaly is different from Codd's (see end of section 2). The essence of their explanation is that an insert anomaly arises because sometimes (e.g. when an employee tuple to be added is the first member of a department) we must give a completely specified tuple whereas sometimes we may only partially specify such a tuple and the missing information can be deduced by means of functional dependencies. E. g. let us consider the relation given in FIGURE 2. Then for a first member of department v we have to give a tuple like

$$(9, a, v, 16, g)$$

whereas we may add the tuple

$$(10, a, x, \perp_1, \perp_2,)$$

and the missing information is deduced to yield

$$\perp_1 = 11$$
$$\perp_2 = g$$

Evidently, this notion of insert anomaly which is based on redundancy resulting from functional dependencies is a quite different one than that of Codd and Ullmann. For Codd and Ullman an insert anomaly is, that there are problems of representing certain relevant "subinformation": if there is no employee in a certain department then M¤, CT information for that department cannot be well represented in the relation scheme W, i.e. we have a partial information of the form

$$(\perp_1, \perp_2, V, 16, g)$$

where the undefined elements may cause problems for later operations and clearly here, values for $\perp_1$, $\perp_2$ cannot be deduced.

## 3.3 HOW TO HANDLE DELETE/INSERT ANOMALIES

Summarizing 3.1 and 3.2 we observe that the existing formal investigations do not clarify the nature of delete and insert anomalies (as presented by Codd) and their relation to normal forms.

We suggest to take the following attitude towards these anomalies: at the design stage for a relation scheme over the attributes $A_1, \ldots A_n$ one has to make the decision which attribute combinations constitute "relevant information". For every such attribute combination, e.g. $A_1A_2A_3$, we introduce a new attribute N and add the functional dependency

$$A_1A_2A_3 \rightarrow N$$

to the other conditions for the relation scheme. Let us assume that there is no functional dependency relating $A_1 \ldots A_n$ in the original design. If we suceed to give a loss- lessjoin decomposition $\rho = \{R_1, \ldots R_k\}$ of the relation scheme into BCNF preserving functional dependencies, then we know that there will be an element

$$R_i = A_1A_2A_3\ N^{(*)} \text{ in } \rho.$$

$R_i$ represents the relevant information which can now be maintained when other parts of information are deleted or not yet provided. So, in a certain sense BCNF helps us in getting rid of delete/insert anomalies. The problem is more difficult, however, for update anomalies, as can be seen in the following.

## 3.4 UPDATE ANOMALY (change, replacement)

As far as the examples are concerned, Codd, Ullmann and Bernstein and Goodman what an update anomaly is at least in the context of a single relation scheme with an associated set of functional dependencies. The functional dependencies induce the existence of "redundant" information, which causes problems, when updating occurs. Decomposition into BCNF is proposed as a cure [4,7].

There are, however, other forms of "redundancy" and "update-anomalies" than displayed in the above examples.

a) There are other forms of constraints that induce the existence of "redundant" information, which causes problems when updating occurs. Consider as an example a relation scheme over the attributes A, B, C

(*) Clearly we do not have to actually store values for N.

together with the multivalued dependency A -> -> C and no functional dependencies (hence trivially in BCNF) and the relation

| A | B | C |
|---|---|---|
| a | b | c |
| a | b' | c' |
| a | b'' | c'' |
| a' | b | c |
| a | b' | c |
| a | b | c' |
| a | b' | c'' |
| a | b'' | c' |
| a | b'' | c |
| a | b | c'' |

**FIGURE 7**

Here e. g. the following can happen

i) the A-value of the first tuple is changed into $\bar{a}$ then the tuple (a b c) has to be added in order to satisfy A -> -> C

ii) the A-value of the fourth tuple is changed into a then a number of tuples depending on the actual relation has to be added

iii) the C-value of a tuple is changed. Then a number of tuples, depending on the actual relation has to be modified as well. Of course , we might try to decompose a relation scheme with multivalued dependencies into 4NF. But can we be sure in general that then the above problems go away? Probably not, see b).

b) Even if we restrict ourselves for the moment to relation schemes with functional dependencies then the decomposition of a relation scheme into BCNF can produce undesired effects. For this consider the example of a relation scheme R over the attributes ABCD and the functional dependencies F = {A -> B, B -> C D} and the decomposition $\rho_1$ = {AB, BC, BCD}. $\rho_1$ is a loss-less join decomposition of R that preserves functional dependencies. Each $R_i$ is in BCNF. Now consider a relation r for R that contains the tuple t = (a, b, c, d) in the decomposition of r we find

$$(a, b) \in \Pi_{AB}(r) = r_1$$

$$(b, c) \in \Pi_{BC}(r) = r_2$$

$$(b, c, d) \in \Pi_{BCD}(r) = r_3$$

If we now modify c to c' we have to perform this update in $r_2$ and in $r_3$, a situation that is probably undesirable.
So, even though the relation schemes of the decomposition are in BCNF and the decomposition is a loss-less join decomposition and preserves functional dependencies, we have some form of "redundancy" and hence undesired update behaviour.

