Extending Classical Functional Dependencies for Physical Database Design

Tok Wang LING Cheng Hian GOH** Mong Li LEE**

* Department of Information Systems & Computer Science
National University of Singapore

** Sloan School of Management
Massachusetts Institute of Technology

Abstract

Traditionally, database design activities are partitioned into distinct phases in which a logical design phase precedes physical database design. The objective of the logical design step is to eliminate redundancies and updating anomalies using the notion of data dependencies, while leaving the physical design step to consider how the database schema may be restructured to provide more efficient access. We argue in this paper that the separation of these two steps often result in physical database design not being able to benefit from knowledge of the semantics of data captured in the earlier phases of the database design life cycle. As a step towards overcoming this problem, we demonstrate how classical functional dependencies can be extended to capture data semantics relevant to the design of database schemas which are more desirable from the efficiency point of view. This is accomplished via the introduction of strong and weak functional dependencies, as well as three new formal forms — the relaxed 3NF, replicated 3NF and relax-replicated 3NF — induced by the extended functional dependencies. These new normal forms provide a framework for designing database schemas which are more efficient, while not compromising the integrity of the underlying database.

1 Introduction

The design of an integrated database is a complex process. Traditionally, this is accomplished using a multi-step framework consisting of requirements analysis, conceptual design, logical design, followed by physical database design [15]. More specifically, the task of logical design is to arrive at a database schema which is devoid of updating anomalies or redundancies. The theory underlying logical database design, frequently referred to as *normalization*, has been well-researched [1]. Depending on the types of constraints (called *data dependencies*) which are accounted for, one can arrive at a number of *normal forms* which guarantees that certain types of anomalies will not occur. The objective of the physical database design step, on the other hand, is to identify how the database can be optimized for application-specific database accesses. Recognizing that normalization frequently leads to expensive joins in query processing, many practitioners have advocated *denormalization* as an integral activity in physical database design [10,12]. The methods proposed however do not take into account the semantics of data involved and often leaves the enforcement of data integrity to the application programmer in an ad hoc fashion.

We advocate that the logical and physical design steps should not be undertaken as disjoint activities. This paper takes a first step towards the integration of the two by proposing how classical functional dependencies can be extended to provide for more intelligent choices during physical design, providing guidelines for the design of more "efficient" schemas while not compromising the integrity of the database.

The rest of this paper is organized as follows. In section 2, we give a brief review of classical normalization theory and motivate the need for integrating the logical and physical design steps with a number of examples. Section 3 formalizes the extensions to classical functional dependencies and their associated normal forms. Section 4 proposes various approaches to preserving the integrity of the database in the a schema complying with the extended normal forms. Finally, section 5 summarizes our contribution and provides some suggestions for further work.

2 Normalization: Theory and Practice

The concepts of data dependencies and normalization lie at the heart of relational database theory. It has been shown that redundancies and various updating anomalies (threatening the integrity of a database) can be avoided by designing relation schemes which conform to certain *normal forms* [1]. The example which follows examines these anomalies. The subsequent discussions assume that the reader is familiar with the fundamental concepts of relational databases at the level of [16].

Example 1 Consider the *Supplier-Part* database which captures information of suppliers, parts, and the quantity of parts ordered from suppliers. These information can be kept using one *unnormalized* relation, called `SUP_INFO`

`SUP_INFO(sno, pno, sname, addr, pname, color, qty)`

We can immediately identify several problems associated with such a scheme [16]:

1. *Redundancy.* The information of each supplier and part are repeated many times over.
2. *Potential inconsistency (update anomalies).* As a result of redundant representation of supplier and part information, update to supplier or part information is costly and there is a possibility of having inconsistent updates.
3. *Insertion anomalies.* We cannot record information concerning a particular supplier until he begins to supply us with a part. (Note that we are not able to put null values in `pno` since `sno` and `pno` together make up the key of the relation).
4. *Deletion anomalies.* Should we delete all the items supplied by a supplier, the supplier information is lost. The converse applies to part information. □

Functional dependencies (FDs) are introduced as one of the ways in which redundancies can be identified. According to the extent in which these redundancies are removed, we obtained relations which are in an assortment of normal forms. The FD-related normal forms which are commonly used as tools for logical design of databases include the *Codd third normal form (3NF)* [6], *Boyce-Codd normal form (BCNF)* [7], and the *improved third normal form (improved 3NF)* [14].

