Polynomial Time Designs toward Both BCNF and Efficient Data Manipulation

KE WANG

Chongqing University, PRC

Current address
Department of Computing Science
University of Alberta
Edmonton, Alberta, CANADA T6G 2H1

Abstract: We define the independence-reducibility based on a modification of key dependencies, which has better computational properties and is more practically useful than the original one based on key dependencies Using this modification as a tool, we design BCNF databases that are highly desirable with respect to updates and/or query answering In particular, given a set U of attributes and a set F of functional dependencies over U, we characterize when F can be embedded in a database scheme over U that is independent and is BCNF with respect to F, a polynomial time algorithm that tests this characterization and produces such a database scheme whenever possible is presented The produced database scheme contains the fewest possible number of relation schemes Then we show that designs of embedding constant-time-maintainable BCNF schemes and of embedding independence-reducible schemes share exactly the same method with the above design Finally, a simple modification of this method yields a polynomial time algorithm for designing embedding separable BCNF schemes

1. INTRODUCTION

The *Boyce-Codd normal formal (BCNF)* [Co] is one of the most important database normal forms aimed at reducing data redundancy and update anomalies Unfortunately, given a set F of functional dependencies [A], the problem "does there exist a non-BCNF database scheme that is embedding F" is NP-complete [BB] (The NP-completeness was proved in [BB] for cover embedding BCNF schemes But since the scheme constructed there happened to embed the given functional dependencies, our statement is still correct) Thus unless P=NP, no polynomial time algorithm for designing embedding BCNF database schemes is likely to be found

Recent work on database design addressed some properties that would allow data updates and/or query answering to be performed efficiently In particular, within the context of the *weak instance model* [H,M,MUV,V,Y], there has been a good deal of work being done on proposing and identifying such nice "data-manipulation" properties Among them there are independence [GY,S1,S2], independence-reducibility [CH], constant-time-maintainability [HC,GW,W], and separability [CM] However, very little has been known about how to actually design databases with these properties in general The design theory should eventually provide algorithms and guides for deisgning the goals it has proposed

In this paper, the above "data manipulation" properties are considered to be equally important as normalization of databases That is, we believe that useful systems should be free of redundancy/anomalies as well as allow efficient data updates and/or query answering We will focus on designs of databases toward such combined goals It turns out that BCNF interacts with these properties in such a nice way that the above intractability disappears

Under the weak instance model, *independence* takes the following form A database state within which each relation satisfies the dependencies local to it has a weak instance, i e , is *consistent* [H,V,Y] Hence, only local dependencies need be enforced in the process of updates if independence is provided Independence meets the aesthetic principle of "separation" or "one thing in one place" [BBG] and therefore is highly desirable in a distributed environment where data transmissions between sites are supposed to be minimized Independent schemes with dependencies given by keys of relations were stuied by [S1,S2] and those with functional dependencies plus the join dependency [ABU] of the database scheme were studied by [GY,S3] Specially, it has been shown that independent schemes are highly desirable with respect to query answering as well [AC,IIK,S2,S3]

Chan and Hernandez [CH] defined a generalization of Sagiv-independent schemes [S2], called *independence-reducible* schemes This is exactly the class of database schemes obtained from decomposing, based on a set of so called *key dependencies*, Sagiv-independent schemes in a dependency preserving manner Independence-reducible schemes inhert most desirable properties of independent schemes and we shall see that they are always in BCNF In this paper, independence-reducibility will be treated as a design goal and more importantly used as a design tool In particular, we will show that independence-reducibility of certain database scheme characterizes when our combined design goals are possible to fulfil, for a given set of functional dependencies, and guides the design procedure whenever possible However, to fit into our framework, some modification to the original notion of key dependencies seems to be necessary We will show how to do this and will argue that the modified notion has beeter computational properties and is more practially useful than the original one

The notion of constant-time-maintainability was proposed as a systematic generalization of independence of [GY] Very informally, a database scheme is *constant-time-maintainable* [GW] if some algorithm can determine whether consistency of a database state is preserved by an insertion of a new tuple in time independent of the state size (This will depend on the computational model used) Independent schemes with only functional dependencies are constant-time-maintainable because only local functional dependencies need be enforced Constant-time-maintainability is a systematic generalization of independence in the sense that the constant time solution captured by it is not necessarily achieved by assuming the *uniqueness* [S2] and simple types of dependencies, but rather is inherent in the scheme itself This property is highly desirable in a large and dynamic environment where scanning entire database state is not acceptable for enforcing constraints in the process of updates Recognition of constant-time-maintainable schemes and some nice behaviours with respect to query answering have recently been established in [HC,HW,W,WG], of them [HC] gave a polynomial time test of constant-time-maintainable schemes, provided that schemes are in BCNF

The notion of separability by Chan and Mendelzon [CM] concerned with both consistency and completeness of information of locally satisfying states A database state is *complete* if any tuple that can be derived from the existing tuples and the constraints are already given explicitly in the state A database scheme is *separable* if local satisfaction implies both consistency and completeness of the state As mentioned in [CM], this property captures the design goal of independently updatable decomposition and it is equivalent to a specification of the abstract independent mapping defined by Bancilhon and Spyratos [BS] We shall also consider design of separability

The central problems we shall address in this paper are the following Given a set U of attributes and a set F of functional dependencies over U, we characterize under what conditions there exists a database scheme over U that is embedding, independent, and in BCNF with respect to F Then we address how to test such conditions and, if the test succeeds, how to produce a database scheme satisfying these properties Very interestingly, we will show that if F cannot be embedded in an independent BCNF scheme, then F cannot be embedded in any constant-time-maintainable BCNF scheme nor in any independence-reducible scheme Therefore, our method for designing embedding independent BCNF schemes is exactly those for designing embedding constant-time-maintainable BCNF schemes and embedding independence-reducible schemes This then would suggest that, within the context of BCNF scheme design, not much needs be studied for constant-time-maintainable schemes and independence-reducible schemes other than independent schemes, provided that no constraints (other than functional dependencies) are imposed The time of our tests is bounded by $O(|F|^2||F||+|U|)$, where $|F|$ is the number of functional dependencies in F, $||F||$ is the size of the description of F, and $|U|$ is the number of attributes in U Finally, a simple modification of the above method yields a polynomial time algorithm for designing embedding separable BCNF schemes

