

## **FINE-GRAINED ACCESS CONTROL WITH ORACLE8I'S VIRTUAL PRIVATE DATABASE FEATURE**

*Douglas Scherer, Core Paradigm, Inc.*

Oracle8i provides a virtual private database feature that is used to associate a security policy with a specific table or view. This feature enables you to create a contextual environment for every application that is used against the database. Complex rules can be shared by multiple tables and views and can be combined with the ability to set up the users' environments at logon time. This leads to the strict enforcement of security rules no matter what tool is used. The virtual private database feature has two main capabilities:

- Fine-grained access control: The ability to assign security policies to views and tables
- Application context: The ability to securely populate and retrieve cached attributes containing information about your application

### **USING DBMS\_RLS TO PROVIDE ROW LEVEL AND APPLICATION ORIENTED SECURITY**

It is possible to provide a fine-grained level of security by applying programmatic and dynamic characteristics to your security policy. The means to do this are provided in the DBMS\_RLS built-in package. The DBMS\_RLS package, which is available only in the Enterprise Edition of the Oracle database, can enforce security policies such as:

Allowing hospital physicians to see only the records of their own patients, customers to see only their own orders, managers to see only the employees in their own department

Supporting different security policies for different types of SQL statements. For example, allowing a user to SELECT all of the records in the emp table, but update only those employees in their department.

Enforcement of these rules is accomplished by defining security policies in a package and then assigning that set of policies to a table. Each policy dynamically creates a predicate that will be attached to the WHERE clause of any DML and SELECT statement that is performed against that table. The enforcement policy is active regardless of the application the client is using to access the data.

The following six steps will create an application context to supply related security information for a Human Resources application system using a customized context in conjunction with DBMS\_RLS.

1. Plan the policy
2. Create the context
3. Create an application security package that defines the attributes for the context
4. Set up the application security package to have its attributes automatically populated when the user logs on
5. Create a table policy security package to be used in a policy to secure the emp table. The security function in the package will make use of the attributes in the application context.
6. Assign the policy to the table or view

### **PLAN THE POLICY**

Building a complex security policy package first requires specifying the policy to be enforced. The following security policy will be enforced against the emp table when all of the steps are completed.

The user Scott will own the application tables and views. Only Scott and the owner of the security package will have the right to issue SELECT or UPDATE statements directly against emp. All other's rights to directly query or update the emp table will be done via a view called employees.

SELECT and UPDATE rights will be determined at runtime based on the job level of the employee who is attempting to perform the action as follows:

Managers and above can SELECT all of employee records, but can update only the records of employees that they directly manage

Other employees can SELECT only the records of other employees in their department, and may not UPDATE any employee records

Anyone else logged on to the system (i.e. those who are not listed as employees in the emp table), cannot see any employees.

A package that contains a policy for querying from scott.emp will be defined in this example. This SELECT policy will enforce the rule that users who do not have the rank of manager or above can see only employees

in their own department, while managers and above can see all employee records. For this example, you will give the scott.emp table an additional column called oracle\_username VARCHAR2(30) NOT NULL and populate it with the corresponding emp.ename. In other words the new column called oracle\_username will contain the same value as the value in ename. Additionally, you will also add a view to Scott's schema with the following definition:

```
CREATE OR REPLACE VIEW employees
AS SELECT *
FROM emp;
```

Adding the employees view on scott.emp avoids causing an infinite loop for policies that must query the policy table from within the security function. The security function in the example queries the emp table to find the job and department of the currently logged on user. If this function were assigned to the emp table itself, then the query inside the function would cause another query to occur against emp which would cause another, and so on.

To enforce the security on the data in the emp table, revoke SELECT rights on the table to all accounts except the one managing security. In this case, assume that the account managing Human Resources security is called HR\_SECURITY\_MANAGER. HR\_SECURITY\_MANAGER will be the manager of the Human Resources application, not a database administrator. Do not grant DBA to HR\_SECURITY\_MANAGER. SELECT rights should be granted on the employees view to all who may need to see it.

## CREATE THE CONTEXT

Note: The HR\_SECURITY\_MANAGER account should be granted the CREATE ANY CONTEXT privilege.

