THE INGRES PROTECTION SYSTEM

by

Michael Stonebraker and Peter Rubinstein

Memorandum No. ERL-M594

6 July 1976

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE INGRES PROTECTION SYSTEM*

by

MICHAEL STONEBRAKER and PETER RUBINSTEIN
Department of Electrical Engineering and Computer Science
and Electronics Research Laboratory
University of California, Berkeley, Ca.

This paper presents the design of a protection system being implemented for the INGRES relational data base management system. A brief description of the INGRES system and its operational environment is first presented to provide the setting for the protection scheme. Mechanisms for protecting physical data files and enforcing sophisticated access control rules for shared relations are then presented. Lastly, the important design decisions concerning protection are discussed.

I. INTRODUCTION

In [STON74a] a proposal was presented for the basis of a protection system to be used by the INGRES data base management system

*This work was sponsored by National Science Foundation Grant DCR75-03839 and ENG74-06651-A01.

[HELD75,ZOOK76,STON76]. In this paper we present the design and implementation of a complete protection system. This includes an explanation of the design and policy decisions made, including

a) requiring the notion of a data base administrator

b) protecting physical relations rather than "views" [STON75, CHAM75]

c) not reporting protection violations to the DBA.

d) not validating users

e) supporting a single data sublanguage statement as the "atom" of protection.

f) handling protection at run time rather than at compile time

g) not supporting user supplied protection routines (such as formularies [HOFF70]).

A variant of the scheme suggested in [STON74a] for protection of shared relations and mechanisms for physical protection of files are also described. Moreover, a protection role for a specific user, called a Data Base Administrator, DBA, is identified and his power explained. The remainder of this paper is organized as follows. Section II briefly discusses INGRES and the environment created by UNIX [RITC74], the operating system on which it is operational. In Section III we indicate the nature of physical protection used for files containing INGRES data. Then, in Section IV the design decisions made are explained. The language by which the DBA can protect shared relations is specified in Sec-

tion V. The algorithm which enforces these protection statements and its implementation are sketched in Section VI. Lastly, in Section VII, we comment on the compatibility of the approach to protection taken and the present version of INGRES.

II The INGRES DATA SUBLANGUAGE

The user interface for INGRES consists of the data sublanguage, QUEL, and assorted utility commands. QUEL (QUEry Language) has points in common with Data Language/ALPHA [CODD71], SQUARE [BOYC73] and SEQUEL [CHAM74] in that it is a complete [CODD72] 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 [STON74b].

The QUEL examples in this section concern the following relations.

```
EMPLOYEE (NAME, DEPT, SALARY, MANAGER, AGE)
DEPT     (DEPT, FLOOR#)
```

These relations indicate, respectively, information about employees and departments in a company.

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

```
RANGE OF variable-list IS relation-name
```

The symbols declared in the range statement are called TUPLE VARIABLES. The purpose of this statement is to specify the rela-

tion over which each variable ranges.

Moreover, an interaction includes one or more statements of the form:

```
Command [Result-name] ( Target-list )
        [ WHERE Qualification ]
```

Here, Command is either RETRIEVE, APPEND, REPLACE, or DELETE. For RETRIEVE and APPEND, Result-name is the name of the relation which qualifying tuples will be retrieved into or appended to. For REPLACE and DELETE, Result-name is the name of a tuple variable which identifies the affected relation. The Target-list is a list of the form

```
Result-domain = Function,...
```

Here, the Result-domain's are domain names in the result relation which are to be assigned the value of the corresponding function.

The following three examples illustrate the language. A complete description can be found in [HELD75].

Example 2.1 Find the birth year of employee Jones

```
RANGE OF E IS EMPLOYEE
RETRIEVE INTO W (BYEAR = 1976 - E.AGE)
WHERE E.NAME = "Jones"
```

Here, E is a tuple variable which ranges over the EMPLOYEE relation and all tuples in that relation are found which satisfy the qualification E.NAME = "Jones". The result of the query is a new relation, W, which has a single domain, BYEAR, that has been calculated for each qualifying tuple. If the result relation is omitted, qualifying tuples are either written on the user's terminal or returned to a calling program in a prescribed format.