For each combined design goal considered, it is shown that the produced database scheme contains the fewest possible number of relation schemes Thus, data redundancy is prevented both within (by BCNF) and between (by minimizing the number of relations) relations We shall also discuss how to modify the produced scheme to make it lossless without affecting the goals that have been designed for it

Our choice of embedding, rather than cover embedding, functional dependencies is justified as follows Let $X \rightarrow Y$ be any given functional dependency $X \rightarrow Y$ not only represents an integrity constraint on the database, but also represents a relationship that the database is intended to store In other words, we consider the given functional dependencies to express as well the information about what attributes at least should be put into one relation scheme Intuitively, such a design will depend on the choice of dependency covers in general It turns out, however, that the design result is not affected by applications of union and decomposition rules of functional dependencies On the other hand, the treatment of cover embedding is a rather syntactic one based on Armstrong's axioms [A], and in the end, not every cover embedding database scheme matches so well our intuition about what information should be tabulated in the database

2. DEFINITIONS AND NOTATION

We now briefly describe the notation and definitions required for the rest of this paepr

2.1 Relations, Schemes, and States

A *(database) scheme*, denoted $(\mathbf{R},\Sigma)$, consists of a collection of relation schemes $\mathbf{R}=\{R_1, \ ,R_m\}$ and a finite set of dependencies Σ over $\cup\mathbf{R}$ defined below, where $\cup\mathbf{R}$ is the abbreviation of the unions $R_1\cup \ \cup R_m$ Very often, a database scheme refers to $\mathbf{R}$ alone if Σ is not of interest A *(database) state* over $\mathbf{R}$, usually denoted ρ, is an assignment of relations to relation schemes of $\mathbf{R}$, with $\rho(R_i)$ denoting the relation assigned to R_i by ρ Let t be a tuple over some R_i in $\mathbf{R}$ $\rho\cup\{t\}$ denotes the state ρ' $\rho'(R_j)=\rho(R_j)$, for $R_j\in\mathbf{R}-\{R_i\}$, and

$\rho'(R_i)=\rho(R_i)\cup\{t\}$

2 2 Dependencies and Normal Forms

An *functional dependency (fd)* [A] over a relation scheme R_i is a statement of the form $X\rightarrow Y$, where X and Y are sets of attributes such that $R_i\supseteq XY$, and they are called the *left-hand-side* and *right-hand-side* of the fd, respectively $F\models F'$ denotes F *(logically) implies* F′ If $F\models G$ and $G\models F$, denoted $F\equiv G$, F is said to be *equivalent* to (or to be a *cover* of) G F^+ is the set of all fd's implied by F Let X be a set of attributes X_F^+ is the set of attributes A such that $F\models X\rightarrow A$ Fd's can be unioned and decomposed using the following rules [A,Ma,U]

- **Union rule.** $\{X\rightarrow Y, X\rightarrow Z\}\models X\rightarrow YZ$
- **Decomposition rule**· Let $X\rightarrow Y$ be an fd and $Z\subseteq Y$ Then $X\rightarrow Y\models X\rightarrow Z$

An fd $X\rightarrow Y$ is *embedded* in a relation scheme R_i if $R_i\supseteq XY$ Let F be a set of fd's F/R_i denotes the fd's of F that are embedded in R_i and F/**R** denotes the fd's of F that are embedded in elements of **R** F is *embedded* in **R**, or **R** is *embedding* F, if every fd in F is embedded in some element of **R** A database scheme **R** is *cover embedding* F [BH] if there is a cover of F that is embedded in **R**

The *join dependency (jd)* [ABU] defined by database scheme $\mathbf{R}=\{R_1, \ ,R_m\}$, denoted $*\mathbf{R}$, is satisfied by a relation I over $\cup\mathbf{R}$ if $\Pi_{R_1}(I)* \ *\Pi_{R_m}(I)=I$, that is, the original relation can be reconstructed from joins of its projections onto **R** A database scheme **R** is *lossless* with respect to (wrt) a set of Σ of dependencies [ABU] if $\Sigma\models *\mathbf{R}$ Evidently, when jd $*\mathbf{R}$ is included in Σ, **R** is trivially lossless wrt Σ

Let F be a set of fd's, R_i a relation scheme of **R**, and $X\subseteq R_i$ X is called a *key* of R_i wrt F if $F\models X\rightarrow R_i$ and no proper subset of X has this property A *superkey* of R_i wrt F is any set of attributes in R_i that contains a key of R_i A relation scheme R_i is in *Boyce-Codd normal form (BCNF)* wrt F [Co] if for every nontrivial fd $X\rightarrow Y\in F^+/R_i$, X is a superkey of R_i wrt F We say that a database scheme **R** is in BCNF wrt F if R_i is in BCNF wrt F for every R_i in **R** More generally, **R** is in BCNF wrt a set of dependencies Σ if **R** is in BCNF wrt the fd's implied by Σ

2.3 Weak Instances, Chase, and Representative Instances.

Let (**R**,Σ) be a database scheme, ρ a state over **R**, and I a relation over $\cup\mathbf{R}$ I is called a *weak instance* of ρ wrt Σ if I satisfies Σ and $\Pi_{R_i}(I)\supseteq\rho(R_i)$ for each R_i in **R** A state ρ is a *consistent state* of (**R**,Σ), or ρ is *consistent* wrt Σ, if there exists a weak instance of ρ wrt Σ [H,M,V,Y] CONS(**R**,Σ) denotes the collection of all consistent states of (**R**,Σ)

We can test whether a database state ρ is consistent wrt a set Σ of dependencies by applying the *chase process* [MMS] to the universal relation aug(ρ), where aug(ρ) is obtained from ρ by augmenting out to $\cup\mathbf{R}$ every tuple of ρ with unique variables The chase process modifies this relation by applying rules associated with dependencies in Σ to aug(ρ) as far as possible, until either a *contradiction* is found, i e, two constants are equated, in which case ρ is inconsistent, or no rule can further modify the relation, in which case ρ is consistent --- the final relation is a weak instance of ρ If ρ is consistent wrt Σ, we shall denote by $\mathrm{CHASE}_\Sigma(\mathrm{aug}(\rho))$ the final chased relation and call it the *representative instance* of ρ wrt Σ [M,S2] Given a consistent state ρ and any set X of attributes of $\cup\mathbf{R}$, the *X-total projection* of the representative instance of ρ wrt F, denoted by $\Pi_X^{\downarrow}(\mathrm{CHASE}_\Sigma(\mathrm{aug}(\rho)))$, is the set of tuples (projected onto X) of the representative instance of ρ that contain no variables over X

