Virtual Private Database Features in Oracle 10g.

SAGE Computing Services

Customised Oracle Training Workshops and Consulting.

www.sagecomputing.com.au

Christopher Muir – Senior Systems Consultant

Agenda

- Modern security requirements
- Virtual Private Database (VPD)
  - Row Level Security (RLS) and Policies
  - Contexts and the after_logon_db trigger
  - Column Level Security
  - Column Level Masking
- Fine Grained Auditing (FGA)
- VPD lessons-learnt!
- Online resources

Traditional Oracle RDBMS Security

- One central schema user account
- Multiple user accounts
- DB roles granted privileges on schema objects
- DB roles granted to user accounts
- DBA responsible for user account administration

Contemporary Security Requirements

- No-longer just a technical issue
- It's not just about the password!
- Makes business sense
- Upcoming Australian legislation
- Security studies show greatest threat is from within, not outside an organisation

Contemporary Security – Web Driven

- Scalable architectures: 1-1000s users
- Mid-tier persistence frameworks using connection pooling
  - 1 schema account & 1 Oracle user account
  - Users and permissions handled by application, not database
- Common example: Bank Account website

Potential Security Solutions

- Modify all queries:
  - Where clauses use functions to determine if rows returned
  - Columns use functions to determine if you can see column data
- An expensive exercise with a maintenance overhead
- Will future programmes remember to implement this?
- Oracle's virtual private database (VPD) features provide a solution to all these issues

Row Level Security (RLS)

- Enforce an additional predicate (where clause) against all queries on a table

```
SELECT * FROM employees;

Becomes

SELECT * FROM employees WHERE department_id = 80;
```

- Use to limit access to rows

Policies & dbms_rls

- Create a database “Policy” object
- Use *dbms_rls* PL/SQL package

```
dbms_rls.add_policy(
     object_schema    => 'HR'
    ,object_name      => 'EMPLOYEES'
    ,policy_name      => 'EMP_RLS_POLICY'
    ,function_schema  => 'HR'
    ,policy_function  => 'emp_mgmt_pk.f_emp_rls_policy'
    ,statement_types  => 'SELECT');
```

Policy Functions

- Policies are based on PL/SQL functions

```
PACKAGE BODY emp_mgmt_pk AS

  FUNCTION f_emp_rls_policy
    (object_schema IN VARCHAR2,object_name IN VARCHAR2)
  RETURN VARCHAR2 IS
  BEGIN
    RETURN 'department_id = 80';
  END;

END;
```

Policy Types

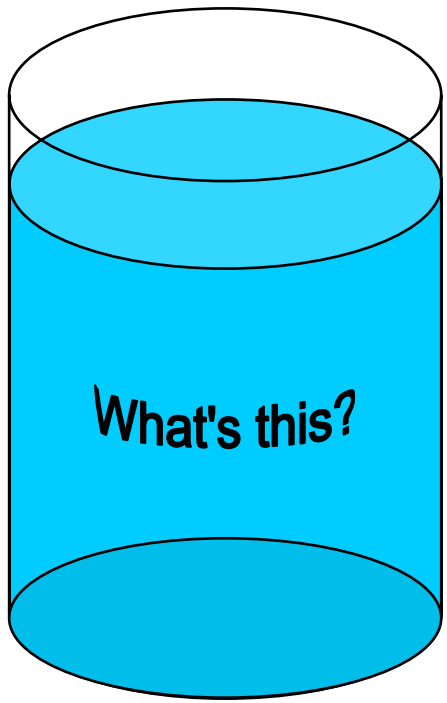
- Specify via *dbms_rls.add_policy* call
- Dynamic
  - Default
  - Re-parsed and evaluated each time
- Static
  - Parsed and evaluated once
  - Cached in the SGA for speed
- Hybrids

RLS Implementation

- Allowed on tables, views and synonyms
- Not limited to select, can be extended to all DML on an object
- Multiple policies on same object are allowed (ANDed)
- Applies to all user & users' DML on that object
- Sys is exempt from VPD policies

RLS Examples

- Common Examples
  - Bank Accounts
  - Employees - Department Head
  - Others?