Example 2.2 Fire everybody on the first floor

```
RANGE OF E IS EMPLOYEE
RANGE OF D IS DEPT
DELETE E WHERE   E.DEPT = D.DEPT
             AND  D.FLOOR# = 1
```

Here E specifies that the EMPLOYEE relation is to be modified. All tuples are to be removed which have a value for DEPT which is the same as that of some department on the first floor.

Example 2.3 Give a 10 percent raise to Jones if he works on the first floor

```
RANGE OF E IS EMPLOYEE
RANGE of D IS DEPT
REPLACE E(SALARY=1.1 * E.SALARY)
WHERE E.NAME = "Jones" AND
      E.DEPT = D.DEPT AND D.FLOOR# = 1
```

Here, E.SALARY is to be replaced by 1.1*E.SALARY for those tuples in EMPLOYEE for which the qualification is true.

In addition to the above QUEL commands INGRES also supports a variety of utility commands. The only ones of interest in a protection context are COPY, PRINT and CREATE. The COPY command transfers a relation to or from a UNIX file . It is a "bulk transfer" mechanism and therefore must be protected. PRINT is a simple report generator that writes a relation onto a user's terminal. CREATE sets up an empty relation and prepares it for use.

III PROTECTION LANGUAGE

INGRES manages a collection of data bases, each of which is made up of a set of relations. Each data base is associated with a special user called the Data Base Administrator (DBA). A list of

allowable DBA's is kept by INGRES which may only be changed by a user who can log on as INGRES. This user will be referred to as the INGRES "super-user".

Only a data base's DBA may create shared relations in that data-base. Relations created by other users are guaranteed private by INGRES. Each data base contains certain relations which are "system catalogs". These relations may only be updated by INGRES utility routines. Such relations are owned by the DBA, however, and he may grant RETRIEVE permission to portions of the catalogs if he wishes.

INGRES makes available to the DBA the following command to specify access permissions for shared relations.

PERMIT relname TO object FOR command (tarqlist;
idlist) WHERE qualification

The fields of this command have the following meanings:

RELNAME: specifies the name of the relation to be protected.

OBJECT: specifies the object to be controlled. This may be a user, a teletype, or the keyword ALL, which will cause the restriction to apply to all users.

COMMAND: specified which type of access is to be allowed. The types of access allowed are RETRIEVE, APPEND, DESTROY, REPLACE, DELETE, COPY, and PRINT.

TARGLIST: A list of domains of the form tuple-variable.domain which may be accessed.

IDLIST: A list of domains of the form tuple-variable.domain. which can be accessed but not updated. These domains may

appear in a qualification clause of an update, but may not appear in the target list. This field is optional, along with the semicolon which precedes it. However, only one tuple variable can be present in the TARGETLIST and IDLIST.

QUALIFICATION: Any valid QUEL qualification containing any number of tuple variables. It specifies the subset of the relation which may be accessed by an interaction of the type specified by the command. Three extensions to allowable qualifications are supported. First, using the keyword CLOCKTTT, the DBA may specify a range of times during which the protection data is to be used (i.e. clockttt>800 and clockttt<1500). Lastly, # may appear in the qualification to specify the UNIX logon name of the user currently invoking INGRES.

The data base administrator can enter an arbitrary number of protection statements governing access to his relations. Each protection statement is given a unique id and stored in the PROTECTION relation. This relation has domains of relation name, command, object, targetlist, idlist, qualification, protectionid. This relation is normally compressed and hashed on the first two domains. Removal of protection statements is accomplished by the DBA using the utiltiy command DENY (protection-id).

