Research sponsored by the Air Force Office of Scientific Research Grant 78-3596 and the U.S. Army Research Office Grant DAAG29-79-C-0182.
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 brackets ([ ]) 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)
- newemp(name, sal, age)
- 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).

ACKNOWLEDGMENTS
We would like to acknowledge the people who have worked on INGRES in the past:
INTRODUCTION (INGRES) 3/1/81

Rick Birman
Bob Epstein
James Ford
Paula Hawthorn
Gerald Held
Peter Kreps
Dan Ries
Peter Rubinstein
Mike Ubell
Nick Whyte
Karel Youssefi
William Zook

FOOTNOTE
UNIX is a trademark of Bell Laboratories.
APPEND(QUEL) — append tuples to a relation
append [to] relname (target_list) [where qual]

COPY(QUEL) — copy data into/from a relation from/into a UNIX file.
copy relname (domname1 = format {, domname2 = format })
   direction "filename"

CREATE(QUEL) — create a new relation
create relname (domname1 = format {, domname2 = format })

DEFINE(QUEL) — define subschema
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

DELETE(QUEL) — delete tuples from a relation
delete tuple_variable [where qual]

DESTROY(QUEL) — destroy existing relation(s)
destroy relname {, relname}
destroy [permit | integrity] relname [integer {, integer] all]

HELP(QUEL) — get information about how to use INGRES or about relations in the
database.
help [relname] ["section"] {, relname}{, "section"}
helper view relname {, relname}
helper permit relname {, relname}
helper integrity relname {, relname}

INDEX(QUEL) — create a secondary index on an existing relation.
index on relname is indexname (domain1 {, domain2})

INTEGRITY(QUEL) — define integrity constraints
define integrity on var is qual

MACROS(QUEL) — terminal monitor macro facility

MODIFY(QUEL) — convert the storage structure of a relation

MONITOR(QUEL) — interactive terminal monitor

PERMIT(QUEL) — add permissions to a relation
define permit oplist { on | of | to } var [ (attlist) ]
to name [ at term ] [ from time to time ] [ on day to day ] [ where qual ]

PRINT(QUEL) — print relation(s)
print relname {, relname}

QUEL(QUEL) — QUEry Language for INGRES

RANGE(QUEL) — declare a variable to range over a relation
range of variable is relname

REPLACE(QUEL) — replace values of domains in a relation
replace tuple_variable (target_list) [where qual]

RETRIEVE(QUEL) — retrieve tuples from a relation
retrieve [[into] relname] (target_list) [where qual]
retrieve unique (target_list) [where qual]
SAVE(QUEL) — save a relation until a date.
    save relname until month day year

VIEW(QUEL) — define a virtual relation
    define view name (target-list) [ where qual ]

COPYDB(UNIX) — create batch files to copy out a database and restore it.
    copydb [ -uname ] database full-path-name-of-directory [ relation ... ]

CREATDB(UNIX) — create a database
    creatdb [ -uname ] [ -e ] [ -m ] [ ±c ] [ ±q ] dbname

DESTROYDB(UNIX) — destroy an existing database
    destroydb [ -s ] [ -m ] dbname

EQUEL(UNIX) — Embedded QUEL interface to C
    equel [ -d ] [ -f ] [ -r ] file.q...

HELPR(UNIX) — get information about a database.
    helpr [ -uname ] [ ±w ] database relation ...

INGRES(UNIX) — INGRES relational database management system
    ingres [ flags ] dbname [ process_table ]

PRINTR(UNIX) — print relations
    printr [ flags ] database relation ...

PURGE(UNIX) — destroy all expired and temporary relations
    purge [ -t ] [ ±p ] [ -a ] [ ±w ] [ database ... ]

RESTORE(UNIX) — recover from an INGRES or UNIX crash.
    restore [ -a ] [ -s ] [ ±w ] [ database ... ]

SYSMOD(UNIX) — modify system relations to predetermined storage structures.
    sysmod [ -s ] [ -w ] dbname [ relation ] [ attribute ] [ indexes ] [ tree ] [ protect ] [ integrities ]

USERSETUP(UNIX) — setup users file
    /bin/usersetup [ pathname ]

DAYFILE(FILES) — INGRES login message

DBTMLTPLT(FILES) — database template

ERROR(FILES) — files with INGRES errors

LIBQ(FILES) — Equel run-time support library

PROCKETTAB(FILES) — INGRES runtime configuration information

STARTUP(FILES) — INGRES startup file

USERS(FILES) — INGRES user codes and parameters

INTRODUCTION(ERROR) — Error messages introduction

EQUEL(HEADER) — EQUEL error message summary
    Error numbers 1000 — 1999.

