PRELIMINARY DESIGN OF INGRES

PART II - PROTECTION, CONCURRENCY, AND GRAPHICS

by

Nancy McDonald, Michael Stonebraker and Eugene Wong

Memorandum No. ERL-M436

9 May 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

PRELIMINARY DESIGN OF INGRES

by

Nancy McDonald, Michael Stonebraker and Eugene Wong

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

Part I Query Language, Data Storage and Access

Part II - Protection, Concurrency, and Graphics

Abstract

INGRES (Interactive Graphics and Retrieval System) is designed to be a general purpose data base management system supporting a relational view of data. This report documents the query language QUEL (QUery Language) available to the user, the data management features provided, the protection mechanism supported, the update concurrency allowed, and the available commands for the display of geographic oriented data.

Research supported in part by the National Science Foundation Grant GK-10656X3, the Army Research Office, Durham, Grant DAHC04-74-G0087, the Joint Services Electronics Program Contract F44620-71-C-0087, the Naval Electronics System Command Grant N0039-71-C-0255 and the Sloan Foundation.

Forward

Project INGRES Has its origin in two related projects. One was a computer-graphics system that had been developed by P, Macri and P. P. Varaiya to deal with map related data. The second was a workshop on data base systems conducted by M. R. Stonebraker and E. Wong, the objective of which was to design and implement a system incorporating certain land-use data of the City of San Francisco. In late 1973, certain common interests and goals became apparent to both groups, of which the foremost was the need for a hardware system. Thanks to financial support from the National Science Foundation, the Sloan Foundation, the College of Engineering and the Department of Electrical Engineering and Computer Sciences at Berkeley, sufficient funds were obtained to purchase a PDP-11/40 based computer graphics system. It is expected that the first version of a graphics and retrieval system will be operational before the end of 1974. In this report we shall describe the basic design goals and philosophy as well as the preliminary system specifications.

Division of labor among the three listed co-authors was as follows: M.R.S. wrote sections 1, 4, 5, 6, 7 and Appendix C, e. W. wrote sections 2, 3 and appendices A and B, N. M. was responsible for section 8. P. P. Varaiya has been the principal moving spirit in the project from its inception. But for him, the project would not exist. Primary responsibility for implementing the retrieval portion of the system is borne by Karel Youssefi (Query Processor) and Peter Kreps (Access Processor). The graphics portion of the system is being implemented by Nancy McDonald and Barbara Cottrell. Members of the Database Workshop (EECS 298-15)

have implemented a preliminary version of the system commands given in Appendix C and have contributed ideas at every stage of the project.[*] We are also grateful to Peter Groat of the Planning Department of the City of San Francisco, to Dean E. S. Kuh of the College of Engineering and to Chairman T. E. Everhart of the Department of Electrical Engineering and Computer Sciences for their enthusiastic support.

[*] In addition to those already mentioned, the members include W. Chang, J. I. Chapela, J. M. Ford, P. Jahanian, P. G. Lehot, J. C. Liang, J. H. Liou, J. N. Yang.

5. Protection in INGRES

It was decided not to rework the UNIX protection system. Basically, six protection modes exist for a file: owner read, owner write, non-owner read, non-owner write, execute and special execute.

The first five modes have the obvious interpretations; the sixth allows the file only to be executed. Moreover, during execution UNIX changes the ID of the running process to that of the owner of the f ile.

In order to provide a semblance of controlled sharing under UNIX, INGRES must run in special execute mode. Files containing relations managed by INGRES will have protection status: owner read, owner write, and no other access will be allowed. There will be one user, called the DATA CZAR, who will be the only user allowed access to INGRES files without executing INGRES. He will be the owner of the file in which the code for INGRES is stored.

One CZAR is allowed for each data base (a data base is an arbitrary collection of relations). In this case several physical copies of the INGRES code must exist, one owned by each of the CZARS. Also, one CZAR may serve several data bases, in which case each individual data base will have an owner, called the DATA BASE ADMINISTRATOR, who has powers (such as deciding relation storage and access controls) not available to the average user. The ADMINISTRATOR is distinguished from the CZAR in that he cannot access his files except through INGRES.

Of special interest here is the fact that the DATA BASE ADMINISTRATOR can enforce access controls on other users, using QUEL-like statements known as <u>interactions</u>. It will be seen that these provide a flexible and general protection mechanism.

We indicate now the two relations that will be used for examples in this section. The first is an employee relation with domains NAME, DEPT, SALARY and MANAGER as follows

	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

Each employee has a manager (except for Harding who is presumably the company president), a salary and is in a department.

The second relation utilized will be a DEPARTMENT relation.

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

5.1 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.

Example 5.1

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.

Example 5.2

Find the salary of Jones.

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

The above statements will automatically be modified to:

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

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 single relation or the key work '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.

Access Control Algorithm

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 $I_R = \{I_1*,\dots,I_j*\}$ and their qualifications by $Q_1*,\dots,Q_j*$.
4. Replace Q, the qualification in R, by $Q \wedge (Q_1* \vee Q_2* \vee \dots \vee Q_j*)$

The resulting QUEL statement can then be executed.[1] Moreover, the resulting query has the following properties.

