A TUTORIAL ON INGRES

by
Robert Epstein

Memorandum No. ERL - M77-25
December 15, 1977
(Revised)

Electronics Research Laboratory
College of Engineering
University of California, Berkeley
94720

A Tutorial on INGRES

This tutorial describes how to use the INGRES data base management system. You should be able to follow the the examples given here and observe the same results.

The data manipulation language supported by the INGRES system is called QUEL (QUEry Language). Complete information on QUEL and INGRES appears in the INGRES reference manual. This tutorial does not attempt to cover every detail of INGRES.

Begin by logging onto UNIX, the time sharing system under which INGRES runs. If at all possible, use a terminal that has both upper and lower case letters; otherwise life is going to be miserable for you. If you are on an upper case only terminal, type "\\" everywhere "\" appears in the tutorial.

There should currently be a "%" printed on your terminal. To start using INGRES type the command:

```
% ingres demo
```

This requests "UNIX" to invoke INGRES using the data base called "demo". After a few seconds, the following will appear:

```
INGRES version 6.1/0 login
Tue Aug 30 14:52:23 1977

COPYRIGHT
The Regents of the University of California
1977

This program material is the property of the
Regents of the University of California and
may not be reproduced or disclosed without
the prior written permission of the owner.

go
*
```

The first two lines include the INGRES version number (in this case version 6.1) and the current date. Following that is the "dayfile", which includes messages related to the INGRES system. The "go" indicates that INGRES is ready for your interactions.

The INGRES monitor prints an asterisk ("*") at the beginning of each line to remind you that INGRES is waiting for input.

Type the command:

```
* print parts
* \g
Executing . . .
```

The line "print parts" requests a printout of some data stored in the data base. The "\g" means "go". The message "Executing . . ." indicates that INGRES is processing your query. The following then appears:

```
parts relation

|pnum  |pname               |color   |weight|qoh   |
|-------------------------------------------------|
|     1|central processor   |pink    |    10|     1|
|     2|memory              |gray    |    20|    32|
|     3|disk drive          |black   |   685|     2|
|     4|tape drive          |black   |   450|     4|
|     5|tapes               |gray    |     1|   250|
|     6|line printer        |yellow  |   578|     3|
|     7|l-p paper           |white   |    15|    95|
|     8|terminals           |blue    |    19|    15|
|    13|paper tape reader   |black   |   107|     0|
|    14|paper tape punch    |black   |   147|     0|
|     9|terminal paper      |white   |     2|   350|
|    10|byte-soap           |clear   |     0|   143|
|    11|card reader         |gray    |   327|     0|
|    12|card punch          |gray    |   427|     0|
|-------------------------------------------------|

continue
*
```

What is printed on your terminal is the "parts relation". Intuitively, a relation is nothing more that a table with rows and columns.

In this case the relation name is "parts". There are five columns (we call them domains) named pnum (part number), pname (part name), color, weight, qoh (quantity on hand). Each row of the relation (called a tuple) represents one entry, which in this case represents one part in a computer installation. A relation can have up to 49 domains and a virtually unlimited number of tuples.

Notice that after the query is executed, INGRES prints "continue", while when we first entered INGRES it printed "go". As you enter a query INGRES saves what you type in a "workspace". If you ever mistype a query, typing "\r" will "reset" (ie. erase) your workspace. (Later on we will learn ways to edit mistakes so we don't have to retype the entire query.)

At any time you can see what is in the workspace by typing "\p". Try typing "\p":

```
* \p
print parts
*
```

The current contents of the workspace is printed. Now try typing "\r":

```
* \r
go
*
```

The workspace is now empty. Whenever INGRES types "continue" the workspace is non-empty; whenever INGRES types "go" the workspace is empty.

After a query is executed, INGRES typically types "continue". If you then type a new query, INGRES automatically erases the previous query, so you don't have to type "\r" after every query. This will be further explained as we proceed.

Using the "retrieve" command we can write specific queries about relations. As an example, let's have INGRES print only the "pname" domain of the parts relation. Type the command:

```
* range of p is parts
* retrieve (p.pname)
* \g
Executing . . .


|pname               |
|--------------------|
|central processor   |
|memory              |
|disk drive          |
|tape drive          |
|tapes               |
|line printer        |
|l-p paper           |
|terminals           |
|paper tape reader   |
|paper tape punch    |
|terminal paper      |
|byte-soap           |
|card reader         |
|card punch          |
|--------------------|

continue
*
```

The output is just the pname domain from the parts relation. What we did required two steps. First we declared what is called a "tuple variable" and assigned it to range over the parts relation.

range of p is parts

What this means in English is that the letter "p" represents the parts relation. It may be thought of as a marker which moves down the "parts" relation to keep our place. INGRES remembers the association so that once p is declared to range over parts, we don't have to repeat the range declaration. This is useful when we are working with more than one relation, as will be seen later on.

Next we used the retrieve command. Its form is

retrieve (list here what you want retrieved)

"p" by itself refers to the parts relation. "p.pname" refers to the pname domain of the parts relation, so

saying:

retrieve (p.pname)

means retrieve the pname domain of the parts relation.

Try the query to retrieve pname and color:

```
* retrieve p.pname, p.color
* \g
Executing . . .

2500: syntax error on line 1
last symbol read was: .

continue
*
```

Unfortunately we've made an error. INGRES tells us that it found a syntax error on the first line of the query. "Syntax error" means that we have typed something which INGRES cannot recognize. The error occured on line 1. INGRES makes a sometimes helpful and sometimes feeble attempt at diagnosing the problem. Whenever possible, INGRES tells us the last thing it read before it got confused.