PARSER(HEADER) — Parser error message summary
    Error numbers 2000 — 2999.

QRYMOD(HEADER) — Query Modification error message summary
    Error numbers 3000 — 3999.

OVQP(HEADER) — One Variable Query Processor error message summary
    Error numbers 4000 — 4499.

DECOMP(HEADER) — Decomposition error message summary
    Error numbers 4500 — 4999.
DBU(ERROR) — Data Base Utility error message summary
Error numbers 5000 — 5999
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 rel-
name. Format indicates the format the UNIX file should have for the correspond-
ding 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 sys-
tem 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 per-
mission 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 neces-
sary.

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. Delimiters for vari-
able length strings and for dummy domains can be selected from the list of:

```
nl — new line character
tab — tab character
sp — space
null 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 delimiter can be turned off by preceding the delimi-
tor with a \"\`. The type specifier can optionally be in quotes ("codelim"). This is
usefully if you wish to use a single character delimiter which has special mean-

ing 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:

- **c0delim** - The data in the UNIX file is a variable length character string terminated by the delimiter `delim`. If `delim` is missing then the first comma, tab, or newline encountered will terminate the string. The delimiter 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=c0z` - string ending in the character 'z'.
  - `pnum="c0%"` - string ending in the character '%'.

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

- **d0delim** - The data in the UNIX file is a variable length character string delimited by `delim`. The string is read and discarded. The delimiter rules are identical for `c0` and `d0`. The domain name is ignored.

- **dl, 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 written 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 '-f', '-f', and '-c' flags (see ingres(unix)).

- **c0delim** - The domain will be converted according to the rules for `c0` above. The one character delimiter will be inserted immediately after the domain.

- **dl, d2, ..., d255** - The domain name is taken to be the name of the delimiter. 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.

- **d0delim** - 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**

```plaintext
/* 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 (domname1 = format {, domname2 = format })

DESCRIPTION
Create will enter a new relation into the database. 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 state-
ment, 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 rela-
tion.

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 rename { , rename }
destroy [ permit | integrity ] rename [ 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 permit 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 permit relname {, relname}\} — Prints permissions on specified relations.
help integrity relname {, relname}\} — Prints integrity constraints on specified relations.

The permit 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 permit 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 (domain1 { ,domain2})