[1]This can be done by processing the qualification statement into disjunctive normal form. The result is a valid QUEL interaction which can then be executed. An alternate approach, which is attractive only when disjunctions are present, is to use the qualification to restrict the range of tuple variables directly. An alternate (and equivalent) form of the modified query in Example 5.2 using this mechanism is

```
RANGE EMPLOYEE (Y)  :  W1(X)
RETRIEVE         W1 :  Y.ALL      : Y.NAME = SMITH
RETRIEVE         W  :  X.SALARY : X.NAME = JONES
```

Here the qualification restricts X to a workspace W1 from its original range. Note that this approach ensures that all interactions are valid QUEL statements and avoids the necessity of processing them into disjunctive normal form.

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 $I_m, I_n \in I$ 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 5.2.

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 5.3

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 5.4

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 5.5

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 5.6

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 5.6. 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.

5.2 Protection for Interactions Containing Aggregates

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

Example 5.8

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 5.9

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).

Example 5.10

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 5.11

```
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 5.12

Suppose Adams is allowed to access only tuples of 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.

5.3 Efficiency Considerations

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 <u>single variable condition</u> if for all $T_m, T_n \in T$ such $T_m \cap T_n \neq \phi$, there 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 5.6). 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.

6. Locking in INGRES

Two basic problems occur when two or more processes concurrently update a shared data base. They are:

1) ensuring consistency
2) preventing deadlock

The second problem arises during attempts to solve the first. Hence, it is appropriate to discuss consistency first.

6.1 Consistency

Consider two updates for an inventory relation containing PART, SUPPLIER and quantity on hand (QOH) as follows:

```
                RANGE       INVENTORY        (X)
        (u6.1) REPLACE   INVENTORY:X.QOH  =  5   :   X.PART=16
        (u6.2) REPLACE   INVENTORY:X.QOH  =  3   :   X.PART=16
```

If u6.1 and u6.2 can proceed in parallel with no controls it is possible for one update to be lost and for neither user to be aware of that fact. If u6.1 and u6.2 update the relation believing that they are selling items from an inventory, then the resulting relation will be inconsistent.

A second more subtle example is the following. Consider two somewhat facetious updates on a data base containing employee information including NAME, SALARY, DEPARTMENT and HAIR color.

```
                RANGE        EMPLOYEE(X)
(u6.3)  REPLACE EMPLOYEE:X.HAIR  =  red   :  X.HAIR = brown
(u6.4)  REPLACE EMPLOYEE:X.HAIR  =  brown:  X.HAIR = red
```

Here, u6.3 changes the hair color of all brown haired employees to red while u6.4 changes red haired employees to brown. If for the moment we assume that everybody initially has either red or brown hair then the alarming result is

