CLING/SQL – COMMON LISP TO INGRES/SQL INTERFACE

by

Dave Charness and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/40

8 May 1990

CLING/SQL – COMMON LISP TO INGRES/SQL INTERFACE

by

Dave Charness and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/40

8 May 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CLING/SQL – COMMON LISP TO INGRES/SQL INTERFACE

by

Dave Charness and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/40

8 May 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CLING/SQL - Common LISP to INGRES/SQL Interface[†]

Dave Charness
Lawrence A. Rowe

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

This paper describes CLING/SQL, a package that provides an interface between Common LISP and INGRES using SQL. A full set of SQL commands is supported, including queries, cursors, and dynamic SQL. INGRES databases may be accessed from Common LISP and results returned to the LISP environment.

1. Introduction

CLING/SQL (Common LISP INGres interface using SQL) is a package that provides an interface between Common LISP and the commercial INGRES database system sold by INGRES, Inc. (formerly known as Relational Technology, Inc.). This version of CLING works only with INGRES version 6 or later versions. Database commands can be executed from Common LISP programs. Common LISP functions are defined for every INGRES command. CLING/SQL uses the SQL query language. In addition, functions are provided to call several operating system level commands.

The remainder of this document is organized as follows. Section 2 shows an example that illustrates how to use CLING/SQL. Section 3 describes important list structures used to call CLING/SQL functions. Section 4 describes the Common LISP function calls that execute INGRES commands. The last section describes how to install the CLING/SQL package. This document assumes a knowledge of INGRES, SQL, and Common LISP.

Software is available from U.C. Berkeley via ftp. Email to cling-info@postgres.berkeley.edu for further information. Bug reports to cling-bugs@postgres.berkeley.edu.

1.1. About the Format for Definitions

In this document, optional items are placed between brackets (the '[' and ']'). Optional items may be omitted from a function call or list. For example, the specification for the *db-createdb* function is

```
(db-createdb database-name [:flags flags])
```

which specifies that the **:flags** argument may be omitted when the function is called.

Ellipses (...) indicate that one or more items may be used. They can be included with a bracketed expression, such as in the specification for the format list

```
(format-item1 [format-item2 ...])
```

When one of many items must be chosen, the items are placed within braces ('{' and '}') and separated by the 'I' character. For example, the specification for the *db-copy* function is

```
(db-copy table-name format-list {:into I :from} file-name)
```

which specifies that either **:into** or **:from** may be used to indicate that data is to be copied into or from the database.

[†] This research was supported by National Science Foundation Grant MIP-8715557.

2. An Example Session Using CLING/SQL

This section shows an example of a CLING/SQL session. The example demonstrates how to load the CLING/SQL package, create and open a database, create tables, append tuples to the tables, and retrieve those tuples. The Common LISP environment shown is based on Franz Allegro Common LISP.

The first step to using CLING/SQL is to load it.

```
<cl> (load "cling.cl")

T
```

Doing this will load the entire package. To use the package, the *use-package* function is executed:

```
<cl> (use-package 'cling)

T
```

The next step will be to create and open a database. A private database called "testdb" will be created. The flag "-p" (for private) ensures that access to the database is restricted.

```
<cl> (db-createdb 'testdb :flags "-p")

T
```

CLING/SQL may access any INGRES database including databases created by other users. Any INGRES database created by either a CLING/SQL program or an INGRES command (e.g. the shell command *createdb*) can be accessed by CLING/SQL. Before the newly created database can be accessed, the connection to the database must be opened, using the *db-connect* function:

```
<cl> (db-connect 'testdb)

T
```

Remember that INGRES version 6.x is a server DBMS. Consequently, the server must be running before the *db-connect* command is executed. If you have problems, contact your INGRES system administrator.

Now that the database has been created and opened, some tables can be created. Tables are created with the *db-create-table* command.

```
<cl> (db-create-table 'paytable '((name (char 10)) (hourly-pay float) (job (char 20))))

NIL
<cl> (db-create-table 'citytable '((name (char 10)) (city (char 20))))

NIL
```

The table "paytable" has a "name" column that is a 10 byte character string, an "hourly-pay" column that is a four-byte floating point number, and a "job" column that is a 20 byte character string. Table "citytable" has two columns. "Name" has 10 characters, and "city" has 20.

