THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

THE USAGE DEPENDENCY MODEL FOR LOGICAL DATABASE DESIGN

E. Hevia
T.J. Teorey

CRL-TR-19-84

MARCH 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

The University of Michigan

Computing Research Laboratory

The Usage Dependency Model for Logical Database Design

E. Hevia
T. J. Teorey

March 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109-1109
USA
Tel: (313)763-8000

ABSTRACT

The usage dependency model is an extension to the traditional functional dependency model of relational database theory, that specifies the frequency of joint data element usage, based on process and data correspondence identified in the user requirements.

The combined application of usage dependency, which is a process-oriented view of data correspondence, and functional dependency, which is a nonprocessing or natural view of data correspondence, can produce a normalized database design that is efficient for current processing requirements as well as one which is adaptable to future requirements.

The usage dependency model is defined and an example is developed to show its applicability to real database environments.

Categories and Subject descriptions: H.2.1
[Database Management]: Logical Design - Normal Forms, Schema and Subschema.
General Terms: Algorithms, Design, Performance, heory.
Additional keywords and phrases: Relational database, functional dependencies.

TABLE OF CONTENTS

1. INTRODUCTION

Logical database design involves the derivation of a database schema from a set of end-user database system requirements. Traditionally, it has been divided into two phases: conceptual design, which derives a process-independent conceptual schema that represents the real-world organization of information to be computerized, and database schema design, which derives a process-dependent database management system (DBMS) - processible schema that minimizes access cost. The schema is further refined during physical design, where cost can be absolutely defined in terms of real time or monetary value.

Techniques to assist the traditional logical database designer have been limited by their inability to evolve from a human-intensive process to a computer-automated process. Consequently, although good designs have been attained by this methodology, the individual decision steps have not been made reproducible, so that the expertise is not easily transferable from one designer to the next. In other words, two designers using current logical design methodologies today are very likely to produce widely different designs because of human variability and the open-endedness of known techniques.

A well-known exception to these limited techniques is associated with the functional dependency model for database normalization [1]. Once semantic assertions can be made about data element relationships based on stated user requirements, functional dependencies can be defined and normalized relations

can be derived that capture the natural data relationships in the organization under investigation. This approach satisfies the real-world representation requirement for logical database design, but does not address the efficiency issue.

We define an extension to the functional dependency concept, a complementary approach named "usage dependency". Both human and computer processing requirements are defined so that the correspondence between processes and the data they manipulate can be quantified in matrix form, and data usage frequencies can be derived. Usage dependency is the combined application of functional dependency, based on data-data relationships, and joint data usage frequency, based on data-process relationships. Their intersection produces a DBMS-independent schema that has short-term efficiency properties. Their union produces a DBMS-independent schema that has long term flexibility for future changes in processing requirements. The objective of usage dependency is to guarantee a logical database design that is accurate, efficient and flexible. It is succinctly defined such that the basic process is algorithmic and deterministic. The mathematical properties of the usage dependency model, as well as those of the functional dependency model, provide for its potential use as a canonical form for algorithmic (and therefore automated) transformation to existing data models defined in currently known DBMS software.

2. CURRENT METHODOLOGIES

Most of the existing methodologies for logical database design take the semantic approach, where database design is considered as a process of picturing the real world as it "exists" by means of informal semantic views of data and the relationships between the data elements of each semantic view. Synthesis algorithms have been developed, and some are now utilized to produce normalized database schemas [2,3,4].

Another approach suggests a "usage" structure of data. In this approach, the database design process is based on anticipated information demands that the database must satisfy. Synthesis procedures have been developed to produce a database schema from the relations between processes and data, either weighted by its frequency of usage or not [5,6,7,8]. Since this design approach uses as input a set of anticipated queries (or primitive processes) along with the data elements used in each query, the output (database schema) is a "usage" design rather than an "existential" design.

