Usage Refinement for ER-to-Relation Design Transformations

TOBY J. TEOREY

Center for Information Technology Integration (CITI) and Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122

and

DONGQING YANG

Computer Science and Technology, Peking University, Beijing, The People's Republic of China

ABSTRACT

Database schema refinement based on usage is proposed as a useful next step in a practical database design methodology founded upon entity-relationship (ER) conceptual modeling and transformation to normalized relations. A simple cost model is defined and applied to several examples and a case study, illustrating the important trade-offs among query and update costs, storage requirements, and degree of normalization with its data integrity implications.

1. INTRODUCTION

Database design techniques for network and hierarchical systems often make use of processing requirements to refine the conceptual (entity-relationship) or logical (DBMS-processible) schema before the physical design phase if there are obvious large efficiency gains to be made [1, 4, 10]. The justification for this approach is that once physical design begins, the logical schema is considered to be fixed and is thus a constraint on efficiency. The database designer would often like to remove this inflexibility if possible.

A similar technique could be applied to relational databases if it would produce more efficient database schemas without loss of data integrity and would be relatively easy to implement. Our goal is to define a relational schema refinement algorithm based on a process-oriented or usage view that could increase the database efficiency for current processing requirements, and yet retain all the information content of the functional dependency or natural view of data. Thus the database would still be an accurate representation of real-world relationships and potentially more adaptable to future processing requirements.

The application of a usage refinement algorithm is the logical next step in practical database design methodologies suggested by [3, 5, 11], starting with conceptual modeling [2], transforming from the ER to the relational model, and normalization. Usage refinement could be used to specify alternative logical structures to be considered during physical design, and thus provide the physical designers with more feasible solutions to choose from. More efficient databases are therefore likely to be defined.

2. RELATION USAGE REFINEMENT

We assume that all attributes are initially assigned to relations based on functional dependencies, and that the relations are at least 3NF (normal forms) [7, 8]. This establishes the requirement for an accurate representation of reality and for flexibility of the design for future processing requirements. Efficiency for the current query requirements should increase by redundantly adding attributes, used together in a query, to an existing relation so that all attributes needed for that query reside in a new relation, called a join relation. This is known as materializing the join [9]. Access time will now be greatly reduced because fewer joins will be needed. However, the side effects of this redundancy include an increase in storage space required, an increase in the update cost, potential denormalization and loss of integrity, and program transformations for all applications containing joins that are materialized. These effects require careful consideration.

As an example of such an effect, let us assume that the relation PROJECT is associated with additional relations PART and SUPPLIER, as shown in Figure 1. We use query by example (QBE) to illustrate processes because of its extensive processing semantics [12]. The extension of the PART relation is shown as a means of reducing the number of joins required in the query. This extension results in a denormalization, with the side effects of add and update anomalies. However, the delete anomaly cannot occur because the original data are redundant in the extended schema. For example, SUPP-NO $\rightarrow$ SUPP-CITY in the extended PART relation (EXT-PART) is reproducible from PART-NO,PROJ-NAME $\rightarrow$ SUPP-NO in relation PART and SUPP-NO $\rightarrow$ SUPP-CITY in relation SUPPLIER.

The storage and processing cost of a logical relational database is to be computed for both the existing and new join relations:

$$\text{COST} = C_p^*(T_q + T_u) + C_s^* V_s \qquad (1)$$

Original relations and process (query)

PART(PART-NO,PROJ-NAME,SUPP-NO,PRICE)
SUPPLIER(SUPP-NO,SUPP-CITY,SUPP-MGR)
PROJECT(PROJ-NAME,HQ-CITY)

Query: For a given project, display the supplier numbers, supplier cities, and project headquarters city.

Functional dependencies

PART-NO,PROJ-NAME --> SUPP-NO | PRICE
SUPP-NO --> SUPP-CITY | SUPP-MGR
PROJ-NAME --> HQ-CITY

QBE representation of the query

PART	PART-NO	PROJ-NAME	SUPP-NO	PRICE
		*	P. X	

SUPPLIER	SUPP-NO	SUPP-CITY	SUPP-MGR
	X	P. Y	

PROJECT	PROJ-NAME	HQ-CITY
	*	P. Z

Extended relation PART in 1NF

EXT-PART (PART-NO,PROJ-NAME,SUPP-NO,SUPP-CITY,HQ-CITY,PRICE)
SUPPLIER (SUPP-NO,SUPP-CITY,SUPP-MGR)
PROJECT (PROJ-NAME,HQ-CITY)

Fig. 1. Relation extension causing denormalization.

where

C_p = unit cost per second for query or update processes

C_s = unit cost per byte for stored data

T_q = I/O service time for query processes (seconds)

T_u = I/O service time for update processes (seconds)

V_s = total volume in bytes for stored data

Unit costs are selected on the basis of the computing environment defined in the requirements specification. I/O service time for query and update can be determined from the processing operations, their frequencies, and the hardware device characteristics given, while stored data volume can be obtained from the size of the relations defined [4, 10]. Each query process must be expressed in terms of basic relational algebra operations such as selection, projection, and join. Some initial assumptions are made about sequential and random accesses needed to efficiently accomplish the query or update at this point, but detailed use of indexes, sorting, and so forth are deferred to physical design when the final configuration decisions are made.