Example 2 The relation `SUP_INFO` in Example 1 can be transformed to the following relation schemes which is in BCNF (hence, 3NF):

SUPPLIER(<u>sno</u>, sname, addr)
PART(<u>pno</u>, pname, color)
SUPPLY(<u>sno, pno</u>, qty)

No redundancies exist and none of the anomalies identified in Example 1 can occur in this schema. □

While no one would dispute the elegance of relational normalization theory, having a database which is normalized to the highest possible degree may not be at all desirable from a practical point of view. There are at least two scenarios why this may be so. First, normalization theory advocates that data should be organized into clusters of "singleton" relationships from which more complex associations can be composed. This often translates to a high degree of fragmentation which is undesirable for efficient query processing. Second, classical normalization theory is build on the foundation of data dependencies, which defines in very precise terms the set of integrity constraints real world data must obey. This is fine if rules are strictly adhered to: unfortunately, exceptions to the rule are often the norm in the real world. Since one must always design the database to accommodate the real world and not vice versa, it is often the case that we are not able to exploit the "norm" with existing normalization theory. These two situations are best illustrated with the following examples.

Example 3 Consider once again the *Supplier-Part* database which is represented by the database schemes in Example 2. Assume that the enterprise requires frequent reporting of the information held in the relation `SUPPLY`. Since numbers mean little to human beings, both the supplier name (`sname`) and part name (`pname`) must be reported along with the quantity (`qty`). This effectively means computing a join on all the three relations each time a piece of information is needed from the relation `SUPPLY`. This is an expensive and time-consuming operation which is highly undesirable. In an attempt to cut down the cost of this transaction, the database designer may wish to include `sname` and `pname` in the relation `SUPPLY`. As a result, the relation `SUPPLY` will not be in third normal form and this is certain to cause some disputes concerning its "goodness". Furthermore, one cannot help but wonder if the integrity of the database would be violated because of the redundant representation. □

Example 4 Consider a database whose intention is to capture information of employees in an enterprise. These information may be captured in the relation `EMP`:

`EMP(emp#, empname, phone#, designation, ... )`

The underlying assumption here is that an employee has only one telephone number. This assumption may be valid except for a handful of individuals in the highest management echelon, in which case he may have two or more telephones in his office. To accommodate these exceptions, the database designer has two alternatives. The first option is to create a new attribute:

`EMP(emp#, empname, phone#, alt_phone#, designation, ...)`

This new attribute (`alt_phone#`) results in higher storage cost since disk space will be allocated for it even though a large proportion of tuples in this relation will not have a value. Furthermore, this scheme will not be able to capture the information of an employee who has, say, three telephones in his office.

A second alternative (which classical relational database theory would advocate) is to model the relationship between `emp#` and `phone#` (in Example 5) would be modelled using a *multivariate dependency (MVD)* [8] `emp#` $\rightarrow\!\!\!\rightarrow$ `phone#`. This will lead to removing `phone#` from relation `EMP` to form a new relation (`EMP_PHONE`) through decomposition:

`EMP(emp#, empname, designation, ...)`
`EMP_PHONE(emp#, phone#)`

There are two problems with this second approach. This first is similar to that raised in Example 3: retrieval of employee phone numbers based on employee name or designation is slow since a join is always required. This is particularly uncomfortable since we know that almost all the employees (with few exceptions) have only one phone each. The second is more philosophical in nature: i.e., a functional dependency which is violated only in special cases is semantically not the same as an MVD. □

Example 3 above demonstrates that contrary to the motivations behind classical normalization theory, controlled redundancies can be beneficial as it can reduce dramatically the effort needed to access information which are needed frequently. This is all the more reasonable if the redundant attribute is not updatable or whose update need not be reflected in real-time. In the first case, update anomalies cannot occur if the right-hand-side attribute is never updated. In the second case, an update which need not be reflected in real-time can be performed offline, which makes it possible for us to ensure that the update is consistent. Therefore the benefits outweigh the extra disk space required.