### **SYS\_CONTEXT**

Extends USERENV to 20 Attributes

Interesting new attributes such as IP\_ADDRESS

Syntax: SYS\_CONTEXT(<'context namespace'>,  
<'attribute name'>  
);

Example 1:

```
SELECT SYS_CONTEXT( 'USERENV' , ' IP_ADDRESS ' )
FROM dual;
```

```
IP_ADDRESS
-----
172.334.22.41
```

Example 2 [Given an additional column (ip\_address) added to the emp table]:

```
ALTER TABLE emp
ADD (CONSTRAINT emp_ip_address_chk
CHECK (ip_address LIKE
SUBSTR(SYS_CONTEXT( 'USERENV' ,
' IP_ADDRESS ' ), 1, 7
) || '%'
);
```

```
CREATE OR REPLACE CONTEXT hr_context
  USING hr_security_manager.hr_application_security;
```

### CREATE AN APPLICATION SECURITY PACKAGE THAT DEFINES THE ATTRIBUTES FOR THE CONTEXT

The application security package will populate the application context attributes and will later be configured to run at logon time, automatically setting the user's application context. The DBMS\_SESSION.SET\_CONTEXT built-in will be used in the package to set the attributes.

```
CREATE OR REPLACE PACKAGE hr_application_security
AS
  PROCEDURE set_hr_context_attributes;
END;

CREATE OR REPLACE PACKAGE BODY hr_application_security
AS
  PROCEDURE set_hr_context_attributes
  IS
    vr_emp scott.emp%ROWTYPE;
  BEGIN
    SELECT empno, oracle_username, job, deptno
      INTO vr_emp.empno, vr_emp.oracle_username,
           vr_emp.job, vr_emp.deptno
    FROM scott.emp
    WHERE ename = SYS_CONTEXT('USERENV', 'SESSION_USER');
    -- populate the application context attributes that will
    -- later be read with the SYS_CONTEXT function.
    -- Example: SELECT SYS_CONTEXT('HR_CONTEXT', 'JOB') FROM dual;
    DBMS_SESSION.SET_CONTEXT('HR_CONTEXT', 'EMPNO',
                             vr_emp.empno
                            );
    DBMS_SESSION.SET_CONTEXT('HR_CONTEXT', 'ORACLE_USERNAME',
                             vr_emp.oracle_username
                            );
    DBMS_SESSION.SET_CONTEXT('HR_CONTEXT', 'JOB', vr_emp.job);
    DBMS_SESSION.SET_CONTEXT('HR_CONTEXT', 'DEPTNO', vr_emp.deptno);
  EXCEPTION
    WHEN OTHERS
    THEN
      -- Allow the HR_CONTEXT attributes to be NULL. The various table
      -- policies will detect this and disallow access to table.
      NULL;
  END;
END;
```

### SET UP THE APPLICATION SECURITY PACKAGE TO HAVE ITS ATTRIBUTES AUTOMATICALLY POPULATED WHEN THE USER LOGS ON

Although you can set the application context attributes from a call within the application, it may be preferable to set them initially and automatically when the user connects to the database. A database event trigger can be created that will fire a user when they connect to the database. The trigger calls a package that populates the attributes. In this example, the after\_logon\_al trigger calls the hr\_application\_security.set\_hr\_context\_attributes procedure, which populates the attributes of the user's HR\_CONTEXT application context.

*Caution: Since database triggers affect the database at a fundamental level, they should be created only by a designated DBA account, not the HR\_SECURITY\_MANAGER application management account. Grant execute on hr\_application\_security to the DBA creating the after\_logon\_al trigger.*

```
CREATE OR REPLACE TRIGGER after_logon_al
  AFTER LOGON ON DATABASE
BEGIN
  -- Populate the application context attributes
  hr_security_manager.hr_application_security.set_hr_context_attributes;
END;
```

*Note: If you have problems connecting after creating (or attempting to create) this trigger, connect as - sys as sysdba - and drop or disable the trigger.*