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

```bash
/* 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
integration — 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 diag-
nostic is issued and the statement is rejected.

In the current implementation, integrity constraints are not flagged — bad up-
dates are simply (and silently) not performed.

Qual must be a single variable qualification and may not contain any aggre-
gates.

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 re-
moved 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 exam-
ple, the template "ret" causes the corresponding macro to be invoked upon en-
countering the word "ret" in the input stream. When a macro is encountered,
the template part is removed and replaced with the replacement part. For ex-
ample, if the replacement part of the "ret" macro was "retrieve", then all in-
stances of the word "ret" in the input text would be replaced with the word "re-
trieve", as in the statement

    ret (p.all)

Macros may have parameters, indicated by a dollar sign. For example, the tem-
plate "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 ex-
ample, 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; $rj

where $t and $r are the template and replacement parts of the macro, respec-
tively.

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 $rj

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 preceding 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.
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};; \n {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 hash 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 abbreviated 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 \texttt{minpages} and \texttt{maxpages} are specified in a \texttt{modify}, \texttt{minpages} cannot exceed \texttt{maxpages}.

Default values for \texttt{fillfactor}, \texttt{minpages}, and \texttt{maxpages} are as follows:

\begin{tabular}{|c|c|c|}
\hline
\textbf{FILLFACTOR} & \textbf{MINPAGES} & \textbf{MAXPAGES} \\
\hline
hash & 50 & no limit \\
chash & 75 & no limit \\
isam & 80 & NA \\
cisam & 100 & NA \\
\hline
\end{tabular}

\textbf{EXAMPLES}

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

\texttt{modify emp to isam on name}

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

\texttt{modify emp to isam}

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

\texttt{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 */

\texttt{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 
and a maximum of 43 primary pages */

\texttt{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 */

\texttt{modify supply to chash on num, quan where minpages = 16}

/* modify parts to a heap storage structure */

\texttt{modify parts to heap}

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

\texttt{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 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 pertinent 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
\go
\print
\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 preceded 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.

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

\print  Print the current query. The contents of the buffer are printed on the user's terminal.
\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.
\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.
\go    Process the current query. The contents of the buffer are macro processed, transmitted to INGRES, and run.
\append Append to the query buffer. Typing \append 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.
\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.
\quit  Exit from INGRES.
\cd    Change the working directory of the monitor to the named directory.
\include \read  Switch input to the named file. Backslash characters in the file will be processed as read.
\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 preceded 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 ".inges" in the user’s home directory. The startup file might be used to define certain macros, execute common range statements, and so forth. 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**

inges(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.
You have entered the UNIX text editor.
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 \( on \mid of \mid to \) 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*' 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` statement.

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 no one 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)

BUGS
Print does not handle long lines of output correctly — no wrap around.
Print should have more formatting features to make printouts more readable.
Print should have an option to print on the line printer.
NAME
queL – QUEry Language 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:

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

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:

$$\{<\text{dig}>\} [.<\text{dig}>] [e|E [+-] <\text{dig}>]$$

Where $<\text{dig}>$ 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:

$$\text{variable}, \text{domain}$$

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 referred 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)
- **f8**: 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:

<table>
<thead>
<tr>
<th></th>
<th>i1</th>
<th>i2</th>
<th>i4</th>
<th>f4</th>
<th>f8</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>i1</td>
<td>i2</td>
<td>i4</td>
<td>f4</td>
<td>f8</td>
</tr>
<tr>
<td>i2</td>
<td>i2</td>
<td>i2</td>
<td>i4</td>
<td>f4</td>
<td>f8</td>
</tr>
<tr>
<td>i4</td>
<td>i4</td>
<td>i4</td>
<td>i4</td>
<td>f4</td>
<td>f8</td>
</tr>
<tr>
<td>f4</td>
<td>f4</td>
<td>f4</td>
<td>f4</td>
<td>f4</td>
<td>f8</td>
</tr>
<tr>
<td>f8</td>
<td>f8</td>
<td>f8</td>
<td>f8</td>
<td>f4</td>
<td>f8</td>
</tr>
</tbody>
</table>

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 occurs 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 atname is a_expr**

Result atname 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)
\texttt{cos(n) – f8 (cosine) }
\texttt{exp(n) – f8 (exponential of n) }
\texttt{gamma(n) – f8 (log gamma) }
\texttt{log(n) – f8 (natural logarithm) }
\texttt{mod(n, b) – same as b \ (n \ modulo \ b. \ n \ and \ b \ must \ be \ i1, \ i2, \ or \ i4) }
\texttt{sin(n) – f8 (sine) }
\texttt{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.

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

15.2. Simple aggregate

\texttt{aggregation\_operator \ (a\_expr \ [ \ where \ qual \ ] \ )}

A simple aggregate evaluates to a single scalar value. \texttt{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 \texttt{sum} and \texttt{avg} require numeric type \texttt{a\_expr}; \texttt{count}, \texttt{any}, \texttt{max} and \texttt{min} permit a character type attribute as well as numeric type \texttt{a\_expr}.

\textit{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 \texttt{count} and checking whether the return is zero or non-zero. Using \texttt{any} instead of \texttt{count} is more efficient since processing is stopped, if possible, the first time a tuple satisfies a qualification.

\texttt{any} returns 1 if the qualification is true and 0 otherwise.

15.4. Aggregate functions

\texttt{aggregation\_operator \ (a\_expr \ by \ by\_domain \ 
\{, \ by\_domain\} \ [ \ where \ qual \ ] \ )}

Aggregate functions are extensions of simple aggregates. The \texttt{by} operator groups (i.e. partitions) the set of qualifying tuples by \texttt{by\_domain} values. For more than one \texttt{by\_domain}, the values which are grouped by are the concatenation of individual \texttt{by\_domain} values. \texttt{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 print-
ing 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 permission 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 rela-
tion will have domain names as specified in the target_list result_attnames. The
new relation will be saved on the system for seven 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 in-
voked (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 pro-
cess.

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.

SYNONYMS
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(quei), retrieve(quei), 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 statement; 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 anomalies 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)
COPYDB (UNIX) 3/14/79

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

SYNOPSIS
copydb [ -uname ] database full-path-name-of-directory [ relation ... ]

DESCRIPTION
Copydb creates two INGRES command files in the directory: Copy.out, which contains Quel instructions 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 r1 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 database

SYNOPSIS
creatdb [-uname ] [-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 +z turns an option on, while -z turns an option off. The -c flag turns off the concurrency control scheme (default on). The +q flag turns on query modification (default on).

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 600. 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 -u eric -q erics_db
creatdb -e +q -c -u:av erics_db

FILES

.../files/dbtmpl7
.../files/data/base/*
.../files/datadir/* (for compatibility with previous versions)

SEE ALSO

demodb(unix), destroydb(unix), users(files), chmod(1), 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 da-
tabase.

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 [-a] [-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 -a
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

Equet provides the user with a method of interfacing the general purpose pro-
gramming 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 filen.q are the source input file names, which must end with .q. The output
is written to the file "filen.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 pro-
  cess space. Defaults to on.

- 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 then be compiled using the C compiler:

cc file1.c file2.c ...

The -lq requests the use of the EQUEL object library.

All EQUEL routines and globals begin with the characters "Il", and so all globals
variables and procedure names of the form Ilzzz 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 ter-
inal 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 asciit function may be used for this purpose.)

Also, the following EQUEL commands are permitted.