2.4 Independence, Separability, and Constant-time-maintainability

Sagiv [S2] defined and studied the notion of independent schemes when fd's are given by keys of relations Let $\mathbf{R}=\{R_1, \ ,R_m\}$ and let $F=F_1\cup \ \cup F_m$ be a set of fd's such that F_i, $1\le i\le m$, contains only (but not necessarily all) fd's of the form $X\rightarrow R_i-X$, where X is a key of R_i wrt F **R** is said to be *independent* [S1,S2] wrt F if every state ρ such that $\rho(R_i)$ satisfies F_i for $1\le i\le m$ is consistent wrt F It was shown in [S2] that **R** is independent wrt F if and only if, for all $1\le i\le m$,

R_i satisfies the *uniqueness condition* [S2] for no $1\le j\le m$ with $j\ne i$ does $(R_i)^+_{F-F_j}$ contain a left-hand-side K of some fd in F_j and an attribute A in R_j-K

Graham amd Yannakakis [GY] studied independent schemes in more general cases Let (**R**,F) be a database scheme , where F is a set of fd's over $\cup\mathbf{R}$ A database scheme **R** is *independent* wrt F if every state ρ such that $\rho(R_i)$ satisfies F^+/R_i for every $R_i\in\mathbf{R}$ is consistent wrt F By a result in [GY], when dependencies are given by keys of relations, this definition coincides with the above Sagiv's definition Let G be the set of fd's implies by $F\cup\{*\mathbf{R}\}$ **R** is *independent* wrt $F\cup\{*\mathbf{R}\}$ if every state ρ such that $\rho(R_i)$ satisfies G^+/R_i for every $R_i\in\mathbf{R}$ is consistent wrt $F\cup\{*\mathbf{R}\}$ More results on independent schemes, including some desirable properties with respect to query answering, can be found in [AC,IIK,S3]

Let (**R**,Σ) be a database scheme A consistent state ρ of (**R**,Σ) is *complete* if $t\in\Pi_{R_i}^{\downarrow}(\mathrm{CHASE}_\Sigma(\mathrm{aug}(\rho)))$ implies $t\in\rho(R_i)$, $R_i\in\mathbf{R}$ As explained in [GMV], the idea is that a complete state contains explicitly all the tuples whose existence can be derived from the state and the dependencies **R** is *everywhere-complete* wrt Σ if every consistent state of (**R**,Σ) is complete [GMV] **R** is *separable* wrt Σ if **R** is both independent and everywhere-compelete wrt Σ [CM]

The *maintenance problem* of a database scheme (**R**,Σ) is the following decision problem [GW,GY] Let ρ be a consistent state of (**R**,Σ) and we want to insert a tuple (over some R_i in **R**) into ρ, called an *instance* <ρ,t> below, is $\rho\cup\{t\}$ a consistent state of (**R**,Σ)? Assume that the database state ρ is stored in a device that responds to requests of the form $<R_j,\Psi>$ by returning, if it exists, an arbitrary tuple satisfying Ψ from the relation $\rho(R_j)$, where $R_j\in\mathbf{R}$, and Ψ is a Boolean combination of equality of form B=b, for some attribute $B\in R_j$ and constant 'b' in the domain of B Furthermore, every request $<R_j,\Psi>$ obeys the *no guess assumption* [GW] in the sense that the constants used in equalities of Ψ appear either in the inserted tuple or in some previously returned tuples Suppose that some algorithm A solves the maintenance problem of **(R**,Σ) by making requests to the current state as above For any instance $<\rho,t>$, we define $\#A(\rho,t)$ to be the number of requests made on $<\rho,t>$ by A in determining consistency of $\rho\cup\{t\}$ A database scheme **R** is said to be *constant-time-maintainable (ctm)* wrt Σ if there exists an integer $k \geq 0$ such that $k \geq \#A(\rho,t)$ for all instance $<\rho,t>$ of the maintenance problem of **(R**,Σ) Ctm schemes are important in the case where states are large and modifications are frequent More results on ctm database schemes were reported in [HC,W,WG]

It should be noted that the properties of BCNF, independence, separability, and constant-time-maintainability are all insensitive to the choice of dependency covers

3 A MODIFIED NOTION OF KEY DEPENDENCIES

As a design goal in capturing efficient query answering and constraint enforcement, Chan and Hernandez [CH] defined a notion of independence-reducibility wrt a set of so called "key dependencies" We shall use this notion as a tool in our design framework Essentially, our design algorithm is a modification of the 3NF synthesizing method [B] First, we shall construct, for each given fd, one relation scheme consisting of the attributes in that fd Then, while the embedding 3NF is always possible to fulfil for the given fd's, we show that embedding independent BCNF design goal is possible to fulfil exactly when the constructed database scheme is independence-reducible Third, when the design is possible, instead of merging relation schemes corresponding to the fd's with the equivalent left-hand-sides as in [B], we merge the relation schemes that are equivalent wrt their embedded key dependencies, in other words, we find "the key-equivalent partition" [CH] of the constructed database scheme and merge the relation schemes in each block of the partition By some results in [CH], such a merged database scheme is independent and in BCNF However, certain modification to the key dependencies seems to be necessary for our purpose In this section we present such a modification and show a few nice properties of it We will returan to a presentation of design algorithm in the subsequent sections

[CH] defined a notion of independence-reducibility as follows A *partition* of a set S is a collection of nonempty subsets of S such that elements in the collection are pairwise disjoint and the union of the collection is S Each subset in a partition is called a *block* Given a set of fd's F and a relation scheme R, if K is a key of R wrt F and $A \in R-K$, $K \to A$ is said to a *key dependency* in R wrt F A set of fd's G is *a set of key dependencies in (a relation scheme)* R wrt F if G is equivalent to the set $\{K \to A \mid K \to A$ is a key dependency in R wrt F $\}$, i e , the set of *all* key dependencies in R F is a *set of key dependencies in (a database scheme)* $\mathbf{R}=\{R_1, \ ,R_m\}$ if $F=F_1\cup \ \cup F_m$, where each F_i is a set of key dependencies in R_i wrt F Let $F=F_1\cup \ \cup F_m$ be a set of key dependencies in $\mathbf{R}=\{R_1, \ ,R_m\}$, and let $S \subseteq \mathbf{R}$ F(S) denotes $\cup\{F_j \mid R_j \in S\}$ S is *key-equivalent* wrt F(S) if for any $R_i,R_j \in S$, $(R_i)^+_{F(S)}=(R_j)^+_{F(S)}$ **R** is said to be *independence-reducible* wrt F if there is a partition $\mathbf{T}=\{T_1, \ ,T_k\}$ of **R** such that