### **CREATE A SECURITY POLICY PACKAGE TO BE USED TO SECURE THE EMP TABLE.**

The security function in the package will make use of the attributes in the application context. This package should be created as the security manager user HR\_SECURITY\_MANGER.

```
CREATE OR REPLACE PACKAGE hr_table_security
AS
  cons_disallow_all_reads CONSTANT CHAR(3) := '1=2';
  SUBTYPE predicate_type IS VARCHAR2(2000);
  FUNCTION update_employees_policy
    (i_object_schema all_objects.owner%TYPE,
     i_object_name all_objects.object_name%TYPE
    )
    RETURN VARCHAR2;
  --
  FUNCTION select_employees_policy
    (i_object_schema all_objects.owner%TYPE,
     i_object_name all_objects.object_name%TYPE
    )
    RETURN VARCHAR2;
END;
```

```

CREATE OR REPLACE PACKAGE BODY hr_table_security
AS
  -- -----
  -- PRIVATE CODE --
  -- -----
  FUNCTION is_at_least_a_manager
    (i_job scott.emp.job%TYPE)
    RETURN BOOLEAN
  IS
  BEGIN
    RETURN i_job IN ('MANAGER', 'PRESIDENT');
  END;
  -- -----
  -- PUBLIC CODE --
  -- -----
  FUNCTION update_employees_policy
    (i_object_schema all_objects.owner%TYPE,
     i_object_name all_objects.object_name%TYPE
    )
    RETURN VARCHAR2
  IS
    v_predicate predicate_type := cons_disallow_all_reads;
  BEGIN
    -- Enforce the rule that only managers and above
    -- can update employees.
    IF is_at_least_a_manager(SYS_CONTEXT('HR_CONTEXT','JOB'))
    THEN
      -- The user is managerial. Allow them to update only
      -- employees records for their own department.
      v_predicate := 'deptno = ' || SYS_CONTEXT('HR_CONTEXT', 'DEPTNO');
    END IF;
    RETURN v_predicate;
  EXCEPTION
    WHEN OTHERS
    THEN
      RETURN cons_disallow_all_reads;
  END;
  --
  FUNCTION select_employees_policy
    (i_object_schema all_objects.owner%TYPE,
     i_object_name all_objects.object_name%TYPE
    )
    RETURN VARCHAR2
  IS
    v_predicate predicate_type := cons_disallow_all_reads;
  BEGIN
    IF is_at_least_a_manager(SYS_CONTEXT('HR_CONTEXT','JOB'))
    THEN
      -- The user is managerial. Let the user
      -- see all the records in emp.
      v_predicate := NULL;
    ELSIF SYS_CONTEXT('HR_CONTEXT', 'DEPTNO') IS NOT NULL
    THEN
      -- The user is not managerial. Restrict their access.
      v_predicate := 'deptno = ' ||
        SYS_CONTEXT('HR_CONTEXT', 'DEPTNO');
    ELSE
      -- The user is not a valid employee, make sure that
      -- cons_disallow_all_reads is the value in v_predicate.
      v_predicate := cons_disallow_all_reads;
    END IF;
    RETURN v_predicate;
  EXCEPTION
    WHEN OTHERS

```

```
    THEN
      RETURN cons_disallow_all_reads;
    END;
END;
```

### **ASSIGN THE POLICY TO THE TABLE OR VIEW**

Note: Execute privilege must be granted on DBMS\_RLS to HR\_SECURITY\_MANAGER.

```
BEGIN
  DBMS_RLS.ADD_POLICY('SCOTT', 'EMPLOYEES', 'SELECT_EMPLOYEES_POLICY',
                    'HR_SECURITY_MANAGER',
                    'hr_table_security.select_employees_policy',
                    'SELECT'
                    );
  DBMS_RLS.ADD_POLICY('SCOTT', 'EMPLOYEES', 'UPDATE_EMPLOYEES_POLICY',
                    'HR_SECURITY_MANAGER',
                    'hr_table_security.update_employees_policy',
                    'UPDATE'
                    );
END;
```

### **ABOUT THE AUTHOR**

**Douglas Scherer** (dscherer@coreparadigm.com) is president of Core Paradigm, a management-consulting firm that assists C-level executives to identify their IT needs and access the resources to fulfill those needs. He is a frequent presenter at international conferences, author of two books on Oracle technology, and a teacher at Columbia University.