```c
## ingres [ingres flags] database_name
```

This command starts INGRES running, and directs all dynamically following queries to the database `database_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
```

```c
## 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:

```c
## 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:

```c
/* 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:

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

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

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

```c
## int Gint;

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

demo = "my_database";

```c
/* 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 occurred as its single argument. The error message corresponding to the error number returned by `IIprint_err()` will be printed. Printing will be suppressed if `IIprint_err()` returns 0. `IIprint_err` may be reassigned to, and is useful 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 retrieve 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:

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

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

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

### FILES

.../files/error?

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 11 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 [ -uname ] [ ±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
ingles(unix), help(quei)

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 in-
dices. 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 de-
bugging and is strongly discouraged for normal use.

-uname  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.

-cN  Set the minimum field width for printing character domains to N. The
default is 6.

-UN  Set integer output field width to N. l may be 1, 2, or 4 for 11's, i2's, or
i4's respectively.

-llxM.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.

-wX  Set the column seperator for retrieves to the terminal and print com-
mands to be X. The default is vertical bar.

-rM  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 de-
fault is "cheapsort".

-nM  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 precisely if the standard input is from a
terminal), otherwise the wait option is invoked.
Process file 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 compatibility with previous versions
.../files/proctab? — 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
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 exe-
cute this command.

Flags accepted are `-u`, `±w`, `-c`, `-l`, `-f`, and `¬w`. Their meanings are identical to
the meanings of the same flags in INGRES.

SEE ALSO
ingres(unix), print(quei)

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 is 'y'. Any other response is interpreted as 'no'.

Purge will lock the database while it is being processed, since errors may occur if the database is active while purge is working on the database. If a database is busy purge will report this and go on to the next database, if any. If standard input is not a terminal purge will wait for the database 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(quei), 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 database after an INGRES or UNIX crash. It should always be run after any abnormal termination to ensure the integrity of the database.

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 -a flag.

If the -a flag is specified you will be asked before restore takes any 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 newly created or destroyed relations. Purge must be called with the -f flag to accomplish this.

Restore locks the database while it is being processed. If a database is busy restore will report this and go on to the next database. If standard input is not a terminal restore will wait for the database 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.
"rename 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 database administrator for the specified database, or be the INGRES superuser and have the -s flag stated.

Sysmod should be run on a database 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 database is being used while sysmod is running, errors will occur. Therefore, sysmod will lock the database while it is being processed. If the database is busy, sysmod will report this. If standard input is not a terminal sysmod will wait for the database 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, protect, and integrities relations have the minpages value set at 5.

SEE ALSO
modify(QUEL)
NAME
usersetup — setup users file

SYNOPSIS
.../bin/usersetup [ flags [ 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 path-
name 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
 0000001, except for user ingres, which is initialized to all permission bits set. If
 the status parameter is provided, the field is set to this instead. The "initializa-
tion 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 en-
tirely, 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 edited 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 as-
sign that user a code in the form "aa" and enter a line in the users file in the

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
every 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 =proctab7), 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/dayfile7 — 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/dbtmtplt - 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
reilowner c2
reispec i1

attribute:−c−20
attrrelid c12
attowner c2
attname c12

(other relation descriptors)

SEE ALSO
creatdb(unix)
NAME
.../files/error7_? - files with INGRES errors

DESCRIPTION
These files contain the INGRES error messages. There is one file for each thousands digit; e.g., error number 2313 will be in file error7_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.

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
libq — Equel run-time support library

DESCRIPTION
Libq contains all the routines necessary for an equel program to load. It typically resides in /usr/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 llatoi(buf, i)
char *buf;
int i;

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

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

char *llitos(i)
int i;

int l1lsequal(s1, s2)
char *s1, *s2;

int lllength(string)
char *string;

llsyserr(string, arg1, arg2, ...);
char *string;

Ilatoi Ilatoi is equivalent to atoi(UTIL).
Ilbmove Moves len bytes from source to destination, returning a pointer to the location after the last byte moved. Does not append a null byte.
Ilconcatv 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.
Ilitos Ilitos is equivalent to itoa(III).
Ilsequal Returns 1 iff strings s1 is identical to s2.
Illength Returns max(length of string without null byte at end, 255)
Ilsyserr Ilsyserr is different from syserr(util) only in that it will print the name in llproc_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:

```c
int IIerrflag;
char *IImainpr;
char (*IIprint_err)();
int IIret_errp;
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 occurred. This remains valid from the time the error occurs to the time when the next equal statement is issued. This may be used just after an equal statement to see if it succeeded.

**IImainpr** This is a string which determines which ingress 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 ingress 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 occurred.

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

**IIno_err** Returns 0. Used to have (*IIprint_err)() suppress error message printing. IIno_err is used when an error in a parametrized equal statement 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/proctab7 - INGRES runtime configuration information

DESCRIPTION
The process table describes the runtime configuration of the INGRES system. Each line of the process table has a special meaning depending on the first character of the line. Blank lines and lines beginning with an asterisk are comments. All other lines have a sequence of fields separated by commas. Pipe descriptor fields are lower case letters or digits; if they are digits they are replaced by file descriptors from the `flag or the @ flag.

D defines a macro. The first field is a single character macro name. The second field is the string to use as the value. Macros are expanded using "$x" where x is the macro name. The macro "P" is predefined to be the pathname of the INGRES subtree.

P introduces a process description. All lines up to an end of file or another P line describe a single process. The first field is the process number. The next field is the pathname of the binary to execute for this process. The third field is the name of the process to use for printing messages. The fourth field must be a single character lower case letter representing the input pipe that is normally read when nothing special is happening, or a vertical bar followed by a single digit, meaning to read from that file descriptor. The next field is a set of flags in octal regarding processing of this process; these are described below. The final field is a single letter telling what trace flag this process uses.

L defines what modules are defined locally by this process. The first field is the module number used internally. The second field is a set of flags describing processing of this module: the only bit defined is the 0001 bit which allows this module to be executed directly by the user. The third field is the function number in the process which defines this module. The final field is the module number to be executed after this module completes; zero is nothing (return).

R defines modules that are known to this process but which must be passed to another process for execution. The first field is the process number the modules will be found in. The second field is the pipe to write to get to that process. The third field is the pipe to read to get a response from that process. The fourth field is a set of flags: 0001 means to write the output pipe if you get a broadcast message, 0002 means that the process is physically adjacent on the read pipe, and 0004 means that the process is adjacent on the write pipe. The fifth and subsequent fields are the module numbers that are defined by this process.

