

ORACLE®



ORACLE®

External Routines in Rdb

Ian Smith
Oracle Rdb Product Architect





Overview

- Terminology and History
- External Routine Architecture
- External Routine Definition
- Using System Sharable Images
- Using Modules
- Using NOTIFY routines
- Final Comments



Terminology

- Routine - any named executable code
- Function - routine that is passed IN parameters and returns a result via the routine name. Acts as a value expression
- Procedure - routine activated by a CALL statement that allows IN, OUT and INOUT parameters



Passing Modes

- IN - read only parameter, may be literal values
- OUT - write only result, must be host variable, procedures parameter or local/global variable
- INOUT - modified parameters - must also be a variable or parameter.



External Routine History

- V6.0
External functions introduced along with stored procedures (written in SQL)
- V7.0
External procedures introduced
- V7.0.6 & V7.1
Modules included external routines
- V7.1
Alter function and alter procedure
(later enhanced in V7.1.2)

External Routine Definition

Architecture





Protecting the Data

- OpenVMS provides 4 levels of memory protection
 - KERNEL mode (most protected)
 - EXECUTIVE mode
 - SUPERVISOR mode
 - USER mode (least protected)
- For example, most of RMS, DBMS and Rdb run in EXECUTIVE mode
- DCL uses SUPERVISOR mode
- Most applications run in USER mode



Protected Memory

- Memory at each level can be protected
- For instance data structures created by OpenVMS can be protected from an erroneous or malicious application so they cannot interfere with any other user



EXEC Mode

- Running part of Oracle Rdb at an elevated mode allows primarily memory protection of database data and structures
- SQL runtime, Rdb/Dispatch run in USER mode
- Rdb server runs in EXEC mode
- SQL queries may consist of several calls through the USER/EXEC layer
- CLIENT/SERVER layers within a process



Drop down to User Mode

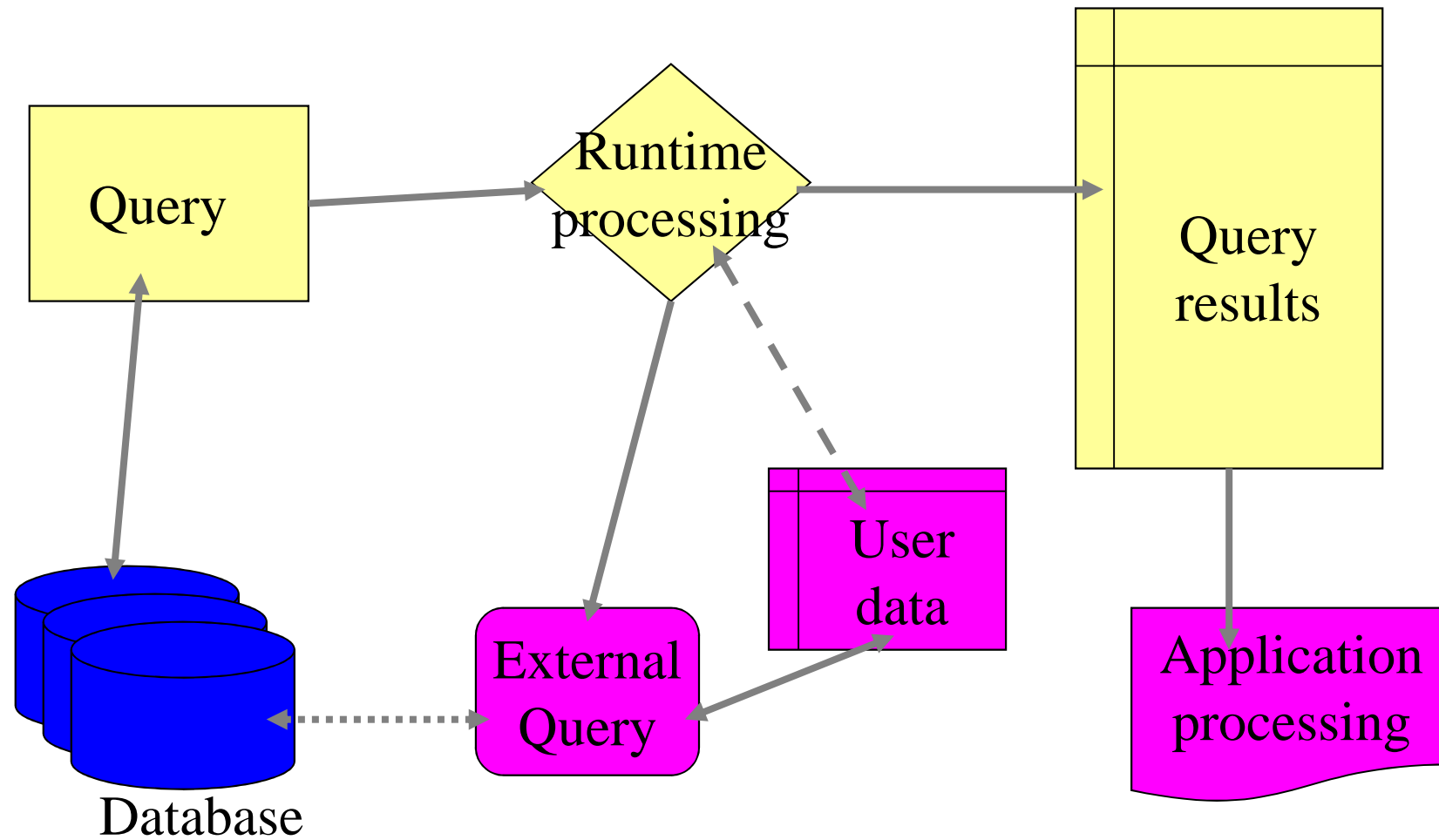
- Rdb reads database pages into EXEC mode memory
- Your external routine may not access that memory (as you might not be authorized to read some of the data on the page)
- So Rdb executes external routines in USER mode
- Passes copies of parameters via a piece of USER mode virtual memory



