Copyright © 1974, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

ACCESS CONTROL IN A RELATIONAL DATA BASE MANAGEMENT SYSTEM BY QUERY MODIFICATION

by

Michael Stonebraker and Eugene Wong

Memorandum No. ERL-M438

14 May, 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ACCESS CONTROL IN A RELATIONAL DATA BASE MANAGEMENT SYSTEM BY QUERY MODIFICATION

by

Michael Stonebraker and Eugene Wong
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

This work describes the access control system being implemented in INGRES (INteractive Graphics and REtrieval System). The scheme can be applied to any relational data base management system and has several advantages over other suggested schemes. These include:

a) implementation ease

b) small execution time overhead

c) powerful and flexible controls

d) conceptual simplicity

The basic idea utilized is that a user interaction with the data base is modified to an alternate form which is guaranteed to have no access violations. This modification takes place in a high level interaction language. Hence, the processing of a resulting interaction can be accomplished with no further regard for protection. In particular, any procedure calls in the access paths for control purposes, such as in [1,2], are avoided.

Research sponsored by the Naval Electronic Systems Command, Grant N00039-71-C-0255.

I. INTRODUCTION

The need for access control in a data base management system has been discussed at length elsewhere [1,3,4,5] and will not be further considered here. What is clearly required is an access control system which is flexible and powerful, yet simultaneously does not require undue execution time overhead. Moreover, desirable characteristics of a system include implementation ease, simplicity, and small overhead for storage of access control information.

This paper describes a system with all above characteristics which is being implemented on a relational data base management system under construction at Berkeley [6]. It has some points in common with the access control system indicated in MacAIMS [7,8]. However, as will be noted in the sequel, it has significant conceptual and implementation advantages.

The basic notion pursued is one of interaction modification. Users interact with a data base through a high level query language. Any such interaction is immediately modified into an interaction which contains no violations. The resulting interaction is compiled into a sequence of simpler interactions which are executed without further concern for access control. In particular, no controls on access paths to data need be present.

In order to discuss the mechanism involved, we must give an indication of our query language. This language, QUEL, is discussed in the next section. Subsequent sections indicate the access control mechanisms appropriate to aggregate free interactions and to interactions containing aggregates. In section V, the efficiency of the access control mechanism is considered. Lastly, section VI draws conclusions and comparisons to other access control schemes.

II. QUERY LANGUAGE

In this and subsequent sections, we discuss access control only for RETRIEVE operations. Also contained in the query language, QUEL (QUEry Language), are REPLACE, DELETE, and COMBINE[1] statements for which access control is handled analogously [6]. This section gives the flavor of RETRIEVE statements in QUEL by doing several examples.

QUEL has points in common with Data Language/ALPHA [9], SQUARE [10] and SEQUEL [11] in that it is a complete [12] query language which frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data. As such, it facilitates a considerable degree of data independence [13]. Since basic entities in QUEL are relations, we define them and indicate the sample relations which will be used in the examples in this paper.

Given sets $D_1,\ldots,D_n$ (not necessarily distinct) a relation $R(D_1,\ldots D_n)$ is a subset of the Cartesian product $D_1 x D_2 x \ldots x D_n$. In other words, R is a collection of n-tuples $x = (x_1,\ldots,x_n)$ where $x_i \in D_i$ for $1 \leq i \leq n$. The sets $D_1,\ldots,D_n$ are called domains of R and R has degree n. The only restriction put on relations in QUEL is that they be normalized [14]. Hence, every domain must be simple i.e., it cannot have members which are themselves relations.

Clearly, R can be thought of as a table with elements of R

[1]The latter command is a generalization of the traditional insert operation. The other commands have the obvious interpretation.

appearing as rows and with columns labeled by domain names as illustrated by the following example.

	NAME	DEPT	SALARY	MANAGER
EMPLOYEE	Smith	toy	10,000	Jones
	Jones	toy	15,000	Johnson
	Adams	candy	12,000	Baker
	Evans	candy	14,000	Todd
	Baker	admin	20,000	Harding
	Harding	admin	40,000	none

The above indicates an EMPLOYEE relation with domains NAME, DEPT, SALARY and MANAGER. Each employee has a manager (except for Harding who is presumably the company president), a salary and is in a department.