In the usage approach, associations among attributes are obtained based on the frequency of joint usage by different processes. Even though it is realistic to assume that a strong relation exits between two attributes whenever their frequency of joint usage is high, it is not possible to conclude from this fact that either attribute uniquely determines the other one, or vice versa. However, attributes that are used together very often can be viewed as part of the same entity.

On the other hand, the existential approach is based on a given set of relations known as functional dependencies. In this approach, any two attributes with a functional dependency relation are represented in the database as such, even though they may never be used together. However, these functional dependencies are synthesized into a smaller set of relations of assorted degrees, based only on their given associations.

Since both approaches, existential and usage, have advantages of their own, it may be desirable to combine both in order to maximize their advantages, while at the same time reducing their limitations. The usage dependency model represents this combined approach.

3. THE USAGE DEPENDENCY MODEL

We now define the components of a Usage Dependency Model (UDM) for logical database design based on the combined existential and usage views of data. The main ideal behind UDM is to perform a partition of all the data elements (attributes) to be represented in the schema, by the use of an optimality criterion based on both functional dependencies and usage dependencies, the latter being obtained by frequency of joint data usage. The result is a partition of the set of attributes such that those attributes with functional dependency relations, which are used together very often, are clustered together in the same partition (relation).

Let V be a given universe of discourse where

$$V = \{ A, P, R, F, FP \}$$

Every member of V is a class of objects from the universe of discourse under consideration, and

$$A = \{ a_1, a_2, \ldots\ldots, a_n \}$$

is a set of attributes such that $A \in V$.

$$P = \{ p_1, p_2, \ldots\ldots, p_m \}$$

is a set of processes and $P \in V$.

Every process in P is assumed to be independent, in the sense that the set of attributes used by every $p \in P$ to perform its function does not depend on the outputs of any other process.

$$R = \{ r \mid r = (p_i, a_j),\ p_i \in P,\ a_j \in A \}$$

Every $r = (p_i, a_j) \in R$ is a binary relation between processes and attributes. This relation is defined as the "uses" relation, e.g., if $r = (p_1, a_3)$ then process 1 uses attribute 3.

Let X, Y be any two sets of attributes, then

$$F = \{ f \mid f = (X,Y),\ X,Y \subseteq A,\ X \neq \phi,\ X \rightarrow Y \}$$

Every $f = (X,Y) \in F$ is a functional dependency from X to Y. The first coordinate of f is the determinant of f, denoted LHS(f). The second coordinate of f, denoted RHS(f), is the determinacy of f.

Let FP be the mXm frequency matrix, where

$$FP(i,i) = \text{frequency of process } i$$

$$= 0 \text{ for } i \neq j$$

Thus, FP is a diagonal matrix whose entries are not necessarily distinct.

Let |X| denote the cardinality of the set X. The set L = {all determinants of F} can be partitioned into two subsets:

$$S = \{ X | X = LHS(f),\ X \subseteq L,\ f \in F,\ |X| = 1 \}$$

$$C = \{ X | X = LHS(f),\ X \subset L,\ f \in F,\ |X| > 1 \}$$

then S is the set of all single determinants of L, and C is the set of all composite determinants of L.

Let FD be the k×k functional dependency matrix over the field of the binary numbers, where k = |A| + |C| and let H = AUC; then if i ε H, i has an associated vector which is a row in FD. If j ε H, then j has an associated vector which is a column of FD. Then

FD(i,j) = 1 iff i→j.

= 0 otherwise.

FD is a functional dependency matrix with potential anomalies. It will be assumed that relations represented in FD are in 1NF. Let FD* be the transitive closure of FD and subject to the same anomalies as FD.

Let PDU be the m×k process-data usage matrix over the field of the binary numbers, where m = |P| and k = |A| + |C|. Then if i ε P, i is a process and has an associated row vector in PDU. If j ε H then j has an associated column vector in PDU and j is either an attribute or a composite determinant. Then

PDU(i,j) = 1 if process i uses attribute j

= 0 otherwise.