Example 4 suggests that the concept of functional dependency is inadequate because it does not consider cases in which the functional relationship between two attributes holds *in general* but may be violated in rare cases. This is significant because such relationships occur frequently in real life and has given rise to poor design and subsequently, poor performances.

3 Extending Classical Functional Dependencies and their associated Normal forms

In this section, we demonstrate how classical functional dependencies (FDs) can be extended to circumvent the problems highlighted in the earlier section. Specifically, we introduce the notion of *strong* FDs and *weak* FDs in addition to classical FDs.

Definition 1 [Strong Functional Dependency (SFD)] Let $X \rightarrow Y$ be a FD such that for each $Z \in Y$, $X \rightarrow Z$ is a full FD. Then $X \rightarrow Y$ is a SFD (denoted by $X \overset{\mathtt{S}}{\rightarrow} Y$) if all the attributes in Y will not be updated, or if the update need not be performed at real-time or on-line.

For instance, in Example 1, the functional dependencies `sno` $\rightarrow$ `sname` and `pno` $\rightarrow$ `pname` are also SFDs.

Definition 2 [Weak Functional Dependency (WFD)] Let R be a relation schema. A WFD, denoted by $X \overset{\mathtt{W}}{\rightarrow} Y$, between two sets of attributes X and Y that are subsets of R states that $X \rightarrow\!\!\!\rightarrow Y$ holds and most of the X-values are associated with a unique Y-value in R, except for a handful of X-values which may be associated with more than one Y-value. If we remove these handful of exceptional tuples from R, then for each $Z \in Y$, $X \rightarrow Z$ is a full FD in R.

For instance, in Example 4, we have `emp#` $\overset{\mathtt{W}}{\rightarrow}$ `phone#`. The functional dependency `emp#` $\rightarrow$ `phone#` does not hold as we have a handful of employees who may have more than one phone.

Corollary 1 $X \overset{s}{\rightarrow} Y \Rightarrow X \rightarrow Y \Rightarrow X \overset{w}{\rightarrow} Y$.

Proof. The proof follows trivially from the definitions of strong and weak FDs. Note however that the converse is not true (i.e., $X \rightarrow Y \not\Rightarrow X \overset{s}{\rightarrow} Y$ and $X \overset{w}{\rightarrow} Y \not\Rightarrow X \rightarrow Y$). □

The purpose of the strong and weak FDs is to allow database designer to capture those aspects of data semantics which would otherwise be unavailable during physical database design. Our motivation stems from the observation that classical FDs are not able to capture these semantics adequately. The strong and weak FDs therefore serve to provide the link between the logical design and physical design steps by allowing the database designer to make more intelligent decisions about how the database can be structured without compromising the integrity the logical design step so painstakingly seek to preserve.

Definition 3 [Replicated 3NF] Let $R = \{R_1, R_2, ..., R_n\}$ be a database schema and A_j be the set of attributes in R_j, for j = 1, 2, ..., n. A relation schema $R_i \in R$ is said to be in replicated 3NF if

1. for each $X \overset{s}{\rightarrow} Y$, $X \cup Y \subseteq A_i$, 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 exists a unique $R_j \in R$, $j \neq i$, such that X is a key of R_j and $Y \subseteq A_j$ (R_j is said to be the *primary instance* of R_i wrt the attributes in $X \cup Y$);
   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, and

2. let $\beta = \{ B \mid X \overset{s}{\rightarrow} B, \{X, B\} \subseteq A_i, X$ is not a key of R_i, and B is non-prime$\}$; the relation schema obtained from R_i after removing all attributes in β is in 3NF.

Example 5 Consider the database schema presented in Example 3, which is reproduced below:

`SUPPLIER(sno, sname, addr)`
`PART(pno, pname, color)`
`SUPPLY(sno, pno, sname, pname, qty)`

Clearly, `SUPPLY` is not in Codd 3NF since `sno` $\rightarrow$ `sname` but `sname` is nonprime and `sno` is not a candidate key. However, it is in replicated 3NF by Definition 1 since

1. sno $\stackrel{s}{\rightarrow}$ sname and pno $\stackrel{s}{\rightarrow}$ pname, and moreover, by Case 1 of Definition 3, SUPPLIER and PART are the primary instances of SUPPLIER wrt the attribute sets {sno, sname} and {pno, pname} respectively; and

2. the relation scheme obtained by removing attributes sname and pname from SUPPLY is in Codd 3NF.