(a) database scheme $\mathbf{D}=\{\cup T_p \mid T_p \in \mathbf{T}\}$ is independent wrt F, and

(b) for any $T_p \in \mathbf{T}$, T_p is key-equivalent wrt $F(T_p)$

It has been shown in [CH] that independence-reducible database schemes inherit most of the properties of independent schemes and are highly desirable with respect to query answering and constraint enforcement The test algorithm of independence-reducibility in [CH] (i e , function KEP and Algorithm 4) has assumed that the key dependencies are explicitly given as $F_1, \quad ,F_m$, where F_i is a set of key dependencies in R_i

Some Negative Results of Key Dependencies. Let F be a set of fd's over U, let $R \subseteq U$ be a relation scheme, and let G be some set of fd's embedded in R By slightly modifying the proof of the NP-completeness of the *additional key problem* in [BB] (i e , Theorem 5), we can show that the problem "Is G a set of key dependencies in R wrt F?" is CoNP-complete This result strongly suggests that no polynomial time algorithm for generating a set of key dependencies in R (for a given set F of fd's over U) is likely to exist Therefore it is unreasonable to assume that any set equivalent to a set of key dependencies in **R** is always given explicilty as sets F_i's of key dependencies in R_i's Moreover, as illustrated by Example 3 1 below many redundant fd's are included in sets F_i's of key dependencies, we doubt that any algorithms taking such sets as input in a non-trivial manner can be considered "truly" efficient To get over these problems as well as to fit into our design framework, we now modify the above notion of key dependencies as follows

A Modified Notion of Key Dependencies. Let $\mathbf{R}=\{R_1, .,R_m\}$ and $F=F_1\cup \ \cup F_m$, where F_i is a set of fd's embedded in R_i, $1 \leq i \leq m$ F_i is a *set of key-dependencies* (i e , with "-" in between for distinction) in R_i if for every fd $X \to Y$ in F_i, $F_i \models X \to R_i$ F is called a *set of key-dependencies* in **R** if F is equivalent to the unions $F_1\cup \ \cup F_m$, where F_i is a set of key-dependencies in R_i, $1 \leq i \leq m$ Assume that $F_1\cup \ \cup F_m$ is a set of key-dependecies in $\mathbf{R}=\{R_1, \ ,R_m\}$, and let $S \subseteq \mathbf{R}$ It follows that the $\cup\{F_j \mid R_j \in S\}$ is a set of key-dependencies in sub-scheme S

Example 3.1 Consider the database scheme **(R**,F), where

$$\mathbf{R}=\{R_1(AB),R_2(BC),R_3(AC),R_4(AD),R_5(DEF)R_6(DEG)\},$$

$$F=\{A\to B,B\to C,C\to A,A\to D,D\to EF,D\to EG\}$$

We claim that F is a set of key-dependencies in **R** In particular, $F=F_1\cup \ \cup F_6$, where $F_1=\{A\to B\}$, $F_2=\{B\to C\}$, $F_3=\{C\to A\}$, $F_4=\{A\to D\}$, $F_5=\{D\to EF\}$, $F_6=\{D\to EG\}$, and each F_i is a set of key-dependencies in R_i Note that F_j, for $1 \leq j \leq 3$, is not equivalent to the set $\{X \to A \mid X \to A$ is a key dependency in R_j wrt F}, and thus F_j is not a set of key dependencies in R_j. To make F a set of key dependencies in **R**, F needs be given as $F'=F_1' \cup \ldots \cup F_6'$, where $F_1'=\{A \rightarrow B, B \rightarrow A\}$, $F_2'=\{B \rightarrow C, C \rightarrow B\}$, $F_3'=\{C \rightarrow A, A \rightarrow C\}$, $F_4'=\{A \rightarrow D\}$, $F_5'=\{D \rightarrow EF\}$, $F_6'=\{D \rightarrow EG\}$, each F_i' being a set of key dependencies in R_i. □

Better Computational Properties of Key-dependencies. As pointed out above, there is a lack of polynomial time algorithm for testing and generating key dependencies. Now we show that key-dependencies are free of these problems. Given a set of fd's F, it is not hard to see the following. If F is equivalent to a set of key-dependencies in **R**, then F is cover embedded in **R**, and if F is embedded in **R**, then F is a set of key-dependencies in **R** if and only if, for each fd $X \rightarrow Y$ in F, there is at least one relation scheme $R_i \in \mathbf{R}$ such that $R_i \supseteq XY$ and X is a superkey of R_i wrt F. Thus we can test whether F is equivalent to a set of key-dependencies in $\mathbf{R}=\{R_1, \ldots, R_m\}$, and find such an equivalent set if it is, in polynomial time as follows.

First, by a polynomial time algorithm in [BH] or [GY], we test whether F is cover embedded. If not, then F is not equivalent to a set of key-dependencies in **R**, otherwise, that algorithm also returns an embedded cover F′ of F. Then we verify, for each fd $X \rightarrow Y \in F'$, that there exists at least one relation scheme $R_i \in \mathbf{R}$ such that $R_i \supseteq XY$ and X is a superkey of R_i wrt F. F is equivalent to a set of key-dependencies in **R** if and only if no violation of these verification is found. In the case of no violation, the sets of key-dependencies F_i's in R_i's, such that $F \equiv F_1 \cup \ldots \cup F_m$, are constructed as follows. For each fd $X \rightarrow Y \in F'$, choose arbitrarily exact one relation scheme $R_i \in \mathbf{R}$, such that $R_i \supseteq XY$ and X is a superkey of R_i wrt F, and include fd $X \rightarrow R_i - X$ in F_i.

Clearly, the above test and transformation take polynomial time in the number of relation schemes and in the size of description of F. More specifically, $|F_i| \leq |F'/R_i|$, for all $R_i \in \mathbf{R}$, where F′ is the embedded cover of F used in the above transformation.

