INGRES

VERSION 6.2

REFERENCE MANUAL

7/20/79

by

John Woodfill
Nick Whyte
Mike Ubell
Polly Siegel
Dan Ries
Marc Meyer
Paula Hawthorn
Bob Epstein
Rick Berman
Eric Allman

Memorandum No. UCB/ERL M79/43

20 July 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Research Sponsored by the Air Force Office of Scientific Research Grant 78-3596, U. S. Army Research Office Grant DAAG29-76-G-0245, Office of Naval Research Contract N00014-78-C-0185, National Science Foundation Grant MCS75-03839-A01, and Joint Services Electronics Program Contract F49620-79-C-0178.

This manual is a reference manual for the INGRES data base system. It documents the use of INGRES in a very terse manner. To learn how to use INGRES, refer to the document called "A Tutorial on INGRES".

The INGRES reference manual is subdivided into four parts:

Quel	describes the commands and features which are used inside of INGRES.
Unix	describes the INGRES programs which are executable as UNIX commands.
Files	describes some of the important files used by INGRES.
Error	lists all the user generatable error messages along with some elaboration as to what they mean or what we think they mean.

Each entry in this manual has one or more of the following sections:

NAME section
This section repeats the name of the entry and gives an indication of its purpose.

SYNOPSIS section
This section indicates the form of the command (statement). The conventions which are used are as follows:

Bold face names are used to indicate reserved keywords.
Lower case words indicate generic types of information which must be supplied by the user; legal values for these names are described in the DESCRIPTION section.
Square brakets ([]) indicate that the enclosed item is optional.
Braces ({}) indicate an optional item which may be repeated. In some cases they indicate simple (non-repeated) grouping; the usage should be clear from context.

When these conventions are insufficient to fully specify the legal format of a command a more general form is given and the allowable subsets are specified in the DESCRIPTION section.

DESCRIPTION section
This section gives a detailed description of the entry with references to the generic names used in the SYNOPSIS section.

EXAMPLE section
This section gives one or more examples of the use of the entry. Most of these examples are based on the following relations:

emp(name,sal,mgr,bdate)
and
newemp(name,sal,age)
and
parts(pnum, pname, color, weight, qoh)

SEE ALSO section
This section gives the names of entries in the manual which are closely related to the current entry or which are referenced in the description of the current entry.

BUGS section
This section indicates known bugs or deficiencies in the command.

To start using INGRES you must be entered as an INGRES user; this is done by the INGRES administrator who will enter you in the "users" file (see users(files)). To start using ingres see the section on ingres(unix), quel(quel), and monitor(quel).

ACKNOWLEDGEMENTS
We would like to acknowledge the people who have worked on INGRES in the past:

William Zook
Karel Youssefi
Peter Rubinstein

Peter Kreps
Gerald Held
James Ford

FOOTNOTE

UNIX is a trademark of Bell Laboratories.

NAME

append — append tuples to a relation

SYNOPSIS

append [**to**] relname (target_list) [**where** qual]

DESCRIPTION

Append adds tuples which satisfy the qualification to *relname*. *Relname* must be the name of an existing relation. The *target_list* specifies the values of the attributes to be appended to *relname*. The domains may be listed in any order. Attributes of the result relation which do not appear in the *target_list* as result_attnames (either explicitly or by default) are assigned default values of 0, for numeric attributes, or blank, for character attributes.

Values or expressions of any numeric type may be used to set the value of a numeric type domain. Conversion to the result domain type takes place. Numeric values cannot be directly assigned to character domains. Conversion from numeric to character can be done using the **ascii** operator (see quel(quel)). Character values cannot be directly assigned to numeric domains. Use the **int1**, **int2**, etc. functions to convert character values to numeric (see quel(quel)).

The keyword **all** can be used when it is desired to append all domains of a relation.

An *append* may only be issued by the owner of the relation or a user with *append* permission on the given relation.

EXAMPLE

```
/* Make new employee Jones work for Smith */
  range of n is newemp
  append to emp(n.name, n.sal, mgr = "Smith", bdate = 1975-n.age)
                where n.name = "Jones"
/* Append the newemp1 relation to newemp */
  range of n1 is newemp1
  append to newemp(n1.all)
```

SEE ALSO

copy(quel), permit(quel), quel(quel), retrieve(quel)

DIAGNOSTICS

Use of a numeric type expression to set a character type domain or vice versa will produce diagnostics.

BUGS

Duplicate tuples appended to a relation stored as a "paged heap" (unkeyed, unstructured) are not removed.

NAME

copy — copy data into/from a relation from/into a UNIX file.

SYNOPSIS

copy relname (domname = format {, domname = format })
direction "filename"

DESCRIPTION

Copy moves data between INGRES relations and standard UNIX files. *Relname* is the name of an existing relation. In general *domname* identifies a domain in *relname*. *Format* indicates the format the UNIX file should have for the corresponding domain. *Direction* is either **into** or **from.** *Filename* is the full UNIX pathname of the file.

On a copy **from** a file to a relation, the relation cannot have a secondary index, it must be owned by you, and it must be updatable (not a secondary index or system relation).

Copy cannot be used on a relation which is a view. For a copy **into** a UNIX file, you must either be the owner of the relation or the relation must have retrieve permission for all users, or all permissions for all users.

The formats allowed by copy are:

i1,i2,i4 — The data is stored as an integer of length 1, 2, or 4 bytes in the UNIX file.

f4,f8 — The data is stored as a floating point number (either single or double precision) in the UNIX file.

c1,c2,...,c255 — The data is stored as a fixed length string of characters.

c0 — Variable length character string.

d0,d1,...,d255 — Dummy domain.

Corresponding domains in the relation and the UNIX file do not have to be the same type or length. *Copy* will convert as necessary. When converting anything except character to character, *copy* checks for overflow. When converting from character to character, *copy* will blank pad or truncate on the right as necessary.

The domains should be ordered according to the way they should appear in the UNIX file. Domains are matched according to name, thus the order of the domains in the relation and in the UNIX file does not have to be the same.

Copy also provides for variable length strings and dummy domains. The action taken depends on whether it is a copy **into** or a copy **from.** Delimitors for variable length strings and for dummy domains can be selected from the list of:

nl — new line character
tab — tab character
sp — space
nul or **null** — null character
comma — comma
colon — colon
dash — dash
lparen — left parenthesis
rparen — right parenthesis
x — any single character 'x'

The special meaning of any delimitor can be turned off by preceeding the delimitor with a '\'. The delimitor can optionally be in quotes ("delim"). This is usefully if you wish to use a single character delimitor which has special meaning to the QUEL parser.

When the *direction* is from, *copy* appends data into the relation **from** the UNIX file. Domains in the INGRES relation which are not assigned values from the UNIX file are assigned the default value of zero for numeric domains, and blank for character domains. When copying in this

direction the following special meanings apply:

c0*delim* — The data in the UNIX file is a variable length character string terminated by the delimitor *delim*. If *delim* is missing then the first comma, tab, or newline encountered will terminate the string. The delimitor is not copied.

For example:

pnum=c0 — string ending in comma, tab, or nl.
pnum=c0nl — string ending in nl.
pnum=c0sp — string ending in space.
pnum=c0"Z" — string ending in the character "Z".
pnum=c0"%" — string ending in the character "%".

A delimitor can be escaped by preceeding it with a '\'. For example, using **name = c0**, the string "Blow\, Joe," will be accepted into the domain as "Blow, Joe".

d0*delim* — The data in the UNIX file is a variable length character string delimited by *delim*. The string is read and discarded. The delimitor rules are identical for **c0** and **d0**. The domain name is ignored.

d1,d2,...,d255 — The data in the UNIX file is a fixed length character string. The string is read and discarded. The domain name is ignored.

When the direction is **into,** *copy* transfers data **into** the UNIX file from the relation. If the file already existed, it is truncated to zero length before copying begins. When copying in this direction, the following special meanings apply:

c0 — The domain value is converted to a fixed length character string and writted into the UNIX file. For character domains, the length will be the same as the domain length. For numeric domains, the standard INGRES conversions will take place as specified by the '**−i**', '**−f**', and '**−c**' flags (see ingres(unix)).

c0*delim* — The domain will be converted according to the rules for **c0** above. The one character delimitor will be inserted immediately after the domain.

d1,d2,...,d255 — The domain name is taken to be the name of the delimitor. It is written into the UNIX file 1 time for **d1,** 2 times for **d2,** etc.

d0 — This format is ignored on a copy **into.**

d0*delim* — The *delim* is written into the file. The domain name is ignored.

If no domains appear in the copy command (i.e. copy relname () into/from "filename") then *copy* automatically does a "bulk" copy of all domains, using the order and format of the domains in the relation. This is provided as a convenient shorthand notation for copying and restoring entire relations.

To *copy* into a relation, you must be the owner or all users must have all permissions set. Correspondingly, to *copy* from a relation you must own the relation or all users must have at least retrieve permission on the relation. Also, you may not *copy* a view.

EXAMPLE

```
/* Copy data into the emp relation */
  copy emp (name=c10,sal=f4,bdate=i2,mgr=c10,xxx=d1)
          from "/mnt/me/myfile"

/* Copy employee names and their salaries into a file */
  copy emp (name=c0,comma=d1,sal=c0,nl=d1)
          into "/mnt/you/yourfile"

/* Bulk copy employee relation into file */
  copy emp ()
          into "/mnt/ours/ourfile"
```

```
/* Bulk copy employee relation from file */
  copy emp ()
          from "/mnt/thy/thyfile"
```

SEE ALSO

append(quel), create(quel), quel(quel), permit(quel), view(quel), ingres(unix)

BUGS

Copy stops operation at the first error.

When specifying *filename*, the entire UNIX directory pathname must be provided, since INGRES operates out of a different directory than the user's working directory at the time INGRES is invoked.

NAME

create — create a new relation

SYNOPSIS

create relname (domname*1* = format {, domname*2* = format })

DESCRIPTION

Create will enter a new relation into the data base. The relation will be "owned" by the user and will be set to expire after seven days. The name of the relation is *relname* and the domains are named *domname1, domname2,* etc. The domains are created with the type specified by *format.* Formats are described in the quel(quel) manual section.

The relation is created as a paged heap with no data initially in it.

A relation can have no more than 49 domains. A relation cannot have the same name as a system relation.

EXAMPLE

```
/* Create relation emp with domains name, sal and bdate */
  create emp (name = c10, salary = f4, bdate = i2)
```

SEE ALSO

append(quel), copy(quel), destroy(quel), save(quel)

BUGS

NAME

define — define subschema

SYNOPSIS

define view name (target list) [**where** qual]
define permit oplist { **on** | **of** | **to** } var [(attlist)]
to name [**at** term] [**from** time **to** time]
[**on** day **to** day] [**where** qual]
define integrity on var **is** qual

DESCRIPTION

The *define* statement creates entries for the subschema definitions. See the manual sections listed below for complete descriptions of these commands.

SEE ALSO

integrity(quel), permit(quel), view(quel)

NAME

delete — delete tuples from a relation

SYNOPSIS

delete tuple_variable [**where** qual]

DESCRIPTION

Delete removes tuples which satisfy the qualification *qual* from the relation that they belong to. The *tuple_variable* must have been declared to range over an existing relation in a previous *range* statement. *Delete* does not have a *target_list*. The *delete* command requires a tuple variable from a *range* statement, and not the actual relation name. If the qualification is not given, the effect is to delete all tuples in the relation. The result is a valid, but empty relation.

To *delete* tuples from a relation, you must be the owner of the relation, or have *delete* permission on the relation.

EXAMPLE

```
/* Remove all employees who make over $30,000 */
  range of e is emp
  delete e where e.sal > 30000
```

SEE ALSO

destroy(quel), permit(quel), quel(quel), range(quel)

BUGS

NAME

destroy — destroy existing relation(s)

SYNOPSIS

destroy relname {, relname}
destroy [**permit** | **integrity**] relname [integer {, integer}| **all**]

DESCRIPTION

Destroy removes relations from the data base, and removes constraints or permissions from a relation. Only the relation owner may destroy a relation or its permissions and integrity constraints. A relation may be emptied of tuples, but not destroyed, using the delete statement or the modify statement.

If the relation being destroyed has secondary indices on it, the secondary indices are also destroyed. Destruction of just a secondary index does not affect the primary relation it indexes.

To destroy individual permissions or constraints for a relation, the *integer* arguments should be those printed by a **help permit** (for **destroy permit**) or a **help integrity** (for **destroy integrity**) on the same relation. To destroy all constraints or permissions, the **all** keyword may be used in place of individual integers. To destroy constraints or permissions, either the *integer* arguments or the **all** keyword must be present.

EXAMPLE

```
/* Destroy the emp relation */
   destroy emp
   destroy emp, parts

/* Destroy some permissions on parts, and all integrity
 * constraints on employee
 */
   destroy permit parts 0, 4, 5
   destroy integrity employee
```

SEE ALSO

create(quel), delete(quel), help(quel), index(quel), modify(quel)

NAME

help — get information about how to use INGRES or about relations in the database.

SYNOPSIS

help [relname] ["section"] {, relname}{, "section"}
help view [relname {, relname}]
help protect [relname {, relname}]
help integrity [relname {, relname}]

DESCRIPTION

Help may be used to obtain sections of this manual, information on the content of the current data base, information about specific relations in the data base, view definitions, or protection and integrity constraints on a relation. The legal forms are as follow:

help "*section* " — Produces a copy of the specified section of the INGRES Reference Manual, and prints it on the standard output device.
help — Gives information about all relations that exist in the current database.
help *relname* {, *relname*} — Gives information about the specified relations.
help "" — Gives the table of contents.
help view *relname* {, *relname*} — Prints view definitions of specified views.
help protect *relname* {, *relname*} — Prints permissions on specified relations.
help integrity *relname* {, *relname*} — Prints integrity constraints on specified relations.

The **protect** and **integrity** forms print out unique identifiers for each constraint. These identifiers may be used to remove the constraints with the *destroy* statement.

EXAMPLE

```
help
help help  /* prints this page of the manual */
help quel
help emp
help emp, parts, "help", supply
help view overp_view
help protect parts, employee
help integrity parts, employee
```

SEE ALSO

destroy(quel)

BUGS

Alphabetics appearing within the section name must be in lower-case to be recognized.

NAME

index — create a secondary index on an existing relation.

SYNOPSIS

index on relname **is** indexname (domain*1* [,domain*2*])

DESCRIPTION

Index is used to create secondary indices on existing relations in order to make retrieval and update with secondary keys more efficient. The secondary key is constructed from relname domains 1, 2,...,6 in the order given. Only the owner of a relation is allowed to create secondary indices on that relation.

In order to maintain the integrity of the index, users will NOT be allowed to directly update secondary indices. However, whenever a primary relation is changed, its secondary indices will be automatically updated by the system. Secondary indices may be modified to further increase the access efficiency of the primary relation. When an index is first created, it is automatically modified to an isam storage structure on all its domains. If this structure is undesirable, the user may override the default isam structure by using the **−n** switch (see ingres(unix)), or by entering a *modify* command directly.

If a *modify* or *destroy* command is used on *relname*, all secondary indices on *relname* are destroyed.

Secondary indices on other indices, or on system relations are forbidden.

EXAMPLE

```
/* Create a secondary index called "x" on relation "emp" */
  index on emp is x(mgr,sal)
```

SEE ALSO

copy(quel), destroy(quel), modify(quel)

BUGS

At most 6 domains may appear in the key.