The two tables have been created, but they are empty. *db-insert* is used to add tuples to the tables.

```
<cl> (db-insert 'paytable '(name hourly-pay job) '("Jeff" 70 "Research Assistant"))

NIL

<cl> (db-insert 'paytable '(name hourly-pay job) '("Melvin" 3.50 "Lawyer"))

NIL

<cl> (db-insert 'paytable '(name hourly-pay job) '("Lois" 3.35 "Professor"))

NIL

<cl> (db-insert 'citytable '(name city) '("Jeff" "Berkeley"))

NIL

<cl> (db-insert 'citytable '(name city) '("Susan" "Manila"))

NIL

<cl> (db-insert 'citytable '(name city) '("Larry" "Berkeley"))

NIL
```

Tuples are retrieved using the *db-select* command. The following example returns a list of tuples:

```
<cl> (db-select '(p.name p.hourly-pay) :from '((paytable p) (paytable j))
        :where '(and (< p.hourly-pay j.hourly-pay) (= j.name "Jeff")))

(("Larry" 3.35) ("Melvin" 3.5))
```

The name and pay of all tuples whose hourly pay is less than Jeff's are returned.

Tuple manipulation functions are provided that make manipulating tuples easier. In the following example, a sorted list of the weekly pay of all people in paytable is returned.

```
<cl> (db-mapcar-select #'(lambda (x) (* 40.0 (car x))) '(hourly-pay) '(paytable) :order-by

(134.0 140.0 2800.0)
```

Tuples are returned as lists, so that *car* must be used in the map function.

To close the connection to the database, the *db-disconnect* function is used.

```
<cl> (db-disconnect)

T
```

Other databases may now be opened.

3. CLING/SQL List Structures

CLING/SQL functions take a number of differently structured lists as input. This section details these lists, showing their format, giving examples of their usage, and listing functions that use them. It should be noted that CLING/SQL allows table and column names greater than 24 characters, but will truncate them before sending them to the INGRES backend. This is done because the maximum length of INGRES names is 24 characters.

3.1. Format Lists

A format list is used to specify the data format of a table or a file. In the CLING/SQL definitions, it is referred to as a format-list.

List Format:

(format-item1 [format-item2 ...])

where *format-item* is a format element **(column-name format)**
column-name is the name of a column in a table, and
format is an INGRES format specification, as listed below:

format	type
(char n)	Character (n = 1-255)
integer1	1-byte integer
integer2	2-byte integer
integer4	4-byte integer
float4	4-byte floating
float8	8-byte floating
date	date(25 bytes)
money	8-byte floating

QUEL type names are also supported.

Example Usage:

((name (char 10)) (pay money) (job (char 20)))

INGRES Equivalent To Example:

(name char(10), pay money, job char(20))

CLING/SQL Functions Using this List:

db-copy
db-create-procedure
db-create-table

3.2. Qualification Lists

This list specifies a qualification that selects rows from a table. It is most often used as the argument following the keyword :where. In the CLING/SQL definitions, it is referred to as qual-list. Note in the example below that symbols are coerced to strings with all characters converted to upper-case (the default in LISP).

List Format:

(boolean-expression)

Example Usage:

(AND (< pay 150) (= job "CEO") (= city 'Berkeley))

INGRES Equivalent To Example:

pay < 150 and job = 'CEO' and city = 'BERKELEY'

CLING/SQL Functions Using this List:

db-create-integrity
db-delete
db-insert
db-select
db-update
db-mapc-select
db-mapcar-select
db-mapcan-select

3.3. Target Lists

A target list specifies tuples to be returned to the user by a query (e.g. *db-select*). When a target list is used in a select command, values other than strings must be preceded by a **:type** specifier. The default behavior is to compute type specifiers for each column in a query and to expand target lists which include the "all columns" ("*") specifier. The variable **cling::expand-target-lists** can be set to *nil* to disable these computations. In the CLING/SQL definitions, a target list is specified by target-list.

