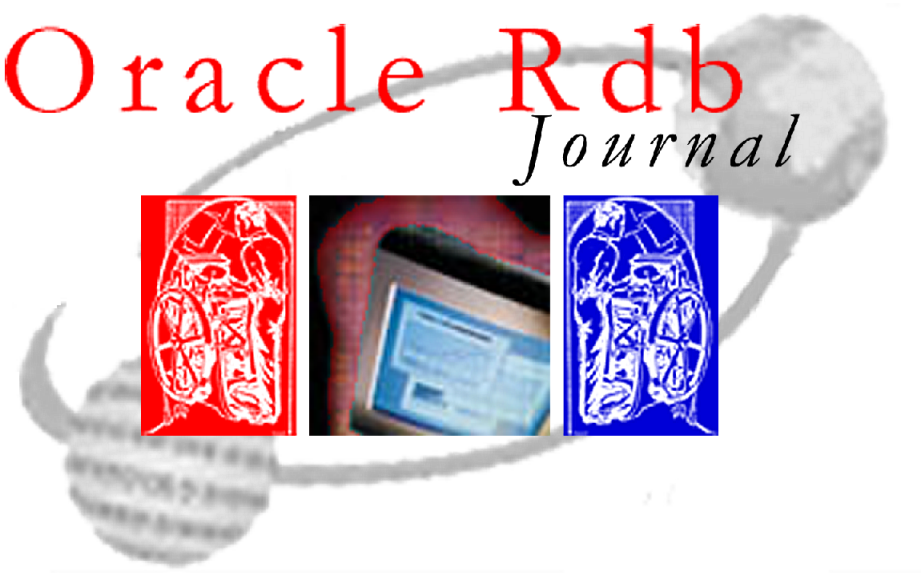


# CREATE MODULE Now Supports External Routines

**A Feature Article from the Rdb Journal**

**September 15, 2001**

# CREATE MODULE Now Supports External Routines

*...syntax for external functions and procedures is now permitted within a CREATE MODULE statement. Both SQL and external routine definitions can be mixed in a single module.*

In prior versions of Oracle Rdb the CREATE MODULE statement allowed only SQL stored functions and procedures to be defined.

With this release of Rdb7 syntax for external functions and procedures is now permitted within a CREATE MODULE statement. Both SQL and external routine definitions can be mixed in a single module.

Please refer to the Oracle Rdb7 SQL Reference Manual for information on the CREATE MODULE syntax, and the <routine-clause> of CREATE FUNCTION and CREATE PROCEDURE which is described in the CREATE ROUTINE section. The Oracle Rdb Release Notes for V7.0.6 include revised syntax diagrams for these changes.

This change provides the following benefits:

- Both stored and external routines can be collected within a single module. This allows simplified DROP, GRANT and REVOKE commands which will operate on multiple external routines in a single statement. For instance, a DROP MODULE can be used to remove external and stored routines for an application in a single command. GRANT and REVOKE can be applied to a larger set of routine.

- Related routines, whether external or stored, can be grouped together thus providing simplified database maintenance.

- External routines within the same module now share the same database environment. This allows, for instance, one external routine to OPEN a cursor, another to FETCH rows and another to CLOSE the cursor.

  In contrast when an external routine is created using the CREATE FUNCTION or CREATE PROCEDURE syntax only that routine uses the database environment.

## Usage Notes

- The name of the stored module need not be the same as that used for the SQL module language module, or SQL pre-compiled module which implements the external functionality. However, keeping identical or similar names may assist future maintenance of the application.

- The shared module database environment is only significant for those externalroutines which execute SQL statements.

- If an external routine attaches to a database, it will be implicitly disconnected when the invoking session is terminated.

  However, Oracle recommends that the current transaction, open cursors and session started for the external function be terminated before using DISCONNECT. This can be done explicitly by calling an external routine which terminates the transaction and disconnects in the same context as the invoking routine, or it can be done implicitly when using a NOTIFY routine.

## Example

### Using External Cursors

This example uses multiple external routines to manage a table cursor in the external routine database environment. This management includes the OPEN, FETCH and CLOSE of a single cursor.