The *copy* command cannot be used to copy into a relation which has secondary indices.

The default structure isam is a poor choice for an index unless the range of retrieval is small.

NAME

integrity — define integrity constraints

SYNOPSIS

define integrity on var **is** qual

DESCRIPTION

The *integrity* statement adds an integrity constraint for the relation specified by *var*. After the constraint is placed, all updates to the relation must satisfy *qual*. *Qual* must be true when the *integrity* statement is issued or else a diagnostic is issued and the statement is rejected.

In the current implementation, *integrity* constraints are not flagged — bad updates are simply (and silently) not performed.

Qual must be a single variable qualification and may not contain any aggregates.

integrity statement may be issued only by the relation owner.

EXAMPLE

```
/* Ensure all employees have positive salaries */
  range of e is employee
  define integrity on e is e.salary > 0
```

SEE ALSO

destroy(quel)

NAME

macros — terminal monitor macro facility

DESCRIPTION

The terminal monitor macro facility provides the ability to tailor the QUEL language to the user's tastes. The macro facility allows strings of text to be removed from the query stream and replaced with other text. Also, some built in macros change the environment upon execution.

Basic Concepts

All macros are composed of two parts, the *template* part and the *replacement* part. The template part defines when the macro should be invoked. For example, the template "ret" causes the corresponding macro to be invoked upon encountering the word "ret" in the input stream. When a macro is encountered, the template part is removed and replaced with the replacement part. For example, if the replacement part of the "ret" macro was "retrieve", then all instances of the word "ret" in the input text would be replaced with the word "retrieve", as in the statement

ret (p.all)

Macros may have parameters, indicated by a dollar sign. For example, the template "get $1" causes the macro to be triggered by the word "get" followed by any other word. The word following "get" is remembered for later use. For example, if the replacement part of the "get" macro where

retrieve (p.all) where p.pnum = $1

then typing "get 35" would retrieve all information about part number 35.

Defining Macros

Macros can be defined using the special macro called "define". The template for the define macro is (roughly)

{define; $t; $r}

where $t and $r are the template and replacement parts of the macro, respectively.

Let's look at a few examples. To define the "ret" macro discussed above, we would type:

{define; ret; retrieve}

When this is read, the macro processor removes everything between the curly braces and updates some tables so that "ret" will be recognized and replaced with the word "retrieve". The define macro has the null string as replacement text, so that this macro seems to disappear.

A useful macro is one which shortens range statements. It can be defined with

{define; rg $v $r; range of $v is $r}

This macro causes the word "rg" followed by the next two words to be removed and replaced by the words "range of", followed by the first word which followed "rg", followed by the word "is", followed by the second word which followed "rg". For example, the input

rg p parts

becomes the same as

range of p is parts

Evaluation Times

When you type in a define statement, it is not processed immediately, just as queries are saved rather than executed. No macro processing is done until the query buffer is evaluated. The commands \go, \list, and \eval evaluate the query buffer. \go sends the results to INGRES, \list prints them on your terminal, and \eval puts the result back into the query buffer.

It is important to evaluate any define statements, or it will be exactly like you did not type them in at all. A common way to define macros is to type

```
{define . . . }
\eval
\reset
```

If the \eval was left out, there is no effect at all.

Quoting

Sometimes strings must be passed through the macro processor without being processed. In such cases the grave and acute accent marks (` and ´) can be used to surround the literal text. For example, to pass the word "ret" through without converting it to "retrieve" we could type

```
`ret´
```

Another use for quoting is during parameter collection. If we want to enter more than one word where only one was expected, we can surround the parameter with accents.

The backslash character quotes only the next character (like surrounding the character with accents). In particular, a grave accent can be used literally by preceeding it with a backslash.

Since macros can normally only be on one line, it is frequently useful to use a backslash at the end of the line to hide the newline. For example, to enter the long "get" macro, you might type:

```
{define; get $n; retrieve (e.all) \
where e.name = "$n"}
```

The backslash always quotes the next character even when it is a backslash. So, to get a real backslash, use two backslashes.

More Parameters

Parameters need not be limited to the word following. For example, in the template descriptor for define:

```
{define; $t; $r}
```

the $t parameter ends at the first semicolon and the $r parameters ends at the first right curly brace. The rule is that the character which follows the parameter specifier terminates the parameter; if this character is a space, tab, newline, or the end of the template then one word is collected.

As with all good rules, this one has an exception. Since system macros are always surrounded by curly braces, the macro processor knows that they must be properly nested. Thus, in the statement

```
{define; x; {sysfn}}
```

The first right curly brace will close the "sysfn" rather than the "define". Otherwise this would have to be typed

```
{define; x; `{sysfn}´}
```

Words are defined in the usual way, as strings of letters and digits plus the underscore character.

Other Builtin Macros

There are several other macros built in to the macro processor. In the following description, some of the parameter specifiers are marked with two dollar signs rather than one; this will be discussed in the section on prescanning below.

{define; $$t; $$r} defines a macro as discussed above. Special processing occurs on the template part which will be discussed in a later section.

{rawdefine; $$t; $$r} is another form of define, where the special processing does not take place.

{remove; $$n} removes the macro with name $n. It can remove more than one macro, since it actually removes all macros which might conflict with $n under some circumstance. For example, typing

```
{define; get part $n; . . . }
{define; get emp $x; . . . }
{remove; get}
```

would cause both the get macros to be removed. A call to

```
{remove; get part}
```

would have only removed the first macro.

{type $$s} types $s onto the terminal.

{read $$s} types $s and then reads a line from the terminal. The line which was typed replaces the macro. A macro called "{readcount}" is defined containing the number of characters read. A control-D (end of file) becomes −1, a single newline becomes zero, and so forth.

{readdefine; $$n; $$s} also types $s and reads a line, but puts the line into a macro named $n. The replacement text is the count of the number of characters in the line. {readcount} is still defined.

{ifsame; $$a; $$b; $t; $f} compares the strings $a and $b. If they match exactly then the replacement text becomes $t, otherwise it becomes $f.

{ifeq; $$a; $$b; $t; $f} is similar, but the comparison is numeric.

{ifgt; $$a; $$b; $t; $f} is like ifeq, but the test is for $a strictly greater than $b.

{substr; $$f; $$t; $$s} returns the part of $s between character positions $f and $t, numbered from one. If $f or $t are out of range, they are moved in range as much as possible.

{dump; $$n} returns the value of the macro (or macros) which match $n (using the same algorithm as remove). The output is a rawdefine statement so that it can be read back in. {dump} without arguments dumps all macros.

Metacharacters

Certain characters are used internally. Normally you will not even see them, but they can appear in the output of a dump command, and can sometimes be used to create very fancy macros.

\| matches any number of spaces, tabs, or newlines. It will even match zero, but only between words, as can occur with punctuation. For example, \| will match the spot between the last character of a word and a comma following it.

\^ matches exactly one space, tab, or newline.

\& matches exactly zero spaces, tabs, or newlines, but only between words.

The Define Process

When you define a macro using define, a lot of special processing happens. This processing is such that define is not functionally complete, but still adequate for most requirements. If more power is needed, rawdefine can be used; however, rawdefine is particularly difficult to use correctly, and should only be used by gurus.

In define, all sequences of spaces, tabs, and newlines in the template, as well as all "non-spaces" between words, are turned into a single \| character. If the template ends with a parameter, the \& character is added at the end.

If you want to match a real tab or newline, you can use \t or \n respectively. For example, a macro which reads an entire line and uses it as the name of an employee would be defined with

```
{define; get $n\n; \
   ret (e.all) where e.name = "$n"}
```

This macro might be used by typing

```
get *Stan*
```

to get all information about everyone with a name which included "Stan". By the way, notice that it is ok to nest the "ret" macro inside the "get" macro.

Parameter Prescan

Sometimes it is useful to macro process a parameter before using it in the replacement part. This is particularly important when using certain builtin macros.

For prescan to occur, two things must be true: first, the parameter must be specified in the template with two dollar signs instead of one, and second, the actual parameter must begin with an "at" sign ("@") (which is stripped off).

For an example of the use of prescan, see "Special Macros" below.

Special Macros

Some special macros are used by the terminal monitor to control the environment and return results to the user.

{begintrap} is executed at the beginning of a query.

{endtrap} is executed after the body of a query is passed to INGRES.

{continuetrap} is executed after the query completes. The difference between this and endtrap is that endtrap occurs after the query is submitted, but before the query executes, whereas continuetrap is executed after the query executes.

{editor} can be defined to be the pathname of an editor to use in the \edit command.

{shell} can be defined to be the pathname of a shell to use in the \shell command.

{tuplecount} is set after every query (but before continuetrap is sprung) to be the count of the number of tuples which satisfied the qualification of the query in a retrieve, or the number of tuples changed in an update. It is not set for DBU functions. If multiple queries are run at once, it is set to the number of tuples which satisfied the last query run.

For example, to print out the number of tuples touched automatically after each query, you could enter:

```
{define; {begintrap}; {remove; {tuplecount}}}
{define; {continuetrap}; \
   {ifsame; @{tuplecount}; {tuplecount};; \
     {type @{tuplecount} tuples touched}}}
```

SEE ALSO

monitor(quel)

NAME

modify — convert the storage structure of a relation

SYNOPSIS

modify relname **to** storage-structure [**on** key1 [: *sortorder*] [{ , key2 [: *sortorder*] }]] [**where** [**fillfactor** = *n*] [, **minpages** = *n*] [, **maxpages** = *n*]]

DESCRIPTION

Relname is modified to the specified storage structure. Only the owner of a relation can modify that relation. This command is used to increase performance when using large or frequently referenced relations. The storage structures are specified as follows:

isam — indexed sequential storage structure
cisam — compressed isam
hash — random hah storage structure
chash — compressed hash
heap — unkeyed and unstructured
cheap — compressed heap
heapsort — heap with tuples sorted and duplicates removed
cheapsort — compressed heapsort
truncated — heap with all tuples deleted

The paper "Creating and Maintaining a Database in INGRES" (ERL Memo M77—71) discusses how to select storage structures based on how the relation is used.

The current compression algorithm only suppresses trailing blanks in character fields. A more effective compression scheme may be possible, but tradeoffs between that and a larger and slower compression algorithm are not clear.

If the *on* phrase is omitted when modifying to isam, cisam, hash or chash, the relation will automatically be keyed on the first domain. When modifying to heap or cheap the *on* phrase must be omitted. When modifying to heapsort or cheapsort the *on* phrase is optional.

When a relation is being sorted (isam, cisam, heapsort and cheapsort), the primary sort keys will be those specified in the *on* phrase (if any). The first key after the *on* phrase will be the most significant sort key and each successive key specified will be the next most significant sort key. Any domains not specified in the *on* phrase will be used as least significant sort keys in domain number sequence.

When a relation is modified to heapsort or cheapsort, the *sortorder* can be specified to be **ascending** or **descending.** The default is always **ascending.** Each key given in the *on* phrase can be optionally modified to be:

key:descending

which will cause that key to be sorted in descending order. For completeness, **ascending** can be specified after the colon (':'), although this is unnecessary since it is the default. **Descending** can be abbreviated by a single '**d**' and, correspondingly, **ascending** can be abreviated by a single '**a**'.

Fillfactor specifies the percentage (from 1 to 100) of each primary data page that should be filled with tuples, under ideal conditions. *Fillfactor* may be used with isam, cisam, hash and chash. Care should be taken when using large fillfactors since a non-uniform distribution of key values could cause overflow pages to be created, and thus degrade access performance for the relation.

Minpages specifies the minimum number of primary pages a hash or chash relation must have. *Maxpages* specifies the maximum number of primary pages a hash or chash relation may have. *Minpages* and *maxpages* must be at least one. If both **minpages** and **maxpages** are specified in a modify, **minpages** cannot exceed **maxpages.**

Default values for **fillfactor**, **minpages**, and **maxpages** are as follows:

	FILLFACTOR	*MINPAGES*	*MAXPAGES*
hash	50	10	no limit
chash	75	1	no limit
isam	80	NA	NA
cisam	100	NA	NA

EXAMPLES

```
/* modify the emp relation to an indexed
      sequential storage structure with
      "name" as the keyed domain */

modify emp to isam on name

/* if "name" is the first domain of the emp relation,
      the same result can be achieved by */

modify emp to isam

/* do the same modify but request a 60% occupancy
      on all primary pages */

modify emp to isam on name where fillfactor = 60

/* modify the supply relation to compressed hash
      storage structure with "num" and "quan"
      as keyed domains */

modify supply to chash on num, quan

/* now the same modify but also request 75% occupancy
      on all primary, a minimum of 7 primary pages
      pages and a maximum of 43 primary pages */

modify supply to chash on num, quan
      where fillfactor = 75, minpages = 7,
      maxpages = 43

/* again the same modify but only request a minimum
      of 16 primary pages */

modify supply to chash on num, quan
      where minpages = 16

/* modify parts to a heap storage structure */

modify parts to heap

/* modify parts to a heap again, but have tuples
      sorted on "pnum" domain and have any duplicate
      tuples removed */

modify parts to heapsort on pnum

/* modify employee in ascending order by manager,
      descending order by salary and have any
      duplicate tuples removed */

modify employee to heapsort on manager, salary:descending
```

SEE ALSO

sysmod(unix)

NAME

monitor — interactive terminal monitor

DESCRIPTION

The interactive terminal monitor is the primary front end to INGRES. It provides the ability to formulate a query and review it before issuing it to INGRES. If changes must be made, one of the UNIX text editors may be called to edit the **query buffer.**

Messages and Prompts.

The terminal monitor gives a variety of messages to keep the user informed of the status of the monitor and the query buffer.

As the user logs in, a login message is printed. This typically tells the version number and the login time. It is followed by the dayfile, which gives information pertinant to users.

When INGRES is ready to accept input, the message "go" is printed. This means that the query buffer is empty. The message "continue" means that there is information in the query buffer. After a \go command the query buffer is automatically cleared if another query is typed in, unless a command which affects the query buffer is typed first. These commands are \append, \edit, \print, \list, \eval, and \go. For example, typing

```
help parts
\go
print parts
```

results in the query buffer containing

```
print parts
```

whereas

```
help parts
\go
\print
print parts
```

results in the query buffer containing

```
help parts
print parts
```

An asterisk is printed at the beginning of each line when the monitor is waiting for the user to type input.

Commands

There are a number of commands which may be entered by the user to affect the query buffer or the user's environment. They are all preceeded by a backslash ("\"), and all are executed immediately (rather than at execution time like queries).

Some commands may take a filename, which is defined as the first significant character after the end of the command until the end of the line. These commands may have no other commands on the line with them. Commands which do not take a filename may be stacked on the line; for example

```
\date\go\date
```

will give the time before and after execution of the current query buffer.

\r
\reset — Erase the entire query (reset the query buffer). The former contents of the buffer are irretrieveably lost.

\p
\print — Print the current query. The contents of the buffer are printed on the user's terminal.

\l
\list — Print the current query as it will appear after macro processing. Any side effects of macro processing, such as macro definition, will occur.

\eval — Macro process the query buffer and replace the query buffer with the result. This is just like \list except that the output is put into the query buffer instead of to the terminal.

\e
\ed
\edit
\editor — Enter the UNIX text editor (see ED in the UNIX Programmer's Manual); use the ED command 'w' followed by 'q' to return to the INGRES monitor. If a filename is given, the editor is called with that file instead of the query buffer. If the macro "{editor}" is defined, that macro is used as the pathname of an editor, otherwise "/bin/ed" is used. It is important that you do not use the "e" command inside the editor; if you do the (obscure) name of the query buffer will be forgotten.