List Format:

(target-item1 [target-item2 ...])

where target-item can be expr or (**:type** expr)
expr can be
target-name
(arithmetic-op target-name [target-name])
(= result-column-name expression)
aggregate
target-name can be
column-name
table-name.column-name
correlation-name.column-name
column-name can be
the name of a column
*
:type can be
:string
:float
:integer

Example Usage:

(paytable.* (:float (= weeklypay (* paytable.hourly-pay 40))))

INGRES Equivalent To Example:

(paytable.*, weeklypay=paytable.hourly-pay*40)

CLING/SQL Functions Using this List:

db-select
db-map-select
db-mapcar-select
db-mapcan-select

3.4. Sort Lists

A sort list is used to specify the sort order in which tuples are returned. It will be specified in the definitions as sort-list.

List Format:
(sort-item1 [sort-item2 ...])

where sort-item is a column name or **(column-name order)**
order is a sort direction (**asc** for ascending and **desc** for descending)

Example Usage:
(job (name desc))

INGRES Equivalent To Example:
job, name desc

CLING/SQL Functions Using this List:
db-modify
db-select
db-map-select
db-mapcar-select
db-mapcan-select

3.5. Aggregates

An aggregate is a function that computes a result over a collection of data.* It may be placed anywhere in an expression where a constant can be used. The following are valid aggregates: *count, sum, avg, max, min.*

List Format:
(aggregate column-name)

Example Usage:
(avg paytable.hourly-pay)

INGRES Equivalent To Example:
avg(paytable.hourly-pay)

*These functions are known as *set functions* in SQL terminology. There are called *aggregates* in this paper to avoid confusion with the functions which parallel the embedded SQL set construct.

4. CLING/SQL Function Reference

CLING/SQL has four groups of functions. The first group implements Unix shell commands. The second group consists of CLING/SQL connection and disconnection operations. The third and largest group of functions map directly to SQL commands. The fourth group uses the SQL select command to manipulate tuples in ways useful to Common LISP. All CLING/SQL functions are prefixed by "db-" to distinguish them from other LISP functions. Except where otherwise specified, all functions return NIL.

4.1. System Level Calls

These three function calls correspond to operating system level INGRES commands. These commands cannot be used with INGRES/NET.

4.1.1. cling:db-createdb

This function creates a database. It takes the same flags as the standard INGRES command. The flags must be contained within a string.

Usage:

(db-createdb database-name [:flags flag-string])

Returns:

T - if successful
NIL - if unsuccessful (database already exists)

INGRES Equivalent:

createdb

Example Usage:

(db-createdb 'testdb :flags "-p -usedayao")

INGRES Equivalent To Example:

createdb -p -usedayao testdb

4.1.2. cling:db-dropdb

This function drops a database. Attempting to drop a database while it is open (after *db-connect* has been executed and before *db-disconnect* has been performed) will fail. Flags are the same as the standard INGRES command and must be passed as a string.

Usage:

(db-dropdb database-name [:flags flag-string])

Returns:

T - if successful
NIL - if unsuccessful (flags used improperly)

INGRES Equivalent:

dropdb

Example Usage:

(db-dropdb 'testdb :flags "-s")

INGRES Equivalent To Example:

dropdb -s testdb

4.1.3. cling:db-sysmod

This function optimizes the system catalogs of a database. Flags are the same as in the standard INGRES command and must be passed as a string.

Usage:

(db-sysmod database-name [:flags flag-string])

Returns:

T - if successful
NIL - if unsuccessful

INGRES Equivalent:

sysmod

Example Usage:

(db-sysmod 'testdb :flags "-w")

INGRES Equivalent To Example:

sysmod -w testdb

4.2. CLING/SQL Calls

These two functions connect to and disconnect from a database. The connect command opens a thread in the INGRES server for the program to access a specified database. Only one connection to a database may be open at a time.

4.2.1. cling:db-connect

This function connects to a database. It must be executed before any commands are attempted, otherwise, an error will be returned. *Db-connect* is very sensitive to errors, and a single error can kill the Common LISP process. Only one flag is permitted in the string. If more than one flag is specified or if a flag is specified incorrectly, the Common LISP process will die. When used with INGRES/NET, the database name must be surrounded by quotes. For example, when using database *testdb* on node *argon*, the call

(db-connect "argon::testdb")

connects to the remote database.

Usage:

(db-connect database-name [:flags flag-string])

Returns:

T

INGRES Equivalent:

sql

Example Usage:

(db-connect 'testdb :flags "-l")

INGRES Equivalent To Example:

sql -l testdb to the shell or **exec sql connect testdb options = '-l'** in an embedded SQL program

4.2.2. cling:db-disconnect

This function disconnects from a database. To use another database, the current database connection must be closed and a new one opened.