Relations owned by other users are guaranteed private by INGRES. Therefore only the DBA is permitted the use of the PERMIT command. Since system catalogs contain data about the data base, their integrity must be carefully guarded. Consequently, no user (including the DBA) is allowed to update system catalogs using

QUEL. All modifications to catalogs are made only by INGRES in response to INGRES commands concerning other relations. However, RETRIEVE permission to portions of catalogs may be granted to others by the DBA.

IV Algorithm and Implementation.

The implementation of access control has two major facets. The first is the representation of information in the PROTECTION relation. If data were stored as a text string, the string would have to be completely parsed for each interaction to which it applies. The overhead of this strategy is considered unreasonable.

Consequently a protection statement will be parsed when entered and stored as a tree structure in the PROTECTION relation. When an interaction is to be modified, the qualification trees of the protection interactions need only be attached to the qualification tree of the interaction. The id and target lists are stored as lists of domain numbers which are the internal representation for domain names.

The second facet of the implementation is the enforcing of the protection interactions. This is done by essentially the same algorithm as that presented in [STON74a].

1 While an interaction is being parsed, two sets of domains are constructed for each tuple variable used. One consists of domains in the target list, the other of domains in the qualification.

2 For a given tuple variable the PROTECTION relation is searched for all tuples with the correct user and relation name whose id and target lists contain the lists of the interaction, and whose command type and object match those of the interaction. If no matching tuples are found, the proposed query is aborted due to lack of permission. This enforces a default to denial policy. Should the DBA forget to issue the proper PERMIT statements all access is denied.

3 The resulting tuples are then compared. Any tuples whose target and id lists contain those of another tuple are removed from further consideration. This may be done safely since the protection interactions are assumed to be properly nested [STON74a]. This ensures that the least restrictive protection interactions are applied to the interaction.

4 Repeat steps 2 and 3 for the terminal from which the interaction originates.

5 The qualifications of the resulting tuples are disjoined and the result is conjoined with the qualification of the original query.

6 This entire procedure is repeated for each relation referenced by a tuple variable in the interaction.

7 Execute the modified interaction normally.

The following example illustrates the algorithm at work.

Example 5.1 Each manager can read salaries of employees who work for him.

```
Protection interaction
   RANGE of E is EMPLOYEE
   PERMIT ALL TO EMPLOYEE FOR RETRIEVE
   (E.SALARY; E.NAME)
   WHERE E.MANAGER = *

An interaction by Jones
   RETRIEVE (E. SALARY) WHERE E. NAME="Smith"
would be modified to
   RETRIEVE (E.SALARY) WHERE E. NAME="Smith"
                AND E. MANAGER= "Jones"
```

V PROTECTION OF PHYSICAL FILES

In order to enforce the protection specifications, INGRES must guarantee that users cannot subvert the system by machinations outside of INGRES. To do so INGRES must make use of operating system features to physically protect data that it manages.

To present the strategy used by INGRES, we must first explore a few details of the UNIX environment. A process in UNIX is an address space (64K bytes or less on a PDP-11/40 and 128K bytes on 11/45's and 11/70's) which is associated with a user-id and is the unit of work scheduled by the UNIX scheduler. Each valid user-id is associated with a logon name and optionally a password. Processes may "fork" subprocesses; consequently, a parent process can be the root of a process subtree. Furthermore, a process can request that UNIX execute a file in a descendent process. UNIX provides an interprocess communication facility for such processes.