Image Activation

- Rdb uses LIB\$FIND_IMAGE_SYMBOL to activate the named routine in a known sharable image
- It is possible to test external routines yourself prior to using as external functions by writing a simple application that calls this runtime library routine.

Data Flow





Accessing a database

- The user mode routine may access Rdb databases
- The current context is suspended awaiting the results of the external routine call
- A second attach and second transaction must be used to execute queries from an external routine
- Any commit/rollback will not affect outer session
- No 2PC between inner and outer session (it is suspended)

External Routine Definition

Interface model for SQL





Interface

- SQL needs information about the routine so that it can activate it and pass data
- Will use simple examples to highlight the interface definition



Creating the Interface

- External routine interfaces can be described by
 - CREATE FUNCTION
 - CREATE PROCEDURE
 - FUNCTION and PROCEDURE clauses of CREATE MODULE
 - ADD clauses of ALTER MODULE
- Various clauses can be changed using ALTER

```
SQL> alter function SAMPLE  
cont> location '....';
```



LP_LINES

```
create function LP_LINES ( )  
returns integer;  
  external  
    name LIB$LP_LINES  
    location 'SYS$SHARE:LIBRTL.EXE'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns lines per page';
```

LP_LINES

```
create function LP_LINES ()  
returns integer;  
external  
  name LIB$LP_LINES  
  location 'SYS$SHARE:LIB$LP_LINES.dll'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns line number of the first line of the input text'
```

Name used by SQL
queries

LP_LINES

```
create function LP_LINES (  
returns integer;  
  external  
    name LIB$LP_LINES  
    location 'SYS$SHARE:LIB$LP_LINES.dll'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns line number of the first line of the file'
```

Parameter list;
empty in this case

LP_LINES

```
create function LP_LINES ()  
returns integer;  
  external  
    name LIB$LP_LINES  
    location 'SYS$SHARED:LIB$LP_LINES'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns line number'
```

Returned data type;
may be a domain name

LP_LINES

```
create function LP_LINES ()  
returns integer;  
external  
  name LIB$LP_LINES  
  location 'SYS$ORACLE:LIB$LP_LINES'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns line number'
```

Defines this
routine as being
external

LP_LINES

```
create function LP_LINES ()  
returns integer;  
external  
  name LIB$LP_LINES  
  location 'SYS$SHARE:LIB$LP_LINES.dll'  
  language GENERAL  
  parameter style GENERAL  
  deterministic  
  comment is 'Returns line number of the first line of the file'
```