Usage:
(db-disconnect)

Returns:
T

INGRES Equivalent:
quit in the terminal monitor or **exec sql disconnect** in an embedded SQL program

4.3. SQL Calls

These functions map to SQL commands. A database must be opened before any of these functions can be called.

4.3.1. cling:db-close

This function closes an open cursor. Note that a *db-commit*, *db-rollback*, or *db-disconnect* implicitly closes all open cursors.

Usage:
(db-close cursor-name)

INGRES Equivalent:
close

Example Usage:
(db-close 'c1)

INGRES Equivalent To Example:
close c1

4.3.2. cling:db-commit

This function commits any changes made during the current transaction. The standard SQL transaction semantics is that the program is always in a transaction. One transaction can be ended and another transaction begun by executing *db-commit*.

Usage:
(db-commit)

INGRES Equivalent:
commit

4.3.3. cling:db-copy

This function copies a table to or from a file. Note that the filename must be a string. CLING/SQL converts a file name into its full path name for INGRES use if the file is not specified with a full path name.

Usage:

```
(db-copy table-name format-list {:into | :from} file-name
        [:on-error {t | nil}] [:error-count n]
        [:rollback {t | nil}] [:log filename])
```

INGRES Equivalent:

copy

Example Usage:

```
(db-copy 'citytable '((name c0tab) (city c0nl)) :into "cityfile")
```

INGRES Equivalent To Example:

```
copy table citytable(name=c0tab,city=c0nl)
        into "/b/users/sedayao/CLING.cbw/cityfile"
```

4.3.4. cling:db-create-index

This function creates a secondary index on an existing table. If the index name is a two element list, the first element is taken to be a database catalog, and the second is the index name.

Usage:

```
(db-create-index index-name table-name
        (column-name1 [{asc | desc}] [column-name2 [{asc | desc}]] ...)
        [:structure {[c]btree | isam | hash}]
        [:key (column-name1 ...)]
        [:fillfactor n] [:minpages n] [:maxpages n]
        [:leaffill n] [:nonleaffill n] [:maxindexfill n]
        [:location location-name])
```

INGRES Equivalent:

create index

Example Usage:

```
(db-create-index 'paytable 'index1 '(name job))
```

INGRES Equivalent To Example:

```
create index index1 on paytable(name, job)
```

4.3.5. cling:db-create-integrity

This function defines an integrity constraint. The qual-list must be a single-table, aggregate-free qualification.

Usage:

(db-create-integrity table-name qual-list)

INGRES Equivalent:

create integrity

Example Usage:

(db-create-integrity 'personneltable '(>= pay 0))

INGRES Equivalent To Example:

create integrity on personneltable is pay >= 0

4.3.6. cling:db-create-procedure

This function creates a named database procedure. Where argument values are to be substituted for parameters in the body functions, prepend a colon and surround it with vertical bars (e.g. "foo" is specified by |:foo|).

Usage:

(db-create-procedure name ([(param1 param1-type) ...]) body-function1 ...)

INGRES Equivalent:

create procedure

Example Usage:

```
(db-create-procedure 'transfer-emp
  '((empname (char 10) 'not-null) (jobname (char 20)))
  '(db-update paytable ((= job |:jobname|)) :where (= name |:empname|)))
```

INGRES Equivalent To Example:

```
create procedure transfer_emp
(empname char(10) not null, jobname char(20))
begin
  update paytable set job = :jobname where name = :empname;
end;
```

4.3.7. cling:db-create-table

This function creates a table.

Usage:

```
(db-create-table table-name
        {format-list | (column-name1 [column-name2 ...]) :as subselect}
        [:location location] [:journaling {t | nil}]
        [:duplicates {t | nil}])
```

INGRES Equivalent:

create

Example Usage:

```
(db-create-table 'paytable '((name (char 10)) (pay money) (job (char 20))))
```

INGRES Equivalent To Example:

```
create table paytable (name char(10), pay money, job char(20))
```

4.3.8. cling:db-create-view

This function defines a view.