Each column in a tabular representation for R can be thought of as a function mapping R into D_i. These functions will be called attributes. An attribute will not be separately designated but will be identified by the domain defining it.

The second relation utilized will be a DEPARTMENT relation as follows.

	DEPT	FLOOR #	# EMP	SALES
DEPARTMENT	toy	B	10	1,000
	candy	1	5	2,000
	tire	1	16	1,500
	admin	4	10	0
	complaints	2	3	0

A QUEL query includes at least one RANGE statement of the form:

```
RANGE   Relation Name (Symbol,...,Symbol):
        Relation Name (Symbol,...,Symbol):
                  •
                  •
                  •
        Relation Name (Symbol,...,Symbol)
```

The symbols declared in the RANGE statement are variables which will be used as arguments for attributes. These are called <u>tuple variables</u>. The purpose of the statement is to specify the relation over which each variable ranges. Moreover, a query includes one (or more) RETRIEVE statements of the form:

RETRIEVE: Workspace Name: Target List: Qualification

The following suggest valid QUEL statements.

<u>Example 2.1</u>

Find the salary of the employee Jones.

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.SALARY:X.NAME= JONES
```

Here, X is a tuple variable ranging over the EMPLOYEE relation and all tuples in that relation are found which satisfy the qualification X.NAME= JONES. The salary attribute for all qualifying tuples is put in a workspace named W.

Queries can involve more than one relation as the following example indicates,

<u>Example 2.2</u>

Find the employees who work on the first floor.

```
RANGE EMPLOYEE(X):DEPARTMENT(Y)
RETRIEVE W:X.NAME:(X.DEPT=Y.DEPT)∧(Y.FLOOR#=1)
```

Here the qualification consists of two clauses connected by 'AND'. The two relations involved are connected by the clause equating DEPT. The meaning of the query is to find all tuples in the EMPLOYEE relation which have a DEPT attribute equal to the DEPT attribute of some tuple in the DEPARTMENT relation which has a FLOOR # of 1. For such qualifying tuples the NAME attribute is returned to W. More than one tuple variable can range over the same relation as follows.

Example 2.3

Find those employees who earn more than their managers.

```
RANGE EMPLOYEE(X,Y)
RETRIEVE W:X.NAME:(X.MANAGER=Y.NAME)∧(X.SALARY>Y.SALARY)
```

One must find the manager of each employee, compare the salaries of the two individuals, and return those employees meeting the conditions specified.

Example 2.4

Find the employees who make more than the average salary of all employees.

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.NAME:X.SALARY>AVE(X.SALARY)
```

Here, AVE(X.SALARY) is an aggregate and indicates the average value of the salary field over all tuples in the EMPLOYEE relation. Valid aggregation operators include AVE, COUNT, MAX, MIN and SUM.

Aggregates can themselves be qualified as the next example indicates

Example 2.5

Find all employees whose salary is smaller than the average toy department salary.

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.EMP:X.SALARY<AVE(X.SALARY;X.DEPT= TOY)
```

Here the aggregate is to be taken only over those tuples which satisfy (X.DEPT= TOY). Aggregates will be qualified or unqualified according to whether their range is restricted or not.

For a complete description of QUEL the reader is referred to [6].

III. ACCESS CONTROL FOR AGGREGATE-FREE INTERACTIONS

The mechanism for RETRIEVE operations will be indicated. However, for COMBINE, DELETE and REPLACE the scheme is analogous. We illustrate the basic approach by an example. Suppose the following access restrictions are placed on user Smith.

<u>Example 3.1</u>

Smith can see only information on himself. This statement specifies a subrelation of the employee relation which can be defined by the following QUEL statement:

```
RANGE EMPLOYEE(X)
RETRIEVE W1:X.NAME,X.DEPT,X.SALARY,X.MANAGER:X.NAME= SMITH
```

Suppose Smith issues a RETRIEVE operation involving the EMPLOYEE relation

<u>Example 3.2</u>

Find the salary of Jones.

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.SALARY:(X.NAME=JONES)
```

The above statements will automatically be modified into:

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.SALARY:(X.NAME=JONES)∧(X.NAME=SMITH)
```

Here Smith's query has been intersected with the portion of the EMPLOYEE relation to which he is allowed access as indicated in Figure 1.

The algorithm which controls this query modification is the following. For each user, U, a set of <u>access control interactions</u>, $I = \{I_1,\ldots I_k\}$ is stored.[2] Each I_j is logically of the form

[2]These interactions are inserted modified and deleted by a data base administrator or by someone whom he designates.

RANGE Relation Name (Symbol,...,Symbol):...:Relation Name (Symbol,...,Symbol)

$$\left\{\begin{array}{l}\text{RETRIEVE}\\ \text{COMBINE}\\ \text{DELETE}\\ \text{REPLACE}\end{array}\right\} : \text{Target List : Qualification}$$

The target list contains attributes from a <u>single</u> relation or the key word 'ALL'. The qualification portion is any valid QUEL qualification containing any number of tuple variables or the key word 'NO ACCESS'.

For each tuple variable, X, appearing in a given user interaction, R, the following algorithm is executed.

<u>Access Control Algorithm</u>

1. If RELATION, the relation which X references, is a workspace, then exit.
2. Find all attributes in the target list or the unmodified qualification statement of R referenced with X. Call this set S.
3. Find all access control interactions with the same command as R and with a target list containing all attributes in S. Denote these by $I_R = \{I_1^*,\ldots,I_j^*\}$ and their qualifications by $Q_1^*,\ldots,Q_j^*$.
4. Replace Q, the qualification in R, by $Q \wedge (Q_1^* \vee Q_2^* \vee \ldots \vee Q_j^*)$

The resulting QUEL statement can be executed normally. Moreover, the resulting query has the following properties.

i) each tuple variable is restricted to a subrelation to which the user has access.

ii) each tuple variable can range over all subrelations to which the user has access.

iii) the workspace in which the result appears contains only information which the user is allowed access to. Hence he can have unrestricted access to it.

iv) the tuple variables in an access control interaction only indicate range. The name of the variable in the appropriate target list of an access control interaction is changed to X. Others which appear are given new unique names and appropriate range statements.

A modification to the above algorithm is made for efficiency purposes. It is expected that all interactions with the same command will have the following nesting property.

A set $I = \{I_1,\ldots,I_q\}$ of interactions is <u>properly nested</u> if for any two interactions I_m, $I_n \varepsilon I$ with target lists T_m, T_n such that $T_m \supset T_n$, then $W_n \supset W_m$ as follows.

RETRIEVE W_n:T_n:Q_n

RETRIEVE W_m:T_n:Q_m

The larger the set of attributes which are specified in a target list, the more restrictive is the allowed access, as indicated in Figure 2.

We assume that all protection interactions for a given user with the same command will be properly nested and can therefore safely add step 3A to the algorithm.

3A. Delete from I_R all interactions with a target list containing the target list of another interaction in I_R.

We now present an example of the algorithm at work. The restrictions are intended more to demonstrate the possibilities than to appear realistic.

Example 3.3

Jones can see all salaries as long as his query does not include name or department in the target list or qualification. Moreover, except for employee Baker, he can see names, departments and managers if salary is not present. Furthermore, he can see names, managers and salaries for all employees who earn more than their managers. Lastly, he can see all attributes for departments which sell more than the average department.

These restrictions may be stated in four access control interactions.

```
     RANGE EMPLOYEE(X,Y):DEPARTMENT(Z)
  1) RETRIEVE X.SALARY
  2) RETRIEVE X.NAME,X.DEPT,X.MANAGER:X.NAME≠BAKER
  3) RETRIEVE X.NAME,X.SALARY,X.MANAGER:Y.NAME=X.MANAGER∧X.SALARY
                                               >Y.SALARY
  4) RETRIEVE ALL:Z.SALES>AVE(Z.SALES)
```

Example 3.4

Suppose Jones issues the query: Find all salaries

```
RANGE EMPLOYEE(X)
RETRIEVE W:  X.SALARY
```

The algorithm finds S = {SALARY} and 1) and 3) contain S. However, step 3A eliminates 3); hence, the resulting query is unchanged from the original since 1) has no qualification statement.

Example 3.5

Now Jones issues the query: Find the manager of employee Adams.