\g
\go — Process the current query. The contents of the buffer are macro processed, transmitted to INGRES, and run.

\a
\append — Append to the query buffer. Typing \a after completion of a query will override the auto-clear feature and guarantees that the query buffer will not be reset.

\time
\date — Print out the current time of day.

\s
\sh
\shell — Escape to the UNIX shell. Typing a control-d will cause you to exit the shell and return to the INGRES monitor. If there is a filename specified, that filename is taken as a shell file which is run with the query buffer as the parameter "$1". If no filename is given, an interactive shell is forked. If the macro "{shell}" is defined, it is used as the pathname of a shell; otherwise, "/bin/sh" is used.

\q
\quit — Exit from INGRES.

\cd
\chdir — Change the working directory of the monitor to the named directory.

\i
\include
\read — Switch input to the named file. Backslash characters in the file will be processed as read.

\w
\write — Write the contents of the query buffer to the named file.

\branch — Transfer control within a \include file. See the section on branching below.

\mark — Set a label for \branch.

\<any other character> — Ignore any possible special meaning of character following '\'. This allows the '\' to be input as a literal character. (See also quel(quel) - strings). It is important to note that backslash escapes are sometimes eaten up by the macro processor also; in general, send two backslashes if you want a backslash sent (even this is too simplistic [sigh] — try to avoid using backslashes at all).

Macros

For simplicity, the macros are described in the section macros(quel).

Branching

The \branch and \mark commands permit arbitrary branching within a \include file (similar to the "goto" and ":" commands in the shell). \mark should be followed with a label. \branch

should be followed with either a label, indicating unconditional branch, or an expression preceeded by a question mark, followed by a label, indicating a conditional branch. The branch is taken if the expression is greater than zero. For example,

```
\branch ?{tuplecount}<=0 notups
```

branches to label "notups" if the "{tuplecount}" macro is less than or equal to zero.

The expressions usable in \branch statements are somewhat restricted. The operators +, −, *, /, <=, >=, <, >, =, and != are all defined in the expected way. The left unary operator "!" can be used as to indicate logical negation. There may be no spaces in the expression, since a space terminates the expression.

Initialization

At initialization (login) time a number of initializations take place. First, a macro called "{pathname}" is defined which expands to the pathname of the INGRES subtree (normally "/mnt/ingres"); it is used by system routines such as demodb. Second, the initialization file .../files/startup is read. This file is intended to define system-dependent parameters, such as the default editor and shell. Third, a user dependent initialization file, specified by a field in the users file, is read and executed. This is normally set to the file ".ingres" in the user's home directory. The startup file might be used to define certain macros, execute common range statements, and soforth. Finally, control is turned over to the user's terminal.

An interrupt while executing either of the initialization files restarts execution of that step.

Flags

Certain flags may be included on the command line to INGRES which affect the operation of the terminal monitor. The −a flag disables the autoclear function. This means that the query buffer will never be automatically cleared; equivalently, it is as though a \append command were inserted after every \go. Note that this means that the user must explicitly clear the query buffer using \reset after every query. The −d flag turns off the printing of the dayfile. The −s flag turns off printing of all messages (except errors) from the monitor, including the login and logout messages, the dayfile, and prompts. It is used for executing "canned queries", that is, queries redirected from files.

SEE ALSO

ingres(unix), quel(quel), macros(quel)

DIAGNOSTICS

go	You may begin a fresh query.
continue	The previous query is finished and you are back in the monitor.
Executing . . .	The query is being processed by INGRES.
>>ed	You have entered the UNIX text editor.
>>sh	You have escaped to the UNIX shell.

Funny character nnn converted to blank

INGRES maps non-printing ASCII characters into blanks; this message indicates that one such conversion has just been made.

INCOMPATIBILITIES

Note that the construct

```
\rprint parts
```

(intended to reset the query buffer and then enter "print parts") no longer works, since "rprint" appears to be one word.

BUGS

NAME

permit — add permissions to a relation

SYNOPSIS

define permit oplist { **on** | **of** | **to** } var [(attlist)]
to name [**at** term] [**from** time **to** time]
[**on** day **to** day] [**where** qual]

DESCRIPTION

The *permit* statement extends the current permissions on the relation specified by *var*. *Oplist* is a comma separated list of possible operations, which can be **retrieve**, **replace**, **delete**, **append**, or **all**; *all* is a special case meaning all permissions. *Name* is the login name of a user or the word **all**. *Term* is a terminal name of the form 'tty*x*' or the keyword **all**; omitting this phrase is equivalent to specifying *all*. *Times* are of the form 'hh:mm' on a twenty-four hour clock which limit the times of the day during which this permission applies. *Days* are three-character abbreviations for days of the week. The *qual* is appended to the qualification of the query when it is run.

Separate parts of a single *permit* statement are conjoined (ANDed). Different *permit* statements are disjoined (ORed). For example, if you include

. . . to eric at tty4 . . .

the *permit* applies only to eric when logged in on tty4, but if you include two *permit* statements

. . . to eric at all . . .
. . . to all at tty4 . . .

then when eric logs in on tty4 he will get the union of the permissions specified by the two statements. If eric logs in on ttyd he will get only the permissions specified in the first *permit* statement, and if bob logs in on tty4 he will get only the permissions specified in the second *permit* statment.

The *permit* statement may only be issued by the owner of the relation. Although a user other than the DBA may issue a *permit* statement, it is useless because noone else can access her relations anyway.

Permit statements do not apply to the owner of a relation or to views.

The statements

```
define permit all on x to all
define permit retrieve of x to all
```

with no further qualification are handled as special cases and are thus particularly efficient.

EXAMPLES

```
range of e is employee
define permit retrieve of e (name, sal) to marc
        at ttyd from 8:00 to 17:00
        on Mon to Fri
        where e.mgr = "marc"

range of p is parts
define permit retrieve of e to all
```

SEE ALSO

destroy(quel)

NAME

print — print relation(s)

SYNOPSIS

print relname {, relname}

DESCRIPTION

Print displays the contents of each relation specified on the terminal (standard output). The formats for various types of domains can be defined by the use of switches when *ingres* is invoked. Domain names are truncated to fit into the specified width.

To print a relation one must either be the owner of the relation, or the relation must have "retrieve to all" or "all to all" permissions.

See ingres(quel) for details.

EXAMPLE

```
/* Print the emp relation */
  print emp
  print emp, parts
```

SEE ALSO

permit(quel), retrieve(quel), ingres(unix), printr(unix) handle long lines of output correctly — no wrap around.

Print should have more formating features to make printouts more readable.

Print should have an option to print on the line printer.

NAME

quel – **QUE**ry **L**anguage for INGRES

DESCRIPTION

The following is a description of the general syntax of QUEL. Individual QUEL statements and commands are treated separately in the document; this section describes the syntactic classes from which the constituent parts of QUEL statements are drawn.

1. Comments

A comment is an arbitrary sequence of characters bounded on the left by "/*" and on the right by "*/":

/* This is a comment */

2. Names

Names in QUEL are sequences of no more than 12 alphanumeric characters, starting with an alphabetic. Underscore (_) is considered an alphabetic. All upper–case alphabetics appearing anywhere except in strings are automatically and silently mapped into their lower–case counterparts.

3. Keywords

The following identifiers are reserved for use as keywords and may not be used otherwise:

abs	**all**	**and**
any	**append**	**ascii**
at	**atan**	**avg**
avgu	**by**	**concat**
copy	**cos**	**count**
countu	**create**	**define**
delete	**destroy**	**exp**
float4	**float8**	**from**
gamma	**help**	**in**
index	**int1**	**int2**
int4	**integrity**	**into**
is	**log**	**max**
min	**mod**	**modify**
not	**of**	**on**
onto	**or**	**permit**
print	**range**	**replace**
retrieve	**save**	**sin**
sqrt	**sum**	**sumu**
to	**unique**	**until**
view	**where**	

4. Constants

There are three types of constants, corresponding to the three data types available in QUEL for data storage.

4.1. String constants

Strings in QUEL are sequences of no more than 255 arbitrary ASCII characters bounded by double quotes (" "). Upper case alphabetics within strings are accepted literally. Also, in order to imbed quotes within strings, it is necessary to prefix them with '\' . The same convention applies to '\' itself.

Only printing characters are allowed within strings. Non-printing characters (i.e. control characters) are converted to blanks.

4.2. Integer constants

Integer constants in QUEL range from −2,147,483,647 to +2,147,483,647. Integer constants beyond that range will be converted to floating point. If the integer is greater than 32,767 or less than −32,767 then it will be left as a two byte integer. Otherwise it is converted to a four byte integer.

4.3. Floating point constants

Floating constants consist of an integer part, a decimal point, and a fraction part or scientific notation of the following format:

{<dig>} [.<dig>] [e|E [+|−] {<dig>}]

Where <dig> is a digit, [] represents zero or one, {} represents zero or more, and | represents alternation. An exponent with a missing mantissa has a mantissa of 1 inserted. There may be no extra characters embedded in the string. Floating constants are taken to be double-precision quantities with a range of approximately -10^{38} to 10^{38} and a precision of 17 decimal digits.

5. Attributes

An attribute is a construction of the form:

variable.domain

Variable identifies a particular relation and can be thought of as standing for the rows or tuples of that relation. A variable is associated with a relation by means of a *range* statement. *Domain* is the name of one of the columns of the relation over which the variable ranges. Together they make up an attribute, which represents values of the named domain.

6. Arithmetic operators

Arithmetic operators take numeric type expressions as operands. Unary operators group right to left; binary operators group left to right. The operators (in order of descending precedence) are:

+,−	(unary) plus, minus
**	exponentiation
*,/	multiplication, division
+,−	(binary) addition, subtraction

Parentheses may be used for arbitrary grouping. Arithmetic overflow and divide by zero are not checked on integer operations. Floating point operations are checked for overflow, underflow, and divide by zero only if the appropriate machine hardware exists and has been enabled.

7. Expressions (a_expr)

An expression is one of the following:

constant
attribute
functional expression
aggregate or aggregate function
a combination of numeric expressions and arithmetic operators

For the purposes of this document, an arbitrary expression will be refered to by the name *a_expr*.

8. Formats

Every *a_expr* has a format denoted by a letter (**c**, **i**, or **f**, for character, integer, or floating data types respectively) and a number indicating the number of bytes of storage occupied. Formats currently supported are listed below. The ranges of numeric types are indicated in parentheses.

c1 – c255	character data of length 1–255 characters
i1	1-byte integer (–128 to +127)
i2	2-byte integer (–32768 to +32767)
i4	4-byte integer (–2,147,483,648 to +2,147,483,647)
f4	4-byte floating (-10^{38} to $+10^{38}$, 7 decimal digit precision)
f4	8-byte floating (-10^{38} to $+10^{38}$, 17 decimal digit precision)

One numeric format can be converted to or substituted for any other numeric format.

9. Type Conversion.

When operating on two numeric domains of different types, INGRES converts as necessary to make the types identical.

When operating on an integer and a floating point number, the integer is converted to a floating point number before the operation. When operating on two integers of different sizes, the smaller is converted to the size of the larger. When operating on two floating point number of different size, the larger is converted to the smaller.

The following table summarizes the possible combinations:

	i1	i2	i4	f4	f8
i1 –	i1	i2	i4	f4	f8
i2 –	i2	i2	i4	f4	f8
i4 –	i4	i4	i4	f4	f8
f4 –	f4	f4	f4	f4	f4
f8 –	f8	f8	f8	f4	f8

INGRES provides five type conversion operators specifically for overriding the default actions. The operators are:

int1(a_expr)	result type i1
int2(a_expr)	result type i2
int4(a_expr)	result type i4
float4(a_expr)	result type f4
float8(a_expr)	result type f8

The type conversion operators convert their argument a_expr to the requested type. *A_expr* can be anything including character. If a character value cannot be converted, an error occures and processing is halted. This can happen only if the syntax of the character value is incorrect.

Overflow is not checked on conversion.

10. Target_list

A target list is a parenthesized, comma separated list of one or more elements , each of which must be of one of the following forms:

a) *result_attname* **is** *a_expr*

Result_attname is the name of the attribute to be created (or an already existing attribute name in the case of update statements.) The equal sign ("=") may be used interchangeably with **is**. In the case where *a_expr* is anything other than a single attribute, this form must be used to assign a result name to the expression.

b) *attribute*

In the case of a *retrieve*, the resultant domain will acquire the same name as that of the attribute being retrieved. In the case of update statements (*append, replace*), the relation being updated must have a domain with exactly that name.

Inside the target list the keyword **all** can be used to represent all domains. For example:

```
range of e is employee
retrieve (e.all) where e.salary > 10000
```

will retrieve all domains of employee for those tuples which satisfy the qualification. **All** can be used in the target list of a *retrieve* or an **append.** The domains will be inserted in their "create" order, that is, the same order they were listed in the *create* statement.

11. Comparison operators

Comparison operators take arbitrary expressions as operands.

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

They are all of equal precedence. When comparisons are made on character attributes, all blanks are ignored.

12. Logical operators

Logical operators take clauses as operands and group left-to-right:

not	(logical not; negation)
and	(logical and; conjunction)
or	(logical or; disjunction)

Not has the highest precedence of the three. **And** and **or** have equal precedence. Parentheses may be used for arbitrary grouping.

13. Qualification (qual)

A *qualification* consists of any number of clauses connected by logical operators. A clause is a pair of expressions connected by a comparison operator:

a_expr comparison_operator a_expr

Parentheses may be used for arbitrary grouping. A qualification may thus be:

clause
not *qual*
qual **or** *qual*
qual **and** *qual*
(*qual*)

14. Functional expressions

A *functional expression* consists of a function name followed by a parenthesized (list of) operand(s). Functional expressions can be nested to any level. In the following list of functions supported (*n*) represents an arbitrary numeric type expression. The format of the result is indicated on the right.

abs(*n*) –	same as *n* (absolute value)
ascii(*n*) –	character string (converts numeric to character)
atan(*n*) –	f8 (arctangent)
concat(*a,b*) –	character (character concatenation. See 16.2)
cos(*n*) –	f8 (cosine)
exp(*n*) –	f8 (exponential of *n*)
gamma(*n*) –	f8 (log gamma)
log(*n*) –	f8 (natural logarithm)
mod(*n,b*) –	same as *b* (*n* modulo *b*. *n* and *b* must be i1, i2, or i4)
sin(*n*) –	f8 (sine)
sqrt(*n*) –	f8 (square root)

15. Aggregate expressions

Aggregate expressions provide a way to aggregate a computed expression over a set of tuples.

15.1. Aggregation operators

The definitions of the aggregates are listed below.

count –	(i4) count of occurrences
countu –	(i4) count of unique occurrences
sum –	summation
sumu –	summation of unique values
avg –	(f8) average (sum/count)
avgu –	(f8) unique average (sumu/countu)
max –	maximum
min –	minimum
any –	(i2) value is 1 if any tuples satisfy the qualification, else it is 0

15.2. Simple aggregate

aggregation_operator (*a_expr* [**where** *qual*])

A simple aggregate evaluates to a single scalar value. *A_expr* is aggregated over the set of tuples satisfying the qualification (or all tuples in the range of the expression if no qualification is present). Operators *sum* and *avg* require numeric type *a_expr; count, any, max* and *min* permit a character type attribute as well as numeric type *a_expr.*

Simple aggregates are completely local. That is, they are logically removed from the query, processed separately, and replaced by their scalar value.

