24

DBMS_APPLICATION_INFO

Application developers can use the `DBMS_APPLICATION_INFO` package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules and debugging.

This chapter contains the following topics:

DBMS_APPLICATION_INFO Overview

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the `V$SESSION` and `V$SQLAREA` views.

DBMS_APPLICATION_INFO Security Model

No further privileges are required. The `DBMSAPIN.SQL` script is already run as a part of standard database creation.

> **Note:**
>
> The public synonym for `DBMS_APPLICATION_INFO` is not dropped before creation so that you can redirect the public synonym to point to your own package.

DBMS_APPLICATION_INFO Operational Notes

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather your own statistics based on module, you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the `SYS` version of the package. The public synonym for `DBMS_APPLICATION_INFO` can then be changed to point to the DBA's version of the package.

Summary of DBMS_APPLICATION_INFO Subprograms

This table lists and describes the DBMS_APPLICATION_INFO package subprograms.

Table 24-1 DBMS_APPLICATION_INFO Package Subprograms

Subprogram	Description
READ_CLIENT_INFO Procedure	Reads the value of the client_info field of the current session
READ_MODULE Procedure	Reads the values of the module and action fields of the current session
SET_ACTION Procedure	Sets the name of the current action within the current module
SET_CLIENT_INFO Procedure	Sets the client_info field of the session
SET_MODULE Procedure	Sets the name of the module that is currently running to a new module
SET_SESSION_LONGOPS Procedure	Sets a row in the V$SESSION_LONGOPS table

READ_CLIENT_INFO Procedure

This procedure reads the value of the client_info field of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO (
   client_info OUT VARCHAR2);
```

Parameters

Table 24-2 *READ_CLIENT_INFO Procedure Parameters*

Parameter	Description
client_info	Last client information value supplied to the SET_CLIENT_INFO procedure.

READ_MODULE Procedure

This procedure reads the values of the module and action fields of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_MODULE (
   module_name OUT VARCHAR2,
   action_name OUT VARCHAR2);
```

Parameters

Table 24-3 READ_MODULE Procedure Parameters

Parameter	Description
module_name	Last value that the module name was set to by calling SET_MODULE.

Table 24-3 (Cont.) READ_MODULE Procedure Parameters

Parameter	Description
action_name	Last value that the action name was set to by calling SET_ACTION or SET_MODULE.

Usage Notes

Module and action names for a registered application can be retrieved by querying V$SQLAREA or by calling the READ_MODULE procedure. Client information can be retrieved by querying the V$SESSION view, or by calling the READ_CLIENT_INFO Procedure.

Examples

The following sample query illustrates the use of the MODULE and ACTION column of the V$SQLAREA.

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';

SQL_TEXT DISK_READS MODULE ACTION
------------------- ---------- ------------------ ----------------
INSERT INTO emp 1 add_employee insert into emp
(ename, empno, sal, mgr, job, hiredate, comm, deptno)
VALUES
(name, next.emp_seq, manager, title, SYSDATE, commission, department)

1 row selected.
```

SET_ACTION Procedure

This procedure sets the name of the current action within the current module.

Syntax

```
DBMS_APPLICATION_INFO.SET_ACTION (
   action_name IN VARCHAR2);
```

Parameters

Table 24-4 SET_ACTION Procedure Parameters

Parameter	Description
action_name	The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.

Usage Notes

The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, subsequent transactions may be logged with the previous transaction's name.

Example

The following is an example of a transaction that uses the registration procedure:

```
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN

-- balance transfer transaction

   DBMS_APPLICATION_INFO.SET_ACTION(
      action_name => 'transfer from chk to sav');
   UPDATE chk SET bal = bal + :amt
      WHERE acct# = :acct;
   UPDATE sav SET bal = bal - :amt
      WHERE acct# = :acct;
   COMMIT;
   DBMS_APPLICATION_INFO.SET_ACTION(null);

END;
```

SET_CLIENT_INFO Procedure

This procedure supplies additional information about the client application.

Syntax

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (
   client_info IN VARCHAR2);
```

Parameters

Table 24-5 SET_CLIENT_INFO Procedure Parameters

Parameter	Description
client_info	Supplies any additional information about the client application. This information is stored in the V$SESSION view. Information exceeding 64 bytes is truncated.