```
u6.3 executed then u6.4 ⇒ all employees have brown hair
u6.4    "        "  u6.3 ⇒ all employees have red hair
u6.3 executed in parallel
        with u6.4 with no
        controls         ⇒ some employees have red hair and some have
                           brown hair
```

Here, the result of the updates depends on the order of execution. Moreover, if they execute in parallel many outcomes are possible. These examples illustrate conflicting updates defined as follows:

Two updates (REPLACE,COMBINE,DELETE) u1 and u2 conflict if

a) one or more tuples satisfies the qualification expression of both updates and the target lists of two updates have an attribute in common.

b) one or more tuples are changed by one update in such a way that at least one tuple either enters or leaves the set of qualifying tuples of the second update.

The first update example illustrates a) while the second explains situation b).

The following two policy alternatives can potentially be enforced when processing conflicting updates.

Policy Alternatives

1) Insist that conflicting updates occur logically sequentially (i.e. as if they were issued sequentially by the same process).

2) Insist that conflicting updates occur sequentially but remove the words "either enter or" from part b) of the definition of conflict.

Note that policy 1 is more comprehensive than policy 2. For example, the first set of updates satisfies both notions of conflict; the second set conflict only by definition 1.

We illustrate the general difficulties involved in each policy situation using the hair updates as an example. We assume that each process has an available mechanism to lock tuples in a relation.

Policy 1

u6.3 and u6.4 respectively must lock all brown and red haired employees. Each then performs the update requested. One then releases his locked records. Since some or all do satisfy the lock condition of the other update, the second user must be backed up so that he can include these new tuples in his lock set and update them appropriately. In general, if one insists on policy 1 then either:

a) there can be no concurrency or

b) some processes may have to be "backed up."

Policy 2

Here u6.3 and u6.4 must lock brown and red haired employees respectively as before so that nobody else can update these records to nullify the qualification statement. In this case no conflict exists and both updates proceed in parallel. Unless it becomes necessary to resolve deadlock, a process need never be backed up.

We feel that the implementation cost of achieving policy 1 is very high and its marginal utility over policy 2 may be quite small. Moreover, the following disucssion shows a clean reasonably efficient implementation of policy 2 which involves no "backing up."

6.2 Locking algorithm

Our basic approach is to force a linear ordering on all resources and insure that users can lock only resources in the ordering greater than those they currently have locked. This is well known to avoid deadlock. Demonstration of the feasibility of this strategy requires an indication of how updates are processed. We discuss the handling of REPLACE operations; the other updates are done similarly.

REPLACE statements involving disjunction are broken into several independent statements each with OR absent from the qualification statement. The resulting commands are processed by first executing a RETRIEVE with the target list and qualification statement of the update. The appropriate set of tuples is ascertained and subsequently new values are substituted back into the appropriate tuples in the original relation. If a new relation is specified by the REPLACE statement, the operation is comparable.

The RETRIEVE portion of a REPLACE statement is handled as discussed in section 2. The tuples satisfying each one variable query must be locked as a RETRIEVE is executed to ensure that policy 2 remains in force. Note, of course, that workspaces are not shared so that tuples in them need not be locked.

Our lock algorithm depends on the validity of the following two statements.

1. Our software conforms to a linear ordering when obtaining tuples from a relation.
2. Each relation in the data base is used as the range of a one-variable query only once when locks must be applied to tuples.

If these statements are valid, the policy can be followed of locking all tuples that satisfy a one variable query whose range is a real relation. If followed one is assured that no other process can change a locked tuple and potentially remove it from those qualifying. One is also assured that no more tuples from a given real relation need ever be locked since subsequent processing will only further restrict qualifying tuples.

Therefore if the relations in the data base are ordered in the same sequence as one variable queries are issued and the tuples within a relation ordered according to our software conventions, a process executing a REPLACE statement will only lock elements greater in the ordering than he has locked already.

We discuss how to ensure the truth of the second statement above; then we indicate in more detail our algorithm. When a one variable query is executed, the range of the corresponding tuple variable is restricted from its original range to a workspace in which the resulting relation is stored. Most of the time the above restriction can happen only once from a real relation to a workspace. Subsequent restriction is from workspace to workspace. There are three easy to remedy cases where this is not true.

a) Tuple Substitution

We consider the situation where tuple substitution is required for a two variable query. As suggested in section 2.10, one relation will be stepped through a tuple at a time; each retrieved tuple will be substituted into the two variable query resulting in a one variable query; this one variable query will then be executed. Notice that many one variable queries are executed for both relations. If one or both are real relations rather than workspaces, statement 2 is not true.

We adopt the brute force solution to this problem, namely, the entirety of both relations will be locked upon starting tuple substitution.

b) Relations with Two Tuple Variables

The same problem arises if two tuple variables range over the same relation. Again the entirety of the relation can be locked when the first one variable query is executed.

c) Aggregates and Aggregate Functions

When an aggregate or aggregate function is defined on the same relation as another tuple variable, we have another instance of b) above. Again, the same solution is taken.

The following indicates our locking algorithm.

<u>Locking Algorithm</u>

1. Collect names of relations upon which

   a) an aggregate or aggregate function is defined

   b) two or more tuple variable are defined

2. Lock the entirety of all such relations, block upon failure
3. Insert each locked relation in the relation ordering as the predecessor to the first relation over which a tuple variable is defined in the given interaction.
4. Inspect the next one variable query, if no more queries gc to 10.
5. If range is a workspace or a relation already locked by current process then execute query, go to 4.
6. If range is a relation in the ordering which is not greater than all existing relations which have a tuple locked by this process, request new ordering, block.
7. If range is a real relation and tuple substitution mode is used lock whole relation (if not already locked), block upon failure, execute query, go to 4.
8. If range is a real relation and query is next one after obtaining the first tuple of a relation by tuple substitution, lock whole relation, block upon failure, execute query, go to 4.

9. Execute query locking qualifying tuples, block if qualifying tuple locked, go to 4 upon completion.

10. Perform remainder of REPLACE statement, unlock all resources, exit.

Associated with each relation in the ordering will be the number of processes currently locking one or more resources in that relation. When this number goes to zero, the relation is removed from the ordering. Processes which are blocked because of an incorrect relation ordering may wait some time before a compatible ordering is arranged. This is the price paid for multivariable updates.

7. Data Independence and Data Definition

This section indicates the data independence provided by INGRES and the commands being implemented to support this goal. Also indicated are the data definition commands

7.1 Data Independence

The notion of data independence is supported in five different ways in INGRES.

a) Relation storage in five different modes is supported and QUEL programs execute correctly regardless of the current storage mode. The command MODIFY allows a user to change the storage of a relation. However, he must be either the DATA BASE ADMINISTRATOR or the DATA CZAR to have this power for relations that are not temporary ones.

b) Redundant indices are supported. Again QUEL programs execute correctly regardless of the presence or absence of indices. The command INDEX allows a user to create an index (again only if he is DATA BASE ADMINISTRATOR or DATA CZAR). The command DELETE allows him to destroy an index (or any other relation).

c) Aliases are supported. Users can create their own aliases for any attributes and issue QUEL statements using these alternate names. The command ALIAS allows the user to declare an alias for an attribute.

d) Conversion of data types are supported. Users may define attributes to be of different types than what is actually stored. QUEL will automatically perform a conversion. The current set of types supported are those available in "C." Another use of the MODIFY command is to declare such alternate types.

e) Virtual Relations are supported. The user by means of a DEFINE statement can specify that a query be stored for subsequent use. This notion is explained by example. Consider the employee relation of section 5 with domains

NAME,DEPT,SALARY and MANAGER.

The statement