Test of Independence-reducibility Based on Key-dependencies. First, we define the independence-reducibility wrt a set of key-dependencies. Let $F=F_1 \cup \ldots \cup F_m$ be a set of key-dependencies in **R**, and let $\mathbf{S} \subseteq \mathbf{R}$. As before, we define $F(\mathbf{S})=\cup\{F_j \mid R_j \in \mathbf{S}\}$, and we say that **S** is *key-equivalent* wrt F(**S**) if for any R_i, R_j in **S**, $(R_i)^+_{F(S)}=(R_j)^+_{F(S)}$. **R** is *independence-reducible* wrt F if there is a partition $\mathbf{T}=\{T_1, \ldots, T_k\}$ of **R** such that

(a) database scheme $\mathbf{D}=\{\cup T_1, \ldots, \cup T_k\}$ is independent wrt F, and

(b) for any $T_p \in \mathbf{T}$, T_p is key-equivalent wrt $F(T_p)$.

We shall say that the above partition **T** is an *independence-reduced partition* of **R** wrt F and the above scheme **D** is an *independence-reduced scheme* of **R** wrt F. Also, if a partition **T** of **R** satisfies condition (b), independently of condition (a), we say that **T** is a *key-equivalent partition* of **R** (wrt F). (Note that this notion is different from "the key-equivalent partition" defined in [CH].) If $\mathbf{T}=\{T_1, \ldots, T_k\}$ is a key-equivalent partition of **R**, then the database scheme $\{\cup T_1, \ldots, \cup T_k\}$ is ***induced*** by **T**. Clearly, F is also a set of key-dependencies in any induced scheme $\{\cup T_1, \ldots, \cup T_k\}$, where the set of key-dependencies in $\cup T_i$ is $F(T_i)$, for $1 \leq i \leq k$. Note that any database scheme independent wrt a set of key-dependencies is trivially independence-reducible.

In the following, we show that replacement of key dependencies by key-dependencies is only a matter of choices of dependency covers in the sense that it gives an equivalent definition and the same test algorithm of independence-reducibility.

Lemma 3.1: (a) Every set of key-dependencies in **R** is equivalent to a set of key dependencies in **R**. (b) Every set of key dependencies in **R** is a set of key-dependencies in **R**. (c) Let F be a set of key-dependencies in **R** and let F′ be an equivalent set of key dependencies in **R**. Then **R** is independence-reducible wrt F if and only if **R** is independence-reducible wrt F′.

An important observation is that the key proofs in [CH] do not need the stronger assumption of having key dependencies and (therefore) the test method of independence-reducible schemes in [CH] still works when a set of key-dependencies rather than a set of key dependencies is taken as input. Then it follows, from Lemma 3.1 and the polynomial transformation of key-dependencies, that we can test independence-reducibility truly efficiently. For our convenience, in the following we borrow from [CH] the test algorithm, with the input replaced by a set of key-dependencies.

Let F be a set of key-dependencies in **R** and let R_i be a relation scheme in **R**. Define $[R_i]$ to be the largest subset of **R** containing R_i such that $[R_i]$ is key-equivalent wrt $F([R_i])$. The collection of $\{[R_i] \mid R_i \in \mathbf{R}\}$ is called the *maximum key-equivalent partition* of **R** (wrt F) (which corresponds to the term "key-equivalent partition" in [CH]). As an important component of testing independence-reducibility, the function KEP below generates the maximum key-equivalent partitions.

Function KEP(**R**,F);

Input: $\mathbf{R}=\{R_1, \ldots, R_m\}$ and $F=F_1 \cup \ldots \cup F_m$, where F_i is a set of key-dependencies in R_i, $1 \leq i \leq m$.

Output: The maximum key-equivalent partition of **R** wrt F.

Notation: $R_i^*=\{R_j \mid (R_j)^+_F=(R_i)^+_F\}$.

Method:

begin

(1) Let PARTITION$=\{R_i^* \mid R_i \in \mathbf{R}\}$;

(2) **if** PARTITION$=\{\mathbf{R}\}$ **then return** $(\{\mathbf{R}\})$

else return $(\mathrm{KEP}(P_1,F(P_1)) \cup \ldots \cup \mathrm{KEP}(P_n,F(P_n)))$, where PARTITION$=\{P_1, \ldots, P_n\}$

end

With **R** and F as input, function KEP(**R**,F) partitions **R**, as in step (1), on the basis of the equivalence of relation schemes under F and invocates a recursive call for each block Since **R** can be "split" at most |**R**|-1 times, the total number of invocations of KEP is bounded by |**R**|-1, where |**R**| is the number of relation schemes in **R** Within each invocation of KEP(**R**,F), PARTITION can be found by computing $(R_i)^+_F$, for each R_i in **R**, and grouping those having the same closure As $(R_i)^+_F$ can be computed in time $O(\|F\|)$ by an algorithm in [BB], where $\|F\|$ is the size of description of F, finding PARTITION in step (1) takes time $O(|\mathbf{R}|\,\|F\|)$ Thus function KEP(**R**,F) is bounded by $O(|\mathbf{R}|^2\|F\|)$

The following algorithm is a rewrite of Algorithm 4 in [CH], except that it now takes a set of key-dependencies, rather than a set of key dependencies, as input

---

Algorithm 1

Input $\mathbf{R}=\{R_1, \ldots, R_m\}$ and $F=F_1\cup \ldots \cup F_m$, where F_i is a set of key-dependencies in R_i, $1\le i\le m$

Output *Accept* or *reject*, if *accept* is output, then the maximum key-equivalent partition of **R** is also output

Method

(1) generate the maximum key-equivalent partition $\{MKE_1, \ldots, MKE_n\}$ of **R** wrt F via KEP(**R**,F),

(2) **if** $\{\cup MKE_1, \ldots, \cup MKE_n\}$ is not independent wrt F **then** output *reject*

else output *accept* and $\{MKE_1, \ldots, MKE_n\}$

The following lemma follows from [CH]

Lemma 3 2: Let F be a set of key-dependencies in **R** When **R** and F are input, Algorithm 1 outputs *accept*, i e, the scheme induced by the maximum key-equivalent partition of **R** wrt F is independent wrt F, if and only if **R** is independence-reducible wrt F