Name used by
external code; may
be different

LP_LINES

```
create function LP_LINES
returns integer;
external
  name LIB$LP_LINES
  location 'SYS$SHARE:LIBRTL.EXE'
  language GENERAL
  parameter style GENERAL
  deterministic
  comment is 'Returns lines per page';
```

Sharable image
for the routine

LP_LINES

```
create function LP_LINES
returns integer;
external
  name LIB$LP_LINES
  location 'SYS$SHARE\LIBRTL.EXE'
  language GENERAL
  parameter style GENERAL
  deterministic
  comment is 'Returns lines per page';
```

Required syntax; allows
Rdb to adapt
to language conventions

LP_LINES

```
create function LP_LINES
returns integer;
external
  name LIB$LP_LINES
  location 'SYS$SHARE:LIBRTL.EXE'
  language GENERAL
  parameter style GENERAL
  deterministic
  comment is 'Returns lines per page';
```

Required; VMS
calling standard

LP_LINES

```
create function LP_LINES
returns integer;
external
  name LIB$LP_LINES
  location 'SYS$SHARE\LIB$LP_LINES.BRTL.EXE'
  language GENERAL
  parameter style GENERAL
  deterministic
  comment is 'Returns lines per page';
```

Optimizer hint;
Given the same input
the same result returned

LP_LINES

```
create function LP_LINES
returns integer;
external
  name LIB$LP_LINES
  location 'SYS$SHARE:LP_LINES.LIB$LP_LINES.EXE'
  language GENERAL
  parameter style GENERAL
  deterministic
  comment is 'Returns lines per page';
```

Multiline comment;
will be displayed by
SHOW command



Use in a query

```
SQL> select lp_lines() from rdb$database;
```

```
        66
```

```
1 row selected
```

- User defined functions always require parameter list - Even when empty
- Contrast to built-in functions such as CURRENT_DATE, SYSDATE, USER, etc

External Routine Definition

Using OpenVMS Sharable Images





Using VMS RTL

- Most of the examples used in this tutorial use OpenVMS libraries
- How do you find the entry point name?
- How do you find the location of the sharable image?



Locating routines

- OpenVMS HELP is the first source
- HELP *routines
- For example here is the HELP entry for OTS\$POWJJ



HELP RTL LIB\$ OTS\$POWJJ

RTL_Routines

OTS\$

OTS\$POWJJ

The Raise a Longword Base to a Longword Exponent routine raises a signed longword base to a signed longword exponent.

Format

OTS\$POWJJ longword-integer-base ,longword-integer-exponent

Additional information available:

Returns Arguments

HELP RTL LIB\$ OTS\$POWJJ

RTL_ROUTINES

OTS\$

OTS\$POWJJ

Returns

OpenVMS usage:longword_signed

type: longword (signed)

access: write only

mechanism: by value

Result of raising a signed longword base to a signed longword exponent.

Integer
Returns
By Value

HELP RTL LIB\$ OTS\$POWJJ Arguments

longword-integer-base

OpenVMS usage:longword_signed
type: longword (signed)
access: read only
mechanism: by value

Integer
IN
By Value

Base. The longword-integer-base argument is a signed longword containing the base.

HELP RTL LIB\$ OTS\$POWJJ Arguments

longword-integer-exponent

OpenVMS usage:longword_signed

type: longword (signed)

access: read only

mechanism: by value

Integer
IN
By Value



OTS\$POWJJ

```
SQL> create function POWER
cont>      (in :longword_integer_base
cont>      integer by value
cont>      ,in :longword_integer_exponent
cont>      integer default 2 by value
cont>      )
cont>      returns integer;
cont> external
cont>      name OTS$POWJJ
cont>      location 'SYS$SHARE:DPML$SHR.EXE'
cont>      language GENERAL
cont>      parameter style GENERAL
cont>      returns null on null input;
```

OTS\$POWJJ - Access

```
create function POWER
  (in :longword_integer_base
    integer by value
  ,in :longword_integer_exponent
    integer default 2 by value
  )
  returns integer;
external
  name OTS$POWJJ
  location 'SYS$SHARE:DP
  language GENERAL
  parameter style GENERAL
  returns null on null input;
```

Access: read-only
means IN
Required to be IN for
functions

OTS\$POWJJ – Names for parameters

```
create function POWER
  (in :longword_integer_base
    integer by value
  ,in :longword_integer_exponent
    integer default 1
  )
  returns integer;
external
  name OTS$POWJJ
  location 'SYS$SHARE:DPML$SHR.EXE'
  language GENERAL
  parameter style GENERAL
  returns null on null input;
```

Name is optional.
Suggest same name
as HELP.