Example 6 Consider next the following database schema:

EMP_MGR(<u>emp#</u>, empname, mgr#, mgrname, address)

The attributes mgr# and mgrname are role names of emp# and empname respectively. We have the following strong functional dependencies in the relation EMP_MGR:

emp# $\stackrel{s}{\rightarrow}$ empname
mgr# $\stackrel{s}{\rightarrow}$ mgrname

The relation EMP_MGR is in replicated 3NF because it satisfies the Case 2 of Definition 3 and the relation scheme obtained by removing the attribute mgrname is in Codd 3NF. □

The replicated normal forms defined above makes provision for inclusion of redundancies in a relation scheme so that the operational efficiency of the database can be enhanced while ensuring that the integrity of the database is not compromised due to various updating anomalies. This is achieved by making sure that

1. only attributes which will not be updated or those which can be updated offline is duplicated. This is enforced through the definition of a strong FD and the second condition in Definition 1; and

2. insertion, deletion, and updating anomalies cannot occur by adhering to a strict updating discipline which will validate the updates against the primary instance of the attribute. How this can be enforced in practice is the subject of the next section.

Hence, the replicated normal forms provide a criteria against which results of denormalization can be verified with. For instance, a denormalized relation schema which is not in replicated 3NF is likely to suffer the updating anomalies presented in Example 1. On the other hand, a relation schema which is not in 3NF but nevertheless, is in replicated 3NF will not succumb to the said anomalies.

We now proceed to define yet another normal form arising from weak FDs.

Definition 4 [Relaxed 3NF] Let $R = \{R_1, R_2, ..., R_n\}$ be a database schema and A_j be the set of attributes in R_j, for j = 1, 2, ..., n. A relation schema $R_i \in R$ 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 7 Consider the scenario given in Example 4:

```
EMP(emp#, empname, phone#, designation, ...)
```

Since most of the employees have only one telephone number except for a handful of individuals in the top management who may have more than one telephones, therefore `emp#` $\xrightarrow{W}$ `phone#` holds in the relation EMP. Relation EMP is in relaxed 3NF because if we replace the WFD `emp#` $\xrightarrow{W}$ `phone#` by its regular counterpart, that is, `emp#` $\rightarrow$ `phone#`, EMP is in Codd 3NF.

The formulation of the relaxed normal forms follow the same rationale as that of the replicated normal form. Since there is only one Y-value associated with each X-value in *most* cases, we can implement this weak FD by treating Y as if the FD $X \rightarrow Y$ holds and accommodate exceptional cases in a separate relation. That is, given a database schema $R = \{R_1, R_2, ..., R_n\}$ and A_j is the set of attributes in R_j, for j = 1, 2, ..., n. For each weak FD $X \xrightarrow{W} Y$ which holds in R_i, we create an all-key relation OR_Y with attributes $X \cup Y$. We call OR_Y the *overflow* relation (schema) with respect to attributes Y. Note that if OR_Y is not in fourth normal form (4NF), then we decompose it into a set of 4NF relations. Retrieving the Y-value corresponding to a given X-value can then be accomplished efficiently. Additional measures however, will be needed to provide for the retrieval of more than one Y-values. This again will be the subject of the next section. It suffice to point out that this notion of relaxed normal forms provide opportunities for database designers to exploit the functional relationship which holds in general and devise a separate scheme for dealing with exceptions within the overall normalization framework.

Example 8 Consider again the scenario in Example 4. We can implement the WFD `emp#` $\xrightarrow{W}$ `phone#` by treating `phone#` as if `emp#` $\rightarrow$ `phone#` holds and accommodate exceptional cases in an overflow relation as follows:

```
EMP(emp#, phone#, empname, designation, ...)
EMP_PHONE_OVERFLOW(emp#, overflow_phone#)
```

The relation represented by `EMP_PHONE_OVERFLOW` is the overflow relation with respect to the attribute `phone#`.