UNIX supports a tree structured file system similar to that of MULTICS. Each file is either a directory (containing references

to descendent files in the file system) or a data file. It can be granted by its owner (the user who created it) any combination of the following protection clauses:

a) owner read

b) owner write

c) non owner read

d) non owner write

e) execute

f) special execute

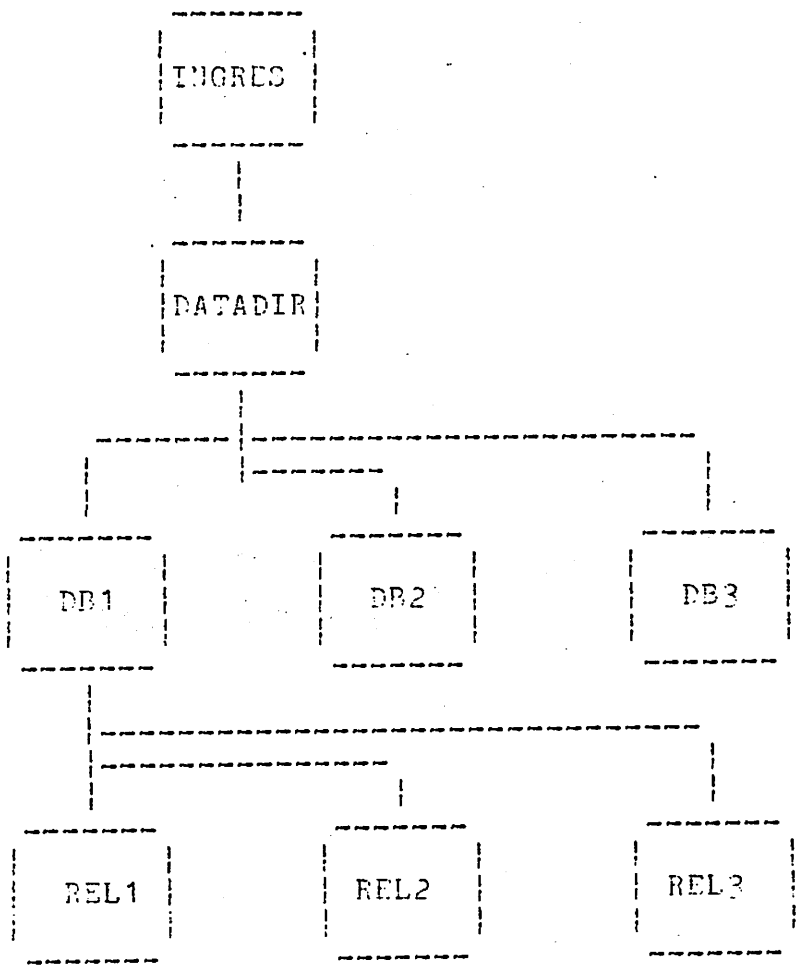
All clauses have an obvious meaning except f). In this case the file can be executed and during execution the user-id of the process in which it runs is changed to that of the owner of the file. Upon termination the user-id is returned to its previous value.

Any file can be read (written) by a process only if the user-id of the process has read (write) permission for that file. Moreover, each process has a "current directory" in the file system. To locate a file a process must specify its relative path from the current directory. In order to read (write) a file the process must also have read (write) permission for all intermediate directories along the path. A file may also be referenced by specifying a complete path from the root of the file system. In this case the above rules hold for all nodes along the path. A process may change its current directory, but only to one for which it has read access according to the above rules.

When the INGRES system is installed at a site, a special user

(with a given user-id) must be created whose logon name is INGRES. INGRES system generation includes creating a variety of directories and data files in a subtree of the file system whose root is INGRES.

Each data base has its own directory. Note that they are owned by the DBA. These directories, as shown in Figure 1 are resident in a directory of databases called DATADIR, which is owned by the INGRES super-user and only he may read or write on them. Each database contains an entry for each relation in the database. Each relation is represented by a file. These files are also owned by the super-user, but he is not permitted to tamper with them. This is ensured because the database directory is owned by the DBA and the super-user may not write on it. Care must be taken that the super-user is not also a DBA, at least with the same userid, for obvious reasons.



DBx = database x RELx = relation x
A Portion of the INGRES File System

Figure 1

We turn now to the mechanism by which a user initiates the INGRES system. When INGRES is invoked, a collection of UNIX processes is created and appropriate interprocess communication set up. The program which accomplishes this is stored in a file which is executed by the invoking user. This file has protection status "special execute, no other access" and belongs to the super-user.