2.1. RELATION USAGE ALGORITHM

The relation usage strategy is to select only the most dominant processes to determine modifications to relations that will most likely improve performance. The basic modification is to add attributes to existing relations in order to reduce join operations.

1. Select the dominant processes on the basis of criteria such as high frequency of execution, high volume of data accessed, response time constraints, or explicit high priority. As a rule of thumb any process with at least a factor of 10 higher frequency of execution or data volume accessed than another process is considered to be dominant.
2. Define join relations, when appropriate, to materialize joins for dominant processes.
3. Evaluate total cost for storage, query, and update for the database schema, with and without the extended relation, and determine which configuration minimizes total cost.
4. Consider also the possibility of denormalization due to a join relation and its side effects. If a join relation schema appears to have lower storage and processing cost and insignificant side effects, then consider that schema for physical design in addition to the original candidate relation schema. Otherwise, consider only the original schema.

In general, joins based on nonkeys should be avoided. They are likely to produce very large relations, thus greatly increasing storage and update cost. For example, if two relations have 100 and 200 tuples, respectively, then a join based on the key of either one will result in a maximum of 200 tuples, but a join based on a nonkey of either one could result in a maximum of 100×200 or 20,000 tuples. Null values are also to be restricted to nonkey attributes so that they will not be inadvertently used in join operations.

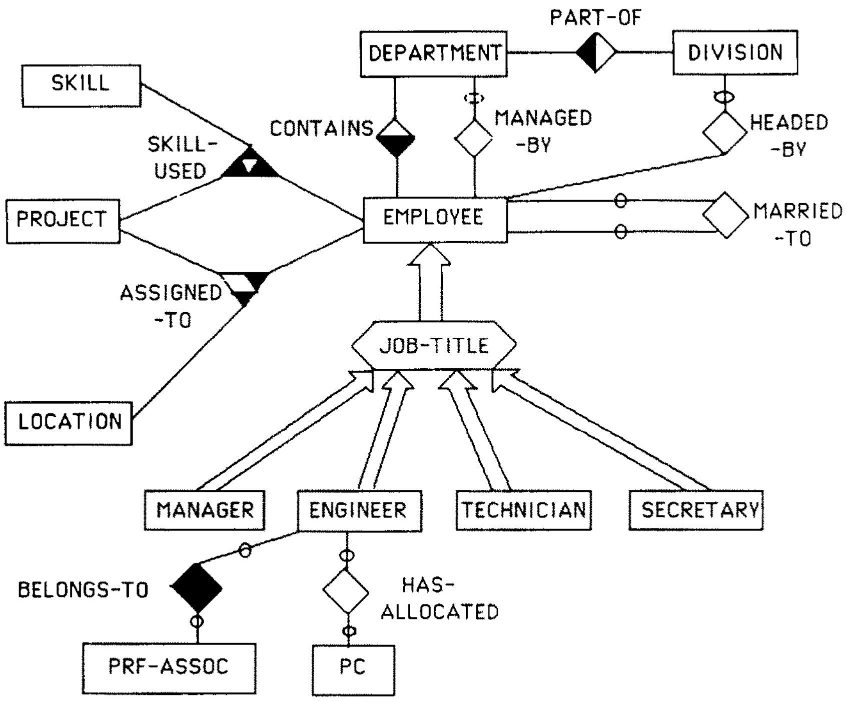

Fig. 2. Company personnel and project database (extended ER diagram).

2.2. *ALGORITHM APPLICABILITY*

The following examples, extending the company personnel and project database design problem defined in [11] and illustrated in Figure 2, show the extremes of applicability and nonapplicability of the relation usage algorithm. In each case we apply the algorithm to a given relational schema and given processing requirements. Cost trade-offs are then evaluated to determine if schema refinement is justifiable.

The ER conceptual model of the company personnel and project database includes extensions of the ER model such as generalization, i.e. the concept of category (EMPLOYEE) and subcategory (MANAGER, ENGINEER, TECHNICIAN, SECRETARY), where subcategory names are based on values of the EMPLOYEE attribute JOB-TITLE. Optional membership (or existence) of an entity in a relationship is designated by a 0 on the arc between that entity and the relationship. Otherwise the membership is assumed to be mandatory.