Let WPD be the m×k matrix over the field of the real numbers such that WPD = FP × PDU. Then WPD is the weighted process-data matrix, weighted by the frequency of each process during a fixed

period of time. Every entry in WPD indicates the number of times attribute j, or composite determinant j, is used by process i in a given period of time.

Let JDU be the k×k matrix over the field of the real numbers such that JDU = PDU' × WPD where PDU' is the transpose of PDU. Thus JDU is a matrix such that every entry indicates the frequency of usage of determinant i jointly with attribute j. Hence, JDU is the (symmetric) joint data usage matrix.

Finally, let UD be the k×k matrix over the field of the real numbers such that

$$UD(i,j) = FD^*(i,j) \cdot JDU(i,j) \qquad \forall i,j$$

thus, matrix UD is the usage dependency matrix. Matrix UD has the following properties:

(1) If $UD(i,j) \neq 0$ then UD(i,j) is the frequency of usage of $i \rightarrow j$ by known processes.

(2) If $UD(i,j) = 0$ $\forall j$, i fixed, then $i \in A$, but i is not a key attribute.

(3) If $UD(i,j) \neq 0$ for some j, i fixed, then i is a key attribute.

Matrix UD still contains anomalies in the sense that if a set of relations were to be derived from matrix UD, they must be subject to the same rules of normalization as the functional dependency matrix, FD.

4. A DESIGN EXAMPLE

The following database example illustrates the computation of usage dependency and its application to both short term efficiency and long term flexibility in relational database design. The example concerns a database to keep track of personnel assigned to projects at different locations. Each employee has a specific job-title and job-level. Employees may be assigned to several projects, each of which is conducted in several different locations. Projects and locations may each accommodate a large number of employees at one time.

There are three currently known processing requirements, although others may be added at a later date:

P1) For a particular employee, display job-level and projects assigned (frequency = 5).

P2) Display employee names for employees with a particular job-level (frequency = 10)

P3) For a given project, display all employee names and their locations (frequency = 3).

An analysis of this problem produces the functional dependencies (FD's) shown in Figure 1. Note that each value of EMPL-NAME is assumed to be unique.

Figure 2 represents the transitive closure of FD (FD*). Figure 3 shows the process-data usage correspondence in the PDU matrix, while Figure 4 shows process frequency in executions per day in the WPD matrix. The joint data usage matrix is derived as JDU=PDU' * WPD and shown in Figure 5, where each element is the

frequency of joint usage of the two data elements specified by row and column number. Finally, the usage dependency matrix, UD, is given in Figure 6.

Based on these computations we have three potential solutions (Figures 7-9). In general there could be more solutions because the normalization process does not necessarily produce a unique result. Normalized relation names are given by R1, R2, etc.

Solutions 1 and 2 are obtained directly from the FD and UD computations, respectively. Solution 2 replaces R1 and R2 with R4 because of the lack of usage of JOB-TITLE in any process and the transitive dependency of JOB-LEVEL on EMPL-NAME. This, of course, increases the short term processing efficiency by reducing access cost and storage space, all at the expense of future flexibility, and the potential loss of data integrity inherent in the relationship between JOB-TITLE and JOB-LEVEL.

Solution 3 provides the retention of the JOB-TITLE and JOB-LEVEL relationship, reduction of access cost, and future flexibility, at the expense of data redundancy. Normally this trade-off is well worth the additional storage space unless there is an extraordinarily large update activity associated with the database.

The functional dependencies in Solution 3 were derived by first incorporating all the FDs in Solutions 1 and 2, and then by searching the joint data usage (JDU) matrix for data usage relationships not already included in the existing FDs. This search reveals that D2 (JOB-LEVEL) and D5 (PROJ-NAME) have usage

but no natural relationship. Therefore they are defined as a composite key in R5. Again, this assignment is not necessarily unique, but a logical consequence of determining where data attribute clustering could increase efficiency and not violate normalization criteria.

Let us now assume the following data volume statistics and illustrate the numerical tradeoffs among these solutions:

(1) 100 employees

(2) 40 job-titles and job-levels

(3) 20 projects

(4) 10 locations

(5) Each employee works at an average of four locations.