Several domains are defined so that parameter data types can be consistently defined in the database that contain the application and also the database upon which the cursor is open.

```
create domain SQLSTATE_T char(5);
create domain STATUS_CODE char(1);
create domain STATUS_NAME char(8);
create domain STATUS_TYPE char(14);
```

*The article originally appeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

The external function interface is contained within a single CREATE MODULE statement. This module also contains the application in a single stored SQL procedure.

```
create module EX
language SQL

-- These procedure define the interface to
-- the external routines that implement the
-- transaction and cursor operations
--
procedure EX_START_READ_TXN
   (inout :ss sqlstate_t);
   external location 'TEST$SCRATCH:EX.EXE'
   language general
   general parameter style
   comment is 'start a READ ONLY transaction';

procedure EX_COMMIT
   (inout :ss sqlstate_t);
   external location 'TEST$SCRATCH:EX.EXE'
   language general
   general parameter style;

procedure EX_OPEN_CURSOR
   (inout :ss sqlstate_t);
   external location 'TEST$SCRATCH:EX.EXE'
   language general
   general parameter style
   comment is 'find all rows in WORK_STATUS order by STATUS_CODE';

procedure EX_CLOSE_CURSOR
   (inout :ss sqlstate_t);
   external location 'TEST$SCRATCH:EX.EXE'
   language general
   general parameter style;
   procedure EX_FETCH_CURSOR
   (inout :ss sqlstate_t,
    out :s_code STATUS_CODE,
    out :s_code_ind integer,
    out :s_name STATUS_NAME,
    out :s_name_ind integer,
    out :s_type STATUS_TYPE,
    out :s_type_ind integer);
   external location 'TEST$SCRATCH:EX.EXE'
   language general
   general parameter style;
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

```
-- This SQL procedures implements a simple
-- application
--
procedure WORK_STATUS
   comment is
      'Use an external cursor to fetch '
   /  'all rows in the '
   /  'WORK_STATUS table';
   begin
   declare :s_code STATUS_CODE;
   declare :s_name STATUS_NAME;
   declare :s_type STATUS_TYPE;
   declare :s_code_ind, :s_name_ind, :s_type_ind integer;
   declare :ss sqlstate_t;

   -- start a read-only transaction on the PERSONNEL database
   call EX_START_READ_TXN (:ss);
   if :ss ^= '00000' then
       SIGNAL :ss;
   end if;

   -- open the cursor on the work-status table
   call EX_OPEN_CURSOR (:ss);
   if :ss ^= '00000' then
       SIGNAL :ss;
   end if;

   -- now loop and fetch all the rows
   FETCH_LOOP:
   loop
       call EX_FETCH_CURSOR (:ss, :s_code, :s_code_ind,
           :s_name, :s_name_ind, :s_type, :s_type_ind);
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

```
        case :ss
            when '02000' then
                -- no more rows to fetch
                leave FETCH_LOOP;
            when '00000' then
                begin
                -- we have successfully fetched a row, so display it
                trace 'Status Code: ', case
                        when :s_code_ind < 0
                        then 'NULL'
                        else :s_code
                    end;
                trace 'Status Name: ', case
                        when :s_name_ind < 0
                        then 'NULL'
                        else :s_name
                    end;
                trace 'Status Type: ', case
                        when :s_type_ind < 0
                        then 'NULL'
                        else :s_type
                     end;
                trace '***';
                end;
            else
                -- signal will implicitly leave the stored procedure
                SIGNAL :ss;
        end case;
    end loop;

    -- close the cursor
    call EX_CLOSE_CURSOR (:ss);
    if :ss ^= '00000' then
        SIGNAL :ss;
    end if;

    -- commit the transaction
    call EX_COMMIT (:ss);
    if :ss ^= '00000' then
        SIGNAL :ss;
    end if;
    end;
end module;
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

The external procedures for this this example are written using the SQL module language. However, any language with embedded SQL, such as C, could have been used.

```
module EX
language GENERAL
parameter colons
-- EX: Sample application
-- Process the WORK_STATUS table using a table cursor
--
declare alias filename 'PERSONNEL'

declare c cursor for
    select status_code,
           status_name,
           status_type
    from WORK_STATUS
    order by status_code

procedure EX_START_READ_TXN
    (sqlstate);
    begin
    -- abort any stray transactions
    rollback;
    -- start a READ ONLY transaction
    set transaction read only;
    end;

procedure EX_COMMIT
    (sqlstate);
    commit work;

procedure EX_ROLLBACK
    (sqlstate);
    rollback work;

procedure EX_OPEN_CURSOR
    (sqlstate);
    open c;

procedure EX_CLOSE_CURSOR
    (sqlstate);
    close c;

procedure EX_FETCH_CURSOR
    (sqlstate,
    :s_code            STATUS_CODE,
    :s_code_ind        integer,
    :s_name            STATUS_NAME,
    :s_name_ind        integer,
    :s_type            STATUS_TYPE,
    :s_type_ind        integer);
    fetch c
    into :s_code indicator :s_code_ind,
         :s_name indicator :s_name_ind,
         :s_type indicator :s_type_ind;

procedure EX_DISCONNECT
    (sqlstate);
    disconnect default;
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

When run the application calls the external procedures to open the cursor and fetch the rows and display them using the TRACE statement.

```
SQL> set flags 'trace';
SQL>
SQL> call WORK_STATUS ();
~Xt: Status Code: 0
~Xt: Status Name: INACTIVE
~Xt: Status Type: RECORD EXPIRED
~Xt: ***
~Xt: Status Code: 1
~Xt: Status Name: ACTIVE
~Xt: Status Type: FULL TIME
~Xt: ***
~Xt: Status Code: 2
~Xt: Status Name: ACTIVE
~Xt: Status Type: PART TIME
~Xt: ***
SQL>
```

Oracle recommends that the cursors be closed, and the external routines database environment be disconnected before the calling session is disconnected. This can be achieved by using NOTIFY routines.