Note that the extended ER constructs also contain ternary relationships, which are defined by the shading of the corners of the triangles that connect the entities involved in the relationship. The shading indicates the functional dependencies (FDs) associated with the relationship. All FDs are defined by the primary keys of the three entities in the relationship, and an unshaded corner of the triangle implies that the primary key of that entity is on the right-hand side of an FD, and the primary keys of the other two entities are the composite determinant of the FD. Thus, the number of unshaded corners equals the number of FDs associated with the relationship. When all three corners are shaded, there are no FDs, and the key of the relationship is the composite of all three primary keys of the involved entities.

Applying the methodology suggested in [11] and summarized in Appendix I, the final normalized relations are the following:

1. DIVISION(DIV-NO,HEAD-EMP-NO,...)
2. DEPARTMENT(DEPT-NO,DEPT-NAME,ROOM-NO,..., DIV-NO,MANAG-EMP-NO)
3. EMPLOYEE(EMP-NO,EMP-NAME,JOB-TITLE,..., DEPT-NO,SPOUSE-EMP-NO,PC-NO)
4. SKILL(SKILL-NO,...)
5. PROJECT(PROJ-NAME,...)
6. LOCATION(LOC-NAME,...)
7. EMP.MANAGER(EMP-NO,...)
8. EMP.ENGINEER(EMP-NO,...)
9. EMP.TECHNICIAN(EMP-NO,...)
10. EMP.SECRETARY(EMP-NO,...)
11. PC(PC-NO,...)
12. PRF-ASSOC(PA-NO,...)
13. BELONGS-TO(PA-NO,EMP-NO)
14. SKILL-USED(EMP-NO,SKILL-NO,PROJ-NAME)
15. ASSIGNED-TO(EMP-NO,LOC-NAME,PROJ-NAME)
16. ROOM(ROOM-NO,PHONE-NO)

2.2.1. Example 1

Example 1 illustrates the most favorable conditions for efficiency improvement with the relation usage algorithm (see Figure 3). The query "display each pair of employee and project in which the project is located in the same city where the employee lives" is executed by a join of EMPLOYEE and

Relations	**Bytes/tuple**	**Tuples**	**Total Bytes**
EMPLOYEE(EMP-NO,EMP-CITY,..)	120	10000	1200 KB
PROJECT(PROJ-NAME,HQ-CITY,..)	200	500	100 KB
ASSIGNED-TO(EMP-NO,PROJ-NAME,..)	20	20000	400 KB

Query: Display each pair of employee and project in which the project headquarters(HQ) is located in the same city where the employee lives.

Update: Delete a given employee from all associated projects.

QBE representation of the query

```
EMPLOYEE | EMP-NO  | EMP-CITY  |          PROJECT | PROJ-NAME  | HQ-CITY |
         |  P.X    |    Z      |                  |    P.Y     |    Z    |

ASSIGNED-TO | EMP-NO  | PROJ-NAME  |
            |   X     |     Y      |
```

QBE representation of the update

```
ASSIGNED-TO | EMP-NO  | PROJ-NAME  |     PROJECT | PROJ-NAME  | HQ-CITY |
         D. |   *     |     X      |             |     X      |         |
```

Unit costs: C@p = 9.00 per disk-hour, C@s = .0031 per page-day
Frequency of all processes: 100/day

Fig. 3. Example 1 relations and processes.

ASSIGNED-TO over EMP-NO, followed by 20,000 random accesses to PROJECT (based on PROJ-NAME) to match HQ-CITY with each EMP-CITY in the temporary relation resulting from the join. To simplify the computation of query time the relations are assumed to be accessed as: EMPLOYEE(sequential, ordered on EMP-NO), PROJECT (indexed on PROJ-NAME), and ASSIGNED-TO (sequential, ordered on EMP-NO).

Using the hardware configuration for the Amdahl 5860 system (currently an IBM 3090-600) at the University of Michigan, the following timing characteristics occur:

Page transfer time (at 4,096 bytes per page): 3.4 ms
Average disk rotation time (half rotation): 8.3 ms
Average disk seek time: 16.0 ms
Average sequential page access = 11.7 ms
Average random page access = 27.7 ms
$C_p = 9.00$ dollars per I/O hour
$C_s = .0031$ dollars per page-day

Given the number of bytes in each of the relations and the searching required for the query, we can calculate the I/O service time (T_q) for the query, and thus the total cost (Equation 1). The remainder of the example is to determine the number of pages for query and update operations and storage space and to calculate total cost.