(6) Each employee is assigned to an average of two projects.

(7) Each project has an average of 20 employees and 10 job-titles.

Let us simplify the numerical computation by assuming that each attribute has one unit length (in bytes). We then hypothesize that two additional processes will be added to the system later, but this new process information is unknown at the time of the original database design:

P4) For a given project, which job-levels are needed?

P5) List job-titles and their associated job-levels.

Storage space (Table 1) is given in attribute "unit" length, and illustrates the reduction of storage due to usage dependency and the increase due to the combined FD and UD Solution 3. Access

cost (Table 2) is computed in terms of estimated random and sequential tuple accesses per execution of a process. The results indicate that the UD Solution 2 and combined FD and UD Solution 3 provide the minimum access cost for the current applications. The UD solution, however, becomes far less efficient, or even infeasible for the future applications (unless the database can be easily reorganized); while the combined solution provides continued efficiency and a greater level of data integrity. In general, the combined approach will provide greater robustness in the design than either the existential or usage approach alone.

5. CONCLUSIONS

The usage dependency model is based on matrix representations of attribute-attribute and process-attribute relationships; it uses the concept of functional dependencies as well as usage of attributes in order to derive a conceptual schema. The main advantage of using matrix representation is to make the design procedure easy to perform algorithmically.

Future research involves the use of the UDM to derive a canonical schema for automatic transformation to both relational and nonrelational database schemas.

ACKNOWLEDGEMENTS

This research was partially supported by the Rome Air Development Center, U.S. Air Force, contract F30602-82-C-003. We are also indebted to Robert Voigt and Rong Chuan Xie for their helpful suggestions on the design example.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. ACM,6 (June 70), pp 377-387.

2. Bernstein, P. A., "Synthesizing 3NF Relations from functional Dependencies," ACMTODS 1,4 Dec. 76, 272-298.

3. Database Design Inc., "Data Designer User's Guide," Ann Arbor, MI., 1980.

4. Maier, David, The Theory of Relational Databases, Computer Science Press, Rockville, MD., 1983.

5. Dyba, J. E., "Data Element Identification," Portfolio 13-04-02, Auerbach Publisher, Inc., Pensanken, NJ, 1977.

6. Martin, J., Managing the Data-Base Environment, Prentice-Hall, Englewood Cliffs, NJ, 1983.

7. Rund, D. S., "Data Base Design Methodology," Parts I and II, Portfolio 23-01-01 nd 23-01-02. Auerbach Publisher, Inc., Pensanken, NJ, 1976.

8. Teorey, T. J. and Fry, J. P., Design of Database Structures, Prentice-Hall, Englewood Cliffs, N.J., 1982.

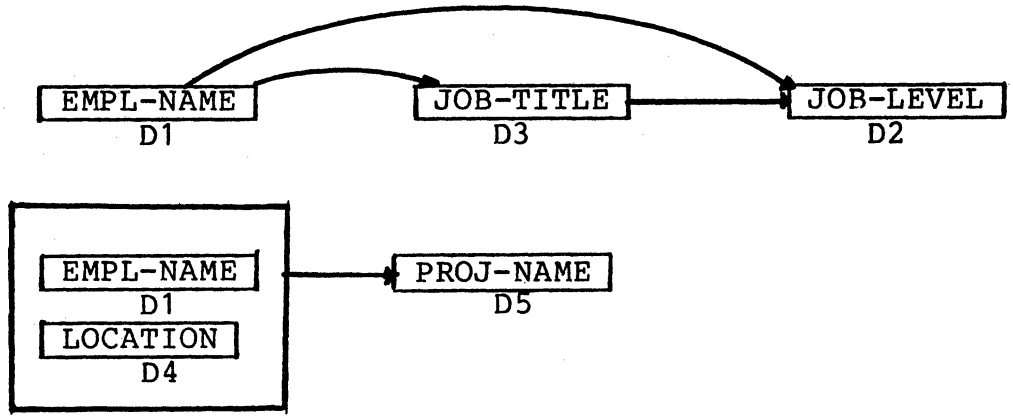

Figure 1. Functional dependencies in the personnel database example.