RLS Limitations?

- Q: What's the limitation with the RLS policy we created?
- A: Hardcoded static predicate "department_id = 80"
- Need the ability to create a dynamic predicate, that is dependent on the user who is currently logged in

Contexts

- A session attribute group called a “Namespace”
- Exist for the life of the session connection

- Applications can use this to store the “context” of the current user

eg. HR_CONTEXT.USER_DEPT = 80

Create Contexts

- A context is created with an associated *trusted* package

```
CREATE CONTEXT hr_context USING hr.emp_mgmt_pk;
```

- Create as Sys
- The package is *trusted* to change the context attributes

Write to a Context

- Call *dbms_session.set_context* in your context package

```
PACKAGE BODY emp_mgmt_pk AS
  PROCEDURE p_set_context IS
  BEGIN
    dbms_session.set_context(
       namespace => 'HR_CONTEXT'
      ,attribute => 'USER_DEPT'
      ,value     => '80');
  END;
END;
```

Reading Contexts

- Via a *sys_context* call

```
SELECT sys_context('HR_CONTEXT','USER_DEPT')
FROM DUAL;

SYS_CONTEXT('HR_CONTEXT','USER_DEPT')
----------------------------------------
80
```

Modified Policy

```
PACKAGE BODY emp_mgmt_pk AS

  FUNCTION f_emp_rls_policy
    (object_schema IN VARCHAR2,object_name IN VARCHAR2)
  RETURN VARCHAR2 IS
  BEGIN
    RETURN 'department_id = ' ||
           'sys_context(''HR_CONTEXT'',''USER_DEPT'')';
  END;

END;
```

USERENV Context

- A default context *USERENV* exists for all sessions
- Provides predefined attributes
- Access via:

sys_context('USERENV','<attribute>')

USERENV Attributes

- Some interesting attributes:
  - DB_NAME — Database name
  - SESSION_USER — Authenticated database user name
  - SESSIONID — Session identifier
  - LANGUAGE — Session language and character set
  - IP_ADDRESS — Client IP address
  - CLIENT_IDENTIFIER — User defined client identifier for the session
  - HOST — Database host name
  - OS_USER — Operating system client username

etc

- Many catches and anomalies in using these so be careful
- See Barry Johnson's presentation (July AUSOUG seminar):
  - http://bkaj.org/oracle/

Enforcing Context – after_logon_db

- Create a database *after_logon_db* trigger
- Fires on every database connection (except Sys)
- Enforces context set for all connections

```
CREATE OR REPLACE TRIGGER after_logon_db
  AFTER LOGON ON DATABASE
BEGIN
  hr.emp_mgmt_pk.p_set_context;
END;
```

RLS Errors

- As RLS policies are based on a literal string:
  - Syntax errors can occur
  - Syntax errors only detected at runtime
- Result in an ORA-28113 'policy predicate error'
- Reported in user trace files
- Presents a problem for developers as they'll need access to these to debug
- DBAs will need to provide access on the network for quick debugging

after_logon_db Errors

- Any error in this trigger will cause all connections to the database to fail, Sys exempted but including OEM
- Be wary of basing trigger on PLSQL with dependencies
- Any changes to dependencies will invalidate PL/SQL then trigger
- I find DBAs get annoyed when you do this for the 10th time
- Suggested approach is developers have their own db
- Only move code to shared development database once testing complete
- Impact analysis during implementation is essential

Policy Groups

- Provides easier maintenance of policies

```
dbms_rls.create_policy_group(
    object_schema => 'HR'
   ,object_name   => 'EMPLOYEES'
   ,policy_group  => 'HR_GRP');
```

- New policies can be created in the policy group
- Also *drop_policy_group*
- You must drop individual policies before group