15.3. "**any**" aggregate

It is sometimes useful to know if any tuples satisfy a particular qualification. One way of doing this is by using the aggregate *count* and checking whether the return is zero or non-zero. Using *any* instead of *count* is more efficient since processing is stopped, if possible, the first time a tuple satisfies a qualification.

Any returns 1 if the qualification is true and 0 otherwise.

15.4. Aggregate functions

aggregation_operator (*a_expr* **by** *by_domain*
{, *by_domain*} [**where** *qual*])

Aggregate functions are extensions of simple aggregates. The *by* operator groups (i.e. partitions) the set of qualifying tuples by *by_domain* values. For more than one *by_domain,* the values which are grouped by are the concatenation of individual *by_domain* values. *A_expr* is as in simple aggregates. The aggregate function evaluates to a set of aggregate results, one for each partition into which the set of qualifying tuples has been grouped. The aggregate value used during evaluation of the query is the value associated with the partition into which the tuple currently being processed would fall.

Unlike simple aggregates, aggregate functions are not completely local. The *by_list,* which differentiates aggregate functions from simple aggregates, is global to the query. Domains in the *by_list* are automatically linked to the other domains in the query which are in the same relation.

Example:

```
/* retrieve the average salary for the employees
working for each manager */
range of e is employee
retrieve (e.manager, avesal=avg(e.salary by e.manager))
```

15.5 Aggregates on Unique Values.

It is occasionally necessary to aggregate on unique values of an expression. The *avgu, sumu,* and *countu* aggregates all remove duplicate values before performing the aggregation. For example:

count(e.manager)

would tell you how many occurrences of *e.manager* exist. But

countu(e.manager)

would tell you how many unique values of *e.manager* exist.

16. Special character operators

There are three special features which are particular to character domains.

16.1 Pattern matching characters

There are four characters which take on special meaning when used in character constants (strings):

*	matches any string of zero or more characters.
?	matches any single character.
[..]	matches any of characters in the brackets.

These characters can be used in any combination to form a variety of tests. For example:

where e.name = "*" — matches any name.
where e.name = "E*" — matches any name starting with "E".
where e.name = "*ein" — matches all names ending with "ein"
where e.name = "*[aeiou]*" — matches any name with at least one vowel.
where e.name = "Allman?" — matches any seven character name starting with "Allman".
where e.name = "[A—J]*" — matches any name starting with A,B,..,J.

The special meaning of the pattern matching characters can be disabled by preceding them with a "\". Thus "*" refers to the character "*". When the special characters appear in the target list they must be escaped. For example:

title = "*** ingres ***"

is the correct way to assign the string "*** ingres ***" to the domain "title".

16.2 Concatenation

There is a concatenation operator which can form one character string from two. Its syntax is "concat(field1, field2)". The size of the new character string is the sum of the sizes of the original two. Trailing blanks are trimmed from the first field, the second field is concatenated and the remainder is blank padded. The result is never trimmed to 0 length, however. Concat can be arbitrarily nested inside other concats. For example:

name = concat(concat(x.lastname, ","), x.firstname)

will concatenate x.lastname with a comma and then concatenate x.firstname to that.

16.3 Ascii (numeric to character translation)

The *ascii* function can be used to convert a numeric field to its character representation. This can be useful when it is desired to compare a numeric value with a character value. For example:

```
retrieve ( ... )
        where x.chardomain = ascii(x.numdomain)
```

Ascii can be applied to a character value. The result is simply the character value unchanged. The numeric conversion formats are determined by the printing formats (see ingres(unix)).

SEE ALSO

append(quel), delete(quel), range(quel), replace(quel), retrieve(quel), ingres(unix)

BUGS

The maximum number of variables which can appear in one query is 10.

Numeric overflow, underflow, and divide by zero are not detected.
When converting between numeric types, overflow is not checked.

NAME

range — declare a variable to range over a relation

SYNOPSIS

range of variable **is** relname

DESCRIPTION

Range is used to declare variables which will be used in subsequent QUEL statements. The *variable* is associated with the relation specified by *relname*. When the *variable* is used in subsequent statements it will refer to a tuple in the named relation. A range declaration remains in effect for an entire INGRES session (until exit from INGRES), until the variable is redeclared by a subsequent range statement, or until the relation is removed with the destroy command.

EXAMPLE

```
/* Declare tuple variable e to range over relation emp */
  range of e is emp
```

SEE ALSO

quel(quel), destroy(quel)

BUGS

Only 10 variable declarations may be in effect at any time. After the 10th range statement, the least recently referenced variable is re—used for the next range statement.

NAME

replace — replace values of domains in a relation

SYNOPSIS

replace tuple_variable (target_list) [**where** qual]

DESCRIPTION

Replace changes the values of the domains specified in the *target_list* for all tuples which satisfy the qualification *qual.* The *tuple_variable* must have been declared to range over the relation which is to be modified. Note that a tuple variable is required and not the relation name. Only domains which are to be modified need appear in the *target_list.* These domains must be specified as result_attnames in the *target_list* either explicitly or by default (see quel(quel)).

Numeric domains may be replaced by values of any numeric type (with the exception noted below). Replacement values will be converted to the type of the result domain.

Only the owner of a relation, or a user with replace pemission on the relation can do *replace.*

If the tuple update would violate an integrity constraint (see integrity(quel)), it is not done.

EXAMPLE

```
/* Give all employees who work for Smith a 10% raise */
  range of e is emp
  replace e(sal = 1.1 * e.sal) where e.mgr = "Smith"
```

SEE ALSO

integrity(quel), permit(quel), quel(quel), range(quel)

DIAGNOSTICS

Use of a numeric type expression to replace a character type domain or vice versa will produce diagnostics.

BUGS

NAME

retrieve — retrieve tuples from a relation

SYNOPSIS

retrieve [[**into**] relname] (target_list) [**where** qual]
retrieve unique (target_list) [**where** qual]

DESCRIPTION

Retrieve will get all tuples which satisfy the qualification and either display them on the terminal (standard output) or store them in a new relation.

If a relname is specified, the result of the query will be stored in a new relation with the indicated name. A relation with this name owned by the user must not already exist. The current user will be the owner of the new relation. The relation will have domain names as specified in the *target_list* result_attnames. The new relation will be saved on the system for seven (7) days unless explicitly saved by the user until a later date.

If the keyword **unique** is present, tuples will be sorted on the first domain, and duplicates will be removed, before being displayed.

The keyword **all** can be used when it is desired to retrieve all domains.

If no result *relname* is specified then the result of the query will be displayed on the terminal and will not be saved. Duplicate tuples are not removed when the result is displayed on the terminal.

The format in which domains are printed can be defined at the time ingres is invoked (see ingres(unix)).

If a result relation is specified then the default procedure is to modify the result relation to an cheapsort storage structure removing duplicate tuples in the process.

If the default cheapsort structure is not desired, the user can override this at the time INGRES is invoked by using the **-r** switch (see ingres(unix)).

Only the relation's owner and users with retrieve permission may *retrieve* from it.

EXAMPLE

```
/* Find all employees who make more than their manager */
  range of e is emp
  range of m is emp
  retrieve (e.name) where e.mgr = m.name
           and e.sal > m.sal
/* Retrieve all domains for those who make more
       than the average salary */
  retrieve into temp (e.all) where e.sal > avg(e.sal)
/* retrieve employees's names sorted */
  retrieve unique (e.name)
```

SEE ALSO

modify(quel), permit(quel), quel(quel), range(quel), save(quel), ingres(unix)

DIAGNOSTICS

BUGS

NAME

save - save a relation until a date.

SYNOPSIS

save relname **until** month day year

DESCRIPTION

Save is used to keep relations beyond the default 7 day life span.

Month can be an integer from 1 through 12, or the name of the month, either abbreviated or spelled out.

Only the owner of a relation can *save* that relation. There is an INGRES process which typically removes a relation immediately after its expiration date has passed.

The actual program which destroys relations is called purge. It is not automatically run. It is a local decision when expired relations are removed.

System relations have no expiration date.

EXAMPLE

```
/* Save the emp relation until the end of February 1987 */
  save emp until feb 28 1987
```

SEE ALSO

create(quel), retrieve(quel), purge(unix)

NAME

view — define a virtual relation

SYNOPSIS

define view name (target-list) [**where** qual]

DESCRIPTION

The syntax of the *view* statement is almost identical to the *retrieve into* statment; however, the data is not retrieved. Instead, the definition is stored. When the relation *name* is later used, the query is converted to operate on the relations specified in the *target-list.*

All forms of retrieval on the view are fully supported, but only a limited set of updates are supported because of anomolies which can appear. Almost no updates are supported on views which span more than one relation. No updates are supported that affect a domain in the qualification of the view or that affect a domain which does not translate into a simple attribute.

In general, updates are supported if and only if it can be guaranteed (without looking at the actual data) that the result of updating the view is identical to that of updating the corresponding real relation.

The person who defines a view must own all relations upon which the view is based.

EXAMPLE

```
range of e is employee
range of d is dept
define view empdpt (ename = e.name, e.sal, dname = d.name)
        where e.mgr = d.mgr
```

SEE ALSO

retrieve(quel), destroy(quel)

NAME

copydb − create batch files to copy out a data base and restore it.

SYNOPSIS

copydb [**−u***name*] database full-path-name-of-directory [relation ...]

DESCRIPTION

Copydb creates two INGRES command files in the directory: **Copy.out,** which contains Quel instuctions which will copy all relations owned by the user into files in the named directory, and **copy.in,** which contains instructions to copy the files into relations, create indexes and do modifies. The files will have the same names as the relations with the users INGRES id tacked on the end. (The directory MUST NOT be the same as the data base directory as the files have the same names as the relation files.) The **−u** flag may be used to run *copydb* with a different user id. (The fact that *copydb* creates the copy files does not imply that the user can necessarily access the specified relation). If relation names are specified only those relations will be included in the copy files.

Copydb is written in Equel and will access the database in the usual manner. It does not have to run as the INGRES user.

EXAMPLE

```
chdir /mnt/mydir
copydb db /mnt/mydir/backup
ingres db <backup/copy.out
tp rl backup
rm −r backup

tp xl
ingres db <backup/copy.in
```

DIAGNOSTICS

Copydb will give self-explanatory diagnostics. If "too many indexes" is reported it means that more than ten indexes have been specified on one relation. The constant can be increased and the program recompiled. Other limits are set to the system limits.

BUGS

Copydb assumes that indexes which are ISAM do not need to be remodified. *Copydb* cannot tell if the relation was modified with a fillfactor or minpages specification. The **copy.in** file may be edited to reflect this.

NAME

creatdb − create a data base

SYNOPSIS

creatdb [−u*name*] [−**e**] [−**m**] [±**c**] [±**q**] dbname

DESCRIPTION

Creatdb creates a new INGRES database, or modifies the status of an existing database. The person who executes this command becomes the Database Administrator (DBA) for the database. The DBA has special powers not granted to ordinary users.

Dbname is the name of the database to be created. The name must be unique among all INGRES users.

The flags ±**c** and ±**q** specify options on the database. The form +*x* turns an option on, while −*x* turns an option off. The −**c** flag turns off the concurrency control scheme (default on). The +**q** flag turns on query modification (default off).

Concurrency control should not be turned off except on databases which are never accessed by more than one user. This applies even if users do not share data relations, since system relations are still shared. If the concurrency control scheme is not installed in UNIX, or if the special file /dev/lock does not exist or is not accessible for read-write by INGRES, concurrency control acts as though it is off (although it will suddenly come on when the lock driver is installed in UNIX).

Query modification must be turned on for the protection, integrity, and view subsystems to work, however, the system will run slightly slower in some cases if it is turned on. It is possible to turn query modification on if it is already off in an existing database, but it is not possible to turn it off if it is already on.

Databases with query modification turned off create new relations with all access permitted for all users, instead of no access except to the owner, the default for databases with query modification enabled.

Database options for an existing database may be modified by stating the −**e** flag. The database is adjusted to conform to the option flags. For example:

creatdb −e +q mydb

turns query modification on for database "mydb" (but leaves concurrency control alone). Only the database administrator (DBA) may use the −**e** flag.

When query modification is turned on, new relations will be created with no access, but previously created relations will still have all access to everyone. The *destroy* command may be used to remove this global permission, after which more selective permissions may be specified with the *permit* command.

The INGRES user may use the −**u** flag to specify a different DBA: the flag should be immediately followed by the login name of the user who should be the DBA.

The −**m** flag specifies that the UNIX directory in which the database is to reside already exists. This should only be needed if the directory if a mounted file system, as might occur for a very large database. The directory must exist (as **.../data/base/***dbname*), must be mode 777, and must be empty of all files.

The user who executes this command must have the U_CREATDB bit set in the status field of her entry in the users file.

The INGRES superuser can create a file in **.../data/base** containing a single line which is the full pathname of the location of the database. The file must be owned by INGRES and be mode 644. When the database is created, it will be created in the file named, rather than in the directory **.../data/base.** For example, if the file **.../data/base/ericdb** contained the line

/mnt/eric/database

then the database called "ericdb" would be physically stored in the directory /mnt/eric/database rather than in the directory .../data/base/ericdb.

EXAMPLE

```
creatdb demo
creatdb -ueric -q erics_db
creatdb -e +q -c -u:av erics_db
```

FILES

.../files/dbtmplt6.2
.../files/data/base/*
.../files/datadir/* (for compatibility with previous versions)

SEE ALSO

demodb(unix), destroydb(unix), users(files), chmod(I), destroydb(quel), permit(quel)

DIAGNOSTICS

No database name specified.
: You have not specified the name of the database to create (or modify) with the command.

You may not access this database
: Your entry in the users file says you are not authorized to access this database.

You are not a valid INGRES user
: You do not have a users file entry, and can not do anything with INGRES at all.

You are not allowed this command
: The U_CREATDB bit is not set in your users file entry.

You may not use the -u flag
: Only the INGRES superuser may become someone else.

<name> does not exist
: With **-e** or **-m**, the directory does not exist.

<name> already exists
: Without either **-e** or **-m**, the database (actually, the directory) already exists.

<name> is not empty
: With the **-m** flag, the directory you named must be empty.

You are not the DBA for this database
: With the **-e** flag, you must be the database administrator.

NAME

destroydb — destroy an existing database

SYNOPSIS

destroydb [**−s**] [**−m**] dbname

DESCRIPTION

Destroydb will remove all reference to an existing database. The directory of the database and all files in that directory will be removed.

To execute this command the current user must be the database administrator for the database in question, or must be the INGRES superuser and have the **−s** flag stated.

The **−m** flag causes *destroydb* not to remove the UNIX directory. This is useful when the directory is a separate mounted UNIX file system.

EXAMPLE

destroydb demo
destroydb −s erics_db

FILES

.../data/base/*
.../datadir/* (for compatibility with previous versions)

SEE ALSO

creatdb(unix)

DIAGNOSTICS

invalid dbname — the database name specified is not a valid name.
you may not reference this database — the database may exist, but you do not have permission to do anything with it.
you may not use the −s flag — you have tried to use the **−s** flag, but you are not the INGRES superuser.
you are not the dba — someone else created this database.
database does not exist — this database does not exist.

NAME

equel − Embedded QUEL interface to C

SYNOPSIS

equel [**−d**] [**−f**] [**−r**] file.q ...

DESCRIPTION

Equel provides the user with a method of interfacing the general purpose programming language "C" with INGRES. It consists of the EQUEL pre-compiler and the EQUEL runtime library.