	D1	D2	D3	D4	D5	D1,D4
D1	0	1	1	0	0	0
D2	0	0	0	0	0	0
D3	0	1	0	0	0	0
D4	0	0	0	0	0	0
D5	0	0	0	0	0	0
D1,D4	0	0	0	0	1	0

D1=EMPL-NAME
D2=JOB-LEVEL
D3=JOB-TITLE
D4=LOCATION
D5=PROJ-NAME
D1,D4=EMPL-NAME, LOCATION

Figure 2. Functional dependency matrix (transitive closure), FD*.

	D1	D2	D3	D4	D5	D1,D4
P1	1	1	0	0	1	0
P2	1	1	0	0	0	0
P3	0	0	0	0	1	1

Figure 3. Process-data usage matrix, PDU.

	D1	D2	D3	D4	D5	D1,D4
P1	5	5	0	0	5	0
P2	10	10	0	0	0	0
P3	0	0	0	0	3	3

Figure 4. Weighted process-data matrix, WPD, using process frequency.

	D1	D2	D3	D4	D5	D1,D4
D1	15	15	0	0	5	0
D2	0	15	0	0	5	0
D3	0	0	0	0	0	0
D4	0	0	0	0	3	3
D5	5	5	0	0	8	3
D1,D4	0	0	0	0	3	3

Figure 5. Joint data usage matrix, JDU.

	D1	D2	D3	D4	D5	D1,D4
D1	0	15	0	0	0	0
D2	0	0	0	0	0	0
D3	0	0	0	0	0	0
D4	0	0	0	0	0	0
D5	0	0	0	0	0	0
D1,D4	0	0	0	0	3	0

Figure 6. Usage dependency matrix, UD.

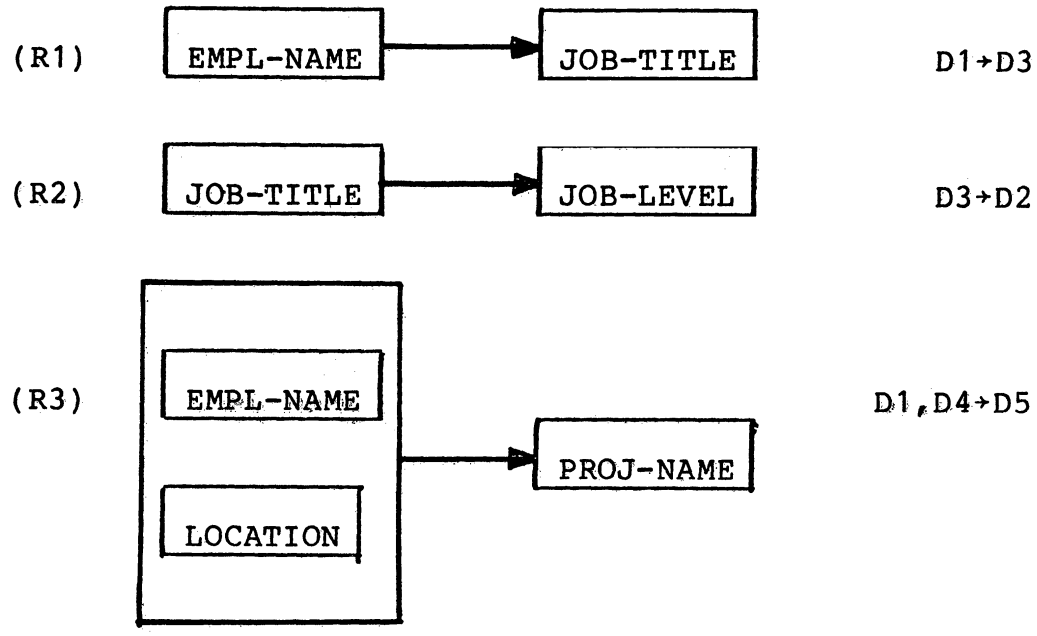

Figure 7. Solution 1: 3NF based on functional dependencies only (see Figure 1).