The invoker of INGRES must also specify a data base which provides the context for any further interactions. The program

"forks" the processes which contain INGRES object code and has UNIX execute them with the user-id of the invoking user and changes the current directory to the appropriate database directory. Then the INGRES processes change their userid back to that of the invoking user.

The DBA may set the protection status for files containing shared relations (using UNIX commands). Only two modes make sense:

a) the file containing the relation may be set to "owner read, owner write, no other access". In this case the data base administrator allows only users with his user-id to access the file. This "non-shared" mode is appropriate for "system catalogs" used for maintenance of the data base.

b) the file containing the relation may be set to "owner read, owner write, nor owner read, non owner write". In this case the data base administrator allows other users to share this relation under the restrictions discussed in Section III.

A relation with this protection status cannot be tampered with by anyone except through the use of INGRES object code. The INGRES super-user cannot access it because he has neither read nor write permission. The DBA cannot access it because he does not have read or write privileges to the higher level directory, DATADIR. Other users are similarly constrained. Hence, the only mechanism available to interact with such files is by executing the INGRES object code. The INGRES object code may access the file if the DBA is the invoking user, since INGRES sets the user's id to

INGRES but can still supply the DBA's id when needed

A relation with this protection status can be accessed by all users, but only by invoking INGRES because the higher level directories only permit INGRES to read them. However, the INGRES super-user is not excluded from tampering with data files using other programs (such as the UNIX editor) because he has read and write permission for the file and all higher level directories. Hence, this shared status requires that the data base administrator trust the INGRES superuser.

Note that in both cases the data base administrator cannot tamper with his relations other than by using INGRES. Hence, his powers stem entirely from the INGRES implementation philosophy.

VI DISCUSSION OF APPROACH TAKEN

6.1 Centralized Data Base Administrator

The DBA has the following powers not available to ordinary users:

1) the ability to create shared relations and to specify access control for them

2) the ability to destroy any relations in his data base (except the system catalogs)

This system allows "one level sharing" in that only the DBA has the above powers and he cannot delegate them to others (as in the file systems of most time-sharing systems). This strategy was implemented for three reasons:

1) Additional generality would have created considerable

problems, such as making revocation of access privileges non-trivial. The benefits of additional generality were not perceived as sufficient to warrant the problems created.

2) It seems appropriate to entrust to the DBA the duty (and power) to resolve the policy decision which must be made when space is exhausted and some relations must be destroyed (or archived). This policy decision becomes much harder (or impossible) if a data base is not in the control of one user.

3) Someone must be entrusted with the policy decision concerning which relations to physically store and which to define as "views". This "data base design" problem is best resolved by a centralized DBA.

6.2 Protection of Views Versus Protection of Real Relations

Algorithms for the support of views are given in [STON75]. Basically a view is a virtual relation which could be created from existing relations but which is not physically stored. Views will be allowed in INGRES to support programs written for obsolete versions of the data base and for user convenience. In the remainder of this section we distinguish the INGRES protection scheme from the view oriented one in [CHAM75] and indicate the rationale behind its use. Consider the following two views:

```
RANGE OF E IS EMPLOYEE
DEFINE RESTRICTION-1 (E.NAME, E. SALARY, E.AGE)
WHERE E.DEPT = "toy"

DEFINE RESTRICTION-2 (E.NAME, E.DEPT, E.SALARY)
WHERE E.AGE < 50
```

and the following two access control statements:

```
RANGE OF E IS EMPLOYEE
PERMIT ALL TO EMPLOYEE FOR RETRIEVE (E.NAME,  E.SALARY,
E.AGE)
WHERE E.DEPT = "toy"

PERMIT ALL TO EMPLOYEE FOR RETRIEVE (E.NAME,  E.SALARY,
E.DEPT)
WHERE E.AGE < 50
```

(See Section V for complete explanation of PERMIT command.)