```
RANGE        EMPLOYEE(X)
DEFINE W  :  X.NAME,X.SALARY  :  X.SALARY > 1000
```

defines a workspace W with the target list specified and obeying the specified qualification. However W is not created physically; only the above definition is stored. If a user indicates W in a RANGE statement the initial version of the system will simply create W by executing the stored query before continuing with the user's request; later versions will attempt to streamline this operation by substituting the stored query into the user's interaction and restructuring the result.

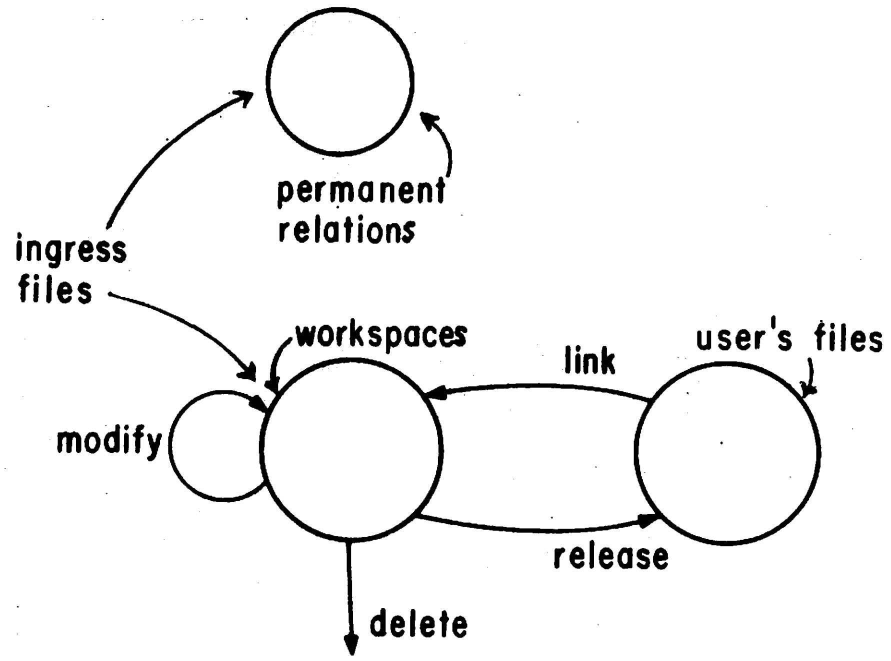
It is expected that virtual relations will be created when one is required to support obsolete views of the data base. Periodically, the DATA BASE ADMINISTRATOR may alter the storage or composition of real relations and define the old version in terms of the new version for any users who require it. Also, users have unrestricted use of the DEFINE command to assist them with stating their interactions.

Using these five mechanisms a reasonable degree of data independence is supported. Although not all one-to-one storage transformations are supported [11], a substantial subset is.

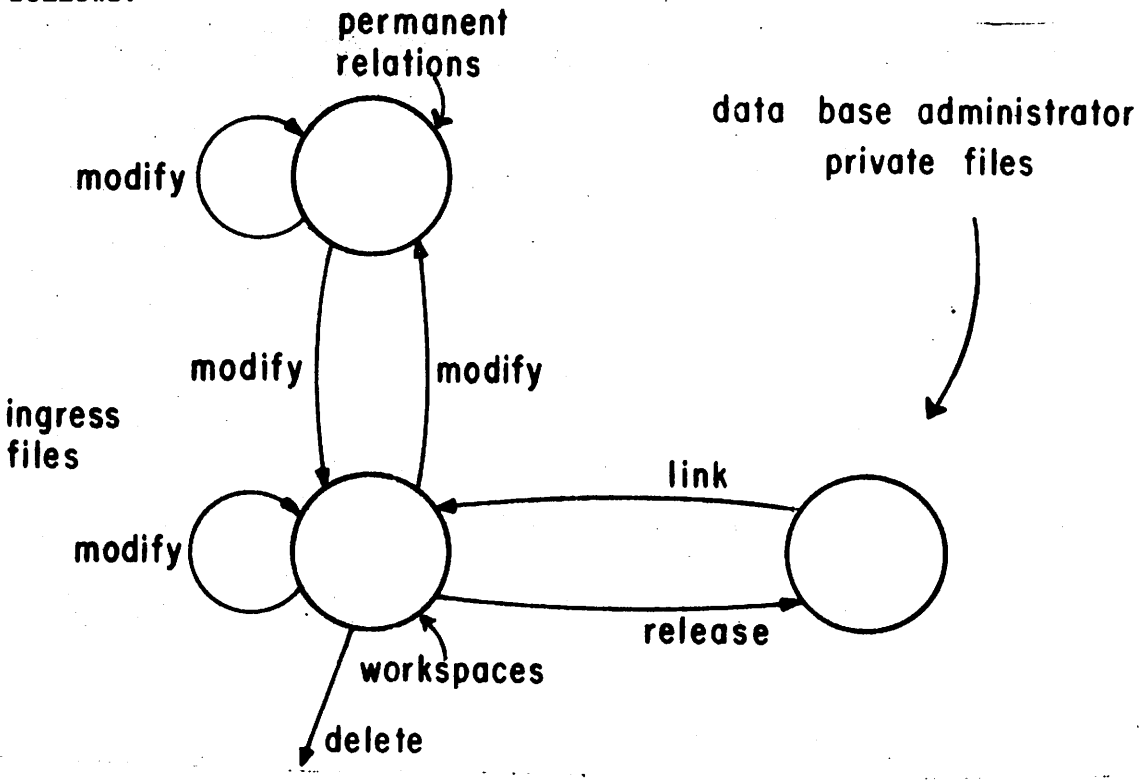
7.2 Data Definition

There are two means by which the DATA BASE ADMINISTRATOR can enter relations into a data base. First, he can utilize the command CREATE to create an empty relation and catalog its existence. Then the administrator can use COMBINE to append tuples or relations to this relation.

The second mechanism is available to the administrator and other users alike. Any workspace can be removed from the system by changing the owner from DATA CZAR to that of the current user. In this way users can preserve private copies of relations of special interest and modify them at will. The command RELEASE has this effect. Conversely, the command LINK transfers a relation from a users' auspices to that of the system. In this way a user can temporarily transfer a relation to the system, use QUEL capabilities and then take the relation back. For all users LINK has the effect of transfering a relation to workspace status, in which case it is destroyed when the user signs off. Hence, general users cannot create permanent relations within the system. Only the DATA BASE ADMINISTRATOR can use MODIFY to change a workspace to a permanent relation. The following diagram indicates relation movement for users.



A different set of transitions are allowed to the DATA BASE ADMINISTRATOR as follows.



It was decided to allow only the DATA BASE ADMINISTRATOR to change relation storage primarily to centralize in one user control of the data base. Otherwise users would, without doubt, alter the data base storage to fit their individual desires. A collection of such users would not harmoniously cooperate.

The full set of commands and their syntax in FORTRAN is given in Appendix C.

8. Display Subsystem

8.1 Introduction

One of the prime users of INGRES will be a group of researchers within the College of Engineering dealing with urban economics, environmental problems, and transportation. This set of users requires the sophisticated retrieval capability of section 2 but additionally needs powerful tools to display results of retrieval queries. Typically, queries are geographically oriented and in one of two forms

a) For various subdivisions of a geographic area (census tracts, parcels, school districts, election districts, counties, etc.) find some A-function(s).

For example if a CENSUS relation contains CENSUS TRACT#, AREA, POPULATION and COUNTY then a typical query of this type would be:

```
RANGE        CENSUS(X)
RETRIEVE W: X.CENSUS TRACT#, X.POPULATION/X.AREA
          : X.COUNTY = SAN FRANCISCO
```

Here a workspace containing population density by census tract is required for San Francisco County.

b) For various arcs in a geographic area (buslines, highways, streets, sewer lines) find some A-function(s).

For example if a BUS relation contains BUS ROUTE#, ANNUAL PATRONAGE, LENGTH and FREQUENCY OF SERVICE, then a typical query would be:

```
RANGE BUS(X)
RETRIEVE W: X.BUS ROUTE# :
        X.LENGTH = MAX(X.LENGTH)
```

Here the longest bus route is required from the relation

In each case a pictorial representation of the resulting workspace would be desirable. In the first situation one might desire the A-function X.POPULATION/X.AREA to be displayed on a census tract map of San Francisco either by placing the value of the A-function in the appropriate map zone or by shading the map zone proportionally to the value of the A-function. In the second example one might want a map of the appropriate bus line rather than simply its number.

To accommodate such desires, INGRES allows an attribute to be of type MAP. In this case it is stored as a variable length character string, typically using the coding facility described in Section 4, and treated specially by display software. Two types of MAPS are recognized:

a) The value of the attribute for a particular tuple is a coded representation for a closed polygon of an arbitrary number of sides.

b) The value of the attribute for a particular tuple is a coded representation for a set of line segments.

Map attributes are treated like any other attributes in QUEL as the following example suggests.

Suppose the CENSUS relation contains CENSUS TRACT#, AREA, POPULATION, COUNTY, and MAP. The following retrieval creates a workspace containing MAP.

```
        RANGE    CENSUS(X)
Q8.3
        RETRIEVE W  :  X.MAP,X.POPULATION/X.AREA : X.COUNTY = SAN FRANCISCO
```

Many of the commands discussed in this section display workspaces assumed to contain at least one MAP. When this is not true, the result from certain display commands will be garbage.

8.2 Display Commands

Users sit before a graphics terminal (here a Digital Equipment Corp GT/42) consisting of a display processor with line and character drawing capability, a light pen and a keyboard. One will be able to enter the DISPLAY MODE from any point in INGRES by typing the command:

```
DISPLAY   filename
```

where filename may be the name (or alias) of a relation. It may also be the name of a previously saved display list for a map. In the latter case the display appears immediately. Otherwise a select phase is entered and the following menu appears.