```
RANGE EMPLOYEE(X)
RETRIEVE W: X.MANAGER:X.NAME=ADAMS
```

Here, S = {MANAGER,NAME} and the algorithm selects 2) and 3) as applicable. Hence, the query is modified to:

```
RANGE EMPLOYEE(X,Y)
RETRIEVE W:X.MANAGER:X.NAME=ADAMS∧(X.NAME≠BAKER∨(Y.NAME
                                  =X.MANAGER∧X.SALARY>Y.SALARY))
```

Example 3.6

Suppose Jones then issues the query: Find all employees who earn more than their managers.

```
RANGE EMPLOYEE(X,Y)
RETRIEVE W:X.NAME:(X.MANAGER=Y.NAME)∧(Y.SALARY<X.SALARY)
```

Here, S_x = {MANAGER,NAME, SALARY}, S_y = {NAME,SALARY}. As a result, 3) is added for both X and Y, yielding

```
RANGE EMPLOYEE(X,Y,Z,U)
RETRIEVE W:X.NAME:(X.MANAGER=Y.NAME)∧(Y.SALARY<X.SALARY)
                  ∧(Z.NAME=Y.MANAGER)∧(Y.SALARY>Z.SALARY)
                  ∧(U.NAME=X.MANAGER)∧(X.SALARY>U.SALARY)
```

The reader can observe that the effect of the modified query is to retrieve those employees who make more than their managers and who have managers who make more than their managers. This is less than what the user requested. However, he can retrieve everything he desires by the following query.

Example 3.7

```
RANGE EMPLOYEE(X)
RETRIEVE W:X.NAME,X.SALARY
```

Here, S = {NAME,SALARY} and the algorithm appends 3) yielding the desired result.

```
RANGE EMPLOYEE(X,Y)
RETRIEVE W:X.NAME,X.SALARY:Y.NAME=X.MANAGER∧X.SALARY>Y.SALARY
```

The reader can note that allowing access control interactions with multivariable target lists and modifying the access control algorithm could alleviate the too strict control placed on Example 3.6. However, such an algorithm would be quite complicated because different qualifications would be added, depending on which variables in a user query were associated with which variables in an interaction target list.

If, for example, 3) had a Y. variable in the target list, different appendages would be applied, depending on association of user variables to interaction variables. The only reasonable policy in this situation would be to append the logical "OR" of all such possibilities. Efficiency and complexity considerations suggest that the added generality is not worthwhile.

IV. PROTECTION FOR INTERACTIONS CONTAINING AGGREGATES

When users issue interactions containing aggregates, an additional policy consideration appears, as illustrated below.

Example 4.1

Suppose Adams can retrieve only salaries for employees in the toy department and has an access control interaction, as follows.

```
RANGE EMPLOYEE(X)
RETRIEVE X.SALARY:X.DEPARTMENT=TOY
```

Moreover, suppose he issues the following query.

Example 4.2

Find the average company salary.

```
RANGE EMPLOYEE(X)
RETRIEVE W: AVE(X.SALARY)
```

Two policies can be taken in this situation.

1) The access control clause can be added inside the aggregate,

producing

```
RANGE EMPLOYEE(X)
RETRIEVE W:AVE(X.SALARY;X.DEPARTMENT=TOY)
```

Adams would then retrieve the average salary of employees in the toy department (which is probably not what he wants).

2) The retrieval can be allowed unprotected since the user is obtaining a statistical quantity on a relation of presumably reasonable size.

In general, the following mechanisms can be potentially enforced concerning aggregates.

1) allow aggregates without restriction
2) allow aggregates without restriction if the minimum number of values aggregated exceeds some threshold
3) allow aggregates without restriction if they are unqualified (i.e. are aggregates over a whole relation)
4) allow aggregates only with access control qualifications appended inside the function.

It can be readily shown that mechanisms 1) and 2) allow the persistent user information from specific tuples to which he may not be permitted access. The following query is representative of potentially unauthorized access using aggregates allowed under mechanism 1).

<u>Example 4.3</u>

Find Smith's salary

```
RANGE EMPLOYEE(X)
RETRIEVE W:AVE(X.SALARY;X.NAME=SMITH)
```

With mechanism 2), the following potentially unauthorized access can take place.

Example 4.4