$$\begin{aligned} T_q &= \text{scan EMPLOYEE} + \text{scan ASSIGNED-TO} \\ &\quad + 20{,}000 \text{ random accesses to PROJECT} \\ &= \text{ceiling}(1{,}200{,}000/4{,}096)*11.7 \\ &\quad + \text{ceiling}(400{,}000/4{,}096)*11.7 + 20{,}000*27.7 \text{ ms} \\ &= 558.575 \text{ sec} \\ &= .155 \text{ hour} \end{aligned}$$

$$\begin{aligned} \text{I/O cost (query)} &= C_p * T_q \\ &= 9.00*.155 \\ &= 1.396 \end{aligned}$$

$$\text{I/O cost (at 100 queries per day)} = 139.6$$

The update operation "delete a given employee from all associated projects" requires a random access to ASSIGNED-TO based on EMP-NO and a scan of an additional page to delete all tuples with a given EMP-NO.

$$\begin{aligned} T_u &= 27.7 \text{ ms} + 11.7 \text{ ms} \\ &= .039 \text{ seconds} \end{aligned}$$

$$\begin{aligned} \text{I/O cost (update)} &= C_p^* T_u \\ &= 9.00*.039/3{,}600 \\ &= .0001 \end{aligned}$$

$$\text{I/O cost (at 100 updates per day)} = .01$$

$$\begin{aligned} \text{Storage cost} &= C_s^* V_s \\ &= .0031 \text{ per page day}* \\ &\quad [\text{ceiling}(1{,}200{,}000/4{,}096) \\ &\quad + \text{ceiling}(100{,}000/4{,}096) \\ &\quad + \text{ceiling}(400{,}000/4{,}096)] \\ &= .0031*416 \text{ pages} \\ &= 1.29 \end{aligned}$$

$$\begin{aligned} \text{Total cost} &= 139.6 + .01 + 1.29 \\ &= 140.9 \end{aligned}$$

The extended join relation solution is to append to ASSIGNED-TO the attributes EMP-CITY and HQ-CITY so that only a single scan of the new relation, which we will call EXT-ASSIGNED-TO, is needed to satisfy the query.

TABLE 1

Summary of Total Cost Per Day (Examples 1 and 2)

	Original relation	Join relation	
Example 1			
$C_p^* T_q$	139.6	.57	Query cost
$C_p^* T_u$	.01	.01	Update cost
$C_s^* V_s$	1.29	1.59	Storage cost
	140.9	2.17	Total cost
Example 2			
$C_p^* T_q$	1.4	1.8	Query cost
$C_p^* T_u$	2.9	3.6	Update cost
$C_s^* V_s$	1.5	1.9	Storage cost
	5.8	7.3	Total cost

EXT-ASSIGNED-TO now has 40 bytes per tuple; therefore at 20,000 tuples it has a total of 800,000 bytes and is double the size of ASSIGNED-TO. Redoing the calculations for query, update, and storage with EXT-ASSIGNED-TO, we obtain the cost figures shown in Table 1. We see that there is a dramatic reduction in cost by using the extended join relation and avoiding the join and random indexing of the original solution.

2.2.2. Example 2

Example 2 illustrates the least favorable conditions for efficiency improvement with the relation usage algorithm. The query given in Figure 4 is executed by a join on the relations EMPLOYEE and DEPARTMENT over the common attribute DEPT-NO. This is accomplished by a scan of EMPLOYEE and DEPARTMENT. DEPARTMENT and EMPLOYEE are assumed to be accessed sequentially based on DEPT-NO.

$$\begin{aligned} T_q &= \text{scan of EMPLOYEE} + \text{scan of DEPARTMENT} \\ &= \text{ceiling}(2{,}000{,}000/4{,}096)^*11.7\text{ ms} \\ &\quad + \text{ceiling}(15{,}000/4{,}096)^*11.7\text{ ms} \\ &= 5{,}713\text{ ms} + 47\text{ ms} \\ &= 5{,}760\text{ ms} \end{aligned}$$

$$\begin{aligned} \text{I/O cost (query)} &= 9.00^*5.76\text{ sec}/3{,}600 \\ &= .014 \end{aligned}$$

$$\text{I/O cost (query at frequency of 100 per day)} = 1.4$$

Relations	Bytes/tuple	Tuples	Total Bytes
EMP (EMP-NO,EMP-NAME,AUTO-TYPE,DEPT-NO)	200	10000	2000 KB
DEPT(DEPT-NO,DEPT-NAME,OFF-NO,EMP-NO)	250	60	15 KB

Query: Display employee number, name, office, and department name for all employees with a given automobile type.

Update: Scan the employee relation and make necessary changes as specified in an in-core update list.