Example 3.2: Consider the database scheme (**R**,F) of Example 3 1, where $\mathbf{R}=\{R_1(AB),R_2(BC),R_3(AC), R_4(AD), R_5(DEF),R_6(DEG)\}$ and $F=\{A\to B,\ B\to C,\ C\to A, A\to D, D\to EF, D\to EG\}$ Since $(R_2)^+_{F-F_1}\supseteq AB$, where $F_1=\{A\to B\}$, R_2 violates the uniqueness condition and thus **R** is not independent wrt F However, **R** is independence-reducible wrt F The maximum key-equivalent partition of **R** is $\{\{R_1,R_2,R_3,R_4\},\{R_5,R_6\}\}$ and its induced scheme $\{D_1(ABCD),D_2(DEFG)\}$ is independent wrt F □

In fact, a close inspection of [CH] discovers that the proofs that concerned with boundedness and algebraic-maintainability of independence-reducible schemes hold just as well when dependencies are given by a set of key-dependencies rather than by a set of key dependencies In other words, independence-reducibility based on key-dependencies not only have the same desirable properties wrt query answering and constraint enforcement as in [CH], but also gains more efficiency of these functions by taking a set of key-dependencies as parameters As a result, our modification is more practially useful

4 CHARACTERIZATION

We now return to the discussion of the central design problem of this paper In this section, we assume our input to be a set F of (nontrivial) fd's We shall characterize the existence of a database scheme that is embedding, independent, and in BCNF wrt F We show that this characterization remains the same for the existence of embedding ctm BCNF schemes and for the existence of embedding independence-reducible schemes The design algorithm is left to the next section

For a given set F of fd's, we define

$$\mathrm{scheme}(F)=\{XW \mid X\to W\in F\}$$

That is, scheme(F) contains, for each fd in F, a relation scheme consisting of all the attributes appearing in that fd Now let

$$\mathrm{scheme}(F)=\{R_1, \ldots, R_m\}$$

and define

$$F_i=\{X\to W \mid X\to W\in F \text{ and } R_i=XW\},\ \text{for } 1\le i\le m$$

It is not hard to see that $F=F_1\cup \ldots \cup F_m$ and F_i is a set of key-dependencies in R_i (but not necessarily a set of key dependencies in R_i), $1\le i\le m$ Thus F is a set of key-dependencies in scheme(F) Note that scheme(F) is not necessarily in 3NF since F is not required to be minimal [U]

Example 4.1. Consider the fd's

$$F=\{BE\to D, D\to B, C\to B, B\to C\}$$

Then $\mathrm{scheme}(F)=\{R_1(BED),R_2(BD),R_3(BC)\}$, and $F_1=\{BE\to D\}, F_2=\{D\to B\}, F_3=\{C\to B, B\to C\}$ Clearly, F is a set of key-dependencies in scheme(F), since $F=F_1\cup F_2\cup F_3$, and each F_i is a set of key-dependencies in R_i □

The following is the main theorem we shall prove in this section

Theorem 4.1: Let F be a set of (nontrivial) fd's The following statements are equivalent

(a) **(Characterization)** scheme(F) is independence-reducible wrt F

(b) There exists a database scheme that is embedding, independent, and in BCNF wrt F

(c) There exists a database scheme that is embedding, ctm, and in BCNF wrt F

(d) There exists a database scheme that is embedding and independence-reducible wrt F

(e) There exists a database scheme such that F is a set of key-dependencies in the scheme and the scheme is independent wrt F

Before proceeding to the proof of Theorem 4 1, let us consider more examples

Example 4.2. Consider the fd's

$$F=\{A\to B, B\to C, C\to A, A\to D, D\to EF, D\to EG\},$$

as given in Example 3 2 Clearly, scheme(F)=**R**, where **R** is the database scheme in that example Therefore scheme(F) is independence-reducible wrt F, and the database scheme induced by its maximum key-equivalent partition, i e, **D**=$\{D_1(ABCD), D_2(DEFG)\}$, is embedding and independent wrt F Also, it is easy to see that **D** is in BCNF wrt F Since independence implies constant-time-maintainability and independence-reducibility, as to be seen below, **D** is also ctm and independence-reducible wrt F □

Example 4.3. Consider fd's

$$F=\{B\to A, A\to C, C\to B, D\to B, D\to C\}$$

We have

$$\text{scheme}(F)=\{R_1(AB), R_2(AC), R_3(BC), R_4(BD), R_5(CD)\}.$$

By Lemma 3 2, scheme(F) is not independence-reducible wrt F The maximum key-equivalent partition of scheme(F) is $\{\{R_1,R_2,R_3\},\{R_4,R_5\}\}$ and the induced scheme $\{D_1(ABC), D_2(BCD)\}$ is not independent wrt F, because $C\to B$ is embedded in both D_1 and D_2 [GY] Therefore, Theorem 4 1 implies that F can not be embedded in any independent (ctm) BCNF scheme nor in any independence-reducible scheme

Now we consider the cover

$$F'=\{B\to A, A\to C, C\to B, D\to B\}$$

of F That is, the relationship on CD is now not required to be tabulated in the database In this case, design becomes possible

$$\text{scheme}(F')=\{R_1(AB), R_2(AC), R_3(BC), R_4(BD)\}$$

is independence-reducible wrt F' The maximum key-equivalent partition of scheme(F') is $\{\{R_1,R_2,R_3\},\{R_4\}\}$ and the induced scheme **D**=$\{D_1(ABC), D_2(BD)\}$ satisfies the uniqueness condition wrt $\{B\to AC, A\to BC, C\to AB, D\to B\}$, a cover of F' in form of [S2] Thus scheme **D** is embedding, independent (and also ctm and independence-reducible), and in BCNF wrt F' □

We now prove Theorem 4 1 by first mentioning a few results

Lemma 4.1 (Theorem 3 2 2 in [WG]) Let F be a set of fd's embedded in **R** If **R** is independent wrt F, then **R** is ctm wrt F

By a result in [HC] and Lemma 3 1, we have

Lemma 4.2: If **R** is embedding, ctm, and in BCNF wrt F, then **R** is independence-reducible wrt F

The following lemma shows that independence-reducibility implies BCNF

Lemma 4.3 Assume that **R** is independence-reducible wrt a set of key-dependencies F Then

(a) any independence-reduced scheme of **R** (wrt F) is in BCNF wrt F, and

(b) **R** is in BCNF wrt F

Proof Immediate from [CH] □

The next lemma tells that, given a set of fd's F, independence-reducibility of scheme(F) is necessary for our design goals