OTS\$POWJJ - Default

```
create function POWER
  (in :longword_integer_base
    integer by value
  ,in :longword_integer_exponent
    integer default 2 by value
  )
  returns integer;
external
  name OTS$POWJJ
  location 'OTS$POWJJ.SHR.EXE'
  language C
  parameter mode is in;
returns null on null input;
```

Use default to allow truncated parameter lists.

OTS\$POWJJ – Passing mechanism

```
create function POWER
  (in :longword_integer_base
    integer by value
  ,in :longword_integer_exponent
    integer default 2 by value
  )
  returns integer;
external
  name OTS$POWJJ
  location 'SYS$SHARE:DPM
  language GENERAL
  parameter style GENERAL
  returns null on null input;
```

Mechanism describes
how the data is passed
Also: REFERENCE,
DESCRIPTOR

OTS\$POWJJ – How to handle NULL arguments

```
create function POWER
  (in :longword_integer_base
   integer by value
  ,in :longword_integer_default
   integer default
  )
  returns integer;
external
  name OTS$POWJJ
  location 'SYS$SHARED$PML$SHR.EXE'
  language GENERATE
  parameter style GENERAL
  returns null on null input;
```

What to do when
NULL is passed.
Default is error!



Usage

- Once define in the database, the function can be called from almost any location.
- DEFAULT for a column, module or routine
- COMPUTED BY or AUTOMATIC columns
- Other expressions

```
SQL> select POWER (100, 2)
cont> from rdb$database;

      10000
1 row selected
```

External Routine Definition

Using Modules





CREATE MODULE

- CREATE MODULE allows SQL and external routines to be defined as a family
- Allows single GRANT, REVOKE and DROP
- Allows restricted access to interface
- The only way to have several routines share a database attach



LIB\$FORMAT_DATE_TIME

- Consider the procedure LIB\$FORMAT_DATE_TIME
- Returns multiple results as parameters
- We want a value expression...
- Create a module with external procedure and jacket SQL function to process the call



LIB\$FORMAT_DATE_TIME

```
create module T_FUNCTIONS
  declare :dt_context integer = 0
  declare :LIB$M_TIME_FIELDS
    constant integer = 1
  declare :LIB$M_DATE_FIELDS
    constant integer = 2
  procedure LIB$FORMAT_DATE_TIME
    ...
    usage is local;
  function FORMAT_DATE_TIME
    (in :ts timestamp(2))
    return varchar(40);
    ...
end module;
```

LIB\$FORMAT_DATE_TIME

```
create module T_FUNCTIONS
  declare :dt_context integer = 0
  declare :LIB$M_TIME_FIELDS
    constant integer = 1
  declare :LIB$M_DATE_FIELDS
    constant integer = 2
  procedure LIB$FORMAT_DATE_TIME
    ...
    usage is local
  function FORMAT_DATE_TIME
    (in :ts
    return varchar2
    ...
end module;
```

Need global context;
Set by first call to
Date/time format
routine

LIB\$FORMAT_DATE_TIME

```
create module T_FUNCTIONS
  declare :dt_context integer = 0
  declare :LIB$M_TIME_FIELDS
    constant integer = 1
  declare :LIB$M_DATE_FIELDS
    constant integer = 2
  procedure LIB$FORMAT_DATE_TIME
    ...
    usage is local,
  function FORMAT_P
    (in :ts t
    return varchar
    ...
end module;
```