In this case, the error is that the list of things to be retrieved (called the target list) must be enclosed in parenthesis. The correct query is:

```
* retrieve (p.pname, p.color)
* \g
Executing . . .


|pname               |color   |
|----------------------------|
|central processor   |pink    |
|memory              |gray    |
|disk drive          |black   |
|tape drive          |black   |
|tapes               |gray    |
|line printer        |yellow  |
|l-p paper           |white   |
|terminals           |blue    |
|paper tape reader   |black   |
|paper tape punch    |black   |
|terminal paper      |white   |
|byte-soap           |clear   |
|card reader         |gray    |
|card punch          |gray    |
|----------------------------|

continue
*
```

You can restrict which tuples are printed by adding a "qualification" to the query. For example to get the name and color of only those parts which are gray, type:

```
* retrieve (p.pname, p.color)
* where p.color = "gray"
* \g
Executing . . .


|pname               |color   |
|----------------------------|
|memory              |gray    |
|tapes               |gray    |
|card reader         |gray    |
|card punch          |gray    |
|----------------------------|

continue
*
```

Notice that INGRES prints only those parts where p.color is gray. Notice also that gray must be in quotes ("gray"). This is necessary. The only way INGRES will recognize character strings (e.g. words) is to enclose them in quotes.

What if we wanted part names for gray or pink parts? We only need to append to the previous query the phrase:

or p.color = "pink"

Remember, however, that if the next line typed begins a new query, INGRES will automatically reset the workspace. The workspace will be *saved* only if the next line begins with a command such as "\p" or "\g". (There are others which we will come to later.) If such a command is typed, the previous query is saved and anything further will be appended to that query.

Thus, by typing:

```
* \p
retrieve (p.pname, p.color)
where p.color = "gray"
*
```

you can see the previous query. Now type:

```
* or p.color = "pink"
*
```

INGRES appends that last line to the end of the query. You can verify this yourself by printing the workspace:

```
* \p
retrieve (p.pname, p.color)
where p.color = "gray"
```

```
or p.color = "pink"
*
```

Now run the query:

```
* \g
Executing . . .


|pname               |color   |
|----------------------------|
|central processor   |pink    |
|memory              |gray    |
|tapes               |gray    |
|card reader         |gray    |
|card punch          |gray    |
|----------------------------|

continue
*
```

The rules about when the workspace is reset may be very confusing at first. In general, INGRES will do exactly what you want without you having to think about it.

We have seen qualifications which used "or" and "=". In general one can use:

```
and
or
not
=      (equal)
!=     (not equal)
>      (greater than)
>=     (greater than or equal)
<      (less than)
<=     (less than or equal)
```

Evaluation occurs in the order the qualification was typed (ie. left to right). Parenthesis can be used to group things in any arbitrary order.

INGRES can do computations on the data stored in a relation. For example, the parts relation has quantity on hand and weight for each item. We might like to know the total weight for each group of parts (i.e. weight multiplied by qoh).

To get the name, part number and total weight for each part type the query:

```
* retrieve (p.pname, p.pnum, p.qoh * p.weight)
* \g
Executing . . .

2500: syntax error on line 1
last symbol read was: *
```

```
continue
*
```

Another error. The problem is that when a computation is done, INGRES does not know how to title the domain on the printout. For a simple domain, INGRES uses the domain name as a title. For anything else, you must create a new domain title by specifying:

```
tot = p.qoh * p.weight
```

More generally the form is:

```
title = expression
```

For example:

```
name = p.pname
computation = p.weight / 2000 * (p.qoh + 2)
```

Let's fix the error by retyping the query. As long as the first line after a query does not begin with a "\p" or "\g" then INGRES will automatically reset the workspace, erasing the previous query for us.

```
* retrieve (p.pname, p.pnum, tot=p.qoh * p.weight)
* \g
Executing . . .


|pname               |pnum  |tot   |
|------------------------------------|
|central processor   |     1|    10|
|memory              |     2|   640|
|disk drive          |     3|  1370|
|tape drive          |     4|  1800|
|tapes               |     5|   250|
|line printer        |     6|  1734|
|l-p paper           |     7|  1425|
|terminals           |     8|   285|
|paper tape reader   |    13|     0|
|paper tape punch    |    14|     0|
|terminal paper      |     9|   700|
|byte-soap           |    10|     0|
|card reader         |    11|     0|
|card punch          |    12|     0|
|------------------------------------|

continue
*
```

In addition to multiplication, INGRES supports:

+	addition
-	subtraction (and unary negation)
/	division

* multiplication
** exponentiation (e.g. 3**10)
abs absolute value (e.g. abs(p.qoh - 50))
mod modulo division

and many others. Please refer to the INGRES reference manual for a brief but complete description of what is supported.