Usage:

```
(db-create-view view-name target-list subselect)
```

Example Usage:

```
(db-create-view 'citypay '(name pay city)
        '(db-select (p.name p.pay c.city)
        :from ((paytable p) (citytable c))
        :where (= p.name c.name)))
```

INGRES Equivalent To Example:

```
create view citypay (name, pay, city) as
        select p.name, p.pay, c.city
        from paytable p, citytable c
        where (p.name=c.name)
```

4.3.9. cling:db-declare-cursor

This function declares a cursor.

Usage:

```
(db-declare-cursor cursor-name select-args
[{:deferred-update | :direct-update} (column1 ...)])
```

INGRES Equivalent:

```
declare cursor_name cursor
```

Example Usage:

```
(db-declare-cursor 'c1 '((name pay) emptab :where (> pay 10.00)))
```

INGRES Equivalent To Example:

```
declare c1 cursor for select name, pay from emptab where pay > 10.00
```

4.3.10. cling:db-delete

This function deletes rows from a table.

Usage:

(db-delete table-name [:where {(current-of cursor-name) | qual-list}])

INGRES Equivalent:

delete

Example Usage:

(db-delete 'paytable :where '(AND (< pay 150) (= job "CEO")))

INGRES Equivalent To Example:

delete from paytable where pay < 150 and job='CEO'

4.3.11. cling:db-describe

This function retrieves type information about a select statement prepared with *db-prepare*, placing the information in the default SQL Descriptor Area or in the supplied descriptor. **cling::sqlda** , the default descriptor area, and its type are defined in low.cl. For direct access to the structure use the C-structure access functions provided by your LISP implementation.

Usage:

(db-describe statement-name [:desc descriptor])

INGRES Equivalent:

describe

Example Usage:

(db-describe 'stmt :desc sqlda)

INGRES Equivalent To Example:

describe stmt into sqlda

4.3.12. cling:db-drop

This function drops tables, views, or indices. Recall that CLING/SQL truncates names to twenty-four characters.

Usage:

(db-drop name1 [name2 ...])

INGRES Equivalent:

drop

Example Usage:

(db-drop 'paytable 'citytable 'personnelview)

INGRES Equivalent To Example:

drop paytable, citytable, personnelview

4.3.13. cling:db-drop-integrity

This function drops integrity constraints. Integrities are specified as integers.

Usage:

(db-drop-integrity table-name {integrity1 | (integrity1 ...) | ALL)})

INGRES Equivalent:

drop integrity

Example Usage:

(db-drop-integrity 'paytable 'all)

INGRES Equivalent To Example:

drop integrity on paytable all

4.3.14. cling:db-drop-permit

This function drops permits. Permits are specified as integers. Specify **:procedure t** to drop a permit on a database procedure.

Usage:

(db-drop-permit table-name {permit1 | (permit1 ...) | ALL} [:procedure t])

INGRES Equivalent:

drop permit

Example Usage:

(db-drop-permit 'paytable '(2 3))

INGRES Equivalent To Example:

drop permit on paytable 2,3

Example Usage:

(db-drop-permit 'updatepaytable 'all :procedure t)

INGRES Equivalent To Example:

drop permit on procedure updatepaytable all

4.3.15. cling:db-drop-procedure

This function removes a procedure definition from the database.

Usage:

(db-drop-procedure procedure-name)

INGRES Equivalent:

drop procedure

Example Usage:

(db-drop-procedure 'raises)

INGRES Equivalent To Example:

drop procedure raises

4.3.16. cling:db-endquery

This function returns a value reflecting whether or not the previous *db-fetch* returned a valid row.

Usage:

(db-endquery)

INGRES Equivalent:

inquire_ingres x = endquery

Returns:

T - *db-fetch* issued after last row of the cursor.
NIL - *db-fetch* returned a valid row.

4.3.17. cling:db-errorno

This function returns the number of the last error encountered in the INGRES session. Note that this value may not have been the result of the last function call. INGRES does not reset this value on successful queries.

Usage:

(db-errorno)

INGRES Equivalent:

inquire_ingres x = errorno

Returns:

INGRES error number

4.3.18. cling:db-errortext

This function returns the text of the last error message issued by the INGRES session. Note that this value may not have been the result of the last function call. INGRES does not reset this value on successful queries.