QBE representation of the query

EMPLOYEE	EMP-NO	EMP-NAME	AUTO-TYPE	DEPT-NO
	P. A	P. B	*	X

DEPARTMENT	DEPT-NO	DEPT-NAME	OFF-NO	EMP-NO
	X	P. C	P. D	

QBE representation of the update

EMPLOYEE	EMP-NO	EMP-NAME	AUTO-TYPE	DEPT-NO
U.	*	*	*	*

Frequency of all processes: 100/day

Fig. 4. Example 2 relations and processes.

The update of department number of every employee is accomplished with a scan (read) and rewrite of EMPLOYEE:

T_u = scan and rewrite of EMPLOYEE
= [ceiling(2,000,000/4,096)*11.7 ms]*2
= 11,443 ms

I/O cost (update) = 9.00*11.443 sec/3,600
= .029

I/O cost (update at frequency of 100 per day) = 2.9

Storage cost = ceiling(2,000,000/4,096)*.0031 + ceiling(15,000/4,096)*.0031
= 1.5 per day

The extended join relation solution is to add the attributes DEPT-NAME and OFF-NO to relation EMPLOYEE, thus increasing the tuple size from 200 to 250 bytes. The size of the entire relation EXT-EMPLOYEE is 2.5 MB, compared to 2 MB for EMPLOYEE. The cost for query, update, and storage space for the extended relation is shown in Table 1. The results show higher cost in all three areas that is due to the extended join relation, mainly because the relation EMPLOYEE is much larger than the relation DEPARTMENT and the extension EXT-EMPLOYEE is larger than EMPLOYEE and DEPARTMENT combined. Thus, the join relation schema is not a candidate for physical design in this case.

To summarize, the extended join relation tends to significantly lower the storage and processing cost for one or more joins if the joined relations are of comparable size, if only the smaller relation is extended, or if it can avoid a large number of random accesses to at least one of the relations.

3. A CASE STUDY

The following case study illustrates how the usage refinement approach easily extends a logical design methodology for simple improvements in performance. The problem definition is followed by the solution steps of ER model definition, functional dependency definition, transformation to relations, normalization and reduction of relations, and usage refinement. Trade-offs among the degree of normalization, storage requirement, and query and update costs are analyzed before the final relation definitions can be specified.

3.1. REQUIREMENT SPECIFICATION

The management of a large retail store would like a database to keep track of their sales activities. The requirement for this database specifies six entities and their unique identifiers as follows:

Entity	Entity identifier	Identifier length	Cardinality
CUSTOMER	CUST-NO	6 char	80,000
JOB	JOB-TITLE	24 char	80
ORDER	ORDER-NO	9 char	200,000
SALESPERSON	SALES-NAME	20 char	150
DEPARTMENT	DEPT-NO	2 char	10
ITEM	ITEM-NO	6 char	5,000

The following assertions describe the data relationships:

1. Each customer has one job title, but different customers may have the same job title.
2. Each customer may place many orders, but only one customer may place a particular order.
3. Each department has many salespersons, but each salesperson must work in one department.
4. Each department has many items for sale, but each item is sold in only one department ("item" means item type, like IBM PC).
5. Each order could be placed for several different items located in the same or different departments.

6. For each order, items ordered in different departments must involve different salespersons, but all items ordered within one department must be handled by exactly one salesperson. In other words, for each order, each item has exactly one salesperson; and for each order, each department has exactly one salesperson.

3.2. *DESIGN PROBLEM*

1. Using the information given above, and in particular the six assertions, derive an ER diagram and a set of functional dependencies (FDs) that represent all the data relationships.

2. Transform the ER diagram into a set of candidate relations. List the relations, their primary keys, and other attributes.

3. Find the minimum set of BCNF relations that are functionally equivalent to the candidate relations. Analyze performance and integrity trade-offs that result from the definition of this minimum set.

4. Given the transactions "select all order numbers assigned to customers who are computer engineers" and "add a new customer and the customer's order to the database," analyze the performance and data integrity trade-offs for strategies to execute these transactions with the minimum-set BCNF schema and a refined schema designed to reduce the number of joins needed for data retrieval.