Compilation

The precompiler is invoked with the statement:

equel [<flags>] file1.q [<flags>] file2.q ...

where file*n*.q are the source input file names, which must end with **.q.** The output is written to the file "file*n*.c". As many files as wished may be specified.
The flags that may be used are:

−d Generate code to print source listing file name and line number when a run-time error occurs. This can be useful for debugging, but takes up process space. Defaults to off.

−f Forces code to be on the same line in the output file as it is in the input file to ease interpreting C diagnostic messages. EQUEL will usually try to get all C code lines in the output file on the same lines as they were in the input file. Sometimes it must break up queries into several lines to avoid C-preprocessor line overflows, possibly moving some C code ahead some lines. With the **−f** flag specified this will never happen and, though the line buffer may overflow, C lines will be on the right line. This is useful for finding the line in the source file that C error diagnostics on the output file refer to.

−r Resets flags to default values. Used to supress other flags for some of the files in the argument list.

The output files may than be compiled using the C compiler:

cc file1.c file2.c ... **−lq**

The **−lq** requests the use of the EQUEL object library.

All EQUEL routines and globals begin with the characters "II", and so all globals variables and procedure names of the form II*xxx* are reserved for use by EQUEL and should be avoided by EQUEL users.

Basic Syntax

EQUEL commands are indicated by lines which begin with a double pound sign ("##"). Other lines are simply copied as is. All normal INGRES commands may be used in EQUEL and have the same effect as if invoked through the interactive terminal monitor. Only retrieve commands with no result relation specified have a different syntax and meaning.

The format of retrieve without a result relation is modified to:

retrieve (C-variable = a_fcn { , C-variable = a_fcn })

optionally followed (immediately) by:

```
##      [ where qual ]
## {
        /* C-code */
## }
```

This statement causes the "C-code" to be executed once for each tuple retrieved, with the "C-variable"s set appropriately. Numeric values of any type are converted as necessary. No conversion is done between numeric and character values. (The normal INGRES *ascii* function may be used for this purpose.)

Also, the following EQUEL commands are permitted.

ingres [ingres flags] data_base_name

This command starts INGRES running, and directs all dynamically following queries to the database *data_base_name*. It is a run-time error to execute this command twice without an intervening "## exit", as well as to issue queries while an "## ingres" statement is not in effect. Each flag should be enclosed in quotes to avoid confusion in the EQUEL parser:

```
## ingres "-f4f10.2" "-i212" demo
```

exit

Exit simply exits from INGRES. It is equivalent to the **\q** command to the teletype monitor.

Parametrized Quel Statements

Quel statements with target lists may be "parametrized". This is indicated by preceding the statement with the keyword "param". The target list of a parametrized statement has the form:

(*tl_var, argv*)

where *tl_var* is taken to be a string pointer at execution time (it may be a string constant) and interpreted as follows. For any parametrized EQUEL statement except a retrieve without a result relation (no "into rel") (i.e. append, copy, create, replace, retrieve into) the string *tl_var* is taken to be a regular target list except that wherever a '%' appears a valid INGRES type (f4, f8, i2, i4, c) is expected to follow. Each of these is replaced by the value of the corresponding entry into *argv* (starting at 0) which is interpreted to be a pointer to a variable of the type indicated by the '%' sequence. Neither *argv* nor the variables which it points to need be declared to EQUEL. For example:

```
char    *argv[10];

        argv[0] = &double_var;
        argv[1] = &int_var;
##      param append to rel
##              ("dom1 = %f8, dom2 = %i2", argv)
##      /* to escape the "%<ingres_type>" mechanism use "%%" */
##      /* This places a single '%' in the string. */
```

On a retrieve to C-variables, within *tl_var,* instead of the C-variable to retrieve into, the same '%' escape sequences are used to denote the type of the corresponding argv entry into which the value will be retrieved.

The qualification of any query may be replaced by a string valued variable, whose contents is interpreted at run time as the text of the qualification.

The *copy* statement may also be parametrized. The form of the parametrized *copy* is analogous to the other parametrized statements: the target list may be parametrized in the same manner as the *append* statements, and furthermore, the **from/into** keyword may be replaced by a string valued variable whose content at run time should be **into** or **from.**

Declarations

Any valid C variable declaration on a line beginning with a "##" declares a C-variable that may be used in an EQUEL statement and as a normal variable. All variables must be declared before being used. Anywhere a constant may appear in an INGRES command, a C-variable may appear. The value of the C-variable is substituted at execution time.

Neither nested structures nor variables of type *char* (as opposed to pointer to char or array of char) are allowed. Furthermore, there are two restrictions in the way variables are referenced within EQUEL statements. All variable usages must be dereferenced and/or subscripted (for arrays and pointers), or selected (for structure variables) to yield lvalues (scalar values). *Char* variables are used by EQUEL as a means to use *strings.* Therefore when using a *char* array or

pointer it must be dereferenced only to a "*char* *". Also, variables may not have parentheses in their references. For example:

```
## struct xxx
## {
        int     i;
##      int     *ip;
## }    **struct_var;

/* not allowed */
##      delete p where p.ifield = *(*struct_var)->ip

/* allowed */
##      delete p where p.ifield = *struct_var[0]->ip
```

C variables declared to EQUEL have either global or local scope. Their scope is local if their declaration is within a free (not bound to a retrieve) block declared to EQUEL. For example:

```
/* globals scope variable */
## int Gint;

func(i)
int     i;
## {
        /* local scope variable */
##      int     *gintp;
        ...
## }
```

If a variable of one of the char types is used almost anywhere in an EQUEL statement the content of that variable is used at run time. For example:

```
##      char    *dbname[MAXDATABASES + 1];
        int     current_db;

        dbname[current_db] = "demo";
##      ingres dbname[current_db]
```

will cause INGRES to be invoked with data base "demo". However, if a variable's name is to be used as a constant, then the non-referencing operator '#' should be used. For example:

```
## char         *demo;

        demo = "my_database";

        /* ingres -d my_database */
##      ingres "-d" demo

        /* ingres -d demo */
##      ingres "-d" #demo
```

The C-preprocessor's #include feature may be used on files containing equel statements and declarations if these files are named *anything*.**q.h.** An EQUEL processed version of the file, which will be #included by the C-preprocessor, is left in *anything*.**c.h.**

Errors and Interrupts

INGRES and run-time EQUEL errors cause the routine **IIerror** to be called, with the error number and the parameters to the error in an array of string pointers as in a C language main routine. The error message will be looked up and printed, before printing the error message, the routine (*IIprint_err)() is called with the error number that ocurred as its single argument. The error message corresponding to the error number returned by (*IIprint_err)() will be printed. Printing will be supressed if (*IIprint_err)() returns 0. IIprint_err may be reassigned to, and is use-

ful for programs which map INGRES errors into their own error messages. In addition, if the "**-d**" flag was set the file name and line number of the error will be printed. The user may write an IIerror routine to do other tasks as long as the setting of IIerrflag is not modified as this is used to exit retrieves correctly.

Interrupts are caught by equel if they are not being ignored. This insures that the rest of INGRES is in sync with the EQUEL process. There is a function pointer, IIinterrupt, which points to a function to call after the interrupt is caught. The user may use this to service the interrupt. It is initialized to "exit()" and is called with -1 as its argument. For example:

```
extern int (*IIinterrupt)();
extern reset();

setexit();
IIinterrupt = reset;
mainloop();
```

To ignore interrupts, signal() should be called before the ## ingres satatement is executed.

FILES

.../files/error6.2_*
: Can be used by the user to decipher INGRES error numbers.

/lib/libq.a
: Run time library.

SEE ALSO

.../doc/other/equeltut.q, C reference manual, ingres(UNIX), quel(QUEL)

BUGS

The C-code embedded in the tuple-by-tuple retrieve operation may not contain additional QUEL statements or recursive invocations of INGRES.

There is no way to specify an **i1** format C-variable.

Includes of an equel file within a parameterized target list, or within a C variable's array subscription brackets, isn't done correctly.

NAME

helpr — get information about a database.

SYNOPSIS

helpr [**−u***name*] [**±w**] database relation ...

DESCRIPTION

Helpr gives information about the named relation(s) out of the database specified, exactly like the *help* command.

Flags accepted are **−u** and **±u**. Their meanings are identical to the meanings of the same flags in INGRES.

SEE ALSO

ingres(unix), help(quel)

DIAGNOSTICS

bad flag — you have specified a flag which is not legal or is in bad format.

you may not access database — this database is prohibited to you based on status information in the users file.

cannot access database — the database does not exist.

NAME

ingres – INGRES relational data base management system

SYNOPSIS

ingres [*flags*] dbname [process_table]

DESCRIPTION

This is the UNIX command which is used to invoke INGRES. Dbname is the name of an existing data base. The optional flags have the following meanings (a "±" means the flag may be stated "+*x*" to set option *x* or "−*x*" to clear option *x*. "−" alone means that "−*x*" must be stated to get the *x* function):

±U Enable/disable direct update of the system relations and secondary indicies. You must have the 000004 bit in the status field of the users file set for this flag to be accepted. This option is provided for system debugging and is strongly discouraged for normal use.

−u*name* Pretend you are the user with login name *name* (found in the users file). If *name* is of the form *:xx*, *xx* is the two character user code of a user. This may only be used by the DBA for the database or by the INGRES superuser.

−c*N* Set the minimum field width for printing character domains to *N*. The default is 6.

−i*lN* Set integer output field width to *N*. *l* may be 1, 2, or 4 for i1's, i2's, or i4's repectively.

−f*lxM.N* Set floating point output field width to *M* characters with *N* decimal places. *l* may be 4 or 8 to apply to f4's or f8's respectively. *x* may be **e, E, f, F, g, G, n,** or **N** to specify an output format. **E** is exponential form, **F** is floating point form, and **G** and **N** are identical to **F** unless the number is too big to fit in that field, when it is output in **E** format. **G** format guarantees decimal point alignment; **N** does not. The default format for both is **n10.3.**

−v*X* Set the column seperator for retrieves to the terminal and print commands to be *X*. The default is vertical bar.

−r*M* Set modify mode on the *retrieve* command to *M*. *M* may be **isam, cisam, hash, chash, heap, cheap, heapsort,** or **cheapsort,** for ISAM, compressed ISAM, hash table, compressed hash table, heap, compressed heap, sorted heap, or compressed sorted heap. The default is "cheapsort".

−n*M* Set modify mode on the *index* command to *M*. *M* can take the same values as the **−r** flag above. Default is "isam".

±a Set/clear the autoclear option in the terminal monitor. It defaults to set.

±b Set/reset batch update. Users must the 000002 bit set in the status field of the users file to clear this flag. This flag is normally set. When clear, queries will run slightly faster, but no recovery can take place. Queries which update a secondary index automatically set this flag for that query only.

±d Print/don't print the dayfile. Normally set.

±s Print/don't print any of the monitor messages, including prompts. This flags is normally set. If cleared, it also clears the **−d** flag.

±w Wait/don't wait for the database. If the **+w** flag is present, INGRES will wait if certain processes are running (purge,restore, and/or sysmod) on the given data base. Upon completion of those processes INGRES will proceed. If the **−w** flag is present, a message is returned and execution stopped if the data base is not available. If the **±w** flag is omitted and the data base is unavailable, the error message is returned if INGRES is running in foreground (more precisly if the standard input is from a terminal), otherwise the wait option is invoked.

Process_table is the pathname of a UNIX file which may be used to specify the run-time configuration of INGRES. This feature is intended for use in system maintenance only, and its unenlightened use by the user community is strongly discouraged.

Note: It is possible to run the monitor as a batch-processing interface using the '<', '>' and '|' operators of the UNIX shell, provided the input file is in proper monitor-format.

EXAMPLE

```
ingres demo
ingres -d demo
ingres -s demo < batchfile
ingres -f4g12.2 -i13 +b -rhash demo
```

FILES

.../files/users – valid INGRES users
.../data/base/* – data bases
.../datadir/* – for compatability with previous versions
.../files/proctab6.2 – runtime configuration file

SEE ALSO

monitor(quel)

DIAGNOSTICS

Too many options to INGRES – you have stated too many flags as INGRES options.

Bad flag format – you have stated a flag in a format which is not intelligible, or a bad flag entirely.

Too many parameters – you have given a database name, a process table name, and "something else" which INGRES doesn't know what to do with.

No database name specified

Improper database name – the database name is not legal.

You may not access database *name* – according to the users file, you do not have permission to enter this database.

You are not authorized to use the *flag* flag – the flag specified requires some special authorization, such as a bit in the users file, which you do not have.

Database *name* does not exist

You are not a valid INGRES user – you have not been entered into the users file, which means that you may not use INGRES at all.

You may not specify this process table – special authorization is needed to specify process tables.

Database temporarily unavailable – someone else is currently performing some operation on the database which makes it impossible for you to even log in. This condition should disappear shortly.

NAME

geoquel — GEO-QUEL data display system

SYNOPSIS

geoquel [**−s**] [**−d**] [**−a**] [**−t***T*] [**−tn***T*] dbname

DESCRIPTION

GEO-QUEL is a geographical interface to INGRES.

Dbname is the name of an existing data base.

The format of the graphic output depends upon the type of terminal in use. GEO-QUEL will look up the terminal type at login time and produce output appropriate for that terminal. If the terminal in use is incapable of drawing graphic output then a display list is generated for a Tektronix 4014 but will only be displayed if the results are saved with SAVEMAP and then redisplayed.

The flags are interpreted as follows:

−s Don't print any of the monitor messages, including prompts. This is inclusive of the dayfile.

−d Don't print the dayfile.

−a Disable the autoclear function in the terminal monitor.

−t*T* Set the terminal type to *T*. *T* can be **gt40, gt42,** or **4014** for DEC's GT40-GT42, and Tektronix's 4014 respectively.

−tn*T* Do not display output but prepare the display list for a terminal of type *T*, where *T* is from the above list.

EXAMPLE

geoquel demo
geoquel −d demo
geoquel −s demo < batchfile

FILES

.../files/grafile6.2
.../files/ttytype

SEE ALSO

GEO-QUEL reference manual

DIAGNOSTICS

The diagnostics produced by GEO-QUEL are intended to be self-explanatory. Occasional messages may be produced by INGRES; for an explanation of these messages see the INGRES system documentation.

NAME

printr — print relations

SYNOPSIS

printr [*flags*] database relation ...

DESCRIPTION

Printr prints the named relation(s) out of the database specified, exactly like the *print* command. Retrieve permission must be granted to all people to execute this command.

Flags accepted are **−u**, **±w**, **−c**, **−i**, **−f**, and **−v**. Their meanings are identical to the meanings of the same flags in INGRES.

SEE ALSO

ingres(unix), print(quel)

DIAGNOSTICS

bad flag — you have specified a flag which is not legal or is in bad format.

you may not access database — this database is prohibited to you based on status information in the users file.

cannot access database — the database does not exist.

NAME

purge − destroy all expired and temporary relations

SYNOPSIS

purge [**−f**] [**−p**] [**−a**] [**−s**] [**±w**] [database ...]

DESCRIPTION

Purge searches the named databases deleting system temporary relations. When using the **−p** flag, expired user relations are deleted. The **−f** flag will cause unrecognizable files to be deleted, normally purge will just report these files.

Only the database administrator (the DBA) for a database may run purge, except the INGRES superuser may purge any database by using the **−s** flag.