Usage:

(db-errortext)

INGRES Equivalent:

inquire_ingres x = errortext

Returns:

INGRES error message

4.3.19. cling:db-execute

This function executes a statement prepared with *db-prepare*.

Usage:
(db-execute statement-name [:values (value1 ...)])

INGRES Equivalent:
execute

Example Usage:
(db-execute 'stmt :values '(1.0))

INGRES Equivalent To Example:
execute stmt using 1.0

4.3.20. cling:db-execute-procedure

This function invokes a database procedure. Take care that the LISP types of the argument values match the parameter declarations because implicit type conversions are not performed.

Usage:
(db-execute-procedure procedure-name [(arg-name1 arg-value1) ...])

INGRES Equivalent:
execute procedure

Example Usage:
(db-execute-procedure 'raises '(amount 1.0))

INGRES Equivalent To Example:
execute procedure raises (amount = 1.0)

4.3.21. cling:db-fetch

This function fetches one row from a database cursor.

Usage:
(db-fetch cursor-name)

Returns:
Single tuple.

INGRES Equivalent:
fetch

Example Usage:
(db-fetch 'cursor1)

INGRES Equivalent To Example:
fetch cursor1

4.3.22. cling:db-grant

This function adds database priveleges to a table, view, or procedure. Specify **:procedure t** to indicate a procedure is being named.

Usage:

(db-grant ([select] [insert] [delete] [update [(column-name1 ...)]] [execute])
(table1 ...) {(user1 ...) | public} [:procedure t])

INGRES Equivalent:

grant

Example Usage:

(db-grant '(select (update name pay)) 'paytable 'larry)

INGRES Equivalent To Example:

grant select, update(name, pay) on paytable to larry

4.3.23. cling:db-insert

This function inserts rows into a table.

Usage:

(db-insert table-name target-list {value-list | subselect})

INGRES Equivalent:

insert

Example Usage:

(db-insert 'citytable '(name city) '(paytable.name "Chicago")
:where '(= paytable.job "Doctor"))

INGRES Equivalent To Example:

insert to citytable(paytable.name, city='Chicago')
where paytable.job='Doctor'

4.3.24. cling:db-messagenumber

This function returns the last message number issued by a database procedure.

Usage:

(db-messagenumber)

INGRES Equivalent:

inquire_ingres x = messagenumber

Returns:

INGRES message number

4.3.25. cling:db-messagetext

This function returns the text of the last message issued by a database procedure.

Usage:

(db-messagetext)

INGRES Equivalent:

inquire_ingres x = messagetext

Returns:

INGRES message text

4.3.26. cling:db-modify

This function converts the storage structure of a table or index. Various other storage parameters may also be specified.

Usage:

```
(db-modify table structure
        [:on sort-list]
        [:unique unique]
        [:fillfactor fill-factor]
        [:minpages minpages]
        [:maxpages maxpages]
        [:indexfill indexfill]
        [:maxindexfill maxindexfill])
```

INGRES Equivalent:

modify

Example Usage:

```
(db-modify 'citytable 'isam
        :on '(name)
        :unique 'yes
        :fillfactor 80)
```

INGRES Equivalent To Example:

modify citytable to isam unique on name where fillfactor=80

4.3.27. cling:db-open

This function opens a cursor for processing. The cursor must have been declared using *db-declare-cursor*.

Usage:

(db-open cursor-name [:readonly t] [:values (value1 ...)])

INGRES Equivalent:

open

Example Usage:

(db-open 'c1 :readonly t)

INGRES Equivalent To Example:

open c1 for readonly

4.3.28. cling:db-prepare

This function prepares an SQL statement for execution.

Usage:

(db-prepare statement-name statement-string)

INGRES Equivalent:

prepare

Example Usage:

(db-prepare 'stmt "select * from emptab")

INGRES Equivalent To Example:

prepare stmt from 'select * from emptab'

4.3.29. cling:db-relocate

This function relocates a table to a new storage area.

Usage:

(db-relocate table-name location)

INGRES Equivalent:

relocate

Example Usage:

(db-relocate 'citytable 'othersite)

INGRES Equivalent To Example:

relocate citytable to othersite

4.3.30. cling:db-rollback

This function undoes the effects of the current multi-statement transaction.