3.3. *LOGICAL DESIGN*

Our first step is to develop an ER diagram (Figure 5) and a set of FDs to correspond to each of the six assertions given. Normally the ER diagram is developed without knowledge of all the FDs, but in this example the nonkey attributes are omitted so that the entire database can be represented with only a few statements and FDs. The result of this analysis, relative to each of the six assertions given above, is as follows:

	ER construct	Functional dependencies
1.	CUSTOMER(many:JOB(one)	CUST-NO → JOB-TITLE
2.	ORDER(many):CUST-NO(one)	ORDER-NO → CUST-NO
3.	SALESPERSON(many): DEPARTMENT(one)	SALES-NAME → DEPT-NO
4.	ITEM(many):DEPARTMENT(one)	ITEM-NO → DEPT-NO
5.	No meaningful relationship	NONE
6a.	ORDER(many):ITEM(many): SALESPERSON(one)	ORDER-NO,ITEM-NO → SALES-NAME
6b.	ORDER(many): DEPARTMENT(many): SALESPERSON(one)	ORDER-NO,DEPT-NO → SALES-NAME

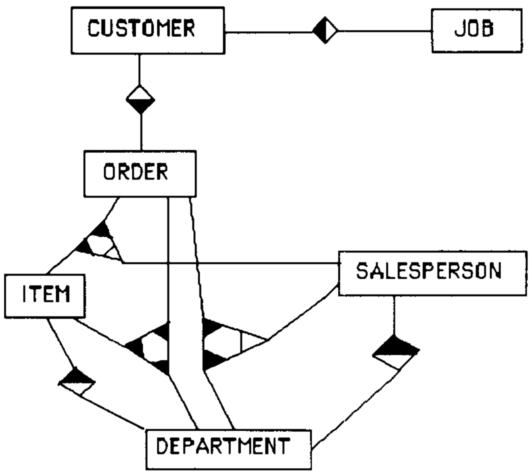

Fig. 5. Extended ER diagram for the retail store database.

The candidate relations needed to represent the semantics of this problem can be easily derived from the constructs for entities and relationships. Primary keys are underlined.

1. CUSTOMER(CUST-NO,JOB-TITLE)
2. JOB(JOB-TITLE)
3. ORDER(ORDER-NO,CUST-NO)
4. SALESPERSON(SALES-NAME,DEPT-NO)
5. DEPARTMENT(DEPT-NO)
6. ITEM(ITEM-NO,DEPT-NO)
7. ORDER-ITEM-SALES(ORDER-NO,ITEM-NO,SALES-NAME)
8. ORDER-DEPT-SALES(ORDER-NO,DEPT-NO,SALES-NAME)

Candidate relations 1–6 are formed directly from entities and are all BCNF. Relation 7 is also BCNF, but relation 8 is only 3NF. Relation 8 has two functional dependencies:

$$\text{ORDER-NO,DEPT-NO} \rightarrow \text{SALES-NAME}$$

$$\text{SALES-NAME} \rightarrow \text{DEPT-NO}$$

which cannot be decomposed into independent BCNF relations if the first functional dependency is still to be preserved. Consequently it must remain in 3NF [3].

This process of decomposition and reduction of relations moves us closer to a minimum set of 3NF or BCNF relations. Additionally, we must consider the relations JOB and DEPARTMENT. Because we have not defined other attributes in these relations, JOB and DEPARTMENT are simple relations consisting of a single key attribute. When this occurs, and the key attribute appears in the other relation as a nonkey, we can consider the elimination of the simple relation. The trade-off is between the decrease in storage space and update cost when we eliminate a relation, and the possible loss of data integrity as a side effect of deletions on another relation in which the key of the eliminated relation has become a nonkey. In our example, if we can justify this trade-off and eliminate the simple relations, we have the following minimum set of 3NF and BCNF relations:

1. CUSTOMER(CUST-NO, JOB-TITLE)	BCNF
2. ORDER(ORDER-NO, CUST-NO)	BCNF
3. SALESPERSON(SALES-NAME, DEPT-NO)	BCNF
4. ITEM(ITEM-NO, DEPT-NO)	BCNF
5. ORDER-ITEM-SALES(ORDER-NO,ITEM-NO, SALES-NAME)	BCNF
6. ORDER-DEPT-SALES(ORDER-NO,DEPT-NO, SALES-NAME)	3NF

In summary, the reductions shown above have decreased storage space and update cost and have maintained the normalization at a minimum of 3NF; but we have potentially higher retrieval cost (e.g., given the transaction "list all job-titles") and have increased the potential for loss of integrity due to the elimination of simple relations with only key attributes.

3.4. SCHEMA REFINEMENT BASED ON USAGE

Let us now look at the quantitative trade-offs of further refinement of relations to accommodate processing efficiency. We shall assume that each of the following transactions are to be executed once per fixed time unit:

1. *Query*, Select all order numbers assigned to customers who are computer engineers.
2. *Update*, Add a new customer and the customer's order to the database.