For example, the external procedure that starts the transaction could be modified as shown below to declare a NOTIFY routine (EX_RUNDOWN) that when called would close the cursors, rollback the transaction and disconnect from the database.

```
procedure EX_START_READ_TXN
    (inout :ss sqlstate_t);
    external location 'TEST$SCRATCH:EX.EXE'
    language general
    general parameter style
    notify EX_RUNDOWN on BIND
    comment is 'start a READ ONLY transaction';
```

The BIND notification ensures that EX_RUNDOWN will be called during the DISCONNECT of the caller and allow the transaction to be rolled back and the session disconnected. ROLLBACK or COMMIT will implicitly close any open cursors, unless the cursor were defined as WITH HOLD. In this case it is important to also close that cursor. Code similar to the following (in C) could implement this rundown routine.

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

```
#include <string.h>
#include <stdio.h>
#define RDB$K_RTX_NOTIFY_ACTV_END 2
#define SQLSTATE_LEN 5
void sql_signal ();
void EX_CLOSE_CURSOR (char sqlstate [SQLSTATE_LEN]);
void EX_DISCONNECT (char sqlstate [SQLSTATE_LEN]);
void EX_ROLLBACK (char sqlstate [SQLSTATE_LEN]);

extern void EX_RUNDOWN
    (int *func_code,
     int *u1,    /* U1, U2, U3 are currently unused */
     int *u2,    /* and are reserved for future use */
     int *u3)
{
char sqlstate [SQLSTATE_LEN];

if (*func_code == RDB$K_RTX_NOTIFY_ACTV_END)
  {
    /* we are running down this external routine, so close the cursor */
    EX_CLOSE_CURSOR (sqlstate);
    if (memcmp ("00000",
                sqlstate,
                SQLSTATE_LEN) != 0
        && memcmp ("24000",
               sqlstate,
               SQLSTATE_LEN) != 0)
        /* we expect success or maybe 24000 (bad cursor state) */
        sql_signal ();

     /* rollback the transaction */
     EX_ROLLBACK (sqlstate);
     if (memcmp ("00000",
                 sqlstate,
                 SQLSTATE_LEN) != 0
         && memcmp ("25000",
                    sqlstate,
                    SQLSTATE_LEN) != 0)
         /* we expect success or maybe 25000 (bad transaction state) */
         sql_signal ();

     /* disconnect from the database */
     EX_DISCONNECT (sqlstate);
     if (memcmp ("00000",
                 sqlstate,
                 SQLSTATE_LEN) != 0)
         /* we expect success */
         sql_signal ();
     }
}
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*

The application can be compiled and built using this fragment of DCL code:

```
$       if f$getsyi("arch_name") .eqs. "VAX"
$       then
$       create ex.opt
universal = EX_START_READ_TXN
universal = EX_COMMIT
universal = EX_ROLLBACK
universal = EX_OPEN_CURSOR
universal = EX_CLOSE_CURSOR
universal = EX_FETCH_CURSOR
universal = EX_DISCONNECT
universal = EX_RUNDOWN
psect_attr = RDB$MESSAGE_VECTOR,noshr
psect_attr = RDB$DBHANDLE,noshr
psect_attr = RDB$TRANSACTION_HANDLE,noshr
sql$user/library
$       else
$       create ex.opt
symbol_vector = (EX_START_READ_TXN = procedure)
symbol_vector = (EX_COMMIT = procedure)
symbol_vector = (EX_ROLLBACK = procedure)
symbol_vector = (EX_OPEN_CURSOR = procedure)
symbol_vector = (EX_CLOSE_CURSOR = procedure)
symbol_vector = (EX_FETCH_CURSOR = procedure)
symbol_vector = (EX_DISCONNECT = procedure)
symbol_vector = (EX_RUNDOWN = procedure)
psect_attr = RDB$MESSAGE_VECTOR,noshr
psect_attr = RDB$DBHANDLE,noshr
psect_attr = RDB$TRANSACTION_HANDLE,noshr
sql$user/library
$       endif
$
$       cc EX_RUNDOWN
$       sql$mod EX
```

*This article originally apeared in the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal)*