```
RANGE EMPLOYEE(X)
RETRIEVE W:COUNT(X.NAME≥EVANS)
RETRIEVE W:AVE(X.SALARY;X.NAME≥EVANS)
RETRIEVE W:AVE(X.SALARY;X.NAME>EVANS)
```

Let A_1, A_2 and A_3, respectively, be the responses to the above RETRIEVE statements. It is easily computed that Evans' salary is

$$A_1 * A_2 - (A_1 - 1) * A_3$$

Because these potential violations can occur using mechanisms 1 and 2, and because preventing such violations requires some sort of complex monitoring of the interaction stream, we are allowing the choice of mechanisms 3) and 4) for each possible aggregate operator. We are aware, however, of the following anomaly concerning mechanism 3).

Example 4.5

Suppose Adams is allowed to access only tuples of those employees in the toy department. Suppose also that he issues the queries:

```
RANGE EMPLOYEE(X)
RETRIEVE W:AVE(X.SALARY)
RETRIEVE W:AVE(X.SALARY;X.NAME>AAAAA)
```

The first query would be answered with the true average salary of all employees since it is an unqualified aggregate. On the other hand, the second query is qualified and therefore would be further qualified to the following:

```
RANGE EMPLOYEE(X)
RETRIEVE W:AVE(X.SALARY;X.NAME>AAAAA∧X.DEPARTMENT=TOY)
```

Therefore, the average salary in the toy department is returned.

Hence, two logically identical queries will yield different answers. This price must be paid for the increased flexibility allowed by mechanism 3) above mechanism 4).

The reader can now note the obvious extensions which must be made to the algorithm of the previous section in order to properly enforce access control for interactions containing aggregates.

V. EFFICIENCY CONSIDERATIONS

Efficiency considerations can only be discussed in view of the search strategy employed to decompose QUEL interactions. The search strategy used resembles that in [15]. Basically, it breaks a query into a sequence of queries each involving only a single tuple variable. These single variable queries involve only a single relation and can be directly executed (in the worst case by a sequential scan of a relation tuple by tuple). Often the relation will be stored in such a way that a complete scan is not needed. Also, any redundant indices which can be used profitably to speed access are utilized.

Usually, the addition of access control qualification will result in a decomposition to the same sequence of one-variable interactions that would occur without access control. Each such interaction is further qualified by one or more protection statements than would otherwise be the case. Such a one-variable interaction will usually be at least as efficient to process as one without protection. In fact, these added protection clauses may allow redundant indices to be employed to speed access that would not otherwise be usable. Hence the cost of protection is usually negligible. Adhering to the following two conditions essentially insures this statement.

1. The following single variable condition holds: A set of interactions, K, with target lists $T = \{T_1,\ldots,T_j\}$ has the