Access control could be based on views as suggested in [CHAM75] and a given user authorized to use views RESTRICTION-1 and RESTRICTION-2. To find the salary of Harding he might interrogate RESTRICTION-1 as follows:

```
RANGE OF R IS RESTRICTION-1
RETRIEVE (R.SALARY) WHERE
R.NAME = "Harding"
```

Failing to find Harding in RESTRICTION-1 he would have to then interrogate RESTRICTION-2. After two queries he would be returned the appropriate salary if Harding was under 50 or in the toy department. Under the INGRES scheme the user can issue

```
RANGE OF E IS EMPLOYEE
RETRIEVE (E.SALARY) WHERE
E.NAME = "Harding"
```

which will be modified by the access control algorithm to

```
RANGE OF E IS EMPLOYEE
RETRIEVE (E.SALARY) WHERE
E.NAME = "Harding"
    AND
(E.AGE < 50 OR E.DEPT = "toy")
```

In this system the user need not manually sequence through his views to obtain desired data but automatically obtains such data if permitted. Note clearly that the portion of EMPLOYEE to which the user has access (the union of RESTRICTION-1 and RESTRIC-

TION-2) is not a relation and hence cannot be defined as a single view.

To summarize, access control restrictions are handled automatically by the INGRES algorithm. In a view oriented scheme, a user must sequence through his views to obtain allowed information.

6.3 Action on Protection Violations

A user is simply allowed to access the portion of a relation for which he is authorized. No attempt is made to check if a security violation has taken place and no violation reporting is done. The system could be extended to detect violations and process the information in any way desired (with some additional overhead).

6.4 Validation of Users

INGRES contains no facilities to validate the authenticity of users. The designers see no reason to duplicate existing facilities in the operating system.

6.5 The Atom used for Protection

It should be noted that only single QUEL statements can be protected; there is no notion of protecting a sequence of such statements (i.e. a transaction). The discussion of the decision not to support transactions as an atom of concurrency control appears in [STON76]. Supporting them only for protection and not for concurrency appears to be unwise. Hence, no notion of a transaction exists in INGRES. Consequently, no control over inferences possible from multiple QUEL statements is provided. Some control is possible by specifying integrity constraints on the

interaction log (e.g. the number of retrieves must be less than a given number per day).

6.6 Run Time Protection

When a user is interacting with INGRES via an interactive terminal monitor which allows him to input,edit,print and execute QUEL statements, there is no notion of "compile time". Hence, query modification must take place at "run time".

A second mechanism by which interaction with INGRES can take place is EQUEL [ALLM76]. This translator embeds all of INGRES in the general purpose programming language "C". A user is thereby allowed to write programs in the combined language. It is possible to completely parse and perform query modification on QUEL statements at the time the translator is run (i.e. at compile time) and do no checking at execution time.

This is not the approach currently followed by INGRES which does all query modification at run time. This decision is discussed in [ALLM76] and basically involves the notion that the schema (or data base description) is not static in INGRES. Hence, it is possible for the relations on which a QUEL statement acts to be created between the time the translator is run and the time the actual program is executed. In this way the schema can change and make it impossible for the protection algorithm to know what clauses to add at the time the EQUEL program is translated.

It is likely that the next version of EQUEL will do query modification at translation time whenever possible and force recompilation of programs if the underlying schema changes.

6.7 User Extensions

There is an obvious desire to allow the DBA to write procedures to determine access control for situations not covered by the above algorithm. INGRES makes no provision for accepting DBA procedures for the following reason. The only efficient mechanism would be to include such procedures with the query modification routines and invoke them by a subroutine call. This would mean that such a procedure would be running in the same process as the query modification routines. If such a procedure were malicious it could "trash" all relations in the data base or never return. There is no way for INGRES to guard against this possibility. The INGRES designers have opted not to trust user supplied procedures and hence do not accept them into the INGRES nucleus.