Definition 5 [Relax-Replicated 3NF] Let $R = \{R_1, R_2, ..., R_n\}$ be a database schema and A_j be the set of attributes in R_j, j = 1, 2, ..., n. A relation schema R_i is said to be in relax-replicated 3NF if whenever every weak FD $X \xrightarrow{\mathtt{W}} Y$ which holds in R_i is replaced by its regular counterpart (i.e., $X \rightarrow Y$), R_i would have been in replicated 3NF.

The relax-replicated normal forms defined above allows for both strong and weak functional dependencies to occur in the same relation schema.

Example 9 Consider the following database schema:

$\mathtt{EMP\_MGR(}\underline{\mathtt{emp\#}}\mathtt{, empname, mgr\#, mgrname, phone\#, address)}$
$\mathtt{EMP\_PHONE\_OVERFLOW(}\underline{\mathtt{emp\#, overflow\_phone\#}}\mathtt{)}$

The attributes `mgr#` and `mgrname` are role names of `emp#` and `empname` respectively. We have the following functional dependencies in the relation EMP_MGR:

$\mathtt{emp\#} \xrightarrow{\mathtt{S}} \mathtt{empname}$
$\mathtt{mgr\#} \xrightarrow{\mathtt{S}} \mathtt{mgrname}$
$\mathtt{emp\#} \xrightarrow{\mathtt{W}} \mathtt{phone\#}$

The relation EMP_MGR is in relax-replicated 3NF because if we replace the WFD $\mathtt{emp\#} \xrightarrow{\mathtt{W}} \mathtt{phone\#}$ by its regular counterpart, that is, $\mathtt{emp\#} \rightarrow \mathtt{phone\#}$, EMP_MGR is in replicated 3NF.

We can similarly define relax-replicated improved 3NF [14], relax-replicated BCNF, and relax-replicated 4NF.

We conclude this section by discussing how we can design a database schema which is in relax-replicated 3NF. There are two main techniques for relational database schema design. In the first technique, often called relational synthesis, which views relational database schema design strictly in terms of functional dependencies specified on the database attributes. To obtain relax-replicated 3NF schemas using this first technique, we first obtain all the regular FDs, SFDs and WFDs. Then we treat all these different types of FDs as the classical FDs and use any algorithm [2,14] to synthesize relations. Note that all the relations obtained are in relaxed 3NF. Moreover, some of the relations may not be in 3NF due to the existence of some WFDs. Finally, we can modify the relation schemas using the SFDs obtained initially to produce relax-replicated 3NF relations which are more efficient from the processing point of view. That is, if $A \xrightarrow{s} B$ holds, A and B are sets of attributes, then whenever A appears in some relation R, we can add B into R.

In the second technique, also known as top down design, a conceptual schema can be designed using a high-level data model, such as the Entity-Relationship (ER) model [5]. The ER model uses the concepts and entity types and relationship sets. An entity type or relationship set has attributes which represents its structural properties. Attributes can be single-valued or multivalued. We can incorporate the notions of SFD and WFD when we use the ER model to design the conceptual schema of a database. Assuming that K is the identifier of an entity type E (or relationship set R), we can identify single-valued attributes A of E such that $K \xrightarrow{s} A$ or multivalued attributes B of E (or R) such that $K \xrightarrow{w} B$. If $K \xrightarrow{w} B$ holds, then we treat B as a single-valued attribute of E (or R). We can translate the ER diagram into a set of relations by using some existing translation algorithm. Note that all the relations obtained are in relaxed 3NF, and some relations may not be in 3NF due to the existence of some WFDs. Finally, we can modify relations translated from relationship sets when necessary using SFDs for more efficient processing. That is, if $K \xrightarrow{s} A$ holds, where K is the identifier of some entity type E and A is the attribute of E, then we can add A to relations involving K. In which case, the relations will be in relax-replicate 3NF.

4 Preserving Database Integrity with Replicated, Relaxed and Relax-Replicated Normal Forms

One of the most important functions of a database system is to ensure that data contained therein is *consistent* at any one time: i.e., it must satisfy all integrity constraints. Any database operation modifying the database must therefore preserve consistency, that is, it must see to it that the database is transformed from one consistent state to another. In the context of classical normalization theory, consistency is preserved by structuring the database to eliminate as much redundancy as possible, and the enforcement of a small subset of constraints (the most common ones being key constraints and referential integrity [9]). In the previous section, we suggested that the stringent constraints imposed by classical normal forms can be relaxed under a number of special but commonly occurring situations. These compromises were made to provide more efficient processing or disk space utilization. The integrity of the database however, should not be compromised in spite of these relaxations. The purpose of this section is to examine how we might be able to preserve the integrity of a database organized under the replicated or relaxed normal forms.