If all we wanted were part numbers 2 or 10, then we could add the qualification:

where p.pnum = 2 or p.pnum = 10

CAUTION: if we just started typing "where p.pnum .... " INGRES would understand this as the beginning of a new query and would reset the workspace. To avoid this you could type "\p" and force INGRES to print the workspace, or you can type "\a" (append). The append command guarantees that whatever else is typed will be appended to what is already in the workspace. This command is only needed immediately after a query is executed. Any other time data will be appended automatically. Try the following:

```
* \a
* where p.pnum = 2 or p.pnum = 10
* \g
Executing . . .


|pname               |pnum  |tot   |
|--------------------------------|
|memory              |     2|   640|
|byte-soap           |    10|     0|
|--------------------------------|

continue
*
```

To include all part numbers greater than 2 and less than or equal to 10:

```
* retrieve (p.pname, p.pnum, tot=p.qoh * p.weight)
* where p.pnum > 2 and p.pnum <= 10
* \g
Executing . . .


|pname               |pnum  |tot   |
|--------------------------------|
|disk drive          |     3|  1370|
|tape drive          |     4|  1800|
|tapes               |     5|   250|
|line printer        |     6|  1734|
|l-p paper           |     7|  1425|
|terminals           |     8|   285|
|terminal paper      |     9|   700|
|byte-soap           |    10|     0|
|--------------------------------|
```

```
continue
*
```

Now, suppose we want to change the previous query to give results for part numbers between 5 and 10 instead of 2 and 10. You are probably annoyed at having to retype the entire query in order to change one character. Consequently, INGRES lets you use the UNIX text editor to make corrections and/or additions to your workspace. At any time you can type "\e" and the INGRES monitor will write your workspace to a file and call the UNIX "ed" program. For example:

```
* \e
>>ed
83
```

The ">>ed" message tells you that you are now using the editor. The number 83 is the number of characters in your workspace.

We can now edit the query by changing the 2 to a 5. Included in the UNIX documentation is a tutorial on using the text editor. Rather than duplicating that tutorial, we will just use a few of the editor commands to illustrate how to do editing:

```
1p
retrieve (p.pname,p.pnum,tot = p.qoh * p.weight)
2p
where p.pnum > 2 and p.pnum <= 10
s/2/5/p
where p.pnum > 5 and p.pnum <= 10
w
83
q
<<monitor
*
```

Very briefly, this is what happens. "1p" and "2p" printed lines 1 and 2. "s/2/5/p" substitutes a 5 for a 2 on the current line (line 2), and then prints that line. "w" writes the query back to the INGRES workspace.

Inside the editor you can use any "ed" command except "e" (since e changes the file name). When you quit the editor (q command), the INGRES monitor will print "<<monitor" to remind you that you are back in INGRES. Notice that you MUST precede the "q" command with a "w" command to pass the corrected workspace back to INGRES.

To verify that the query is correct and to run it, type:

```
* \p\g
retrieve (p.pname,p.pnum,tot = p.qoh * p.weight)
where p.pnum > 5 and p.pnum <= 10
Executing . . .


|pname               |pnum  |tot   |
|--------------------------------|
|line printer        |     6|  1734|
```

```
|l-p paper           |     7|  1425|
|terminals           |     8|   285|
|terminal paper      |     9|   700|
|byte-soap           |    10|     0|
|-----------------------------------|

continue
*
```

Having exhausted the interesting queries concerning the parts relation, lets now look at a new relation called "supply". Type:

```
* print supply
* \g
Executing . . .


supply relation

|snum  |pnum  |jnum  |shipdate|quan  |
|-----------------------------------|
|   475|     1|  1001|73-12-31|     1|
|   475|     2|  1002|74-05-31|    32|
|   475|     3|  1001|73-12-31|     2|
|   475|     4|  1002|74-05-31|     1|
|   122|     7|  1003|75-02-01|   144|
|   122|     7|  1004|75-02-01|    48|
|   122|     9|  1004|75-02-01|   144|
|   440|     6|  1001|74-10-10|     2|
|   241|     4|  1001|73-12-31|     1|
|    62|     3|  1002|74-06-18|     3|
|   475|     2|  1001|73-12-31|    32|
|   475|     1|  1002|74-07-01|     1|
|     5|     4|  1003|74-11-15|     3|
|     5|     4|  1004|75-01-22|     6|
|    20|     5|  1001|75-01-10|    20|
|    20|     5|  1002|75-01-10|    75|
|   241|     1|  1005|75-06-01|     1|
|   241|     2|  1005|75-06-01|    32|
|   241|     3|  1005|75-06-01|     1|
|    67|     4|  1005|75-07-01|     1|
|   999|    10|  1006|76-01-01|   144|
|   241|     8|  1005|75-07-01|     1|
|   241|     9|  1005|75-07-01|   144|
|-----------------------------------|

continue
*
```

The supply relation contains snum (the supplier number), pnum (the part number which is supplied by that supplier), jnum (the job number), shipdate (the date it was shipped), and quan (the quantity

shipped).

To find out what parts are supplied by supplier number 122 type:

```
* retrieve (s.pnum) where s.snum = 122
* \g
Executing . . .

2109: line 1, Variable 's' not declared in RANGE statement

continue
*
```

We have referenced the tuple variable "s" (i.e. s.pnum) without telling INGRES what "s" represents. We are missing a range declaration. Retype the query as follows:

```
* range of s is supply
* retrieve (s.pnum) where s.snum = 122
* \g
Executing . . .


|pnum  |
|------|
|     7|
|     7|
|     9|
|------|

continue
*
```

Supplier number 122 supplies part numbers 7, 7 and 9. Note that 7 is listed twice. When retrieving tuples onto a terminal it is more efficient for INGRES NOT to check for duplicate tuples. INGRES can be forced to remove duplicate tuples. We will come to that later.