Global constants to
Control formatting

LIB\$FORMAT_DATE_TIME

```
create module T_FORMAT_DATE_TIME
  declare :dt_
  declare :LIB
  constant
  declare :LIB$M_FIELDS
  constant integer = 2
  procedure LIB$FORMAT_DATE_TIME
  ...
  usage is local;
  function FORMAT_DATE_TIME
    (in :ts timestamp(2))
    return varchar(40);
  ...
end module;
```

External procedure.
LOCAL restricts calls to
within module

LIB\$FORMAT_DATE_TIME

```
create module T_FUNCTIONS
  declare :dt context integer = 0
  procedure LIB$FORMAT_DATE_TIME
    ...
    usage is local
  function FORMAT_DATE_TIME
    (in :ts timestamp(2))
    return varchar(40);
  ...
end module;
```

Jacket routine to process
The call to the external
routine



FORMAT_DATE_TIME

- Jacket routine may be simple
- Provides local and global variables for output results
- Use module global variable for persistent results (such as context value)
- Test various returned results and format result value

FORMAT_DATE_TIME

```
function FORMAT_DATE_TIME
    (in :ts timestamp(2))
return varchar(40);
begin
declare :res_len integer = 0;
declare :res_str varchar(40);
call LIB$FORMAT_DATE_TIME
    (:res_str, :ts, :dt_context, :res_len,
    :LIB$M_DATE_FIELDS
    +:LIB$M_TIME_FIELDS);
trace :res_len, length (:res_str),
    '<' || :res_str || '>';
return trim(trailing from :res_str);
end;
```



Further examples

- See example using OPEN, CLOSE and FETCH cursor in the Rdb Technical Journal article: *Create Module now supports External Routines*
- When database context is shared between multiple external routine you must use a single module to describe all the external routines

External Routine Definition

Using NOTIFY Routines





NOTIFY

- Called at requested times: BIND, CONNECT, TRANSACTION
- Passes action site value
- Used to attach/detach to secondary databases, open/close files, etc.
- No need for a PROCEDURE definition, as this routine can not be called from SQL
- Can have multiple NOTIFY routines in the sharable image, taking different actions for different routines



An Example

```
procedure EX3_START_READ_TXN
  (inout :ss sqlstate_t);
external location 'APP_DIR:EX3.EXE'
language general
general parameter style
notify EX3_RUNDOWN on BIND
comment is 'start a READ ONLY transaction';
```

An Example

```
procedure EX3_START_READ_TXN
  (inout :ss sqlstate_t);
external location 'APP_DIR:EX3.EXE'
language general
parameter style general
notify EX3_RUNDOWN on BIND
comment is 'start a READ ONLY transaction';
```

When first called this routine
can attach to the database.
NOTIFY will ensure
disconnect on unbind.

The NOTIFY routine (in C)

```
extern void EX3_RUNDOWN
    (int *func_code,
     int *u1,          // U1, U2, U3 are currently unused
     int *u2,          // and are reserved for future use
     int *u3)
{
    int sqlcode = 0;

    if (*func_code == RDB$K_RTX_NOTIFY_ACTV_END)
    {
        // we are running down this external routine, so
        // disconnect from the database
        EX3_DISCONNECT (&sqlcode);
        // Raise an exception if we had a problem
        if (sqlcode != 0) sql_signal ();
    }
}
```



Notes on NOTIFY routine

- The literal value `RDB$K_RTX_NOTIFY_ACTV_END` is defined in `SYS$LIBRARY:DSRI.H`
 - Shipped by recent Rdb versions
- This example uses a SQL Module Language routine to DISCONNECT from the database
- Use SQL Module Language /PROTOTYPE qualifier to generate extern definitions for the routines called by the notify procedure



Building the application

- When using SQL you must link the sharable image with SQL\$USER.OLB
- Hence, also inherit SQL_SIGNAL and SQL_GET_ERROR_TEXT routines. May also add external procedures for them.
- The following example uses sql_get_error_text



Interface for sql_get_error_text

```
procedure EX3_GET_ERROR_TEXT
  (out :errbuf char(100),
   in :buflen integer by value,
   out :reslen smallint);
external name SQL_GET_ERROR_TEXT
      location 'APP_DIR:EX3.EXE'
language general
parameter style general
comment is 'translate the current SQLCODE';
```