Lemma 4.4. Let F be a set of fd's Assume that there exists some database scheme **T** that is embedding, independent, and in BCNF wrt F Then

(a) if every relation scheme of **T** embeds at least one fd in F, **T** is induced by a key-equivalent partition of scheme(F) wrt F,

(b) scheme(F) is independence-reducible wrt F

We are now ready for the

Proof of Theorem 4 1 We prove the implication cycle (a) ==> (b) ==> (c) ==> (d) ==> (e) ==> (a)

(a) ==> (b) This follows from Lemma 4 3(a) and definitions

(b) ==> (c) This follows from Lemma 4 1

(c) ==> (d) This follows from Lemma 4 2

(d) ==> (e) Any independence-reduced scheme of the scheme mentioned in (d) is a scheme required by (e)

(e) ==> (a) Let **R** be the scheme mentioned in (e) Then **R** is trivially independence-reducible wrt F and therefore is in BCNF wrt F by Lemma 4 3(b) Then (a) follows from Lemma 4 4(b) □

5. DESIGN ALGORITHMS

In this section we assume our input to be a set U of attributes and a set F of fd's over U We present a polynomial time algorithm that tests the condition of Theorem 4 1(a) and, if the test succeeds, produces a database scheme over U that is embedding F, independent, and in BCNF wrt F By Theorem 4 1, Lemmas 4 1 and 4 2, the same algorithm also designs embedding ctm BCNF schemes and embedding independence-reducible schemes In all design cases, the produced database scheme, if there is one, contains the fewest possible number of relation schemes The idea of our algorithm is the following Given a set F of fd's, we tests whether scheme(F) is independence-reducible wrt F If not, by Theorem 4 1, the design goal is impossible to fulfil, otherwise, the database scheme induced by the maximum key-equivalent partition of scheme(F) wrt F is returned If some attributes in U are not mentioned in F, a minor modification to this scheme will give a database scheme over U that satisfies the same properties

We now present our design algorithm

Algorithm 2

Input A set U of attributes and a set F of fd's over U

Output *accept* or *reject*, if *accept* is output, a database scheme is also output

Method

(1) Run Algorithm 1 on input scheme(F) and F,

(2) **if** *reject* is output from step (1) **then** output *reject*

else do begin

(3) let **D** be the database scheme induced by the maximum key-equivalent partition returned from the execution of Algorithm 1 in step (1),

(4) let X be the set of attributes of U that are not mentioned in F,

(5) **if** X=∅ **then** output *accept* and **D**

else output *accept* and $\mathbf{D}\cup\{X\}$,

end

Theorem 5.1 let U be a set of attributes and let F be a set of fd's over U (1) Algorithm 2 outputs *accept* if and only if Theorem 4 1(a) holds (2) If Algorithm 2 outputs *accept*, it outputs a database scheme over U that is

(a) embedding, independent, and in BCNF wrt F,

(b) embedding, ctm, and in BCNF wrt F,

(c) embedding, indepedence-reducible, in BCNF wrt F

Moreover, the returned database scheme contains the fewest possible number of relation schemes in each case of (a), (b), and (c)

Proof sketch Part (1) follows from Theorem 4 1 and examination of Algorithm 2 In the following proofs we assume that Algorithm 2 outputs *accept* and a database scheme **R** Let F, **D**, and X be specified as in Algorithm 2 Observe that **D** is the scheme induced by the maximum key-equivalent partition of scheme(F) wrt F and thus **D** satisfies (a), (b), and (c) Clearly, the database scheme **R** returned in step (5) also satisfies (a), (b), and (c)

We now claim that **R** contains no more relation schemes than any database scheme over U that is (c) Then the minimality of **R** follows in all cases because by Lemma 4 1, being (a) implies being (b), which, by Lemma 4 2, implies being (c) The key argument for this claim is the following and it can be proved as a generalization of Lemma 4 4(a) For any database scheme **T** that is (c) and contains the fewest possible number of relation schemes, any independence-reduced scheme of **T** is a scheme induced by a key-equivalent partition of scheme(F) wrt F □

The time complexity of Algorithm 2 is analysed as follows The time of step (1) consists of the time of function KEP and the time on determining independence Since |scheme(F)|≤|F|, KEP(scheme(F),F) is bounded by $O(|F|^2||F||)$ Also by an algorithm in [IIK] or by testing the uniqueness condition, independence can be determined in time $O(|F|^2||F||)$ Therefore, Algorithm 2 is bounded by $O(|F|^2||F||+|U|)$

As illustrated in Example 4 3, the design results will depend on the choice of covers of functional dependencies in general The following theorem shows, however, that applying union and decomposition rules of fd's to the given fd's does not affect the design results The advantage of knowing this is that we can run Algorithm 2 faster by first obtaining a cover of F with smaller size and shorter description using the union rule of fd's

Theorem 5.2: Let F be a set of fd's over U and let F′ be any set obtained from F by applying union and decomposition rules to F Then the output from Algorithm 2 with U and F as input is the same as the output from Algorithm 2 with U and F′ as input

We can always make the produced database scheme (**R**,F) lossless by including the jd ***R** to the constraints What is really important is that this inclusion of the jd preserves the set of consistent states and the designed properties This is stated in a theorem below

Theorem 5.3: Let **R** be the database scheme returned by Algorithm 2 when U and F are input Then (1) CONS(**R**,F∪{***R**})=CONS(**R**,F), (2) **R** is a database scheme over U that is

(a) embedding F, independent, and in BCNF wrt F∪{***R**},

(b) embedding F, ctm, and in BCNF wrt F∪{***R**}

6. DESIGN SEPARABLE BCNF SCHEMES.

Now we extend the above methods to design embedding separable BCNF database schemes The following lemma can be proved by the notion of extensibility of database schemes [M]

Lemma 6.1: Let F be a set of key-dependencies in **R** if a database scheme induced by a key-equivalent partition of **R** wrt F is separable wrt F, then the scheme induced by the maximum key-equivalent partition of **R** wrt F is separable wrt F

In light of Lemma 6 1, we can show the following results for designing separable BCNF schemes

Theorem 6 1· Let F be a set of fd's There exists a database scheme that is embdding, separable, and in BCNF wrt F if and only if the following conditioms hold

(a) scheme(F) is independence-reducible wrt F, and

(b) the database scheme induced by the maximum key-equivalent partition of scheme(F) wrt F is separable wrt F

A polynomial time test of separability was suggested in [CM] when fd's are embedded Thus a polynomial time algorithm for designin embedding separable BCNF schemes can be obtained by inserting a line between steps (3) and (4) in Algorithm 2 that tests whether the scheme **D** in step (3) is separable and output *reject* if not. Moreover, since separability implies independence, Theorem 5.1 should imply that the scheme **R** so produced contains the fewest number of relation schemes among database schemes that are embedding, separable, in BCNF wrt F. By a result in [CM], **R** is also separable wrt $F \cup \{*\mathbf{R}\}$. Therefore, without affecting the separability and BCNF, we can simply add the jd ***R** to the produced database scheme (**R**,F) to enforce losslessness.