We now know that supplier 122 supplies part numbers 7 and 9. If you haven't run this query a few hundred times you probably don't know what part names correspond to part numbers 7 and 9. We could find out simply by running the query:

```
* retrieve (p.pname) where p.pnum = 7 or
* p.pnum = 9
* \g
Executing . . .


|pname               |
|--------------------|
|l-p paper           |
|terminal paper      |
|--------------------|
```

```
continue
*
```

After two queries we know by part name what parts are supplied by supplier number 122. We could do the same thing in one query by asking:

```
* retrieve (p.pname) where p.pnum = s.pnum
* and s.snum = 122
* \g
Executing . . .


|pname               |
|--------------------|
|l-p paper           |
|l-p paper           |
|terminal paper      |
|--------------------|

continue
*
```

Again note that "l-p paper" is duplicated. Look closely at this query. Note that the domain pnum exists in both the parts and supply relations. By saying p.pnum = s.pnum, we are logically joining the two relations.

Suppose we wished to find all suppliers who supply the central processor. We know that we will want to retrieve s.snum. We want only those s.snum's where the corresponding s.pnum is equal to the part number for the central processor.

If we find the p.pname which is equal to "central processor" then that will tell us the correct p.pnum. Finally we want s.pnum = p.pnum. The query is:

```
* retrieve (s.snum) where
* s.pnum = p.pnum and p.pname = "central processor"
* \g
Executing . . .


|snum  |
|------|
|   475|
|   475|
|   241|
|------|

continue
*
```

Let's abandon the parts and supply relations and try another. First, we can see what other relations are in the database by typing:

```
* help \g
* Executing . . .


 relation name     relation owner

 relation          ingres
 attribute         ingres
 indexes           ingres
 integrity         ingres
 constraint        ingres
 item              ingres
 sale              ingres
 employee          ingres
 dept              ingres
 supplier          ingres
 store             ingres
 parts             ingres
 supply            ingres


continue
*
```

Let's look at the "employee" relation. Since we know nothing about the relation we can also use the "help" command to learn about it. Type:

```
* help employee
* \g
Executing . . .


Relation:            employee
Owner:               ingres
Tuple width:         30
Saved until:         Fri Mar 25 11:01:30 1977
Number of tuples:    24
Storage structure:   paged heap
relation type:              user relation

 attribute name    type  length  keyno.

 number           i       2
 name             c      20
 salary           i       2
 manager          i       2
 birthdate        i       2
 startdate        i       2


continue
*
```

The help command lists overall information about the employee relation together with each attribute, its type and its length.

INGRES supports three data types: integer numbers, floating point numbers, and characters strings. Character domains can be from 1 to 255 characters in length. Integer domains can be 1, 2, or 4 bytes in length. This means that integers can obtain a maximum value of 127; 32,767; and 2,147,483,647 respectively. Floating point numbers can be either 4 or 8 bytes. Both hold a maximum value of about 10**38; with 7 or 17 digit accuracy respectively.

To look at all domains we could use the print command or we could use the retrieve command and list each domain in the target list. INGRES provides a shorthand way of doing just that. Try the following:

```
* range of e is employee
* retrieve (e.all)
* \g
Executing . . .


|number|name                |salary|manage|birthd|startd|
|-----------------------------------------------------|
|   157|Jones, Tim          | 12000|   199|  1940|  1960|
|  1110|Smith, Paul         |  6000|    33|  1952|  1973|
|    35|Evans, Michael      |  5000|    32|  1952|  1974|
|   129|Thomas, Tom         | 10000|   199|  1941|  1962|
|    13|Edwards, Peter      |  9000|   199|  1928|  1958|
|   215|Collins, Joanne     |  7000|    10|  1950|  1971|
|    55|James, Mary         | 12000|   199|  1920|  1969|
|    26|Thompson, Bob       | 13000|   199|  1930|  1970|
|    98|Williams, Judy      |  9000|   199|  1935|  1969|
|    32|Smythe, Carol       |  9050|   199|  1929|  1967|
|    33|Hayes, Evelyn       | 10100|   199|  1931|  1963|
|   199|Bullock, J.D.       | 27000|     0|  1920|  1920|
|  4901|Bailey, Chas M.     |  8377|    32|  1956|  1975|
|   843|Schmidt, Herman     | 11204|    26|  1936|  1956|
|  2398|Wallace, Maggie J.  |  7880|    26|  1940|  1959|
|  1639|Choy, Wanda         | 11160|    55|  1947|  1970|
|  5119|Ferro, Tony         | 13621|    55|  1939|  1963|
|    37|Raveen, Lemont      | 11985|    26|  1950|  1974|
|  5219|Williams, Bruce     | 13374|    33|  1944|  1959|
|  1523|Zugnoni, Arthur A.  | 19868|   129|  1928|  1949|
|   430|Brunet, Paul C.     | 17674|   129|  1938|  1959|
|   994|Iwano, Masahiro     | 15641|   129|  1944|  1970|
|  1330|Onstad, Richard     |  8779|    13|  1952|  1971|
|    10|Ross, Stanley       | 15908|   199|  1927|  1945|
|    11|Ross, Stuart        | 12067|     0|  1931|  1932|
|-----------------------------------------------------|

continue
*
```

"All" is a keyword which is expanded by INGRES to become all domains. The domains are not guaranteed to be in any particular order. The previous query is equivalent to:

```
range of e is employee
retrieve (e.number, e.name, e.salary, e.manager
       e.birthdate, e.startdate)
```

Let's retrieve the salary of Stan Ross. At this point we will need to be able to type both upper and lower case letters. If you are on an upper case only terminal, type a single "\" before a letter you wish to capitalize. Thus on an upper case only terminal type "\ROSS, \STAN". If you are on an upper and lower case terminal, use the shift key to capitalize a letter.

Run the query:

```
* retrieve (e.name,e.salary)
* where e.name = "Ross, Stan"
* \g
Executing . . .


|name                |salary|
|--------------------------|
|--------------------------|

continue
*
```

The result is empty. There is no e.name which satisfies the qualification. That's strange because we know there is a Stan Ross. However, INGRES does not know, for example, that "Stanley" and "Stan" are semantically the same.