```
     filename                    Display Types

    DOMAIN NAMES              1.  TABLE
 1.                               (a) PRINTER
 2.                               (b) GRAPHICS
 3.                              2. MAP

 4.                                 SHADE: (a) DOT

 5.                                        (b) HML

 6.                                        (c) ALPHA

 7.                              3. CURVE GRAPH

 8.                                 DISTINGUISH:  (a) LINE

 9.                                               (b) DOT

10.                                               (c) ALPHA

    NEXT SET                     4. POINT GRAPH
    PREVIOUS SET
                                    DISTINGUISH   (a) DOT

                                                  (b) ALPHA

                                 5. OVERLAY .

              ERASE EXIT OK NEWFILE:
```

The domain names (shown here as blanks) will contain the first ten domain names of the relation. NEXT SET is used to page forward through the domains 10 at a time; PREVIOUS SET does the converse.

By pointing to various items in the menu one can formulate 5 different types of displays. These are:

1. Table - appropriate parts of a relation will be presented in tabular form on either the line printer or the graphics terminal.
2. Shaded map - a map will be presented shaded according to the value of a domain. Several shading characters are allowed.
3. Brightness map - a map will be presented with the intensity of an arc proportional to the value of a domain.
4. Curve graph - a plot of one domain against another will be presented. Points in the graph will be connected by lines.

Several graphs may be constructed simultaneously.

5. Point graph - a plot similar to that of 4 will be presented. Points will not be connected, however.

The selection sequence for various specific operations is the following.

1. To print a tabular representation of the relation: select either "PRINTER" or "GRAPHICS" and either a set of specific domain names or "DOMAIN NAMES" which will be interpreted to mean all DOMAINS.
2. To create a shaded map, select a domain from the domain list and the word MAP. This indicates which attribute is to be the map. Then, one at a time, point to a domain on the left and a shading/distinguishing character from the right (typing in one if necessary) to represent that domain until the subset desired has been specified. If no character is specified a dot is the default assumption.
3. To create a map with the brightness of the displayed arc proportional to the value of another attribute proceed as in 2 but select only one domain other than the map domain. If the map is appropriately coded as an arc map (rather than a polygon map), the above display is created.

4. & 5. To create a graph: select a domain from the domain list, which is assumed to be the X-axis, and either "CURVE GRAPH" or "POINT GRAPH". Then select pairs of (domain name, shading/distinguishing character) as in 2.

To move on to the check phase one points to OK. Mistakes can be corrected using ERASE which eliminates all selections indicated thus far. EXIT leaves display mode and returns to the mode from whence one came. NEWFILE allows a new file to be entered through the keyboard for which display selections are desired.

The check phase allows the user to observe and correct if necessary the choices he has made. The various menus are indicated by example

Example 8.1

An example of the check-phase for displaying a table on the printer (indicated by word 'Print') is:

```
    CHECK

       Filename                Table          Print

    DOMAINS:

      1.  Dom 1

      2.  Dom 4

      3.  Dom 7

      4.  Dom 12

   CORRECTION                  RESELECT       EXIT       OK
```

The final line contains the four operations to either correct for an error or end the Check Phase altogether:

1. CORRECTION: by indicating this and the item to be corrected, one can make small changes via the keyboard (e.g. spelling corrections or different shading characters). The filename or display type cannot be changed in this manner. To change them or make many changes, one should indicate RESELECT.

2. RESELECT: returns the user to the Select Phase as though one had just entered with a file name.
3. EXIT: leaves the DISPLAY MODE
4. OK: accepts everything and moves on to the display phase where the following will appear.

```
                    Filename

        Dom 1              Dom 2          ...        Dom 10

   1.
   2.        [the appropriate
   3.                 values
   4.                    will appear
   .                         in this
   .
  25                            space]

     CONTINUE RIGHT       CONTINUE LEFT
       NEXT PAGE          PREVIOUS PAGE
     EXIT RESELECT            O.K.
```

By selecting the CONTINUE RIGHT and OK the next ten domains will be displayed. This is a horizontal continuation. Indicating NEXT PAGE and OK will display the next 25 rows and provides the vertical continuation. EXIT and RESELECT & OK are identical to those options used in earlier phases. PREVIOUS PAGE and CONTINUE LEFT have the obvious meanings.

<u>Example 8.2</u>

An example of a map display with shading follows:

```
CHECK
   Filename                Map          Map Variable: Domain 1
```

DOMAINS

1. Dom 2 - Dot

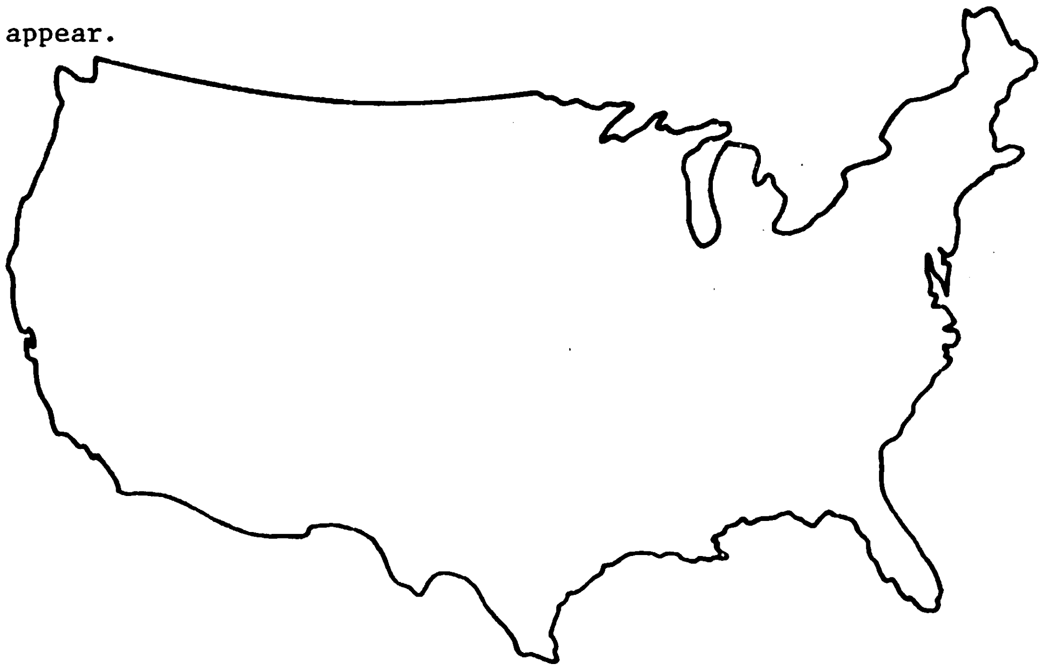
SELECT: OUTLINE

DENSITY SCALE:

SCREEN SCALE:

CORRECTION RESELECT EXIT OK

The SELECT portion of this display provides the user with the option of selecting 0, 1 or 2 items. Choosing the single option OUTLINE provides an intermediate display of an outline map automatically scaled to fit on the screen. The outline option is advisable when displaying a map for the first time to analyze the scale of the map to be sure the detail will show as desired. DENSITY SCALE allows one to override the internal routine which calculates the shading density and SCREEN SCALE refers to the horizontal and vertical span over which the map is displayed upon the screen. Upon selecting outline the following display might appear.

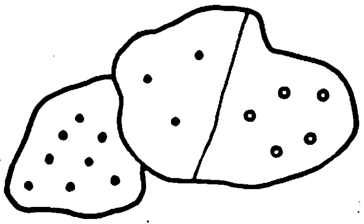


CENTER SCREEN SCALE: RESELECT OK

DENSITY SCALE:

CENTER allows one to select the center of a resulting shaded display. The other commands have the same interpretation as above. Upon selecting OK the following display might appear.

MAP   DIVISIONS: Domain 1



DOMAIN 2: Dot

Scale: x,y

Density:

NOMAP   SAVE: MOVE: CHANGE:

EXIT   RESELECT   OK

Selecting NOMAP followed by one or more of the tracking crosses, then OK causes that portion of the map to be deleted. To combine several sections, select those sections and then O.K. Both of these operations will result in another display-phase image with the same options. Selecting "CHANGE' and any of the listed facts in the upper right corner will allow the user to input a different shading character or scale or density. MOVE followed by inputs from the keyboard represents the X or horizontal distance and Y or vertical distance to be shifted. SAVE allows the display list to be saved for subsequent recall.

Example 8.3

In displaying a curve or point graph the check-phase is exemplified by:

CHECK

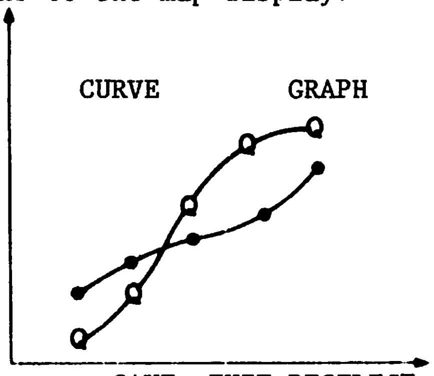
Filename	CURVE GRAPH
X-COORD:	Dom 1
Y-COORD:	Dom 4: dot
	Dom 9: Alpha-Q

SELECT:

SCALE:

CORRECTION RESELECT EXIT OK

SCALE allows the user to specify the X and Y maximums. The X and Y minimum will be data dependent. Selection of OK yields a display with similar options to the map display.



Dom 1: X-COORD

Dom 4: DOT

Dom 9: ALPHA-Q

SCALE: SAVE: EXIT RESELECT OK

In the case of the graph display the scale shows at the extreme ends of the axes and in order to alter them, one must select SCALE and key in the X and Y changes. All other options are as previously discussed. The last example in this section illustrates the overlay feature.

<u>Example 8.4</u>

In overlaying one map with another, the following check-list will appear:

CHECK

Filename	OVERLAY	MAP	Variable = DOM 1

DOMAINS

Dom 2: Dot

Dom 3: HML

Dom 4: Alpha: A

Indicating CENTER and typing a pair allows one to specify the map center. The other options have been as discussed above. Again "OK" allows one to proceed to the next phase where the following will appear.