Using the minimum-set 3NF/BCNF schema, we can execute the transactions in a number of different ways. Let us first assume that the relations are all ordered physically by their primary keys. We use the following strategy for the first transaction: Sort the ORDER relation by CUST-NO, then join relations ORDER and CUSTOMER with a single scan of each, and select only tuples that have JOB-TITLE of computer engineer. We then project on

ORDER-NO for the resulting display. To simplify the analysis we assume that a sort of n tuples takes $n \log_2 n$ tuple accesses (TA) and that computer engineers make up 5% of the customers and orders in the database.

$$\begin{aligned} \text{TA} &= \text{sort ORDER} + \text{scan ORDER} + \text{scan CUSTOMER} + \text{create ORDER-CUST} \\ &\quad + \text{scan ORDER-CUST} + \text{create COMP-ENGR} + \text{project COMP-ENGR} \\ &= (200{,}000 \log_2 200{,}000) + 200{,}000 + 80{,}000 + 200{,}000 \\ &\quad + 200{,}000 + 200{,}000^*.05 + 200{,}000^*.05 \\ &= 200{,}000^*(17.61 + 3.10) + 80{,}000 \\ &= 4{,}222{,}000 \text{ tuple accesses} \end{aligned}$$

Since all tuple accesses are sequential in this strategy, and assuming 10 ms per sequential block access and block size of 1000 bytes, we estimate the I/O service time to process this by first computing the blocking factors for relations ORDER, CUSTOMER, ORDER-CUST and COMP-ENGR: 66, 33, 25, and 25, respectively. We compute the physical block accesses (PBA) as follows:

$$\begin{aligned} \text{PBA} &= \text{ceiling}(200{,}000^*18.61/66) + \text{ceiling}(80{,}000/33) + \text{ceiling}(420{,}000/25) \\ &= 75{,}619 \end{aligned}$$

$$\begin{aligned} \text{IOTIME} &= 75{,}619^*10 \text{ ms} \\ &= 756 \text{ seconds} \end{aligned}$$

The strategy to execute the second transaction using the same schema is to scan each relation (ORDER and CUSTOMER) and rewrite both relations with the new order.

$$\begin{aligned} \text{PBA} &= \text{ceiling}(200{,}000/66)^*2 + \text{ceiling}(80{,}000/33)^*2 \\ &= 10{,}912 \end{aligned}$$

$$\begin{aligned} \text{IOTIME} &= 10{,}912^*10 \text{ ms} \\ &= 109 \text{ seconds} \end{aligned}$$

If we refine the minimum-set 3NF/BCNF schema to avoid the join in the first transaction, the resulting schema will have a single relation ORDER-CUST (ORDER-NO,CUST-NO,JOB-TITLE) instead of separate relations ORDER and CUSTOMER. This avoids not only the join, but also the sort needed to get both relations ordered by CUST-NO. The strategy for the first transaction is now to scan ORDER-CUST once to find the computer engineers, write out the data to disk, and then read back from disk to project the resulting temporary relation COMP-ENGR for the final display.

$$\begin{aligned} \text{PBA} &= \text{ceiling}(200{,}000/25) + [\text{ceiling}(200{,}000{*}.05/25)]{*}2 \\ &= 8{,}800 \end{aligned}$$

$$\begin{aligned} \text{IOTIME} &= 8{,}800{*}10 \text{ ms} \\ &= 88 \text{ seconds} \end{aligned}$$

The strategy for the second transaction using this refined schema is to scan ORDER-CUST once to find the point of insertion, and then to scan again to reorder the relation.

$$\begin{aligned} \text{PBA} &= \text{ceiling}(200{,}000/25){*}2 \\ &= 16{,}000 \end{aligned}$$

$$\begin{aligned} \text{IOTIME} &= 16{,}000{*}10 \text{ ms} \\ &= 160 \text{ seconds} \end{aligned}$$

Common to the two strategies is the addition of an ORDER tuple to the relations ORDER-ITEM-SALES and ORDER-DEPT-SALES. For the sake of simplicity we will assume these relations to be unsorted, so the addition of a new order will require only one tuple access at the end of the relation, and thus negligible IOTIME.

The basic performance and normalization data for these two schemas and the two transactions given above are summarized in Table 2.

The refined schema dramatically reduces the I/O time for the query transaction; but the cost is the loss of performance for the update, more storage space, and significant reduction in the degree of normalization. The

TABLE 2

Comparison of Performance and Integrity of Original Relations and Join Relation

	Minimum set 3NF/BCNF schema (ORDER and CUSTOMER)	Refined schema (ORDER-CUST)
Query	756 seconds	88 seconds
Update	109 seconds	160 seconds
Storage space	5.4 MB	7.8 MB (relevant relations)
Normalization	3 NF	2 NF

TABLE 3

Comparison of Three Schemas for Performance and Integrity

	ORDER-CUSTOMER	ORDER-CUST	All 3 relations
Query	756 seconds	88 seconds	88 seconds
Update	109 seconds	160 seconds	269 seconds
Storage space	5.4 MB	7.8 MB	13.2 MB (relevant relations)
1Normalization	3 NF	2 NF	2 NF, 3NF combination

normalization is reduced because we now have a transitive functional dependency ORDER-NO → CUST-NO → JOB-TITLE in relation ORDER-CUST. The implication of this is, of course, the delete anomaly for JOB-TITLE when a customer deletes an order or the order is filled.