If no databases are specified all databases for which you are the DBA will be purged. All databases will be purged if the INGRES superuser has specified the **−s** flag. The **−a** flag will cause purge to print a message about the pending operation and execute it only if the response if a 'y'. Any other response is interpreted as "no".

Purge will lock the data base while it is being processed, since errors may occur if the database is active while purge is working on the database. If a data base is busy *purge* will report this and go on to the next data base, if any. If standard input is not a terminal *purge* will wait for the data base to be free. If **−w** flag is stated *purge* will not wait, regardless of standard input. The **+w** flag causes *purge* to always wait.

EXAMPLES

purge −p +w tempdata
purge −a −f

SEE ALSO

save(quel), restore(unix)

DIAGNOSTICS

who are you? − you are not entered into the users file.
not ingres superuser − you have tried to use the **−s** flag but you are not the INGRES superuser.
you are not the dba − you have tried to purge a database for which you are not the DBA.
cannot access database − the database does not exist.

BUGS

If no database names are given, only the databases located in the directory **data/base** are purged, and not the old databases in **datadir.** Explicit database names still work for databases in either directory.

NAME

restore — recover from an INGRES or UNIX crash.

SYNOPSIS

restore [**−a**] [**−s**] [**±w**] [database ...]

DESCRIPTION

Restore is used to restore a data base after an INGRES or UNIX crash. It should always be run after any abnormal termination to ensure the integrity of the data base.

In order to run restore, you must be the DBA for the database you are restoring or the INGRES superuser and specify the **−s** flag.

If no databases are specified then all databases for which you are the DBA are restored. All databases will be restored if the INGRES superuser has specified the **−s** flag.

If the **−a** flag is specified you will be asked before restore takes any serious actions. It is advisable to use this flag if you suspect the database is in bad shape. Using /dev/null as input with the **−a** flag will provide a report of problems in the data base. If there were no errors while restoring a database, *purge* will be called, with the same flags that were given to *restore,* to remove unwanted files and system temporaries. *Restore* may be called with the **−f** and/or **−p** flags for *purge.* Unrecognized files and expired relations are not removed unless the proper flags are given. In the case of an incomplete destroy, create or index *restore* will not delete files for partially created or destroyed relations. *Purge* must be called with the **−f** flag to accomplish this.

Restore locks the data base while it is being processed. If a data base is busy *restore* will report this and go on to the next data base. If standard input is not a terminal *restore* will wait for the data base to be free. If the **−w** flag is set *restore* will not wait regardless of standard input. If **+w** is set it will always wait.

Restore can recover a database from an update which had finished filling the batch file. Updates which did not make it to this stage should be rerun. Similarly modifies which have finished recreating the relation will be completed (the relation relation and attribute relations will be updated). If a destroy was in progress it will be carried to completion, while a create will almost always be backed out. Destroying a relation with an index should destroy the index so *restore* may report that a secondary relation has been found with no primary.

If interrupt (signal 2) is received the current database is closed and the next, if any, is processed. Quit (signal 3) will cause restore to terminate.

EXAMPLE

```
restore −f demo
restore −a grants < /dev/null
```

DIAGNOSTICS

All diagnostics are followed by a tuple from a system relations.

"No relation for attribute(s)" — the attributes listed have no corresponding entry in the relation relation

"No primary relation for index" — the tuple printed is the relation tuple for a secondary index for which there is no primary relation. The primary probably was destroyed the secondary will be.

"No indexes entry for primary relation" — the tuple is for a primary relation, the relindxd domain will be set to zero. This is the product of an incomplete destroy.

"No indexes entry for index" — the tuple is for a secondary index, the index will be destroyed. This is the product of an incomplete destroy.

"*relname* is index for" — an index has been found for a primary which is not marked as indexed. The primary will be so marked. This is probably the product of an incomplete index command. The index will have been created properly but not modified.

"No file for" — There is no data for this relation tuple, the tuple will be deleted. If, under the **−a** option, the tuple is not deleted purge will not be called.

"No secondary index for indexes entry" — An entry has been found in the indexes relation for which the secondary index does not exist (no relation relation tuple). The entry will be deleted.

SEE ALSO

purge(unix)

BUGS

If no database names are given, only the databases located in the directory **data/base** are restored, and not the old databases in **datadir.** Explicit database names still work for databases in either directory.

NAME

sysmod − modify system relations to predetermined storage structures.

SYNOPSIS

sysmod [**−s**] [**−w**] dbname [**relation**] [**attribute**] [**indexes**] [**tree**] [**protect**] [**integrities**]

DESCRIPTION

Sysmod will modify the relation, attribute, indexes, tree, protect, and integrities relations to hash unless at least one of the **relation, attribute, indexes, tree, protect,** or **integrities** parameters are given, in which case only those relations given as parameters are modified. The system relations are modified to gain maximum access performance when running INGRES. The user must be the data base administrator for the specified database, or be the INGRES superuser and have the **−s** flag stated.

Sysmod should be run on a data base when it is first created and periodically thereafter to maintain peak performance. If many relations and secondary indices are created and/or destroyed, *sysmod* should be run more often.

If the data base is being used while *sysmod* is running, errors will occur. Therefore, *sysmod* will lock the data base while it is being processed. If the data base is busy, *sysmod* will report this. If standard input is not a terminal *sysmod* will wait for the data base to be free. If **−w** flag is stated *sysmod* will not wait, regardless of standard input. The **+w** flag causes *sysmod* to always wait.

The system relations are modified to hash; the relation relation is keyed on the first domain, the indexes, attribute, protect, and integrities relations are keyed on the first two domains, and the tree relation is keyed on domains one, two, and five. The relation and attribute relations have the minpages option set at 10, the indexes relation has a minpages value of 5.

SEE ALSO

modify(quel)

NAME

timefix − patch INGRES binary code for correct timezone.

SYNOPSIS

timefix [**−u**] [**−s***N*] [**−t***yyyzzz*] [**−d***X*] filename ...

DESCRIPTION

Timefix can be used to change the INGRES system to use local time. As delivered, the INGRES system assumes it is in the Pacific Time Zone, eight hours behind Greenwich Mean Time. *Timefix* can automatically patch the necessary INGRES code to reflect the true timezone of your installation.

INGRES uses the UNIX supplied routine ctime(III) for converting the UNIX base time to local time. *Timefix* is first compiled on your system in order to get the *ctime* routine for your installation. It then can correct the necessary INGRES programs at which point your INGRES system will be correct and not have to be changed again.

To use *timefix* you must be logged in as **ingres** and then use the *setup* command:

setup timefix

This command performs the following steps: (1) compiles timefix.c, (2) runs timefix on the necessary INGRES programs.

If for any reason, timefix.c cannot be compiled on your system, another method exists for updating INGRES. *Timefix* can be run with the local time expressed using the **−s** **−t** and **−d** flag options. Their meanings are:

−s*N*	*N* is the number of seconds ahead/behind GMT.
−t*yyyzzz*	*yyy* and *zzz* are the two, three character timezone names.
−d*X*	if *X* is 0, then daylight savings time is disabled, otherwise daylight savings time will occur according to the rules specified in ctime(III).

The flags can be given in any order. You must be logged in as **ingres** and then use the *setup* command giving the proper time conversions. For example, in Hartford, Connecticut you would type:

setup timefix −s18000 −tESTEDT −d1

If the **−u** flag is specified then *timefix* reports what it would do but does not actually update the files.

EXAMPLES

setup timefix
setup timefix −s18000 −tESTEDT −d1

DIAGNOSTICS

Timefix depends on the existence of a namelist. If the namelist (i.e. symbol table) is not present then an error message is given. If all the symbols related to the *ctime* routine cannot be found then it is assumed that the file need not be corrected. If the file cannot be opened for reading and writing then an error message is issued.

BUGS

Timefix will not work for timezones which are ±10 or ±11 hours from GMT.

SEE ALSO

ctime(III)

NAME

usersetup — setup users file

SYNOPSIS

.../bin/usersetup [pathname]

DESCRIPTION

The **/etc/passwd** file is read and reformatted to become the INGRES users file, stored into **.../files/users.** If *pathname* is specified, it replaces "...". If pathname is "—", the result is written to the standard output.

The user name, user, and group id's are initialized to be identical to the corresponding entry in the **/etc/passwd** file. The status field is initialized to be zero, except for user **ingres,** which is initialized to all permission bits set. The "initialization file" parameter is set to the file **.ingres** in the user's login directory. The user code field is initialized with sequential two-character codes. All other fields are initialized to be null.

After running *usersetup,* the **users** file must be edited. Any users who are to have any special authorizations should have the status field changed, according to the specifications in users(files). To disable a user from executing INGRES entirely, completely remove her line from the users file.

As UNIX users are added or deleted from the **/etc/passwd** file, the **users** file will need to be editted to reflect the changes. For deleted users, it is only necessary to delete the line for that user from the **users** file. To add a user, you must assign that user a code in the form "aa" and enter a line in the users file in the form:

name:cc:uid:gid:status:flags:proctab:initfile::databases

where *name* is the user name (taken from the first field of the **/etc/passwd** file entry for this user), *cc* is the user code assigned, which must be exactly two characters long and must not be the same as any other existing user codes, *uid* and *gid* are the user and group ids (taken from the third and fourth fields in the **/etc/passwd** entry), *status* is the status bits for this user, normally 000000, *flags* are the default flags for INGRES (on a per-user basis), *proctab* is the default process table for this user (which defaults to **=proctab6.2**), and *databases* is a list of the databases this user may enter. If null, she may use all databases. If the first character is a dash ("—"), the field is a comma separated list of databases which she may not enter. Otherwise, it is a list of databases which she may enter.

The *databases* field includes the names of databases which may be created.

Usersetup may be executed only once, to initially create the **users** file.

FILES

.../files/users
/etc/passwd

SEE ALSO

ingres(unix), passwd(V), users(files)

BUGS

It should be able to bring the **users** file up to date.

NAME

.../files/dayfile6.1 — INGRES login message

DESCRIPTION

The contents of the dayfile reflect user information of general system interest, and is more or less analogous to **/etc/motd** in UNIX. The file has no set format; it is simply copied at login time to the standard output device by the monitor if the **−s** or **−d** options have not been requested. Moreover the dayfile is not mandatory, and its absence will not generate errors of any sort; the same is true when the dayfile is present but not readable.

NAME

.../files/dbtmplt6.2 – database template

DESCRIPTION

This file contains the template for a database used by *creatdb*. It has a set of entries for each relation to be created in the database. The sets of entries are separated by a blank line. Two blank lines or an end of file terminate the file.

The first line of the file is the database status and the default relation status, separated by a colon. The rest of the file describes relations. The first line of each group gives the relation name followed by an optional relation status, separated by a colon. The rest of the lines in each group are the attribute name and the type, separated by a tab character.

All the status fields are given in octal, and have a syntax of a single number followed by a list of pairs of the form

±*x*±N

which says that if the ± *x* flag is asserted on the *creatdb* command line then set (clear) the bits specified by *N*.

The first set of entries must be for the relation catalog, and the second set must be for the attribute catalog.

EXAMPLE

```
3-c-1+q+2:010023
relation:-c-20
relid    c12
relowner        c2
relspec  i1

attribute:-c-20
attrelid c12
attowner        c2
attnamec12

        (other relation descriptors)
```

SEE ALSO

creatdb(unix)

NAME

.../files/error6.2_? — files with INGRES errors

DESCRIPTION

These files contain the INGRES error messages. There is one file for each thousands digit; hence, error number 2313 will be in file error6.2_2.

Each file consists of a sequence of error messages with associated error numbers. When an error enters the front end, the appropriate file is scanned for the correct error number. If found, the message is printed; otherwise, the first message parameter is printed.

Each message has the format

errnum <TAB> message tilde.

Messages are terminated by the tilde character ('~'). The message is scanned before printing. If the sequence %*n* is encountered (where *n* is a digit from 0 to 9), parameter *n* is substituted, where %0 is the first parameter.

The parameters can be in any order. For example, an error message can reference %2 before it references %0.

SEE ALSO

error(UTIL)

EXAMPLE

```
1003    line %0, bad database name %1~
1005    In the purge of %1,
a bad %0 caused execution to halt~
1006    No process, try again.~
```

NAME

.../files/grafile6.2 — GEO-QUEL login message

DESCRIPTION

The contents of the grafile (GEO-QUEL dayfile) reflect user information of general system interest, and is more or less analogous to **/etc/motd** in UNIX. The file has no set format; it is simply copied at login time to the standard output device by GEO-QUEL if the **−s** or **−d** options have not been requested. Moreover the grafile is not mandatory, and its absence will not generate errors of any sort; the same is true when the grafile is present but not readable.

NAME

libq – Equel run-time support library

DESCRIPTION

Libq all the routines necessary for an equel program to load. It resides in **/lib/libq.a**, and must be specified when loading equel pre-processed object code. It may be referenced on the command line of *cc* by the abbreviation **–lq.**

Several useful routines which are used by equel processes are included in the library. These may be employed by the equel programmer to avoid code duplication. They are:

```
int     IIatoi(buf, i)
char    *buf;
int     i;

char    *IIbmove(source, destination, len)
char    *source, *destination;
int     len;

char    *IIconcatv(buf, arg1, arg2, ..., 0)
char    *buf, *arg1, ...;

char    *IIitos(i)
int     i;

int     IIsequal(s1, s2)
char    *s1, *s2;

int     IIlength(string)
char    *string;

IIsyserr(string, arg1, arg2, ...);
char    *string;
```

IIatoi	IIatoi is equivalent to atoi(UTIL).
IIbmove	Moves *len* bytes from *source* to *destination*, returning a pointer to the location after the last byte moved. Does not append a null byte.
IIconcatv	Concatenates into *buf* all of its arguments, returning a pointer to the null byte at the end of the concatenation. *Buf* may not be equal to any of the arg-n but arg1.
IIitos	IIitos is equivalent to itoa(III).
IIsequal	Returns 1 iff strings s1 is identical to s2.
IIlength	Returns max(length of *string* without null byte at end, 255)
IIsyserr	IIsyserr is diferrent from syserr(util) only in that it will print the name in IIproc_name, and in that there is no 0 mode. Also, it will always call exit(–1) after printing the error message.

There are also some global Equel variables which may be manipulated by the user:

```
int     IIerrflag;
char    *IImainpr;
char    (*IIprint_err)();
int     IIret_err();
int     IIno_err();
```

IIerrflag — Set on an error from INGRES to be the error number (see the error message section of the "INGRES Reference Manual") that ocurred. This remains valid from the time the error occurrs to the time when the next equel statement is issued. This may be used just after an equel statement to see if it succeded.

IImainpr — This is a string which determines which ingres to call when a "## ingres" is issued. Initially it is "/usr/bin/ingres".

IIprint_err — This function pointer is used to call a function which determines what (if any) error message should be printed when an ingres error occurs. It is called from IIerror() with the error number as an argument, and the error message corresponding to the error number returned will be printed. If (*IIprint_err)(<*errno*>) returns 0, then no error message will be printed. Initially IIprint_err is set to IIret_err() to print the error that ocurred.

IIret_err — Returns its single integer argument. Used to have (*IIprint_err)() cause printing of the error that ocurred.

IIno_err — Returns 0. Used to have (*IIprint_err)() suppress error message printing. IIno_err is used when an error in a parametrized equel statemenr occurs to suppress printing of the corresponding parser error.