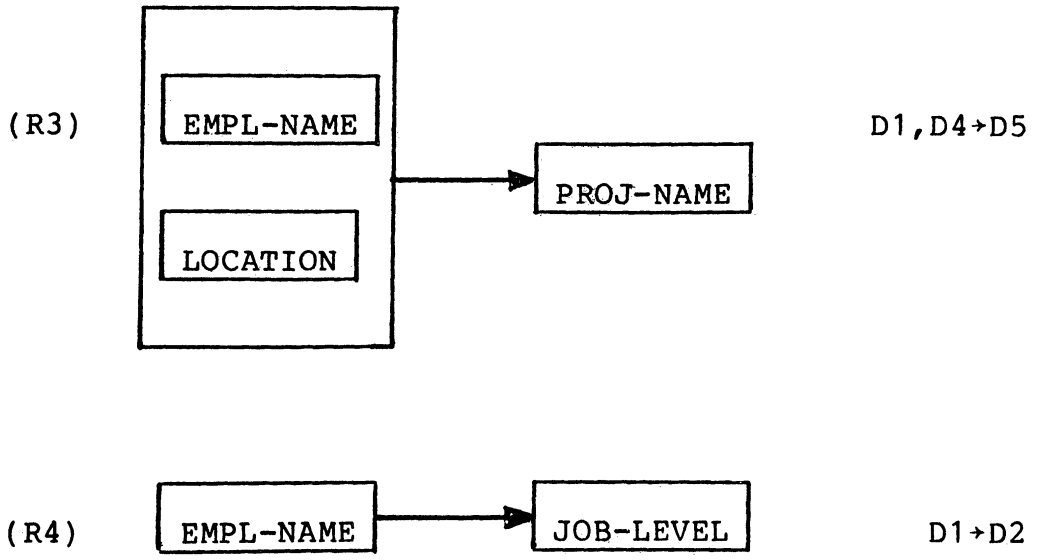

Figure 8. Solution 2: 3NF based on usage dependency only (see Figure 6).