7. CONCLUSION

We have considered design problems of several very desirable properties of relational databases. These designs were aimed at both reducing data redundancy/update anomalies and achieving efficient data manipulations. In particular, we have characterized the condition under which a given set of functional dependencies can be embedded in an independent BCNF database scheme, and have presented a polynomial time algorithm that tested this condition and produced a satisfying database scheme whenever possible. We have shown that the exact same algorithm also worked for designing two generalizations of independent BCNF schemes, that is, ctm BCNF schemes and independence-reducible schemes. This essentially suggested that within the context of BCNF scheme design independent schemes are all we need to study, even we are allowed to consider more general database schemes like ctm schemes and independence-reducible schemes, provided that no constraints (other than functional dependencies) are imposed. Finally, we have also considered designing a restriction of independent BCNF schemes, namely, separable BCNF schemes.

Acknowledgements: This work was supported by a grant from the NSF of China.

[A] W.W. Armstrong, "Dependency structures of data base relationships," *IFIP Congress, 1974*, pp. 580-583.

[ABU] A.V. Aho, C. Beeri, and J.D. Ullman, "The theory of join in relational databases," *ACM TODS, Vol. 4, No. 3*, pp. 297-314.

[AC] P. Atzeni and E.P.F. Chan, "Efficient query answering in the representative instance approach," *ACM PODS 1985*, pp. 181-188.

[B] P.A. Berstein, "Synthesizing third normal form relations from functional dependencies," *ACM TODS, Vol. 1, No. 4, (December 1976)*, pp. 277-298.

[BB] C. Beeri and P.A. Bernstein, "Computational problems related to the design of normal form relation schemes," *ACM TODS, Vol. 4, No. 1, 1979*, pp. 30-59.

[BBG] C. Beeri, P.A. Berstein and N. Goodman, "Sophisticate's introduction to database normalization theory," *VLDB 1978*, pp. 113-124.

[BH] C. Beeri and P. Honeyman, "Preserving functional dependencies," *SIAM J. Comput., Vol. 10, No. 3, 1981*, pp. 647-656.

[BS] F. Bancilhon and N. Spyratos, "Independent components of data bases," *VLDB 1981*, pp. 398-408.

[CH] E.P.F. Chan and H.J. Hernandez, "Independence-reducible schemes," submitted to publication, 1988. (An extended abstract appeared in *ACM PODS 1988*, pp. 163-173).

[CM] E.P.F. Chan and A.O. Mendelzon, "Independent and separable database schemes," *SIAM J. Comput. Vol. 16, No. 5, October, 1987*, pp. 841-851.

[Co] E.F. Codd, "Recent investigation in relational systems," *IFIP 1987*, pp. 1017-1021.

[GMV] M.H. Graham, A.O. Mendelzon, and M.Y. Vardi, "Notions of dependency satisfaction," *JACM, Vol. 33, No. 1 (1986), pp. 105-129*.

[GW] M.H. Graham and K. Wang, "Constant time maintenance or the triumph of fd's," *ACM PODS 1986*, pp. 121-141.

[GY] M.H. Graham and M. Yannakakis, "Independent database schemes," *JCSS, Vol. 28, No. 1, 1984*, pp. 121-141.

[H] P. Honeyman, "Test satisfaction of functional dependencies," *JACM, Vol. 29, No. 3, 1982*, pp. 668-677.

[HC] H.J. Hernandez and E.P.F. Chan, "Constant-time-maintainable BCNF database schemes," submitted to publication, 1988. (An extended abstract appeared as "A characterization of constant-time-maintainable BCNF database schemes" in *ACM SIGMOD 1988*, pp. 209-217.

[HW] H.J. Hernandez and K. Wang, "On the boundedness of constant-time-maintainable database schemes," submitted to publication, 1989.

[IIK] M. Ito, M. Iwasaki and T. Kasami, "Some results on the representative instances in relational databases," *SIAM J. Comput., Vol. 14, No. 2, May, 1985*, pp. 334-353.

[M] A.O. Mendelzon, "Database states and their tableaux," *ACM TODS, Vol. 9, No. 2*, pp. 264-282.

[Ma] D. Maier, *The theory of relational databases*, CSP Inc., 1983.

[MMS] D. Maier, J.D. Ullman, Y. Sagiv, "Testing implication of data dependencies," *ACM TODS, Vol. 4, No. 4, 1979*, pp. 455-469.

[MUV] D. Maier, J.D. Ullman, M.Y. Vardi, "On the foundations of the universal relation model," *ACM TODS, Vol. 9, No. 2, 1984*, pp. 283-308.

[S1] Y. Sagiv, "Can we use the universal instance assumption without using nulls?" *ACM SIGMOD 1981*, pp. 108-120.

[S2] Y Sagiv, "A characterization of globally consistent databases and their correct access pathes," *ACM TODS, Vol 8, No , 2, 1983*, pp 266-286

[S3] Y Sagiv, "Evaluation of queries in independent database schemes," to appear in *JACM*

[U] J D Ullman, *Principles of database systems*, CSP, Inc , 1982

[V] Y Vassiliou, "A formal treatment of imperfect information in database management," Ph D thesis, University of Tornoto, 1980

[W] K Wang, "Can constant-time-maintainability be more practically useful?" *ACM PODS 1989*, pp 120-127

[WG] K Wang and M H Graham, "Constant-time-maintainability a generalization of independence," submitted to publication, 1988 (See [GW] for an extended abstract)

[Y] M Yannakakis, "Algorithms for acyclic database schemes," *VLDB 1981*