To get the correct answer in this situation you may use the special "pattern matching" characters provided by INGRES.

One such character is "*". It matches any string of zero or more characters. Try the query:

```
* retrieve (e.name,e.salary)
* where e.name = "Ross, S*"
* \g
Executing . . .


|name                |salary|
|--------------------------|
|Ross, Stanley       | 15908|
|Ross, Stuart        | 12067|
|--------------------------|

continue
*
```

In the first case "*" matched the string "tanley" and in the second case it matched "tuart".

Here is another example. Find the salaries of all people whose first name is "Paul":

```
* retrieve (e.name,e.salary)
* where e.name = "*,Paul*"
* \g
Executing . . .


|name                |salary|
|--------------------------|
|Smith, Paul         |  6000|
|Brunet, Paul C.     | 17674|
|--------------------------|

continue
*
```

Notice that if we had asked for e.name = "*,Paul" we would not have gotten the second tuple. Also, INGRES ignores blanks in any character comparison whether using pattern matching characters or not. This means that the following would all give the same results:

```
e.name = "Ross,Stanley"
e.name = "Ross,   Stanley   "
e.name = "R o s s,Stanley"
```

Particular characters or ranges of characters can be put in square brackets ([]). For example, find all people whose names start with "B" through "F":

```
* retrieve (e.name,e.salary)
* where e.name = "[B-F]*"
* \g
Executing . . .


|name                |salary|
|--------------------------|
|Evans, Michael      |  5000|
|Edwards, Peter      |  9000|
|Collins, Joanne     |  7000|
|Bullock, J.D.       | 27000|
|Bailey, Chas M.     |  8377|
|Choy, Wanda         | 11160|
|Ferro, Tony         | 13621|
|Brunet, Paul C.     | 17674|
|--------------------------|

continue
*
```

Notice that this last query could be done another way:

```
* retrieve (e.name,e.salary)
* where e.name >"B" and e.name <"G"
* \g
```

```
Executing . . .


|name                |salary|
|--------------------------|
|Evans, Michael      |  5000|
|Edwards, Peter      |  9000|
|Collins, Joanne     |  7000|
|Bullock, J.D.       | 27000|
|Bailey, Chas M.     |  8377|
|Choy, Wanda         | 11160|
|Ferro, Tony         | 13621|
|Brunet, Paul C.     | 17674|
|--------------------------|

continue
*
```

The two results are identical; however, the second way is generally more efficient for INGRES to process.

There are three types of pattern matching constructs. All three can be used in any combination for character comparison. They are:

- `*` matches any length character string
- `?` matches any one (non-blank) character
- `[ ]` can match any character listed in the brackets. If two characters are separated by a dash (-), then it matches any character falling between the two characters.

The special meaning of a pattern matching character can be turned off by preceeding it with a `"\"`. This means that `"\*"` refers to the character `"*"`.

We turn now to the aggregation facilities supported by INGRES. This allows a user to perform computations on whole domains of a relation. For example, one aggregate is average (avg). To compute the average salary for all employees, we enter:

```
* retrieve (avgsal=avg(e.salary))
* \g
Executing . . .


|avgsal    |
|----------|
| 11867.520|
|----------|

continue
*
```

The particular title "avgsal" is arbitrary, but necessary; INGRES needs *some* sort of title for any expression in the target list (other than a simple domain).

We can also find the minimum and maximum salaries:

```
* retrieve (minsal=min(e.salary),maxsal=max(e.salary))
* \g
Executing . . .


|minsal|maxsal|
|------------|
|  5000| 27000|
|------------|

continue
*
```

If we wanted to know the names of the employees who make the minimum and maximum salaries, that query would be:

```
* retrieve (e.name, e.salary)
* where e.salary = min(e.salary) or e.salary = max(e.salary)
* \g
Executing . . .

|name                |salary|
|--------------------------|
|Evans, Michael      |  5000|
|Bullock, J.D.       | 27000|
|--------------------------|

continue
*
```

INGRES supports the following aggregates:

```
count
min
max
avg
sum
any
```

We now indicate the query to list each employee along with the average salary for all employees:

```
* retrieve (e.name,peersal=avg(e.salary))
* \g
Executing . . .


|name                |peersal   |
|-----------------------------|
|Jones, Tim          | 11867.520|
|Smith, Paul         | 11867.520|
|Evans, Michael      | 11867.520|
|Thomas, Tom         | 11867.520|
```

```
|Edwards, Peter      | 11867.520|
|Collins, Joanne     | 11867.520|
|James, Mary         | 11867.520|
|Thompson, Bob       | 11867.520|
|Williams, Judy      | 11867.520|
|Smythe, Carol       | 11867.520|
|Hayes, Evelyn       | 11867.520|
|Bullock, J.D.       | 11867.520|
|Bailey, Chas M.     | 11867.520|
|Schmidt, Herman     | 11867.520|
|Wallace, Maggie J.  | 11867.520|
|Choy, Wanda         | 11867.520|
|Ferro, Tony         | 11867.520|
|Raveen, Lemont      | 11867.520|
|Williams, Bruce     | 11867.520|
|Zugnoni, Arthur A.  | 11867.520|
|Brunet, Paul C.     | 11867.520|
|Iwano, Masahiro     | 11867.520|
|Onstad, Richard     | 11867.520|
|Ross, Stanley       | 11867.520|
|Ross, Stuart        | 11867.520|
|------------------------------|

continue
*
```

An aggregate always evaluates to a single value. To process the last query, INGRES replicated the average salary next to each e.name.