single variable condition if for all T_m, $T_n \varepsilon T$ such $T_m \cap T_n \neq \phi$, $\exists$ a $T_i \in T = T_m \cap T_n$.

This condition insures the truth of the following statements:

a) I_R contains a single interaction as a result of steps 1-3A of the algorithm.

b) In step 4 a user interaction is further qualified by one qualification clause.

c) One qualification clause is appended for each variable in the query.

2. Access control qualifications do not contain two or more tuple variables or the operator "OR".

Here, inefficiency usually results if new variables must be added to the original query (as occurred in Example 3.6). In all probability, more tuple variables would require a longer sequence of one-variable queries. Also, for processing convenience, when "OR" is present in a qualification statement, two one-variable queries are issued and the set union is taken on the results. Hence, one variable queries do not contain "OR". As a result, inclusion of additional "OR" operators to the unmodified query result in a longer sequence of one variable queries. This latter inefficiency, however, is specific to our search strategy and could potentially be avoided. It is felt that little generality is lost by adherence to the two rules above, which guarantee small or negligible efficiency overhead. More general schemes are, of course, allowed but execution speed may be slowed significantly.

VI. COMPARISONS AND CONCLUSIONS

The reader should note the differences between this access control scheme and that proposed by CODASYL [2]. That proposal inserts procedure calls into access paths to check access privileges. It involves both inefficiency (checking at each call to an access method) and less flexibility (it is hard to enforce complicated control such as that in Example 3.3 in this manner). The reader can also note that our protection scheme is appended at the highest level possible and not buried deep within the system. We believe this to be the proper approach.

We also note the differences between our scheme and that of MacAIMS [7,8]. MacAIMS requires several different modes of protection (i.e. manipulate, statistical, print, none). That scheme must keep track of when access to a relation changes from one mode to another (i.e. subsetting away information the user cannot see). Our scheme handles all access in a unified manner, and no such conversions are ever made. Second, since we have a higher level query language than MacAIMS, we can enforce more complex controls (such as constraints which involve aggregation). Third, because users have unrestricted access to workspaces which they create, we avoid any necessity of dealing with the problem of truth mentioned in [7,8]. We admit that our scheme does not include checks on terminal number, time of day, etc. which appear in [7,8]. However, it is simple to include them.

Finally, the advantages of our scheme are:

- Little storage space is required to store access control interactions.

- Implementation is trivial.