Usage:

(db-rollback [:to savepoint])

INGRES Equivalent:

rollback

Example Usage:

(db-rollback :to 'label)

INGRES Equivalent To Example:

rollback to label

4.3.31. cling:db-rowcount

This function returns the number of rows affected by the previous INGRES operation.

Usage:
(db-rowcount)

INGRES Equivalent:
inquire_ingres x = rowcount

Returns:
number of rows affected

4.3.32. cling:db-save

This function saves a table until a specified date.

Usage:
(db-save table-name month day year)

INGRES Equivalent:
save

Example Usage:
(db-save 'citytable 8 28 1988)

INGRES Equivalent To Example:
save citytable until 8 28 1988

4.3.33. cling:db-savepoint

This function declares a savepoint marker within a multistatement transaction.

Usage:
(db-savepoint savepoint-name)

INGRES Equivalent:
savepoint

Example Usage:
(db-savepoint 'label)

INGRES Equivalent To Example:
savepoint label

4.3.34. cling:db-select

This function retrieves rows from a table.

Usage:

```
(db-select target-list
        :from (table [corr-name] [table [corr-name]] ...)
        [:distinct t]
        [:where qual-list]
        [:group-by (column1 ...)]
        [:having qual-list]
        [:union db-select-statement]
        [:order-by sort-list])
```

Returns:

List of tuples

INGRES Equivalent:

select

Example Usage:

```
(db-select '(name pay job) :from '((paytable p))
        :where '(> p.pay 50)
        :order-by '(job (name desc)))
```

INGRES Equivalent To Example:

```
select name, pay, job from paytable p
        where (paytable.pay > 50)
        order by job,name desc
```

4.3.35. cling:db-transaction

This function returns a value reflecting whether or not a transaction is open.

Usage:

```
(db-transaction)
```

INGRES Equivalent:

inquire_ingres x = transaction

Returns:

T	- a transaction is open.
NIL	- no transaction is open.

4.3.36. cling:db-update

This function updates values of columns in a table.

Usage:

(db-update table-name target-list [:where {qual-list | (current-of cursor-name)}])

INGRES Equivalent:

update

Example Usage:

(db-update 'citytable '((= city "New Delhi")) :where '(= name "Suresh"))

INGRES Equivalent To Example:

update citytable set (city='New Delhi') where (name='Suresh')

4.3.37. cling:db-whenever

This function specifies functions to be called when certain conditions become true. Pass the argument *nil* to remove a previous assignment.

Usage:

(db-whenever [:sqlwarning function]
[:sqlerror function]
[:not-found function]
[:sqlmessage function])

INGRES Equivalent:

whenever

Example Usage:

(db-whenever :sqlerror 'sqlprint)

INGRES Equivalent To Example:

whenever sqlerror call sqlprint

4.4. Tuple Manipulation Functions

These three functions are designed to make tuple manipulation easier. They are variations of the *mapc*, *mapcar*, and *mapcan* functions in Common LISP. No INGRES equivalents are given, but they correspond roughly to a select loop in embedded SQL.

4.4.1. cling:db-mapc-select

This function retrieves a list of tuples and applies the specified function to each element of the list. No results are returned. To halt the function, the map function should return NIL.

Usage:

```
(db-mapc-select function target-list
        :from (table [corr-name] [table [corr-name]] ...)
        [:distinct t]
        [:where qual-list]
        [:group-by (column1 ...)]
        [:having qual-list]
        [:union db-select-statement]
        [:order-by sort-list])
```

Example Usage:

```
(db-mapc-select #'(lambda (x) (print (first x)))
        '(name pay job) :from '(paytable)
        :order-by '(name))
```

4.4.2. cling:db-mapcar-select

This function retrieves a list of tuples and applies the specified function to each element of the list. The results are collected in a list and returned.

Usage:

```
(db-mapcar-select function target-list
        :from (table [corr-name] [table [corr-name]] ...)
        [:distinct t]
        [:where qual-list]
        [:group-by (column1 ...)]
        [:having qual-list]
        [:union db-select-statement]
        [:order-by sort-list])
```

Returns:

A list of the results of the function applied to each retrieved tuple.

Example Usage:

```
(db-mapcar-select #'(lambda (x) (if (> (second x) 0) (cons (first x) NIL)))
        '(name children) :from '(personnelvar))
```

4.4.3. cling:db-mapcan-select

This function retrieves a list of tuples and applies the given function to each of element of the list. The results are collected in a list. The return list is created with *nconc* rather than with *cons*.