SEE ALSO

atoi(util), bmove(util), cc(I), equel(unix), exit(II), itoa(III), length(util), sequal(util), syserr(util)

NAME

.../files/proctab6.2 — INGRES runtime configuration information

DESCRIPTION

The file **.../files/proctab6.2** describes the runtime configuration of the INGRES system.

The process table is broken up into logical sections, separated by lines with a single dollar sign. The first section describes the configuration of the system and the parameters to pass to each system module. All other sections contain strings which may be macro-substituted into the first section.

Each line of the first section describes a single process. The lines consist of a series of colon-separated fields.

The first field contains the pathname of the module to be executed.

The second field is a set of flags which allow the line to be ignored in certain cases. If this field is null, the line is accepted; otherwise, it should be a series of items of the form "+−=*X*", where any of "+−=" may be omitted, and *X* is a flag which may appear on the INGRES command line. The characters "+−=" are interpreted as the sense of the flag: "+" will accept the line if the flag "+*X*" is stated on the command line, "−" will accept if "−*X*" is stated, and "=" will accept if the "*X*" flag is not stated at all. These may be combined in the forms "+=" and "−=". For example, the field:

+=&

will acept the line if the EQUEL flag ("&") is stated as "+&" or is not stated, but the line will be ignored if the "−&" flag is stated.

If any flag item rejects the line, the entire line is rejected.

The third field is a status word. The number in this word is expressed in octal. The bits have the following meaning:

000010 close diagnostic output
000004 close standard input
000002 run in user's directory, not database
000001 run as the user, not as INGRES

The fourth field is a file name to which the standard output should be redirected. It is useful for debugging.

The fifth field describes the pipes which should be connected to this process. The field must be six characters long, with the characters corresponding to the internal variables R_up, W_up, R_down, W_down, R_front, and W_front respectively. The characters may be a question mark, which leaves the pipe closed; a digit, which is filled in from the file descriptors provided by the EQUEL flag or the "@" flag; or a lower case letter, which is connected via a pipe with any other pipes having the same letter; this last action is done on the fly to conserve file descriptors.

The sixth and subsequent fields are arbitrary parameters to be sent to the modules. There must be a colon after the fifth field even if no parameters are present, but there need not be a colon after the last parameter.

The last module executed (usually the last line in the first section) becomes the parent of all the other processes.

The second through last sections of the process table consist of a single line which names the section followed by arbitrary text. The pathname field and all parameter fields of each line of the first section are scanned for strings of the form "Sname"; this string is replaced by the text from the corresponding section. For convenience, the name **Spathname** is automatically defined to be the pathname of the root of the INGRES subtree.

The DBU routines want to see two parameters. The first parameter is the pathname of the "ksort" routine. The second parameter is a series of lines of the form:

command_name:place_list

where command_name is the name of one of the possible INGRES commands executed by the DBU routines, and place_list is a comma-separated list of the actual location(s) of that command. Each "place" is a two-character descriptor: the first character is the overlay in which that command resides, and the second character is the function within that overlay. If a command is in more than one place, INGRES will try to avoid calling in another overlay. For example:

create:a0,m1

means that the *create* command may be found in overlay "a" function 0 or in overlay "m" function 1. If already in overlay "a" or "m" the create command resident in that overlay will be called; otherwise, overlay "a" will be called.

EXAMPLE

The following example will execute a three process system unless the "&" flag is specified (as "−&"), when a two-process system will be executed with the monitor dropped out and the calling (EQUEL) program in its place. Notice that there are two lines for the parser entry, one for the EQUEL case and one for the non-EQUEL case. In the EQUEL case, output from the parser is diverted to a file called "debug.out".

```
$pathname/bin/overlaya::000014::bc??23:$pathname/bin/ksort:$dbutab
$pathname/bin/parser:+=&:000014::adcb??:
$pathname/bin/parser:−&:000014:debug.out:01cb??:
$pathname/bin/monitor:+=&:000003::??da??:$pathname/files/startup
$
dbutab
create:a0,m1
destroy:a1,m2
modify:m0
help:a2
$
```

NAME

.../files/startup — INGRES startup file

DESCRIPTION

This file is read by the monitor at login time. It is read before the user startup file specified in the users file. The primary purpose is to define a new editor and/or shell to call with the \e or \s commands.

SEE ALSO

monitor(quel), users(files)

NAME

.../files/ttytype — GEO-QUEL terminal type database

DESCRIPTION

The **ttytype** file describes each terminal on your system. GEO-QUEL will not attempt to display graphical output on terminals that are not capable of displaying it. There is a sample of the file in **.../geoquel/ttytype.sample.** This is a copy of the file in use on the Berkeley system.

The **ttytype** file consists of a series of lines; the first character is the terminal id, and the rest of the line tells the type of the terminal. The first of these characters is a terminal class, and the rest signify the brand, or some other more descriptive indication. A completely blank line terminates the useful part of the file, after which comments may appear unrestricted. In the sample file the currently recognized (defined) terminal types are listed as comments.

BUGS

The current version of GEO-QUEL only looks for terminals of graphic nature and therefore only the graphic terminals need be in this database. If any other system or sub-system wishes to use this file, all terminals must be kept up to date.

NAME

.../files/users — INGRES user codes and parameters

DESCRIPTION

This file contains the user information in fields seperated by colons. The fields are as follows:

* User name, taken directly from **/etc/passwd** file.
* User code, assigned by the INGRES super-user. It must be a unique two character code.
* UNIX user id. This MUST match the entry in the **/etc/passwd** file.
* UNIX group id. Same comment applies.
* Status word in octal. Bit values are:

0000001	creatdb permission
0000002	permits batch update override
0000004	permits update of system catalogs
0000020	can use trace flags
0000040	can turn off qrymod
0000100	can use arbitrary proctabs
0000200	can use the =proctab form
0100000	ingres superuser

* A list of flags automatically set for this user.
* The process table to use for this user.
* An initialization file to read be read by the monitor at login time.
* Unassigned.
* Comma seperated list of databases. If this list is null, the user may enter any database. If it begins with a '−', the user may enter any database except the named databases. Otherwise, the user may only enter the named databases.

EXAMPLE

```
ingres:aa:5:2:177777:−d:=special:/mnt/ingres/.ingres::
guest:ah:35:1:000000:::::demo,guest
```

SEE ALSO

initucode(util)

NAME

Error messages introduction

DESCRIPTION

This document describes the error returns which are possible from the INGRES data base system and gives an explanation of the probable reason for their occurrence. In all cases the errors are numbered *nxxx* where *n* indicates the source of the error, according to the following table:

1 = EQUEL preprocessor
2 = parser
3 = query modification
4 = decomposition and one variable query processor
5 = data base utilities
30 = GEO-QUEL errors

For a description of these routines the reader is referred to *The Design and Implementation of INGRES*. The *xxx* in an error number is an arbitrary identifier.

The error messages are stored in the file **.../files/error6.2_*n*,** where *n* is defined as above. The format of these files is the error number, a tab character, the message to be printed, and the tilde character ("~") to delimit the message.

In addition many error messages have "%*i*" in their body where *i* is a digit interpreted as an offset into a list of parameters returned by the source of the error. This indicates that a parameter will be inserted by the error handler into the error return. In most cases this parameter will be self explanatory in meaning.

Where the error message is thought to be completely self explanatory, no additional description is provided.

NAME

EQUEL error message summary

SYNOPSIS

Error numbers 1000 — 1999.

DESCRIPTION

The following errors can be generated at run time by EQUEL programs.

ERRORS

1000 In domain %0 numeric retrieved into char field.

Equel does not support conversion at run-time of numeric data from the data base to character string representation. Hence, if you attempt to assign a domain of numeric type to a C-variable of type character string, you will get this error message. To convert numerics to characters use the "ascii" function in QUEL.

1001 Numeric overflow during retrieve on domain %0.

You will get this error if you attempt to assign a numeric data base domain to a C-variable of a numeric type but with a shorter length. In this case the conversion routines may generate an overflow. For example, this error will result from an attempt to retrieve a large floating point number into a C-variable of type integer.

1002 In domain %0, character retrieved into numeric variable.

This error is the converse of error 1000.

1003 Bad type in target list of parameterized retrieve "%0".

Valid types are %f8, %f4, %i4, %i2, %c.

1004 Bad type in target list of parameterized statement "%0".

Valid types are %f8, %f4, %i4, %i2, %i1, %c.

NAME

Parser error message summary

SYNOPSIS

Error numbers 2000 – 2999.

DESCRIPTION

The following errors can be generated by the parser. The parser reads your query and translates it into the appropriate internal form; thus, almost all of these errors indicate syntax or type conflict problems.

ERRORS

2100 line %0, Attribute '%1' not in relation '%2'

This indicates that in a given line of the executed workspace the indicated attribute name is not a domain in the indicated relation.

2103 line %0, Function type does not match type of attribute '%1'

This error will be returned if a function expecting numeric data is given a character string or vice versa. For example, it is illegal to take the SIN of a character domain.

2106 line %0, Data base utility command buffer overflow

This error will result if a utility command is too long for the buffer space allocated to it in the parser. You must shorten the command or recompile the parser.

2107 line %0, You are not allowed to update this relation: %1

This error will be returned if you attempt to update any system relation or secondary index directly in QUEL (such as the RELATION relation). Such operations which compromise the integrity of the data base are not allowed.

2108 line %0, Invalid result relation for APPEND '%1'

This error message will occur if you execute an append command to a relation that does not exist, or that you cannot access. For example, append to junk(...) will fail if junk does not exist.

2109 line %0, Variable '%1' not declared in RANGE statement

Here, a symbol was used in a QUEL expression in a place where a tuple variable was expected and this symbol was not defined via a RANGE statement.

2111 line %0, Too many attributes in key for INDEX

A secondary index may have no more than 6 keys.

2117 line %0, Invalid relation name '%1' in RANGE statement

You are declaring a tuple variable which ranges over a relation which does not exist.

2118 line %0, Out of space in query tree - Query too long

You have the misfortune of creating a query which is too long for the parser to digest. The only options are to shorten the query or recompile the parser to have more buffer space for the query tree.

2119 line %0, MOD operator not defined for floating point or character attributes

The *mod* operator is only defined for integers.

2120 line %0, no pattern match operators allowed in the target list

Pattern match operators (such as "*") can only be used in a qualification.

2121 line %0, Only character type domains are allowed in CONCAT operator

2123 line %0, '%1.all' not defined for replace

2125 line %0, Cannot use aggregates ("avg" or "avgu") on character values
2126 line %0, Cannot use aggregates ("sum" or "sumu") on character values
2127 line %0, Cannot use numerical functions (ATAN, COS, GAMMA, LOG, SIN, SQRT, EXP, ABS) on character values
2128 line %0, Cannot use unary operators ("+" or "−") on character values
2129 line %0, Numeric operations (+ − * /) not allowed on character values

Many functions and operators are meaningless when applied to character values.

2130 line %0, Too many result domains in target list

Maximum number of result domains is MAXDOM (currently 49).

2132 line %0, Too many aggregates in this query

Maximum number of aggregates allowed in a query is MAXAGG (currently 49).

2133 line %0, Type conflict on relational operator

It is not legal to compare a character type to a numeric type.

2134 line %0, '%1' is not a constant operator.
Only 'dba' or 'usercode' are allowed.

2135 line %0, You cannot duplicate the name of an existing relation(%1)

You have tried to create a relation which would redefine an existing relation. Choose another name.

2136 line %0, There is no such hour as %1, use a 24 hour clock system
2137 line %0, There is no such minute as %1, use a 24 hour clock system
2138 line %0, There is no such time as 24:%1, use a 24 hour clock system

Errors 2136-38 indicate that you have used a bad time in a *permit* statement. Legal times are from 0:00 to 24:00 inclusive.

2139 line %0, Your database does not support query modification

You have tried to issue a query modification statement (*define*), but the database was created with the **−q** flag. To use the facilities made available by query modification, you must say:

creatdb −e +q *dbname*

to the shell.

2500 syntax error on line %0
last symbol read was: %1

A 2500 error is reported by the parser if it cannot otherwise classify the error. One common way to obtain this error is to omit the required parentheses around the target list. The parser reports the last symbol which was obtained from the scanner. Sometimes, the last symbol is far ahead of the actual place where the error occurred. The string "EOF" is used for the last symbol when the parser has read past the query.

2502 line %0, The word '%1', cannot follow a RETRIEVE command, therefore the command was not executed.
2503 line %0, The word '%1', cannot follow an APPEND command, therefore the command was not executed.
2504 line %0, The word '%1', cannot follow a REPLACE command, therefore the command was not executed.
2505 line %0, The word '%1', cannot follow a DELETE command, therefore the command was not executed.
2506 line %0, The word '%1', cannot follow a DESTROY command, therefore the command was not executed.
2507 line %0, The word '%1', cannot follow a HELP command, therefore the command was

not executed.

2508 line %0, The word '%1', cannot follow a MODIFY command, therefore the command was not executed.

2509 line %0, The word '%1', cannot follow a PRINT command, therefore the command was not executed.

2510 line %0, The word '%1', cannot follow a RETRIEVE UNIQUE command, therefore the command was not executed.

2511 line %0, The word '%1', cannot follow a DEFINE VIEW command, therefore the command was not executed.

2512 line %0, The word '%1', cannot follow a HELP VIEW, HELP INTEGRITY, or HELP PERMIT command, therefore the command was not executed.

2513 line %0, The word '%1', cannot follow a DEFINE PERMIT command, therefore the command was not executed.

2514 line %0, The word '%1', cannot follow a DEFINE INTEGRITY command, therefore the command was not executed.

2515 line %0, The word '%1', cannot follow a DESTROY INTEGRITY or DESTROY PERMIT command, therefore the command was not executed.

Errors 2502 through 2515 indicate that after an otherwise valid query, there was something which could not begin another command. The query was therefore aborted, since this could have been caused by misspelling **where** or something equally as dangerous.

2700 line %0, non-terminated string

You have omitted the required string terminator (").

2701 line %0, string too long

Somehow, you have had the persistence or misfortune to enter a character string constant longer than 255 characters.

2702 line %0, invalid operator

You have entered a character which is not alphanumeric, but which is not a defined operator, for example, "?".

2703 line %0, Name too long '%1'

In INGRES relation names and domain names are limited to 12 characters.

2704 line %0, Out of space in symbol table - Query too long

Your query is too big to process. Try breaking it up with more \go commands.

2705 line %0, non-terminated comment

You have left off the comment terminator symbol ("*/").

2707 line %0, bad floating constant: %1

Either your floating constant was incorrectly specified or it was too large or too small. Currently, overflow and underflow are not checked.

2708 line %0, control character passed in pre-converted string

In EQUEL a control character became embedded in a string and was not caught until the scanner was processing it.

2709 line %0, buffer overflow in converting a number

Numbers cannot exceed 256 characters in length. This shouldn't become a problem until number formats in INGRES are increased greatly.

NAME

Query Modification error message summary

SYNOPSIS

Error numbers 3000 – 3999.

DESCRIPTION

These error messages are generated by the Query Modification module. These errors include syntactic and semantic problems from view, integrity, and protection definition, as well as run time errors – such as inability to update a view, or a protection violation.

ERRORS

3310 %0 on view %1: cannot update some domain

You tried to perform operation %0 on a view; however, that update is not defined.

3320 %0 on view %1: domain occurs in qualification of view

It is not possible to update a domain in the qualification of a view, since this could cause the tuple to disappear from the view.

3330 %o on view %1: update would result in more than one query

You tried to perform some update on a view which would update two underlying relations.