We can illustrate the trade-off between degree of normalization and performance with a simple alternative schema. In this case, instead of replacing ORDER and CUSTOMER by ORDER-CUST, we keep all three relations in the database. Thus normalization for these relations is preserved from the original schema. The cost is not only for increased storage space and update I/O time, but also for a greater choice of retrieval options requiring more query optimization software. The performance changes are summarized in Table 3.

4. CONCLUSION

A practical approach to extending a logical database design methodology for database usage has been presented and illustrated with several detailed examples, extending previous work based upon requirements analysis using the ER model and its transformation to the relational model. The extensions focus on the trade-offs between normalization, minimization of the number of relations, and the relation refinements to improve query processing at the expense of update and storage cost. The degree of normalization provides a well-defined level of data integrity in terms of the delete anomaly.

The significance of these performance and data integrity differences depends upon the global set of objectives and computing environment for the database, and must be analyzed in that context. For instance, the performance differences must be evaluated for all relevant transactions, present and projected. Storage space differences may or may not be significant in the computing environment. Deletion integrity problems need to be evaluated on a case-by-case basis to determine whether the side effects of certain tuple deletions are destructive to the objectives of the database. In summary, the database designer now has the ability to evaluate the trade-offs among query, update, storage space, and integrity associated with normalization. This knowledge can be applied to a variety of database design problems.

APPENDIX I

SUMMARY OF LOGICAL RELATIONAL DATABASE DESIGN STEPS

1. Extended ER (EER) modeling of requirements
   1.1 Identify entities and attach attributes to each.
   1.2 Identify generalization and subset hierarchies.
   1.3 Define relationships.
   1.4 Integrate multiple views of entities, attributes, and relationships.
2. Transformation of the EER model to relations
   2.1 Transform every entity into one relation with the key and nonkey attributes of the entity.
   2.2 Transform every many-to-many binary (or unary) relationship into a relationship relation.
   2.3 Transform every ternary (or higher *n*-ary) relationship into a relationship relation.
3. Normalization of relations
   3.1 Derive the primary FDs from the EER diagram.
   3.2 Examine all the candidate relations for MVDs and secondary FDs.
   3.3 Normalize all candidate relations to the highest degree desired, eliminating any redundancies that occur in the normalized relations.
4. Usage refinement (logical/physical design boundary)
   4.1 Select dominant processes.
   4.2 Define join relations.
   4.3 Evaluate total cost for storage, query, and update.
   4.4 Consider the possible effects of denormalization.

REFERENCES

1. P. Bertaina, A. Dileva, and P. Giolito, Logical design in CODASYL and relational environments, in *Methodology and Tools for Data Base Design* (S. Ceri, ed.), North-Holland, Amsterdam, 1983, pp. 85–117.
2. P. P. Chen, The entity-relationship model—Toward a unified view of data, *ACM Trans. Database System* 1(1):9–36 (March 1976).
3. C. J. Date, *An Introduction to Database Systems*, Vol. 1 (4th ed.), Addison-Wesley, Reading, MA, 1986.

4. I. Hawryszkiewycz, *Database Analysis and Design*, SRA, Chicago, 1984.
5. D. Howe, *Data Analysis and Data Base Design*, Arnold, London, 1983.
6. W. H. Inmon, Optimizing performance with denormalization, *Database Programming and Design* 1(1):34–39 (December 1987).
7. W. Kent, A simple guide to five normal forms in relational database theory, *Comm. ACM* 26(2):120–125 (February 1983).
8. D. Maier, *Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983.
9. M. Schkolnick and P. Sorenson, DENORMALIZATION: A performance oriented database design technique, in *Proceedings of the AICA* 1980 *Congress*, Bologna, Italy, AICA, Brussels, 1980, pp. 363–377.
10. T. Teorey and J. Fry, *Design of Database Structures*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
11. T. J. Teorey, D. Yang, and J. P. Fry, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Computing Surveys* 18(2):197–222 (June 1986).
12. M. M. Zloof, Query by example, *Proc. Natl. Computer Conf.*, Vol. 44, AFIPS Press, 1975, pp. 431–438.

Received 16 March 1988; accepted 30 September 1988