The status bits for the P line are as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>close diagnostic output</td>
</tr>
<tr>
<td>000004</td>
<td>close standard input</td>
</tr>
<tr>
<td>000002</td>
<td>run in user's directory, not database</td>
</tr>
<tr>
<td>000001</td>
<td>run as the user, not as INGRES</td>
</tr>
</tbody>
</table>

The lowest numbered process becomes the parent of all the other processes.

WARNING: Giving a user permission to specify his or her own process table will allow them to bypass all protection provided by INGRES. This facility should be provided for system debugging only!

EXAMPLE
The following example will execute a three process system.

```
DB:$P/bin
DS:$P/source
**** Process 0 - terminal monitor
```
P0: $B/monitor:MONITOR:h:0003:M
L0: 0:0:0
R1: 0:a:h:0007:1
* Process 1 — parser
L3: 1:0:0
R0: 0:h:a:0006:0
R2: 0:b:g:0007:5:6:7
* Process 2 — data base utilities
P2: $B/alldbu:DBU:b:0014:Z
L5: 0:6:0
L6: 0:0:0
L7: 0:1:0
R0: 0:g:b:0000
R1: 0:g:b:0006
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
...

DESCRIPTION
This file contains the user information in fields separated 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 separated 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.

Giving permission to a user to use arbitrary process tables is tantamount to
turning off the protection system for that user.

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 oc-
currence. In all cases the errors are numbered \textit{nxxx} where \( n \) indicates the
source of the error, according to the following table:

\begin{itemize}
  \item \( 1 = \) EQUEL preprocessor
  \item \( 2 = \) parser
  \item \( 3 = \) query modification
  \item \( 4 = \) decomposition and one variable query processor
  \item \( 5 = \) data base utilities
  \item \( 30 = \) GEO-QUEL errors
\end{itemize}

For a description of these routines the reader is referred to \textit{The Design and Im-
plementation of INGRES}. The \textit{xxx} in an error number is an arbitrary identifier.

The error messages are stored in the file \texttt{.../files/error7\_\_\_\_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 "\%\texttt{i}" in their body where \( i \) is a digit inter-
preted 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 er-
ror return. In most cases this parameter will be self explanatory in meaning.

Where the error message is thought to be completely self explanatory, no addi-
tional description is provided.
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
2000 %0 errors were found in quel program
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 recom-
pile 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 re-
lation that does not exist, or that you cannot access. For example, ap-
pend 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 state-
ment.
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 attrib-
utes
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 line %0, The word "%1", cannot follow this command
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.

2501 line %0, The word "%1", cannot follow a RETRIEVE command

2502 line %0, The word "%1", cannot follow an APPEND command
Errors 2502 through 2528 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.

Syntax errors:

- Syntax error on line %0: The word '%%!' cannot follow a REPLACE command
- Syntax error on line %0: The word '%%!' cannot follow a DELETE command
- Syntax error on line %0: The word '%%!' cannot follow a DESTROY command
- Syntax error on line %0: The word '%%!' cannot follow a HELP command
- Syntax error on line %0: The word '%%!' cannot follow a MODIFY command
- Syntax error on line %0: The word '%%!' cannot follow a PRINT command
- Syntax error on line %0: The word '%%!' cannot follow a RETRIEVE UNIQUE command
- Syntax error on line %0: The word '%%!' cannot follow a DEFINE VIEW command
- Syntax error on line %0: The word '%%!' cannot follow a HELP VIEW, HELP INTEGRITY, or HELP PERMIT command
- Syntax error on line %0: The word '%%!' cannot follow a DEFINE PERMIT command
- Syntax error on line %0: The word '%%!' cannot follow a DEFINE INTEGRITY command
- Syntax error on line %0: The word '%%!' cannot follow a DESTROY INTEGRITY or DESTROY PERMIT command

Errors 2502 through 2528 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.