Abort - jacket routine

```
procedure EX3_ABORT_WITH_ERROR
  (in :ss sqlstate_t);
begin
  declare :cl rdb$object_name;
  declare :eb char(100);
  declare :rl smallint;

  get diagnostics :cl = CALLING_ROUTINE;
  call EX3_GET_ERROR_TEXT (:eb, CHAR_LENGTH (:eb), :rl);
  -- signal the callers name and the error text
  SIGNAL :ss (:cl || SUBSTRING (:eb from 1 for :rl));
end;
```



Link Options file

- Make sure that the LINK option exposes the entry point for SQL_GET_ERROR_TEXT as a procedure

```
symbol_vector =(EX3_RUNDOWN =procedure)
symbol_vector =(EX3_START_READ_TXN =procedure)
symbol_vector =(EX3_ROLLBACK =procedure)
symbol_vector =(EX3_SELECT_LN =procedure)
symbol_vector =(SQL_GET_ERROR_TEXT =procedure)
psect_attr = RDB$MESSAGE_VECTOR,noshr
psect_attr = RDB$DBHANDLE,noshr
psect_attr = RDB$TRANSACTION_HANDLE,noshr
sys$share:sql$user/library
```


External Routine Definition

Frequently Asked Questions





Using SQL Module Language

- Yes. External routines can use SQL Module language routines directly
- No. Can not use SQLCA, SQLDA or record parameters
- No. NOTIFY routine can not be a SQL procedure, must use language such as C



Dynamic SQL

- Yes. Can process dynamic SQL
- No. Can not use SQLDA directly in SQL
 - EXECUTE IMMEDIATE
 - EXECUTE ... INTO ... USING
 - PREPARE
 - OPEN
 - CLOSE
 - FETCH ... INTO
- No. DDL is not currently supported in external routines



Constructing SQL on the fly

```
begin
declare :sc, :i integer;
call EX1_START_WRITE_TXN (:sc);
if :sc <> 0 then
    call ABORT_WITH_ERROR (:sc, 'XX001'); end if;
for :i in 1 to 20
do
    call EX1_EXECUTE_IMMEDIATE (:sc,
        'insert into SAMPLE values ('
        || cast(:i as VARCHAR(10)) || ');');
    if :sc <> 0 then
        call ABORT_WITH_ERROR (:sc, 'XX002'); end if;
end for;
call EX1_COMMIT (:sc);
if :sc <> 0 then
    call ABORT_WITH_ERROR (:sc, 'XX004'); end if;
end;
```



Alter command maintenance

- Most external routine attributes can be changed using alter function or alter procedure
- If sharable image changes location use alter so that dependencies are preserved
- Alter function and alter procedure work on routines in a module
- Preferred over DROP and CREATE because relationships are retained



What's in a name?

- Technically SYS\$SHARE:APP.EXE and SYS\$LIBRARY:APP.EXE are the same file
- However, try to keep the location strings the same to avoid multiple image activations
- Older versions of OpenVMS might activate lowercase locations as distinct from uppercase locations [fixed in recent OpenVMS releases]



Privileges

- Prior to 7.2 an image installed with more privileges than the user raised this error:
 - -RDMS-E-RTNUSENOTALL, routine "GET_ONE" can not be used, too many privileges available
- This prevented SQL from activating an image not authorized by the application



Privileges

- Restriction lifted in V7.2
- Use `INSTALL ADD/SHARE`
- The image must be located in a secure directory – `SYS$SHARE`
- When the image is a “known” image and accessed by executive mode logical names then we assume that the image is trusted



BIND ON SERVER

- Invented for operating systems that only provided two memory protections zones
- Very little reason to use it on OpenVMS
- Most applications want process context
- BIND ON SERVER creates extra process to run routines and this overhead is usually not desirable
- V7.2 eliminates the requirement for a separate privilege environment



For More Information

- www.oracle.com/rdb
- metalink.oracle.com
- www.hp.com/products/openvms
- ian.e.smith@oracle.com
- [Oracle Rdb Guide to SQL Programming Manual](#)
- [Guide to SQL Programming: Advanced External Routines](#) (pending)



Questions? & Answers!



ORACLE IS THE INFORMATION COMPANY

ORACLE®