Had we decomposed the scheme by

$$\rho_2 = (AB, BC, BD) = (R'_1, R'_2, R'_3)$$

then this decomposition is again a lossless join decomposition that preserves functional dependencies. The $R'_i$ are in BCNF. In this decomposition the change of a C-value remains local. A change of B-values does not remain local, however, this can be considered as an effect of a different type as B is key in $R'_2$ and $R'_3$ and a foreign key in $R'_1$.

In [6] decompositions like $\rho_1$ that contain "superfluous" attributes (namely the attribute C in BCD) are treated and it is shown how a relation scheme with functional dependencies can be decomposed into 3 NF such that no "superfluous" attributes occur.

However, superfluous attributes are not the only problems with decompositions.

The situation can be more complicated; consider e.g. a relation scheme with attributes BCEF and functional dependencies

BC -> F
CE -> F

Let $\tau_1 = (BCE, BCF, CEF) = (R_1, R_2, R_3)$

then each $R_i$ is in BCNF, the decomposition is lossless and all functional dependencies are preserved.(*) Let us now consider the example given in FIGURE 8.

| | B | C | E | F |
|---|---|---|---|---|
| r = | b | c | e | f |
| | b | c | e' | f |
| | b' | c | e' | f |

A relation for BCEF

| | B | C | E |
|---|---|---|---|
| $r_1$ = | b | c | e |
| | b | c | e' |
| | b' | c | e' |

| | B | C | F |
|---|---|---|---|
| $r_2$ = | b | c | f |
| | b' | c | f |

| | C | E | F |
|---|---|---|---|
| $r_3$ = | c | e | f |
| | c | e' | f |

Its projections onto the $R_i$

**FIGURE 8**

Now changing f to $\tilde{f}$ in the second tuple original relation r (that is not in BCNF) forces us to change $\tilde{f}$ to f also for the other tuples in r. In the decomposed case we have to modify all f's in $r_2$ and $r_3$ !

Now one might argue that this decomposition was akwardly chosen, so let us consider the decomposition that results from the decomposition algorithm [8 ]

$$\tau_2 = \{BCF, BCE\}$$

(*) $\tau_1$ is the only loss-less join decomposition of BCEF that preserves functional dependencies.

which is lossless and both schemes are in BCNF but $\tau_2$ does not preserve functional dependencies. This decomposition leads to

$s_1$ =

| B | C | E |
|---|---|---|
| b | c | e |
| b | c | e' |
| b' | c | e' |

$s_2$ =

| B | C | F |
|---|---|---|
| b | c | f |
| b' | c | f |

**FIGURE 9**

If we now consider the modification of f in the second tuple of the relation r as before then all f's in $S_2$ have to be modified as well.

What is even worse with examples alike scheme BCEF and its functional dependencies is that it is not obvious to find out in the decomposed case what modification have to be carried out as a consequence of the modification of a single value!

Our list, however, of intriguing examples is not yet ended. Consider the scheme BCEF with functional dependencies

CE -> F
B -> E  with key BC

and the following sample relation

r =

| B | C | E | F |
|---|---|---|---|
| b | c | e | f |
| b' | c | e | f |
| b'' | c | e' | f' |
| b''' | c | e' | f' |

$\prod$ (r) = CEF

| C | E | F |
|---|---|---|
| c | e' | f' |
| c | e | f |

$\prod$ (r) = BC

| B | C |
|---|---|
| b | c |
| b' | c |
| b'' | c |

$\prod$ (r) = BE

| B | E |
|---|---|
| b | e |
| b' | e |
| b'' | e' |

**FIGURE 10**

Consider now the update of the EF-value for the second tuple in r where we wish to substitute b' c e f by b' c e' f''. In order to satisfy the constraints we have to modify the third and fourth tuple as well yielding a relation r'; if we had changed e into e'' the third and fourth tuple were not affected, so we have a case of an "update anomaly". Let us now consider the lossless join decomposition into BCNF

$$\rho = \{CEF, BE, BC\}$$

let us now look at the action we have to perform on the projections of r in order to be able to obtain r'. In this case we have to modify the second tuple in $\prod_{CEF}(r)$ even though the first tuple of $\prod_{CEF}(r)$ originally held the respective information for t = (b' c e f). In addition we have to modify the second tuple of $\prod_{BE}(r)$.