Syntax errors:

- Syntax error on line %0: The last symbol read was: '%%!
- Syntax error on line %0: The correct syntax is:
  
  - RETRIEVE [[INTO]rename] (target_list) [WHERE qual]
  - RETRIEVE UNIQUE (target_list) [WHERE qual]
- Syntax error on line %0: The correct syntax is:
  
  - APPEND [TO] rename (target_list) [WHERE qual]
- Syntax error on line %0: The correct syntax is:
  
  - REPLACE tuple_variable (target_list) [WHERE qual]
- Syntax error on line %0: The correct syntax is:
  
  - DELETE tuple_variable [WHERE qual]
- Syntax error on line %0: The correct syntax is:
  
  - COPY rename (domname = format, domname2 = format)) direction
- Syntax error on line %0: The correct syntax is:
  
  - CREATE rename (domname1 = format, domname2 = format)
- Syntax error on line %0: The correct syntax is:
  
  - DESTROY rename {}, rename)
  - DESTROY [PERMIT | INTEGRITY] rename [integer integer] | ALL
- Syntax error on line %0: The correct syntax is:
  
  - INDEX ON rename IS indexname (domain1, domain2)
- Syntax error on line %0: The correct syntax is:
  
  - MODIFY rename TO storage-structure [ON key1 [: sortord]
  [key2 [:sortorder]]) [WHERE [FILLFACTOR = n] [, MINPAGES = n] [, MAXPAGES = n]]
- Syntax error on line %0: The correct syntax is:
  
  - PRINT rename{}, rename
- Syntax error on line %0: The correct syntax is:
  
  - RANGE OF variable IS rename
- Syntax error on line %0: The correct syntax is:
  
  - SAVE rename UNTIL month day year
- Syntax error on line %0: The correct syntax is:
  
  - 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
- Syntax error on line %0: The correct syntax is:
  
  - RETRIEVE UNIQUE (target_list) [WHERE qual]
- Syntax error on line %0: The correct syntax is:
DEFINE VIEW name (target_list) [WHERE qual]

2619 line %0, Syntax error on '%1', the correct syntax is:
HELP VIEW rename[, rename]
HELP PERMIT rename[, rename]
HELP INTEGRITY rename[, rename]

2622 line %0, Syntax error on '%1', the correct syntax is:
DEFINE PERMIT opUst {ON|OFF|TO} var [(attUst)] TO name [AT term] [FROM
time TO time] [ON day TO day] [WHERE qual]

2623 line %0, Syntax error on '%1', the correct syntax is:
DEFINE INTEGRITY ON var IS qual

Errors 2600 through 2623 are generated when a command's syntax has
been violated. The correct syntax is given. If the command cannot be
determined, error 2600 is given.

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 charac-
ter 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 com-
mands.

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
catched 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.

2800 line %0, yacc stack overflow in parsing query
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 %0 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.

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.

4113 compiler text space ran out
4114 compiler ran out of registers

These errors refer to an experimental version of the system that is not currently released.

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 inter-
mediate 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 vari-
ables 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
deomp is too long for the available buffer space. You must either short-
ten your interaction or recompile deomp. 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 1010 bytes wide. Try breaking the aggre-
gate function into two or more parts.

4620 Target list for "retrieve unique" has more than 49 domains or is wider
than 1010 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 1010. This results from the decision that a tuple must fit on one UNIX "page". Assorted pointers require the 14 bytes which separates 1010 from 1024. 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 indices on them.

5304 INDEX: relation %0 is already an index
INGRES does not permit tertiary indices.

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 indices 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

```bash
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 inconsistent 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:

```bash
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.
Sort order may be supplied only when modifying to heapsort or cheap-
sort.
SAVE: cannot save system relation "%0"

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

SAVE: bad month "%0"
SAVE: bad day "%0"
SAVE: bad year "%0"

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

SAVE: relation %0 does not exist or is not owned by you
COPY: relation %0 doesn't exist
COPY: attribute %0 in relation %1 doesn't exist or it has been listed twice
COPY: too many attributes

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

COPY: bad length for attribute %0. Length="%1"
COPY: can't open file %0

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

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.

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.

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.).

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

COPY: bad type for attribute %0. Type="%1"
The only accepted types are i, f, c, and d.

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.

COPY: You are not allowed to update the relation %0
You cannot copy into a system relation or secondary index.
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 dumm-
my 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 trun-
cated.

5822 COPY: Relation %0 is protected.
You are not authorized to access this relation.