Aggregates can have their own qualification. For example, we can retrieve a list of each employee along with the average salary of those employees over 50.

```
* retrieve (e.name,peersal=
* avg(e.salary where 1977-e.birthdate > 50))
* \g
Executing . . .


|name                |peersal   |
|------------------------------|
|Jones, Tim          | 19500.000|
|Smith, Paul         | 19500.000|
|Evans, Michael      | 19500.000|
|Thomas, Tom         | 19500.000|
|Edwards, Peter      | 19500.000|
|Collins, Joanne     | 19500.000|
|James, Mary         | 19500.000|
|Thompson, Bob       | 19500.000|
|Williams, Judy      | 19500.000|
|Smythe, Carol       | 19500.000|
|Hayes, Evelyn       | 19500.000|
|Bullock, J.D.       | 19500.000|
```

```
|Bailey, Chas M.     | 19500.000|
|Schmidt, Herman     | 19500.000|
|Wallace, Maggie J.  | 19500.000|
|Choy, Wanda         | 19500.000|
|Ferro, Tony         | 19500.000|
|Raveen, Lemont      | 19500.000|
|Williams, Bruce     | 19500.000|
|Zugnoni, Arthur A.  | 19500.000|
|Brunet, Paul C.     | 19500.000|
|Iwano, Masahiro     | 19500.000|
|Onstad, Richard     | 19500.000|
|Ross, Stanley       | 19500.000|
|Ross, Stuart        | 19500.000|
|--------------------------------|

continue
*
```

Contrast the previous query with this next one. We will retrieve the names of those employees over fifty and retrieve the average salary for all employees.

```
* retrieve (e.name,peersal=avg(e.salary))
* where 1977-e.birthdate > 50
* \g
Executing . . .


|name                |peersal   |
|--------------------------------|
|James, Mary         | 11867.520|
|Bullock, J.D.       | 11867.520|
|--------------------------------|

continue
*
```

There is a very important distinction between these last two queries. An aggregate is completely self-contained. It is not affected by the qualification of the query as a whole.

In the first case, average is computed only for those employees over fifty, and all employees are retrieved. In the second case, however, average is computed for all employees but only those employees over 50 are retrieved.

If we wanted a list of all employees over fifty together with the average salary of employees over fifty, we would combine the previous two queries into one. That query would be:

```
* retrieve (e.name, peersal=
* avg(e.salary where 1977 - e.birthdate > 50))
* where 1977 - e.birthdate > 50
* \g
Executing . . .
```

```
|name                |peersal   |
|--------------------------------|
|James, Mary         | 19500.000|
|Bullock, J.D.       | 19500.000|
|--------------------------------|

continue
*
```

It is sometimes useful to have duplicate values removed before an aggregation is computed. For example if you wanted to know how many managers there are, the following query will not give the right answer:

```
* retrieve (bosses = count(e.manager))
* \g
* Executing . . .


|bosses       |
|-------------|
|           25|
|-------------|

continue
*
```

Notice that that gives the count of how many tuples there are in employee. What we want to know is how many unique e.manager's there are.

INGRES provides three special forms of aggregation.

```
countu        count unique values
avgu          average unique values
sumu          sum unique values
```

It's interesting to note that minu, maxu, and anyu are not needed. Their values would be the same whether duplicates were removed or not.

The correct query to find the number of managers is:

```
* retrieve (bosses=countu(e.manager))
* \g
Executing . . .


|bosses       |
|-------------|
|            9|
|-------------|

continue
*
```

Another aggregate facility supported by INGRES is called aggregate functions. Aggregate functions group data into categories and perform separate aggregations on each category.

For example, what if you wanted to retrieve each employee, and the average salary paid to employees with the same manager? That query would be:

```
* retrieve (e.name,manageravg=avg(e.salary by e.manager))
* \g
Executing . . .


|name                |manageravg|
|--------------------------------|
|Jones, Tim          | 11117.555|
|Thomas, Tom         | 11117.555|
|Edwards, Peter      | 11117.555|
|James, Mary         | 11117.555|
|Thompson, Bob       | 11117.555|
|Williams, Judy      | 11117.555|
|Smythe, Carol       | 11117.555|
|Hayes, Evelyn       | 11117.555|
|Ross, Stanley       | 11117.555|
|Smith, Paul         |  9687.000|
|Williams, Bruce     |  9687.000|
|Evans, Michael      |  6688.500|
|Bailey, Chas M.     |  6688.500|
|Collins, Joanne     |  7000.000|
|Bullock, J.D.       | 19533.500|
|Ross, Stuart        | 19533.500|
|Schmidt, Herman     | 10356.333|
|Wallace, Maggie J.  | 10356.333|
|Raveen, Lemont      | 10356.333|
|Choy, Wanda         | 12390.500|
|Ferro, Tony         | 12390.500|
|Zugnoni, Arthur A.  | 17727.666|
|Brunet, Paul C.     | 17727.666|
|Iwano, Masahiro     | 17727.666|
|Onstad, Richard     |  8779.000|
|--------------------------------|

continue
*
```

The first nine people all have the same manager and their average salary is 11117.555. The next two people have the same manager and their average salary is 9687. etc.