RLS Alternative Uses

- Not just useful for security
- eg. Time governed queries:

```
SELECT *
FROM   land_assessments
WHERE  effective_from_date <= :p_date
AND   (effective_to_date   >= :p_date
OR     effective_to_date IS NULL);
```

- Stop developers from implementing where clause everywhere
- Instead use RLS so predicate added regardless

Column Level VPD

- Enforcement when specified columns queried
- If columns aren't queried, policy isn't enforced
- Policy function similar to RLS policy function:

```
FUNCTION f_emp_cl_policy
  (object_schema IN VARCHAR2, object_name IN VARCHAR2)
RETURN VARCHAR2 IS
BEGIN
  RETURN 'department_id = 90';
END;
```

Column Level Add Policy

- Policy includes specified columns

```
dbms_rls.add_policy(
   object_schema     => 'HR'
  ,object_name       => 'EMPLOYEES'
  ,policy_name       => 'EMP_CL_POLICY'
  ,function_schema   => 'HR'
  ,policy_function   => 'emp_mgmt_pk.f_emp_cl_policy '
  ,sec_relevant_cols => 'SALARY, COMMISION_PCT');
```

- Create on tables, views but not synonyms
- Early implementation of column masking

Column Masking VPD

- Implemented similar to column-level VPD

```
dbms_rls.add_policy(
   object_schema          => 'HR'
  ,object_name            => 'EMPLOYEES'
  ,policy_name            => 'EMP_CL_POLICY'
  ,function_schema        => 'HR'
  ,policy_function        => 'emp_mgmt_pk.f_emp_cl_policy'
  ,sec_relevant_cols      => 'SALARY, COMMISION_PCT'
  ,sec_relevant_cols_opt => dbms_rls.all_rows);
```

Column Masking Predicates

- All rows returned regardless
- If predicate evaluates as:
  - TRUE Column result returned
  - FALSE Column returned as NULL
- If a NULL predicate is returned:
  - NULL Column result returned
- Create on tables, not views or synonyms.

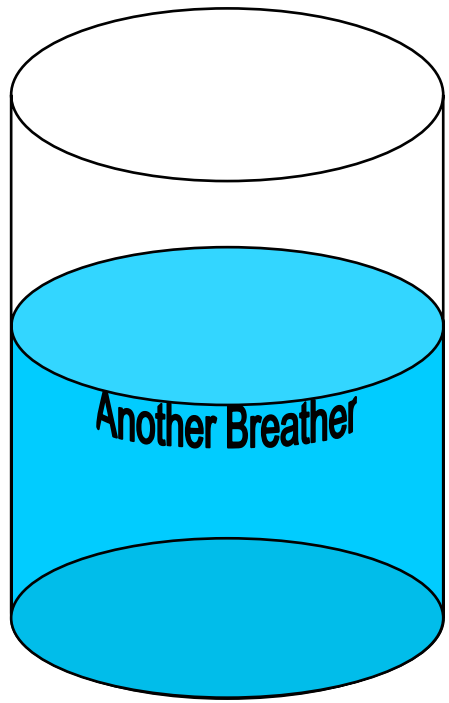
Column Masking Catches

- If masked columns are part of a predefined fk used by queries,
  - and VPD policy returns null,
  - query join will not find any results

- Consider null results in query aggregates

Masking Uses

- Common examples:
  - Credit card numbers
  - People's names
  - Others?



VPD Policy Exemptions

- Sys is always exempted from VPD policies
- Other database users may be exempt through granting EXEMPT ACCESS POLICY

VPD Data Dictionary

all_policies	user_policies
all_policy_groups	user_policy_groups
all_policy_contexts	user_policy_contexts