> **Note:**
>
> CLIENT_INFO is readable and writable by any user. For storing secured application attributes, you can use the application context feature.

SET_MODULE Procedure

This procedure sets the name of the current application or module.

Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
   module_name IN VARCHAR2,
   action_name IN VARCHAR2);
```

Parameters

Table 24-6 SET_MODULE Procedure Parameters

Parameter	Description
module_name	Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.
action_name	Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.

Usage Notes

Example

```
CREATE or replace PROCEDURE add_employee(
  name VARCHAR2,
  salary NUMBER,
  manager NUMBER,
  title VARCHAR2,
  commission NUMBER,
  department NUMBER) AS
BEGIN
  DBMS_APPLICATION_INFO.SET_MODULE(
    module_name => 'add_employee',
    action_name => 'insert into emp');
  INSERT INTO emp
    (ename, empno, sal, mgr, job, hiredate, comm, deptno)
    VALUES (name, emp_seq.nextval, salary, manager, title, SYSDATE,
            commission, department);
  DBMS_APPLICATION_INFO.SET_MODULE(null,null);
END;
```

SET_SESSION_LONGOPS Procedure

This procedure sets a row in the V$SESSION_LONGOPS view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.

Applications may use the SET_SESSION_LONGOPS procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the V$SESSION_LONGOPS view.

Syntax

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (
   rindex      IN OUT BINARY_INTEGER,
   slno        IN OUT BINARY_INTEGER,
   op_name     IN     VARCHAR2       DEFAULT NULL,
   target      IN     BINARY_INTEGER DEFAULT 0,
   context     IN     BINARY_INTEGER DEFAULT 0,
   sofar       IN     NUMBER         DEFAULT 0,
   totalwork   IN     NUMBER         DEFAULT 0,
   target_desc IN     VARCHAR2       DEFAULT 'unknown target',
   units       IN     VARCHAR2       DEFAULT NULL)

set_session_longops_nohint constant BINARY_INTEGER := -1;
```

Parameters

Table 24-7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
rindex	A token which represents the v$session_longops row to update. Set this to set_session_longops_nohint to start a new row. Use the returned value from the prior call to reuse a row.
slno	Saves information across calls to set_session_longops: It is for internal use and should not be modified by the caller.
op_name	Specifies the name of the long running task. It appears as the OPNAME column of v$session_longops. The maximum length is 64 bytes.
target	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of v$session_longops.
context	Any number the client wants to store. It appears in the CONTEXT column of v$session_longops.
sofar	Any number the client wants to store. It appears in the SOFAR column of v$session_longops. This is typically the amount of work which has been done so far.
totalwork	Any number the client wants to store. It appears in the TOTALWORK column of v$session_longops. This is typically an estimate of the total amount of work needed to be done in this long running operation.
target_desc	Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of v$session_longops. The maximum length is 32 bytes.
units	Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes.

Example

This example performs a task on 10 objects in a loop. As the example completes each object, Oracle updates V$SESSION_LONGOPS on the procedure's progress.

```
DECLARE
        rindex    BINARY_INTEGER;
        slno      BINARY_INTEGER;
```

```
  totalwork number;
  sofar     number;
  obj       BINARY_INTEGER;

BEGIN
  rindex := dbms_application_info.set_session_longops_nohint;
  sofar := 0;
  totalwork := 10;

  WHILE sofar < 10 LOOP
    -- update obj based on sofar
    -- perform task on object target

    sofar := sofar + 1;
    dbms_application_info.set_session_longops(rindex, slno,
      "Operation X", obj, 0, sofar, totalwork, "table", "tables");
  END LOOP;
END;
```