Once again, if we wanted to see the same list just for those employees over 50:

```
* retrieve (e.name,manageravg=avg(e.salary by e.manager))
* where 1977-e.birthdate > 50
* \g
Executing . . .
```

```
|name                |manageravg|
|--------------------------------|
|James, Mary         | 11117.555|
|Bullock, J.D.       | 19533.500|
|--------------------------------|

continue
*
```

Aggregate functions (unlike simple aggregates) are not completely local to themselves. The domains upon which the data is grouped (called the by-list) are logically connected to the domains in the rest of the query.

In these last examples, the "e.manager" in the by-list refers to the same tuple as "e.name" in the target list.

If we wanted to compute the average salaries by manager for only managers 33 and 199, then the query would be:

```
* retrieve (e.name,manageravg=
* avg(e.salary by e.manager)
* where e.manager = 199 or e.manager = 33
* \g
Executing . . .


|name                |manageravg|
|--------------------------------|
|Jones, Tim          | 11117.555|
|Thomas, Tom         | 11117.555|
|Edwards, Peter      | 11117.555|
|James, Mary         | 11117.555|
|Thompson, Bob       | 11117.555|
|Williams, Judy      | 11117.555|
|Smythe, Carol       | 11117.555|
|Hayes, Evelyn       | 11117.555|
|Ross, Stanley       | 11117.555|
|Smith, Paul         |  9687.000|
|Williams, Bruce     |  9687.000|
|--------------------------------|

continue
*
```

Suppose we wanted to find out how many people work for each manager, and in addition wanted only to include those employees who have worked at least seven years.

```
* retrieve (e.manager,people=count(e.name by e.manager where
* e.startdate < 1970))
* \g
Executing . . .
```

```
|manage|people       |
|--------------------|
|   199|            8|
|    33|            2|
|    32|            0|
|    10|            0|
|     0|            2|
|    26|            2|
|    55|            1|
|   129|            2|
|    13|            0|
|--------------------|

continue
*
```

Notice that managers 32, 10, and 13 have no employees who started before 1970. Now suppose we want to know the average salary for those employees. Simply change "count" to "avg" and rerun the query.

```
* retrieve (e.manager,people=avg(e.salary by e.manager where
* e.startdate < 1970))
* \g
Executing . . .


|manage|people    |
|-----------------|
|   199| 10882.250|
|    33| 22687.000|
|    32|     0.000|
|    10|     0.000|
|     0| 19533.500|
|    26|  9542.000|
|    55| 13621.000|
|   129| 18771.000|
|    13|     0.000|
|-----------------|

continue
*
```

Notice what INGRES does for managers 32, 10 and 13. The average salary for those manager employees is actually undefined since there are no employees who started before 1970. INGRES always makes undefined values zero in aggregates.

If you want to remove the zero values from the output, a qualification can be added to the query. The following query will find the average salaries only for those which are greater than zero.

```
* retrieve (e.manager,people=avg(e.salary by e.manager where
* e.startdate < 1970))
* where avg(e.salary by e.manager where e.startdate < 1970) > 0
* \g
```

```
Executing . . .


|manage|people    |
|------------------|
|   199| 10882.250|
|    33| 22687.000|
|     0| 19533.500|
|    26|  9542.000|
|    55| 13621.000|
|   129| 18771.000|
|------------------|

continue
*
```

Up until now we have been retrieving results directly onto the terminal. You can also save results by retrieving them into a new relation. This is done by saying:

```
retrieve into newrel ( ... )
where . . .
```

The rules are exactly the same as for retrieves onto the terminal. INGRES will create the new relation with the correct domains, and then put the results of the query in the new relation.

For example, create a new relation called "overpaid" which has only those employees who make more than $8000:

```
* retrieve into overpaid (e.all)
* where e.salary > 8000
* print overpaid
* \g
Executing . . .


overpaid relation

|number|name                |salary|manage|birthd|startd|
|-----------------------------------------------------|
|    10|Ross, Stanley       | 15908|   199|  1927|  1945|
|    11|Ross, Stuart        | 12067|     0|  1931|  1932|
|    13|Edwards, Peter      |  9000|   199|  1928|  1958|
|    26|Thompson, Bob       | 13000|   199|  1930|  1970|
|    32|Smythe, Carol       |  9050|   199|  1929|  1967|
|    33|Hayes, Evelyn       | 10100|   199|  1931|  1963|
|    37|Raveen, Lemont      | 11985|    26|  1950|  1974|
|    55|James, Mary         | 12000|   199|  1920|  1969|
|    98|Williams, Judy      |  9000|   199|  1935|  1969|
|   129|Thomas, Tom         | 10000|   199|  1941|  1962|
|   157|Jones, Tim          | 12000|   199|  1940|  1960|
|   199|Bullock, J.D.       | 27000|     0|  1920|  1920|
|   430|Brunet, Paul C.     | 17674|   129|  1938|  1959|
```

```
|   843|Schmidt, Herman     | 11204|    26|  1936|  1956|
|   994|Iwano, Masahiro     | 15641|   129|  1944|  1970|
|  1330|Onstad, Richard     |  8779|    13|  1952|  1971|
|  1523|Zugnoni, Arthur A.  | 19868|   129|  1928|  1949|
|  1639|Choy, Wanda         | 11160|    55|  1947|  1970|
|  4901|Bailey, Chas M.     |  8377|    32|  1956|  1975|
|  5119|Ferro, Tony         | 13621|    55|  1939|  1963|
|  5219|Williams, Bruce     | 13374|    33|  1944|  1959|
|-----------------------------------------------------|

continue
*
```

On a "retrieve into" nothing is printed. We had to include a "print" command to see the results. Also, the relation name on a "retrieve into" must not already exist. For example, if we tried the same query again:

```
* \g
Executing . . .

5102: CREATE: duplicate relation name overpaid

continue
*
```

There are two special features about a "retrieve into". First, the result relation is automatically sorted and any duplicate tuples are removed. Second, the relation becomes part of the data base and is owned by you. If you don't want it to be saved you should remember to destroy it. The mechanism for destroying a relation will be mentioned a bit later.