There are two generic strategies for accomplishing the above goal. One strategy is to incorporate a *constraint enforcement precompiler* that accepts a user's program and produces a new program guaranteeing that any update to database will be integrity preserving. A second strategy is to propagate updates to the database through the use of *triggers*.

4.1 Preserving Integrity of Replicated 3NF

Integrity preservation of a database in replicated normal forms can be cast as a more general problem: i.e., that of enforcing *inclusion dependencies* [3]. Recall that an inclusion dependency (IND) $R[X] \subseteq S[Y]$ is said to hold in a database d if for any two relations $r, s \in d$ which are extensions of R and S respectively, it is the case that $r[X] \subseteq s[Y]$. Suppose R_i is in replicated 3NF (but not 3NF), and $R_j, i \neq j$, is the primary instance of R_i with respect to attributes $W(= X \cup Y)$, where $W \subseteq A_i$, $W \subseteq A_j$, X is a key of R_j but it is not a key of R_i, and $X \overset{s}{\rightarrow} Y$ holds. Preserving the integrity of the above relations is therefore equivalent to making sure that the inclusion dependency $R_i[W] \subseteq R_j[W]$ holds at all times.

Given this simplification, one must now consider how updates to the database can be modified or propagated. Casanova et al. [4] noted that there are a number of options available for enforcing the constraint $R_i[W] \subseteq R_j[W]$ as described above. However, we can simplify these options as the W in our case is a superkey of R_i as follows:

1. *block deletion*: do not delete (or update) the value of W of a tuple in R_j if there is a tuple in R_i with the same W-value (or the same new W-value);

2. *block insertion*: do not insert (or update) the W-value of a tuple in R_i if there does not exist any tuple in R_j with the same W-value (or the same old W-value);

3. *propagate deletion*: propagate the deletion (or update) of a W-value of a tuple in R_j by deleting those tuples in R_i with the same W-value (or update those tuples in R_i with the same old W-value by the new W-value); and

4. *propagate insertion*: propagate the insertion (or the update) of a tuple μ_i in R_i by creating a tuple μ_j in R_j such that $\mu_i[W] = \mu_j[W]$, if there is no tuple μ in R_j such that $\mu_i[W] = \mu[W]$. The other attribute values of μ_j in this case may be simply set to null, or the user might be prompted to provide them;

As mentioned earlier, there are two strategies for implementing these options. The first is to modify the operations submitted to the database so that all updates and deletions can be made compliant to one or more of these options. The second is to encode these options in the form of *triggers*, so that certain operations may be automatically executed to implement a propagation. The next example expands on Example 5 by demonstrating how the options detailed earlier can be implemented.

Example 10 Consider the database schema presented in Example 5:

SUPPLIER(<u>sno</u>, sname, addr)
PART(<u>pno</u>, pname, color)
SUPPLY(<u>sno, pno</u>, sname, pname, qty)

We have `sno` $\stackrel{s}{\rightarrow}$ `sname` and `pno` $\stackrel{s}{\rightarrow}$ `pname`. The relation SUPPLY is in replicated 3NF. In order to preserve the integrity of this database, we will need to enforce the following inclusion dependencies:

SUPPLY[sno,sname] $\subseteq$ SUPPLIER[sno,sname]
SUPPLY[pno,pname] $\subseteq$ PART[pno,pname]

One example of how an insertion statement might be rewritten to implement the *block insertion* strategy is as follows. Given this insertion:

```
insert into SUPPLY values ("s1","p1","acme","screw, 1in",10);
```

This 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, 1in";
if S=NULL or P=NULL then
      reject transaction
else
      insert into SUPPLY values ("s1","p1","acme","screw, 1in",10);
```

Alternatively, we can implement the above insertion statement using the *propagate insertion* strategy by attaching a trigger to the relation SUPPLY which triggers insertions into both SUPPLIER and PART:

```
On insert(Sno,Pno,Sname,Pname,Qty) Call PG1(Sno,Sname), PG2(Pno,Pname)
```

where PG1 and PG2 are programs which insert into SUPPLIER and PART (respectively) tuples with those key values which are not currently in those relations. For instance, PG1 might be

```
if not exists
     ( select from SUPPLIER
      where SUPPLIER.sno=Sno
      and SUPPLER.sname=Sname; )
then insert into SUPPLIER values (Sno,Sname,NULL);
```

Note that the system will automatically enforce the key constraint. For example, for the insert statement in PG1, the system will ensure that no two tuples in the relation SUPPLIER have the same `sno` value. □

Similarly, by implementing these strategies, we can also preserve the integrity of a database containing relations in replicated 3NF for a deletion or modification request.

4.2 Preserving Integrity of Relaxed 3NF

In the case of relations organized in relaxed 3NF, we need to consider not only insertions, deletions and updates, but also queries. We will first identify the issues related to queries with the following example.

Example 11 Consider again the scenario in Example 8:

```
EMP(emp#, phone#, empname, designation, ...)
EMP_PHONE_OVERFLOW(emp#, overflow_phone#)
```

Suppose a user wishes to query the phone number of an employee by name. This query can be supported by two different strategies as detailed below.

In the first strategy, both schemas are made visible to the user. The implication of this is that the user can choose to query relation EMP directly if he is not concerned about getting *all* the phone numbers of an employee. The advantage of this approach is that queries of this kind can be most efficiently supported. Moreover, the phone number thus obtained will, in most instances, be the only one anyway. If a user wishes, for some reason, to know all the phone numbers of an employee named "Smith", he can then issue a second query which will be more expensive to process:

```
select phone# from EMP
where empname = "Smith"
UNION
select overflow_phone# from EMP_PHONE_OVERFLOW, EMP
where EMP.empname = "Smith"
and EMP.emp# = EMP_PHONE_OVERFLOW.emp#;
```

In the above approach, the user is not insulated from the idiosyncrasies of the database structure. Some authors might argue that exposing the user to the knowledge of the overflow relation might be undesirable. In fact, we could also present only the relation EMP: in this case, the user could query the relation directly, say, with

```
select phone# from EMP where empname = "Smith";
```

but we would need to rewrite this query to range over both relations (i.e., EMP and EMP_PHONE_OVERFLOW). This simplicity is however is achieved at the expense of efficiency in query processing, since the query after transformation will require a join of the two relations. However, the relation EMP_PHONE_OVERFLOW is very small. Therefore a join of the two relations will not be too expensive. Note that if there is another weak FD in the relation EMP, say E# $\xrightarrow{w}$ designation, then we will have another overflow relation called EMP_DESIGNATION_OVERFLOW. □

Similar transformations can be applied to insertion and deletion operations as demonstrated in the following example.

Example 12 Consider once again the earlier example involving employees and phone numbers. The insertion operation:

```
insert into EMP values (Eno,Ephone,...);
```

can be transformed to

```
E :=  select * from EMP where emp# = Eno;
if E = NULL then
      insert into EMP values (Eno,Ephone,...);
else
      if E.phone# = NULL then /* employee has no phone yet */
            update EMP set phone# = Ephone;
      else /* employee has more than one phone */
            insert into EMP_PHONE_OVERFLOW values (Eno,Ephone);
```

Note that the above transformation caters for either the insertion of a tuple into EMP or the insertion of a new telephone number for a particular employee.

Deleting a tuple from `EMP` (for example, delete from `EMP` where emp# = Eno) requires that any other tuples which has the same Eno in `EMP_PHONE_OVERFLOW` be deleted as well. On the other hand, if only the telephone number is made obsolete, that is, delete the telephone number of an employee, additional work might be required to update `EMP` with a phone number in `EMP_PHONE_OVERFLOW`. For example, the deletion operation:

```
update EMP set phone# = NULL where emp#=Eno and phone#=Ephone;
```

will need to be transformed to the following:

```
Select phone# from EMP where emp# = Eno;
if phone# = Ephone then
      S :=  select overflow_phone#
            from EMP_PHONE_OVERFLOW where emp# = Eno;
      If S = NULL then
            update EMP set phone# = NULL where emp# = Eno;
      else /* Move a phone# in the overflow relation to EMP */
            p := any arbitrary element in S;
```

```
            delete from EMP_PHONE_OVERFLOW where emp# = Eno and phone# = p;
            update EMP set phone# = p where emp# = Eno;
    else
        delete from EMP_PHONE_OVERFLOW
            where emp# = Eno and phone# = Ephone;
```

□

To preserve the integrity of a database with relax-replicated 3NF relations, we basically combine the strategies used to preserve the integrity of relations in replicated and relaxed 3NFs.

5 Conclusion

In this paper, we have shown how the constraints imposed by Codd 3NF can be relaxed to facilitate the design of database schemas which are more efficient from the processing and storage point of view. This is accomplished by identifying those circumstances under-which certain relaxations can be made without sacrificing the integrity or performance of the database. This led to the proposal of three new normal forms, which we refer to as relaxed 3NF, replicated 3NF and relax-replicated 3NF. These normal forms are defined with respect to strong and weak FDs which are themselves extensions to classical FDs. Two approaches for designing relax-replicated 3NF relations were presented. We also proposed some methods for preserving the integrity of a database in relax-replicated 3NF. We can similarly define relax-replicated improved 3NF, relax-replicated BCNF and relax-replicated 4NF.

The design of a good database schema is both an art and a science. In the past two decades, we have seen much scientific progress in logical database design, whereas the same is not true of the other phases, such as the physical database design phase. From this perspective, the challenge for database researchers is to identify a unifying theoretical framework which will integrate the different phases of database design in a more coherent manner. There have been some effort directed at integrating the theories of conceptual design and logical design (see for example, [11,13]) but to date, we are not aware of any effort aiming to achieve the same for logical and physical design. The theory we have presented in this paper represents a preliminary step towards allowing physical database design to take advantage of semantics captured in the earlier phases. Furthermore, we can also apply our theory to the process of schema evolution where certain functional dependencies may be violated over time because of some policy changes by the database owner.

For future work, it would be good to have a cost model to determine when it is most beneficial to generate relations in replicated, relaxed, or relax-replicated normal forms based on the classical functional dependencies, strong and weak functional dependencies, application program specification and execution frequency, available storage and response time required.

References

[1] C. Beeri, P. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *Proc. of the 4th VLDB*, pages 113–124, 1978.

[2] P. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1(4):277–298, 1976.

[3] M. Casanova, R. Fagin, and C. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies (*extended abstract*). In *Proc. ACM PODS*, pages 171–176, 1982.

[4] M. A. Casanova, L. Tucherman, and A. L. Furtado. Enforcing inclusion dependencies and referential integrity. In *Proceedings of the 14th VLDB Conference*, pages 38–49, 1988.

[5] P. Chen. The entity – relationship model: towards a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[6] E. Codd. A relational model for large shard data banks. *Commun. ACM*, 13(6):377–387, 1970.

[7] E. Codd. Further normalization of the data base relational model. In *Data Base Systems*, pages 33–64, 1974.

[8] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, 1977.

[9] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.

[10] C. Fleming and B. von Halle. *Handbook of relational database design.* Addison-Wesley Publishing Co., 1989.

[11] C. Goh and T. Ling. Extending the entity relationship formalism for conceptual data modeling to capture more semantics. In *Proc. of the 1st International Conference on Information and Knowledge Management*, Baltimore, Maryland, Nov 1992.

[12] W. Inmon. *Optimizing performances in DB2 software.* Prentice-Hall, Engelwood Cliffs, 1988.

[13] T. Ling and C. Goh. Towards a synergistic integration of the ER and relational formalisms for logical relational database design. Technical report, Department of Information Systems and Computer Science, National University of Singapore, 1992. Also submitted for publication.

[14] T. Ling, F. Tompa, and T. Kameda. An improved third normal form for relational databases. *ACM Trans. Database Syst.*, 6(2):329–346, 1981.

[15] T. Teorey and J. Fry. *Design of database structures.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[16] J. Ullman. *Principles of database and knowledge-base systems, Vol I.* Computer Science Press, 1991.