6.8 Default Access

As noted in Section 4, INGRES defaults to "no permission" when no permission statement is specified. This policy decision causes forgetfulness on the part of the DBA to deny access.

VII COMPATIBILITY ISSUES

A protection scheme using query modification fits into the INGRES code easily. The technique of query modification is also used to implement integrity constraints [STON75], views [STON75], and to put interactions into conjunctive normal form. Because many of the routines necessary for query modification already exist, the implementation of the protection scheme described in this paper will be greatly facilitated. These factors contribute to the ease with which a protection system of the type described can be added

to the present INGRES implementation. It is expected that this code will be running by the time this paper appears.

VIII SECURITY and TRUST

The UNIX operating system supports a user (often called the "root") who has the power to authorize and deauthorize users (including INGRES). Obviously, this user must be trusted not to subvert the database system. Moreover, the UNIX operating system itself must be trusted to operate in a "leakproof" way. Efforts at building security kernels for operating systems are intended to increase such trust. In addition, trust must be placed in the database system itself. The database system must apply the extra qualifications to the queries correctly and then execute the resulting interaction accurately. Certification of the code which modifies the incoming queries appears feasible. However, certification of the resultant execution would seem to be considerably more difficult. To summarize, it is implicit in the design of the INGRES protection system that trust is to be placed in the "root" user, in the operating system, and in the valid operation of the database system.

REFERENCES

ALLM76 Allman, E., Stonebraker, M., and Held, G. "Embedding a Relational Data Sub-language in a General Purpose Programming Language", Proc. ACM SIGPLAN-SIGMOD Workshop on Data, Salt Lake City, Utah, March, 1976.

BOYC73 Boyce, R. et. al., "Specifying Queries as Relational Expressions: SQUARE", IBM Research, San Jose, Ca., RJ 1291, Oct. 1973.

CHAM74 Chamberlin, D. and Boyce, R., "SEQUEL: A Structured English Query Language", Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann

Arbor, Mich., May 1974.

CHAM75 Chamberlin, D., Gray, J.N., and Traiger, I.L., "Views, Authorization and Locking in a Relational Data Base System", Proc. 1975 National Computer Conference, AFIPS Press, May 1975.

CODD71 Codd, E.F., "A Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Ca., Nov. 1971.

CODD72 Codd, E.F., "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposium 6, May 1972.

HELD75 Held, G.D., Stonebraker, M. and Wong, E., "INGRES - A Relational Data Base Management System", Proc. 1975 National Computer Conference, AFIPS Press, 1975.

HOFF70 Hoffman, L. "The Formulary Model for Access Control and Privacy" Stanford Linear Acceleration Report 117, May, 1970

RITC74 Ritchie, D.M. C Reference Manual UNIX Programmer's Manual, Bell Telephone Labs, Murray Hill, N.J. July 1974.

STON74a Stonebraker, M. and Wong, E., "Access Control in a Relational Data Base Management System by Query Modification", Proc. 1974 ACM National Conference, San Diego, Ca., Nov. 1974

STON74b Stonebraker, M. "A Functional View of Data Independence," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor Mich. May 1974

STON75 Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification", Proc 1975 SIGMOD Workshop on Management of Data, San Jose, Ca., May 1975.

STON76 Stonebraker, M.R., Wong, E., Held, G.D. and Kreps, P. "The Design and Implementation of INGRES", ACM Transactions on Data Base Systems, Vol.1, No.3.

ZOOK76 Zook, W., Youssefi, K., Whyte, M., Rubinstein, P., Kreps, P., Held, G., Ford, J., Berman, R., and Allman, E. "INGRES - Reference Manual-Version 6", University of California, Electronics Research Laboratory, Memo. No. ERL-M579, April, 1976.