- Query modification is required to support "views" [6,16]. Hence, access control modification can be done at the same time. The query need only be modified once.

- Success of this scheme does not require any complex operating system or hardware protection support (such as in [17,18]. Hence, it can be widely applied.

- There is small or negligible overhead in most cases.

REFERENCES

1. Hoffman, L., "The Formulary Model for Access Control and Privacy," Stanford Linear Accelerator Center Report #117, May, 1970.

2. "CODASYL Data Description Language," NBS Handbook #113, U.S. Dept. of Commerce, January, 1974.

3. Browne, P. and Steinauer, D., "A Model for Access Control," Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Calif., November, 1971.

4. Weissman, C., "Security Controls in the ADEPT-50 Time Sharing System," Proc. 1969 Fall Joint Computer Conference, November, 1969.

5. Friedman, T., "The Authorization Problem in Shared Files," IBM Systems Journal, No. 4, 1970.

6. McDonald, N., Stonebraker, M., and Wong, E., "Preliminary Specification of INGRES," Electronics Research Laboratory Report #470, University of California, Berkeley, California, May, 1974.

7. Owens, R., "Evaluation of Access Authorization Characteristics of Derived Data Sets," Proc. 1971 SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, November, 1971.

8. Owens, R., "Primary Access Control in Large Scale Time-Shared Decision Systems," Project MAC Report TR-89, M.I.T., Cambridge, Mass., July, 1971.

9. Codd, E., "A Data Base Sublanguage Founded on the Relational Calculus," Proc. 1971 SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, November, 1971.

10. Boyce, R., et al., "Specifying Queries as Relational Expressions: SQUARE," IBM Technical Report RJ1291, IBM Research Laboratory, San Jose, California, October, 1973.

11. Chamberlin, D. and Boyce, R., "SEQUEL: A Structured English Query Language," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Michigan, May, 1974.

12. Codd, E., "Relational Completeness of Data Base Sublanguages," Courant Computer Science Symposium, Vol. 6, Data Base Systems, Prentice Hall, New York, May, 1971.

13. Stonebraker, M., "A Functional View of Data Independence," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

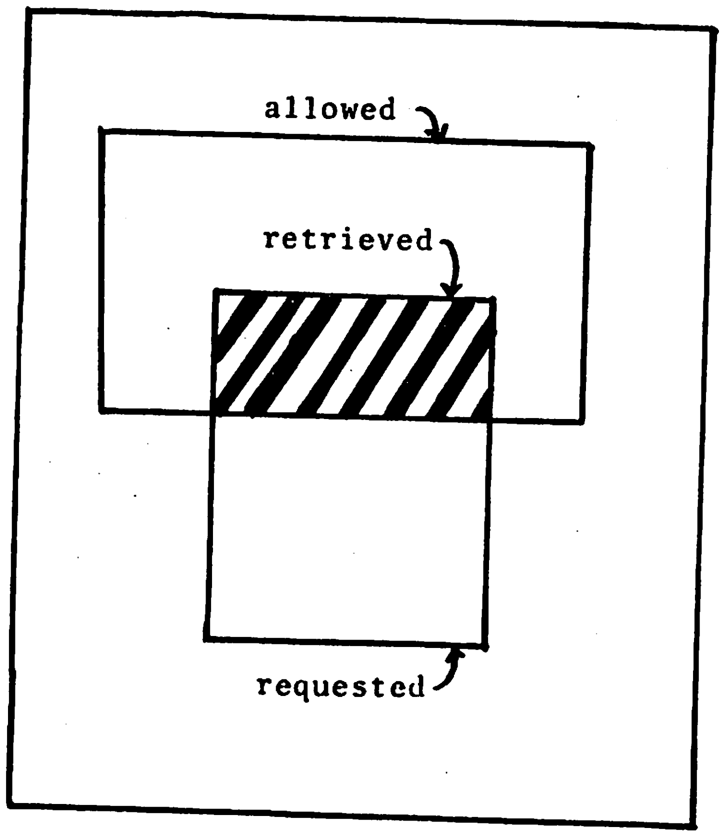
14. Codd, E., "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, No. 6, June, 1970.

15. Rothnie, J., "An Approach to Implementing a Relational Data Management System," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Michigan, May, 1974.

16. Boyce, R. and Chamberlin, D., "Using a Structured English Query Language as a Data Definition Facility," IBM Research Report No. RJ 1318, IBM Research Laboratory, San Jose, California, December, 1973.

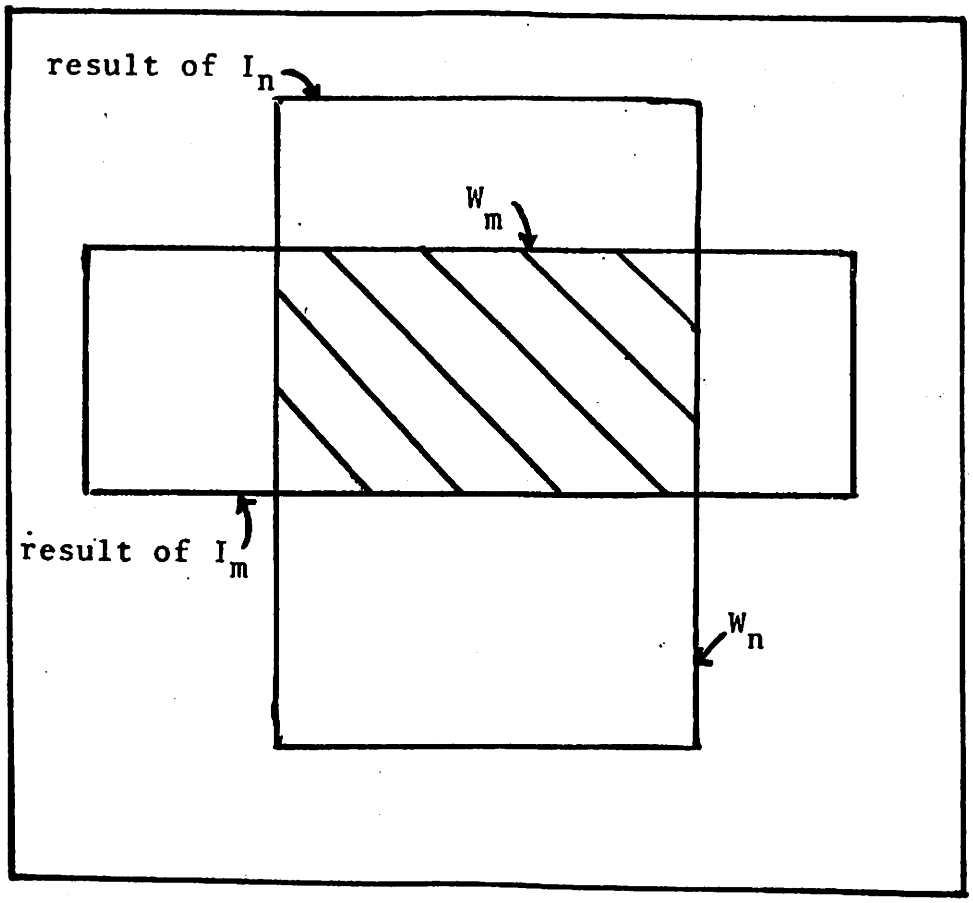
17. Graham, R., "Protection in an Information Processing Utility," CACM, Vol. 11, No. 5, May, 1968.

18. Lampson, B., "Dynamic Protection Structures," Proc. 1969 Fall Joint Computer Conference, November 1969.



The Access Control Scheme at Work

Figure 1



Two Properly Nested Interactions

Figure 2