We consider now the case that we want to modify -in the relation r- the second tuple (b' c e f) into (b' c e'' f).

In this case no other tuples of r have to be modified to preserve consistency. But for the projections we have to do the following steps:

- insert a tuple c e'' f into $\prod_{CEF}(r)$

- substitute the second tuple in $\prod_{BE}(r)$ by b' e''.

In contrast to the examples usually presented in connection with update anomalies, where it is typically demonstrated that decomposition removes the undesired effects [3,4,8 ], our above presented cases demonstrate that after decomposing the situation may become even worse.

The question remains, if in such cases there is a decomposition such that the above undesired update behaviour can be avoided? If not, what conditions must the functional dependencies for a given relation scheme satisfy such that there exists a "useful" decomposition, i.e. a decomposition that is free of "update anomalies?"

From the above we conclude that we need a formal framework which allows us to talk about "redundancy" and "update anomalies" for sets of relation schemes on which certain constraints are given.

Let us now look at the formal models given in [1,5] to capture the meaning of anomalies. In [5] only insert and delete anomalies are treated. In [1] Bernstein and Goodman treat the case of a single relation scheme with functional dependencies. They define a relation scheme with a set F of functional dependencies free of replacement (= update) anomalies iff for each relation for that scheme that satisfies F and for every X -> Y in F there are no two tuples with the same X value. Obviously, under this definition a relation scheme (R, F) is free of update anomalies iff (R, F) in BCNF. The case of constraints other than functional dependencies and the case of a decomposition $\rho$, i.e. a set of schemes is considered in [1] for insert/delete anomalies on the basis of a universal relation assumption but not for replacement.

Summerizing the topic of update (= replacement) anomaly we make the following observations:

O. There does not exist a formal framework for defining and discussing update anomalies

1. If we deal with the case of a single relation scheme with functional dependencies then BCNF guarantees that the scheme does not display update anomalies [1]. This is intuitively obvious, because BCNF means that there are no other functional dependencies than those derived from key information

2. update anomalies originating from conditions other than functional dependencies have not been treated so far even for the case of a single relation scheme

3. if $\rho$ is a functional dependencies preserving loss-join decomposition (e.g. into BCNF) of the relation scheme R with functional dependencies there is no knowledge about it, if and when we got rid of the problem of update anomalies. It even seems that decomposing might make problems worse!

## 4. CONCLUSION AND FURTHER INVESTIGATIONS

We discussed some issues that arise, if we want clarify the benefit of decompositions of a relation scheme according to some normal form. In particular we were interested in the question if decomposing a relation scheme guarantees the disappearance of anomalies. We argue that, in a certain sense, lossless join decompositions into BCNF that preserve functional dependencies help to handle insert/delete anomalies as introduced by Codd [3]. We presented some of intriguing examples that show that such decompositions may fail in getting rid of update anomalies as introduced by Codd [3]. The question, under which circumstances one can get rid of update anomalies is open.

It has to be considered as a deficiency of the data base design theory developed so far, that the notions of anomaly and redundancy, that were the origin for the normal form theory, have not been formally defined and hence no general formal statement about the value of normal forms with respect to anomalies exists.

## REFERENCES


1 Bernstein, P. Goodmann, N. — What does Boyce-Codd Normal Form do? Proc. Int. Conference on very large Data Base (1980) 245-259

2 Codd, E.F. — A relational model of data for large shared data banks. Com. ACM 13, 6 (1970)

3 Codd, E.F. — Further normalization of the data base relational model In: Data Base Systems, Courant Computer Science Symposia 6, Prentice Hall (1972) 65-98

4 Codd, E.F. — Recent investigations in relational data bases. Proc 1974 IFIP Congress, North Holland (1974) 1017-1021

5 Fagin, R. — A Normal Form for Relational Data Bases that is based on Domains and Keys. ACM TODS Vol. 6, No. 3 (1981) 387-415

6 Ling, T. Frank, T. Kameda T. — An Improved Third Normal Form for Relational Data Bases. ACM TODS, Vol. 6, No. 2 (1981) 329-346

7 Maier, D. — The Theory of Relational Data Bases. Computer Science Press (1983)

8 Ullman, J.D. — Principles of Data Base and Knowledge Base Systems. Vol. 1 Computer Science Press 1988

9 Vossen, G. — A new characterization of FD implication with an application to update anomalies. Information Processing Letter 29 (1988) 131-135