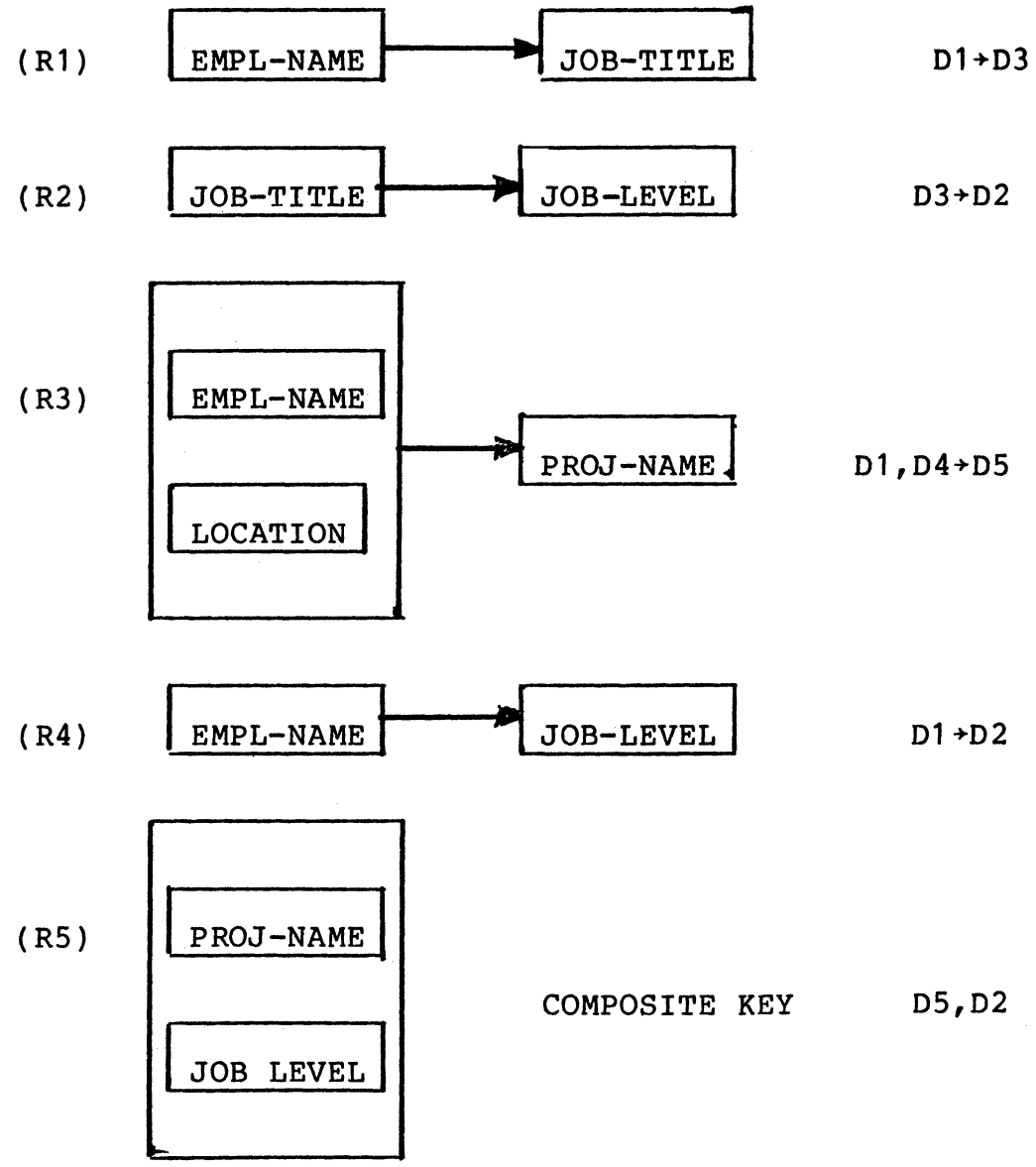

Figure 9. Solution 3: 3NF based on combined functional and usage dependency (see Figures 2,5,6).

	SOLUTION 1 (FD ONLY)	SOLUTION 2 (UD ONLY)	SOLUTION 3 (COMBINED FD AND UD)
R1	100 × 2 = 200		100 × 2 = 200
R2	40 × 2 = 80		40 × 2 = 80
R3	400 × 3 = 1200	400 × 3 = 1200	400 × 3 = 1200
R4		100 × 2 = 200	100 × 2 = 200
R5			200 × 2 = 400
TOTAL STORAGE UNITS	1480	1400	2080

TABLE 1. Comparative storage space requirements

	SOLUTION 1 (FD)	SOLUTION 2 (UD)	SOLUTION 3 (FD/UD)
P1 CURRENT PROCESSES	R1 ACCESS = 1R R2 ACCESS = 1R R3 ACCESS = 400s *	R4 SCAN = 1R R3 SCAN = 400s *	R4 SCAN = 1R R3 SCAN = 400s *
P2	R2 HALF SCAN = 20s R1 SCAN = 100s *	R4 SCAN = 100s *	R4 SCAN = 100s *
P3	R3 SCAN = 400s *	R3 SCAN = 400s *	R3 SCAN = 400s *
P4 FUTURE PROCESSES	R3 SCAN = 400s R1 SCAN = 100s R2 SCAN = 40s	R2 SCAN = 400s R4 SCAN = 100s	R5 SCAN = 200s *
P5	R2 SCAN = 40s *	DATA NOT AVAILABLE	R2 SCAN = 40s *

Table 2. Comparative access costs in random (R) and sequential (S) tuple accesses. (*-denotes minimum cost for a given process).