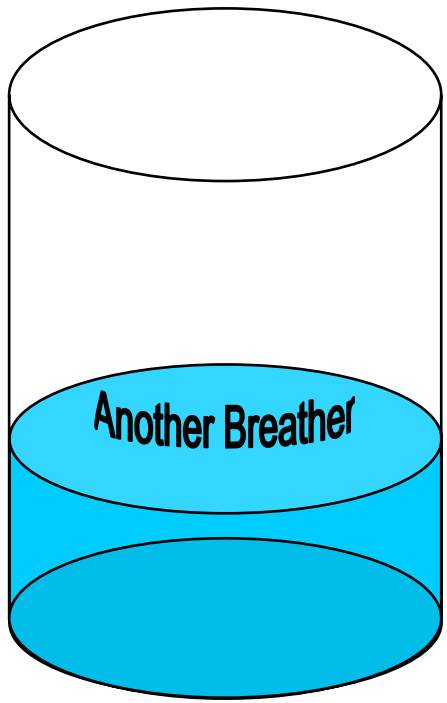
v$vpd_policy

VPD Lessons Learnt

- Start security early to continuously test VPD code
- Your first instinct in debugging with VPD is often wrong
- Ensure mid-tier programmers are aware of VPD constraints

Performance Implications

- Explain plans do show VPD predicates
- Be wary: Trace/TKProf hides VPD predicates
- However indexes etc are used
- Be wary of policy type causing query reparsing



VPD Future (Predictions)

- Reduce technical limitations
- Improved performance analysis support

- 10g Release 2:
  - Transparent Data Encryption (TDE)
  - (not VPD but interesting).

RIP? (VPD Predictions)

- Public synonym should become redundant (security hole anyhow)
- Replaced by private synonyms per connecting pool account and VPD
- Database user accounts will become limited
- Database roles will lose their convenience
- Free DBAs up for other tasks
- Developers will need to consider application security from mid-tier/client-tier

Database Auditing Mechanisms

- Database auditing mechanisms:
  - Statement Auditing
  - Privilege Auditing
  - Schema Object Auditing
  - Fine Grained Auditing

Fine Grained Auditing (FGA)

- Audit on rows and columns returned & modified
- Focus on DML executed on sensitive data
  - 9i Select only
  - 9i database must be running with CBO
  - 10g Select, Insert, Update & Delete
- Tables and views

dbms_fga Policy

```
dbms_fga.add_policy(
   object_schema    => 'HR'
  ,object_name      => 'EMPLOYEES'
  ,policy_name      => 'EMP_FGA_POLICY');

...or....

  ,audit_column    => 'SALARY'
  ,audit_condition => 'SALARY >= 5000');
```

FGA Data Dictionary

```
SELECT * FROM employees;

...results in...

SELECT ... FROM dba_fga_audit_trail;

...populated with...

TIMESTAMP DB_USER OS_USER OBJECT_ OBJECT_N  SQL_TEXT
--------- ------- ------- ------- --------- -----------
15-AUG-05 HR      CHRIS   HR      EMPLOYEES (next line)

SELECT * FROM employees
```

FGA Data Dictionary

all_audit_policies

user_audit_policies

sys.fga_logs$

dba_fga_audit_trail

Alternative FGA Uses

- Not just security auditing:
  - Capture all SQL for index planning
  - Capture bind variables for designing histograms
  - Via *handle* mechanism can fire PL/SQL to (as an example) send emails
  - Or use as a *trigger* to do other work on Select

Fine-Grained Auditing for Real-World Problems 2004 Arup Nanda, OTN.

Consider VPD/FGA....

- Plan for RLS/column masking implementation
- Australian Legislation is changing
- Required by govt and free-enterprise (liability!)
- VPD can be applied retrospectively
- However don't forget regression-testing!
- Immediately consider FGA to audit unsecure data usage
- Put auditing processes in place (don't just collect the data, report on it)

VPD & FGA Availability

- Only available in Oracle 10g Enterprise Edition

Resources

- Oracle Manuals:
  - Security Guide
  - Security Overview
  - PL/SQL Packages and Types References
- Online:
  - OTN Security
  - www.oracle.com/technology/deploy/security

Question & Answers!

enquiries@sagecomputing.com.au

chriscmuir@sagecomputing.com.au

SAGE Computing Services

Customised Oracle Training Workshops and Consulting

www.sagecomputing.com.au