3340 %0 on view %1: views do not have TID's

You tried to use the Tuple IDentifier field of a view, which is undefined.

3350 %0 on view %1: cannot update an aggregate value

You cannot update a value which is defined in the view definition as an aggregate.

3360 %0 on view %1: that update might be non-functional

There is a chance that the resulting update would be non-functional, that is, that it may have some unexpected side effects. INGRES takes the attitude that it is better to not try the update.

3490 INTEGRITY on %1: cannot handle aggregates yet

You cannot define integrity constraints which include aggregates.

3491 INTEGRITY on %1: cannot handle multivariable constraints

You cannot define integrity constraints on more than a single variable.

3492 INTEGRITY on %1: constraint does not initially hold

When you defined the constraint, there were already tuples in the relation which did not satisfy the constraint. You must fix the relation so that the constraint holds before you can declare the constraint.

3493 INTEGRITY on %1: is a view

You can not define integrity constraints on views.

3494 INTEGRITY on %1: You must own '%1'

You must own the relation when you declare integrity constraints.

3500 %0 on relation %1: protection violation

You have tried to perform an operation which is not permitted to you.

3590 PERMIT: bad terminal identifier "%2"

In a *permit* statement, the terminal identifier field was improper.

3591 PERMIT: bad user name "%2"

You have used a user name which is not defined on the system.

3592 PERMIT: Relation '%1' not owned by you

You must own the relation before issuing protection constraints.

3593 PERMIT: Relation '%1' must be a real relation (not a view)

You can not define permissions on views.

3594 PERMIT on %1: bad day-of-week '%2'

The day-of-week code was unrecognized.

3595 PERMIT on %1: only the DBA can use the PERMIT statement

Since only the DBA can have shared relations, only the DBA can issue *permit* statements.

3700 Tree buffer overflow in query modification
3701 Tree build stack overflow in query modification

Bad news. An internal buffer has overflowed. Some expression is too large. Try making your expressions smaller.

NAME

One Variable Query Processor error message summary

SYNOPSIS

Error numbers 4000 — 4499.

DESCRIPTION

These error messages can be generated at run time. The One Variable Query Processor actually references the data, processing the tree produced by the parser. Thus, these error messages are associated with type conflicts detected at run time.

ERRORS

4100 OVQP query list overflowed

This error is produced in the unlikely event that the internal form of your interaction requires more space in the one variable query processor than has been allocated for a query buffer. There is not much you can do except shorten your interaction or recompile OVQP with a larger query buffer.

4101 numeric operation using char and numeric domains not allowed

Occasionally, you will be notified by OVQP of such a type mismatch on arithmetic operations. This only happens if the parser has not recognized the problem.

4102 unary operators are not allowed on character values
4103 binary operators cannot accept combinations of char and numeric fields
4104 cannot use aggregate operator "sum" on character domains
4105 cannot use aggregate operator "avg" on character domains

These errors indicate type mismatches — such as trying to add a number to a character string.

4106 the interpreters stack overflowed -- query too long
4107 the buffer for ASCII and CONCAT commands overflowed

More buffer overflows.

4108 cannot use arithmetic operators on two character fields
4109 cannot use numeric values with CONCAT operator

You have tried to perform a numeric operation on character fields.

4110 floating point exception occurred.

If you have floating point hardware instead of the floating point software interpreter, you will get this error upon a floating point exception (underflow or overflow). Since the software interpreter ignores such exceptions, this error is only possible with floating point hardware.

4111 character value cannot be converted to numeric due to incorrect syntax.

When using int1, int2, int4, float4, or float8 to convert a character to value to a numeric value, the character value must have the proper syntax. This error will occur if the character value contained non-numeric characters.

4112 ovqp query vector overflowed

Similar to error 4100.

4199 you must convert your 6.0 secondary index before running this query!

The internal format of secondary indices was changed between versions 6.0 and 6.1 of INGRES. Before deciding to use a secondary index OVQP checks that it is not a 6.0 index. The solution is to destroy the secondary index and recreate it.

NAME

Decomposition error message summary

SYNOPSIS

Error numbers 4500 — 4999.

DESCRIPTION

These error messages are associated with the process of decomposing a multi-variable query into a sequence of one variable queries which can be executed by OVQP.

ERRORS

4602 query involves too many relations to create aggregate function intermediate result.

In the processing of aggregate functions it is usually necessary to create an intermediate relation for *each* aggregate function. However, no query may have more than ten variables. Since aggregate functions implicitly increase the number of variables in the query, you can exceed this limit. You must either break the interaction apart and process the aggregate functions separately or you must recompile INGRES to support more variables per query.

4610 Query too long for available buffer space (qbufsize).
4611 Query too long for available buffer space (varbufsiz)
4612 Query too long for available buffer space (sqsiz)
4613 Query too long for available buffer space (stacksiz)
4614 Query too long for available buffer space (agbufsiz).

These will happen if the internal form of the interaction processed by decomp is too long for the available buffer space. You must either shorten your interaction or recompile decomp. The name in parenthesis gives the internal name of which buffer was too small.

4615 Aggregate function is too wide or has too many domains.

The internal form of an aggregate function must not contain more than 49 domains or be more than 498 bytes wide. Try breaking the aggregate function into two or more parts.

4620 Target list for "retrieve unique" has more than 49 domains or is wider than 498 bytes.

NAME

Data Base Utility error message summary

SYNOPSIS

Error numbers 5000 – 5999

DESCRIPTION

The Data Base Utility functions perform almost all tasks which are not directly associated with processing queries. The error messages which they can generate result from some syntax checking and a considerable amount of semantic checking.

ERRORS

5001 PRINT: bad relation name %0

You are trying to print a relation which doesn't exist.

5002 PRINT: %0 is a view and can't be printed

The only way to print a view is by retrieving it.

5003 PRINT: Relation %0 is protected.

You are not authorized to access this relation.

5102 CREATE: duplicate relation name %0

You are trying to create a relation which already exists.

5103 CREATE: %0 is a system relation

You cannot create a relation with the same name as a system relation. The system depends on the fact that the system relations are unique.

5104 CREATE %0: invalid attribute name %1

This will happen if you try to create a relation with an attribute longer than 12 characters.

5105 CREATE %0: duplicate attribute name %1

Attribute names in a relation must be unique. You are trying to create one with a duplicated name.

5106 CREATE %0: invalid attribute format "%2" on attribute %1

The allowed formats for a domain are c1–c255, i1, i2, i4, f4 and f8. Any other format will generate this error.

5107 CREATE %0: excessive domain count on attribute %1

A relation cannot have more than 49 domains. The origin of this magic number is obscure. This is very difficult to change.

5108 CREATE %0: excessive relation width on attribute %1

The maximum number of bytes allowed in a tuple is 498. This results from the decision that a tuple must fit on one UNIX "page". Assorted pointers require the 14 bytes which separates 498 from 512. This "magic number" is very hard to change.

5201 DESTROY: %0 is a system relation

The system would immediately stop working if you were allowed to do this.

5202 DESTROY: %0 does not exist or is not owned by you

To destroy a relation, it must exist, and you must own it.

5203 DESTROY: %0 is an invalid integrity constraint identifier

Integers given do not identify integrity constraints on the specified relation. For example: If you were to type "destroy permit parts 1, 2, 3", and 1, 2, or 3 were not the

numbers "help permit parts" prints out for permissions on parts, you would get this error.

5204 DESTROY: %0 is an invalid protection constraint identifier

Integers given do not identify protection constraints on the specified relation. Example as for error 5203.

5300 INDEX: cannot find primary relation

The relation does not exist — check your spelling.

5301 INDEX: more than maximum number of domains

A secondary index can be created on at most six domains.

5302 INDEX: invalid domain %0

You have tried to create an index on a domain which does not exist.

5303 INDEX: relation %0 not owned by you

You must own relations to put indicies on them.

5304 INDEX: relation %0 is already an index

INGRES does not permit tertiary indicies.

5305 INDEX: relation %0 is a system relation

Secondary indices cannot be created on system relations.

5306 INDEX: %0 is a view and an index can't be built on it

Since views are not physically stored in the database, you cannot build indicies on them.

5401 HELP: relation %0 does not exist

5402 HELP: cannot find manual section "%0"

Either the desired manual section does not exist, or your system does not have any on-line documentation.

5403 HELP: relation %0 is not a view

Did a "help view" (which prints view definition) on a nonview. For example: "help view overpaidv" prints out overpaidv's view definition.

5404 HELP: relation %0 has no permissions on it granted
5405 HELP: relation %0 has no integrity constraints on it

You have tried to print the permissions or integrity constraints on a relation which has none specified.

5410 HELP: tree buffer overflowed
5411 HELP: tree stack overflowed

Still more buffer overflows.

5500 MODIFY: relation %0 does not exist

5501 MODIFY: you do not own relation %0

You cannot modify the storage structure of a relation you do not own.

5502 MODIFY %0: you may not provide keys on a heap

By definition, heaps do not have keys.

5503 MODIFY %0: too many keys provided

You can only have 49 keys on any relation.

5504 MODIFY %0: cannot modify system relation

System relations can only be modified by using the *sysmod* command to the shell; for example

sysmod *dbname*

5507 MODIFY %0: duplicate key "%1"

You may only specify a domain as a key once.

5508 MODIFY %0: key width (%1) too large for isam

When modifying a relation to isam, the sum of the width of the key fields cannot exceed 245 bytes.

5510 MODIFY %0: bad storage structure "%1"

The valid storage structure names are heap, cheap, isam, cisam, hash, and chash.

5511 MODIFY %0: bad attribute name "%1"

You have specified an attribute that does not exist in the relation.

5512 MODIFY %0: "%1" not allowed or specified more than once

You have specified a parameter which conflicts with another parameter, is inconsistant with the storage mode, or which has already been specified.

5513 MODIFY %0: fillfactor value %1 out of bounds

Fillfactor must be between 1 and 100 percent.

5514 MODIFY %0: minpages value %1 out of bounds

Minpages must be greater than zero.

5515 MODIFY %0: "%1" should be "fillfactor", "maxpages", or "minpages"

You have specified an unknown parameter to *modify*.

5516 MODIFY %0: maxpages value %1 out of bounds

5517 MODIFY %0: minpages value exceeds maxpages value

5518 MODIFY %0: invalid sequence specifier "%1" for domain %2.

Sequence specifier may be "ascending" (or "a") or "descending" (or "d") in a *modify*. For example:

```
modify parts to heapsort on
pnum:ascending,
pname:descending
```

5519 MODIFY: %0 is a view and can't be modified

Only physical relations can be modified.

5520 MODIFY: %0: sequence specifier "%1" on domain %2 is not allowed with the specified storage structure.

Sortorder may be supplied only when modifying to **heapsort** or **cheapsort.**

5600 SAVE: cannot save system relation "%0"

System relations have no save date and are guaranteed to stay for the lifetime of the data base.

5601 SAVE: bad month "%0"
5602 SAVE: bad day "%0"
5603 SAVE: bad year "%0"

This was a bad month, bad day, or maybe even a bad year for INGRES.

5604 SAVE: relation %0 does not exist or is not owned by you

5800 COPY: relation %0 doesn't exist

5801 COPY: attribute %0 in relation %1 doesn't exist or it has been listed twice

5803 COPY: too many attributes

Each dummy domain and real domain listed in the copy statement count as one attribute. The limit is 150 attributes.

5804 COPY: bad length for attribute %0. Length="%1"

5805 COPY: can't open file %0

On a copy "from", the file is not readable by the user.

5806 COPY: can't create file %0

On a copy "into", the file is not creatable by the user. This is usually caused by the user not having write permission in the specified directory.

5807 COPY: unrecognizable dummy domain "%0"

On a copy "into", a dummy domain name is used to insert certain characters into the unix file. The domain name given is not valid.

5808 COPY: domain %0 size too small for conversion.
There were %2 tuples successfully copied from %3 into %4

When doing any copy except character to character, copy checks that the field is large enough to hold the value being copied.

5809 COPY: bad input string for domain %0. Input was "%1". There were %2 tuples successfully copied from %3 into %4

This occurs when converting character strings to integers or floating point numbers. The character string contains something other than numeric characters (0-9,+,−,blank,etc.).

5810 COPY: unexpected end of file while filling domain %0.
There were %1 tuples successfully copied from %2 into %3

5811 COPY: bad type for attribute %0. Type="%1"

The only accepted types are i, f, c, and d.

5812 COPY: The relation "%0" has a secondary index. The index(es) must be destroyed before doing a copy "from"

Copy cannot update secondary indices. Therefore, a copy "from" cannot be done on an indexed relation.

5813 COPY: You are not allowed to update the relation %0

You cannot copy into a system relation or secondary index.

5814 COPY: You do not own the relation %0.

You cannot use copy to update a relation which you do not own. A copy "into" is allowed but a copy "from" is not.

5815 COPY: An unterminated "c0" field occurred while filling domain %0. There were %1 tuples successfully copied from %2 into %3

A string read on a copy "from" using the "c0" option cannot be longer than 1024 characters.

5816 COPY: The full pathname must be specified for the file %0

The file name for copy must start with a "/".

5817 COPY: The maximum width of the output file cannot exceed 1024 bytes per tuple

The amount of data to be output to the file for each tuple exceeds 1024. This usually happens only if a format was mistyped or a lot of large dummy domains were specified.

5818 COPY: %0 is a view and can't be copied

Only physical relations can be copied.

5819 COPY: Warning: %0 duplicate tuples were ignored.

On a copy "from", duplicate tuples were present in the relation.

5820 COPY: Warning: %0 domains had control characters which were converted to blanks.

5821 COPY: Warning: %0 c0 character domains were truncated.

Character domains in c0 format are of the same length as the domain length. You had a domain value greater than this length, and it was truncated.

5822 COPY: Relation %0 is protected.

You are not authorized to access this relation.

NAME

GEO-QUEL error message summary

SYNOPSIS

Error numbers 30000 – 30999.

DESCRIPTION

The following errors can be generated by the GEO-QUEL subsystem of INGRES. These errors are all associated with errors in the graphics processor.

ERRORS

30210 GEO-QUEL is creating a MAPRELATION relation

30300 line %0, In attempting to do a MAP, SHADE, or OVERLAY the relation '%1' does not exist or is not owned by you

30301 line %0, Failed to obtain information to put a tuple in MAPRELATION relation

30310 line %0, Failed to complete MAP

30330 line %0, Failed while building temporary relation in SHADE command

30331 line %0, Maximum polygon value is 0.0 which causes floating point exception

30332 line %0, Too many sides (maxside) in polygon (zoneid: %1, groupid: %2)

30340 line %0, Failed while trying to obtain range limits for axes in POINTGRAPH or LINE-GRAPH

30341 line %0, The linetype specified (%1) in the linegraph command does not make sense for the device

30342 line %0, %1 does not exist or cannot be accessed by you

30350 line %0, There is no current center definition for this map: %1

30400 line %0, Error in creating destination file '%1' in SAVEMAP

30401 line %0, Too many domains for SHADE(shadedoms)

30402 line %0, Too many domains for POINTGRAPH of LINEGRAPH (graphdoms)

30410 line %0, Error in creating destination file '%1' in SAVEMAP

30420 line %0, You may specify only one boundary when using 'cellcount' and 'cellwidth'

30421 line %0, Only one cellwidth per command has meaning

30422 line %0, Only one cellcount per command has meaning

30423 line %0, Only one upper bound per command has meaning

30424 line %0, Only one lower bound per command has meaning

30425 line %0, %1 does not exist or cannot be accessed by you