```
              Dom 7:  Alpha$
           SELECT:  OUTLINE
                    DENSITY SCALE:
                    SCREEN SCALE:
           TYPE IN NAME OF FILE WHICH THIS SHALL OVERLAY:
           CORRECTION RESELECT EXIT OK
```

Selection of OK causes the map specified to be presented superimposed on the one selected for recall.

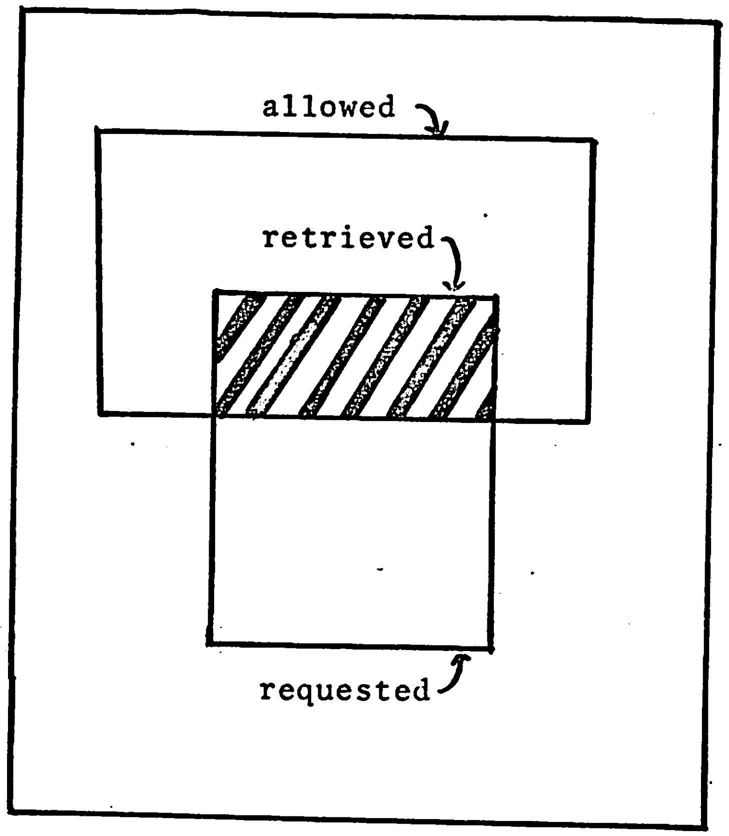
9. Conclusions

The previous sections have indicated the goals of INGRES, the query language adopted, solutions to various data management problems and the display subsystem planned.

In many cases expediency has dictated certain simplifications. These fall into two categories.

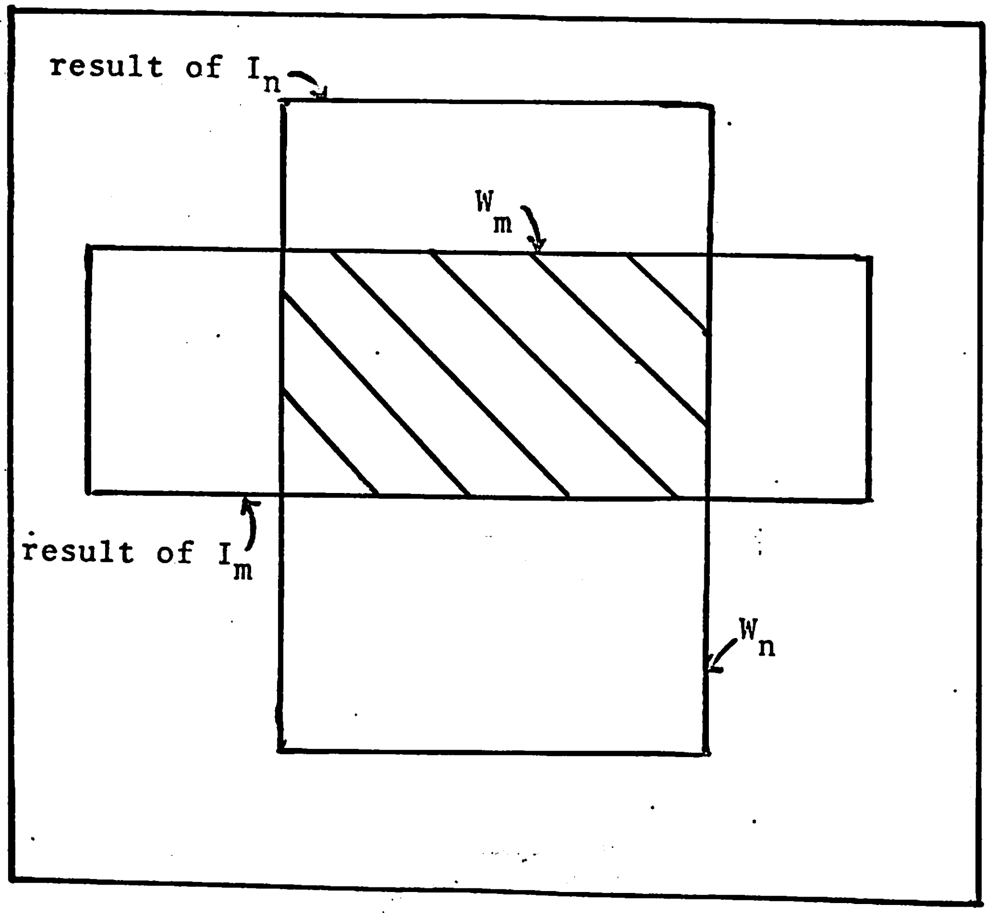
1. Level and form of QUEL. It is anticipated that the unskilled user will have difficulty programming in QUEL. Work is underway to design a user language that is less threatening. We are utilizing the two dimensional medium of our graphics terminal to advantage.
2. Efficiency. Our first goal is to make a straightforward QUEL processor work. At some later time we will attempt to streamline interaction processing. Moreover, we will attempt to have the system give the DATA BASE ADMINISTRATOR assistance in selecting

storage structures in the future. Lastly, the current version is only inefficiently able to be called from inside another "C" program. In the future we will attempt to correct these problems.



The Access Control Scheme at Work

Figure 1



Two Properly Nested Interactions

Figure 2