Usage:

```
(db-mapcan-select function target-list
        :from (table [corr-name] [table [corr-name]] ...)
        [:distinct t]
        [:where qual-list]
        [:group-by (column1 ...)]
        [:having qual-list]
        [:union db-select-statement]
        [:order-by sort-list])
```

Returns:

A list of the results of the function applied to each retrieved tuple, nconc'ed together.

Example Usage:

```
(db-mapcan-select #'(lambda (x) (if (> (second x) 0) (cons (first x) NIL)))
        '(name children) :from '(personnelvar))
```

4.5. Set Functions

These functions change properties of an INGRES session. They map to the embedded SQL *set* function.

4.5.1. cling:db-set-autocommit

This function turns autocommit on or off.

Usage:

```
(db-set-autocommit {t | nil})
```

INGRES Equivalent:

set autocommit

Example Usage:

```
(db-set-autocommit t)
```

INGRES Equivalent To Example:

set autocommit on

4.5.2. cling:db-set-journaling

This function turns on journaling. When journaling is on, all tables created within a session are logged. Selected tables can be logged by specifying them in the *db-set-journaling* command. Tables also can be explicited logged using the journaling option of the *db-create-table* command.

Usage:

(db-set-journaling [:table table])

INGRES Equivalent:

set journaling

Example Usage:

(db-set-journaling :table 'citytable)

INGRES Equivalent To Example:

set journaling on citytable

4.5.3. cling:db-set-nojournaling

This function turns off journaling. This is also the default state of logging.

Usage:

(db-set-nojournaling [:table table])

INGRES Equivalent:

set nojournaling

Example Usage:

(db-set-nojournaling :table 'citytable)

INGRES Equivalent To Example:

set nojournaling

4.5.4. cling:db-set-result-structure

Usage:

(db-set-result-structure storage-structure)

This function sets the storage structure of tables created using the :as option of the *db-create-table* command.

INGRES Equivalent:

set result_structure

Example Usage:

(db-set-result-structure 'cheap)

INGRES Equivalent To Example:

set result_structure "cheap"

4.5.5. cling:db-set-lockmode

This function sets the various parameters involved with locking in an INGRES session.

Usage:

```
(db-set-lockmode [:on table-name]
        [:level level]
        [:readlock readlock]
        [:maxlock maxlock]
        [:timeout timeout])
```

INGRES Equivalent:

set lockmode session

Example Usage:

```
(db-set-lockmode :on 'citytable
        :level 'session
        :readlock 'system
        :maxlocks 10
        :timeout 0)
```

INGRES Equivalent To Example:

```
set lockmode on citytable where level=session, readlock=system, maxlocks=10,
        timeout=0
```

5. Implementation and Installation Notes

CLING/SQL is implemented by connecting Common LISP to an INGRES server. CLING/SQL functions are mapped to SQL commands, which are passed to the INGRES libq routines that are loaded into the Common LISP process. Libq routines connect to the INGRES server and communicate with it.

The libq routines are loaded into Common LISP, which calls C-language programs via a Foreign Function Interface (FFI). The FFI is an implementation-dependent feature of Common LISP that allows a program to use to non-LISP procedures.

CLING/SQL is composed of the following files:

main.cl	user-visible CLING/SQL functions
subs.cl	CLING/SQL utility routines
macros.cl	CLING/SQL macros
libq.c	The INGRES interface library
low.cl	libq foreign function interface

Low.cl contains implementation dependent code. This file must be modified when porting CLING/SQL to other Common LISP implementations. The current release runs with Allegro Common Lisp from Franz, Inc. For other LISP support send mail to cling-info@postgres.berkeley.edu.

To set up CLING/SQL, compile libq.c as follows

```
cc -c libq.c
```

and put the rtingres/bin directory in your path.

To use CLING/SQL, you must be a registered INGRES user. When in Common LISP, load the file cling.cl as follows:

```
(load "cling.cl")
```

Acknowledgements

This work is a continuation of the work started by Jim Larus for use with Franz Lisp and University INGRES. CLING/SQL is based on the CLING package developed by Jeff Sedayao and John Irwin for commercial INGRES.