So far we have only retrieved data but never changed it. INGRES supports three update commands: append, replace, and delete.

For example, to add "Tom Terrific" to the list of overpaid employees and start him off at $10000:

```
* append to overpaid(name = "Terrific, Tom",salary = 10000)
* \g
Executing . . .


continue
*
```

Notice that we specified values for only two of the six domains in "overpaid". That is fine. INGRES will automatically set numeric domains to zero and character domains to blank, if they are not specified.

Notice also that INGRES did not print anything after the query. This is true for all update commands.

Let's give everyone in overpaid a 10% raise. To do this we want to replace o.salary by 1.1 times its value. Type the query:

```
* range of o is overpaid
```

```
* replace o(salary = o.salary * 1.1)
* \g
Executing . . .


continue
*
```

While the append command requires that you give a relation name (e.g. append to overpaid), the replace and delete commands require a tuple variable. Note that the command is:

```
       replace o ( . . . )
              where . . .
```

and not:

```
       replace overpaid ( . . . )
              where . . .
```

Print the results of these last two updates:

```
* print overpaid
* \g
Executing . . .


overpaid relation

|number|name                |salary|manage|birthd|startd|
|-----------------------------------------------------|
|    10|Ross, Stanley       | 17498|   199|  1927|  1945|
|    11|Ross, Stuart        | 13273|     0|  1931|  1932|
|    13|Edwards, Peter      |  9899|   199|  1928|  1958|
|    26|Thompson, Bob       | 14299|   199|  1930|  1970|
|    32|Smythe, Carol       |  9954|   199|  1929|  1967|
|    33|Hayes, Evelyn       | 11109|   199|  1931|  1963|
|    37|Raveen, Lemont      | 13183|    26|  1950|  1974|
|    55|James, Mary         | 13199|   199|  1920|  1969|
|    98|Williams, Judy      |  9899|   199|  1935|  1969|
|   129|Thomas, Tom         | 10999|   199|  1941|  1962|
|   157|Jones, Tim          | 13199|   199|  1940|  1960|
|   199|Bullock, J.D.       | 29699|     0|  1920|  1920|
|   430|Brunet, Paul C.     | 19441|   129|  1938|  1959|
|   843|Schmidt, Herman     | 12324|    26|  1936|  1956|
|   994|Iwano, Masahiro     | 17205|   129|  1944|  1970|
|  1330|Onstad, Richard     |  9656|    13|  1952|  1971|
|  1523|Zugnoni, Arthur A.  | 21854|   129|  1928|  1949|
|  1639|Choy, Wanda         | 12275|    55|  1947|  1970|
|  4901|Bailey, Chas M.     |  9214|    32|  1956|  1975|
|  5119|Ferro, Tony         | 14983|    55|  1939|  1963|
|  5219|Williams, Bruce     | 14711|    33|  1944|  1959|
|     0|Terrific, Tom       | 11000|     0|     0|     0|
```

```
|-----------------------------------------------------|

continue
*
```

Let's fire whoever has the smallest salary:

```
* delete o where o.salary = min(o.salary) \g
Executing . . .


continue
*
```

Notice that the delete command requires a tuple variable (eg. delete o) and not a relation name.

What if we wanted to know who makes more that Tom Terrific? The query to do this is very subtle. First we use a new tuple variable called "t" which ranges over overpaid, and will be used to refer to Tom. t.name must equal "Terrific, Tom". Next, we use a tuple variable called "o" which will scan the whole relation. If we ever find an o.salary > t.salary then o.name must make more than Tom.

The complete query is:

```
* range of t is overpaid
* retrieve (o.name, osal=o.salary, tomsal = t.salary)
* where o.salary > t.salary
* and t.name = "Terrific, Tom"
* \g
* Executing . . .


|name                |osal  |tomsal|
|---------------------------------|
|Ross, Stanley       | 19247| 11000|
|Ross, Stuart        | 14600| 11000|
|Thompson, Bob       | 15728| 11000|
|Hayes, Evelyn       | 12219| 11000|
|Raveen, Lemont      | 14501| 11000|
|James, Mary         | 14518| 11000|
|Thomas, Tom         | 12098| 11000|
|Jones, Tim          | 14518| 11000|
|Bullock, J.D.       | 32668| 11000|
|Brunet, Paul C.     | 21385| 11000|
|Schmidt, Herman     | 13556| 11000|
|Iwano, Masahiro     | 18925| 11000|
|Zugnoni, Arthur A.  | 24039| 11000|
|Choy, Wanda         | 13502| 11000|
|Ferro, Tony         | 16481| 11000|
|Williams, Bruce     | 16182| 11000|
|---------------------------------|

continue
```

```
*
```

If we wanted to give Tom Terrific $50 more than anyone else, the query would be:

```
* replace o(salary = max(o.salary) + 50)
* where o.name = "Terrific, Tom"
* \g
Executing . . .


continue
*
```

Finally, to destroy a relation owned by yourself, type the command:

```
* destroy overpaid
* \g
Executing . . .


Continue
*
```

We are now ready to leave INGRES. This is done either by typing an end-of-file (control/d) or more typically use the "\q" command:

```
* \q
INGRES vers 6.1/0 logout
Tue Aug 30 14:55:20 1977
goodbye bob -- come again
```