Data Security and Privacy

Topic 11: Virtual Private Databases

Based on Prof. Bertino's Slides

Announcements

- Next Quiz on Feb 15

Oracle VPD

- Virtual Private Database (VPD)
  - <u>Fine-grained access control</u>: associate security policies with database objects
  - <u>Application Context</u>: define and access application or session attributes and use them in access control, for example for implementing temporal access control
- By combining these two features, VPD enables administrators to define and enforce row-level access control policies based on session attributes

Why VPD

- Scalability
  - Table Customers contains 1,000 customer records. Suppose we want customers to access their own records only. Using views, we need to create 1,000 views. Using VPD, it can be done with a single policy function.
- Simplicity
  - Say, we have a table T and many views are based on T. Suppose we want to restrict access to some information in T. Without VPD, all view definitions have to be changed. Using VPD, it can be done by attaching a policy function to T; as the policy is enforced in T, the policy is also enforced for all the views that are based on T.
- Security
  - Server-enforced security (as opposed to application-enforced).

Oracle VPD

- How does it work?

  When a user accesses a table (or view or synonym) which is protected by a VPD policy (function):

1. The Oracle server invokes the policy function.
2. The policy function returns a predicate, based on session attributes or database contents.
3. The server dynamically rewrites the submitted query by appending the returned predicate to the WHERE clause.
4. The modified SQL query is executed.

Oracle VPD - Example

- Suppose Alice has (is the owner of) the following table.

  my_table(owner varchar2(30), data varchar2(30));

- Suppose that we want to implement the following policy:
  - Users can access only data that refer to themselves. However Admins should be able to access any data without restrictions.

Oracle VPD - Example

1. Create a policy function

```
Create function sec_function (object_schema varchar2, object_name varchar2)
Return varchar2
As
        user VARCHAR2(100);
Begin
        if ( SYS_CONTEXT(‘userenv’, ‘ISDBA’) ) then
                return ‘ ’;
        else
                user := SYS_CONTEXT(‘userenv’, ‘SESSION_USER’);
                return ‘owner = ‘ || user;
        end if;
End;
```

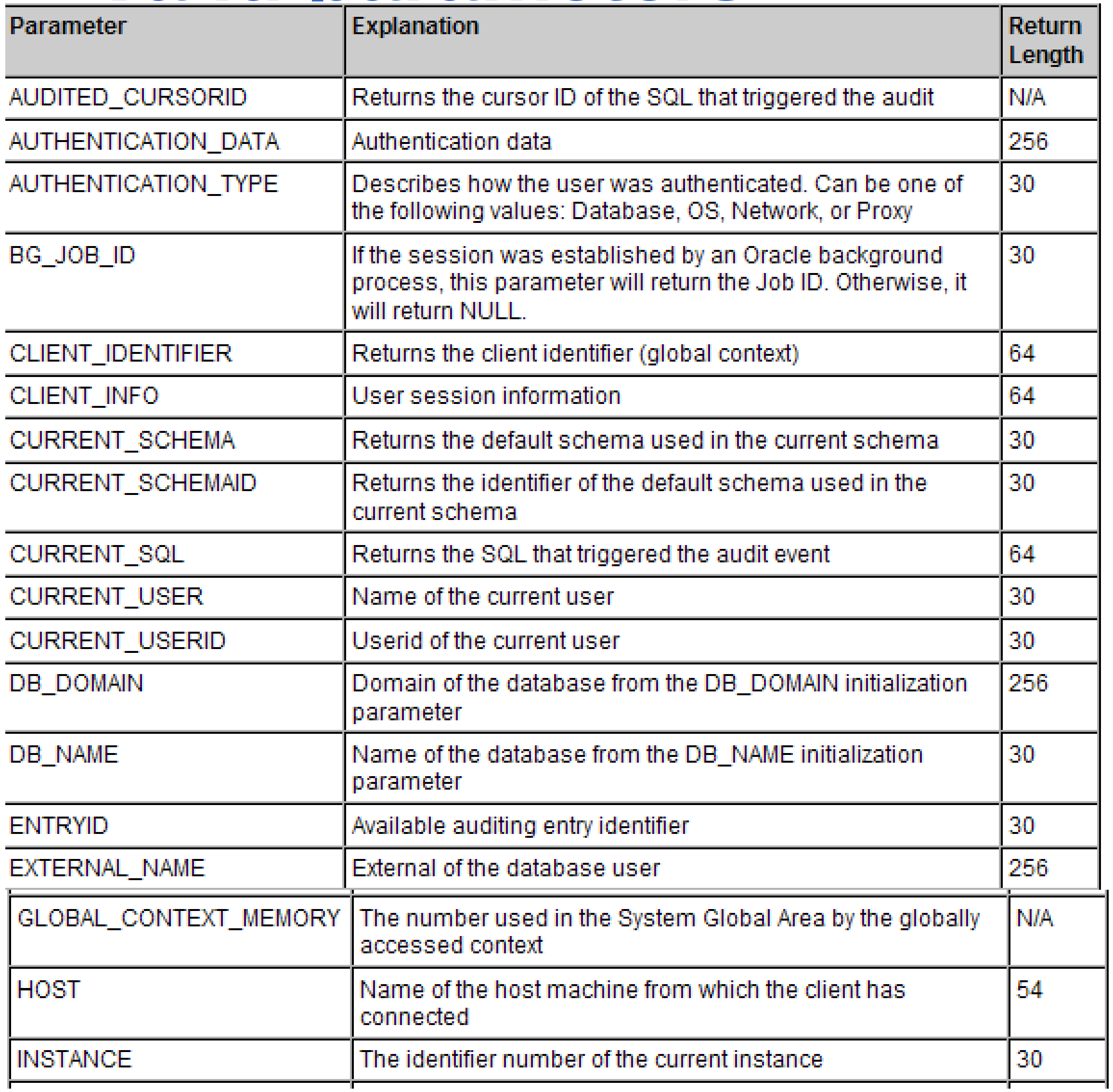
userenv is the pre-defined application context
object_name is the name of table or view to which the policy will apply
object_schema is the schema owning the table or view

SYS_CONTEXT

- In Oracle/PLSQL, the sys_context function is used to retrieve information about the Oracle environment.
- The syntax for the sys_context function is:

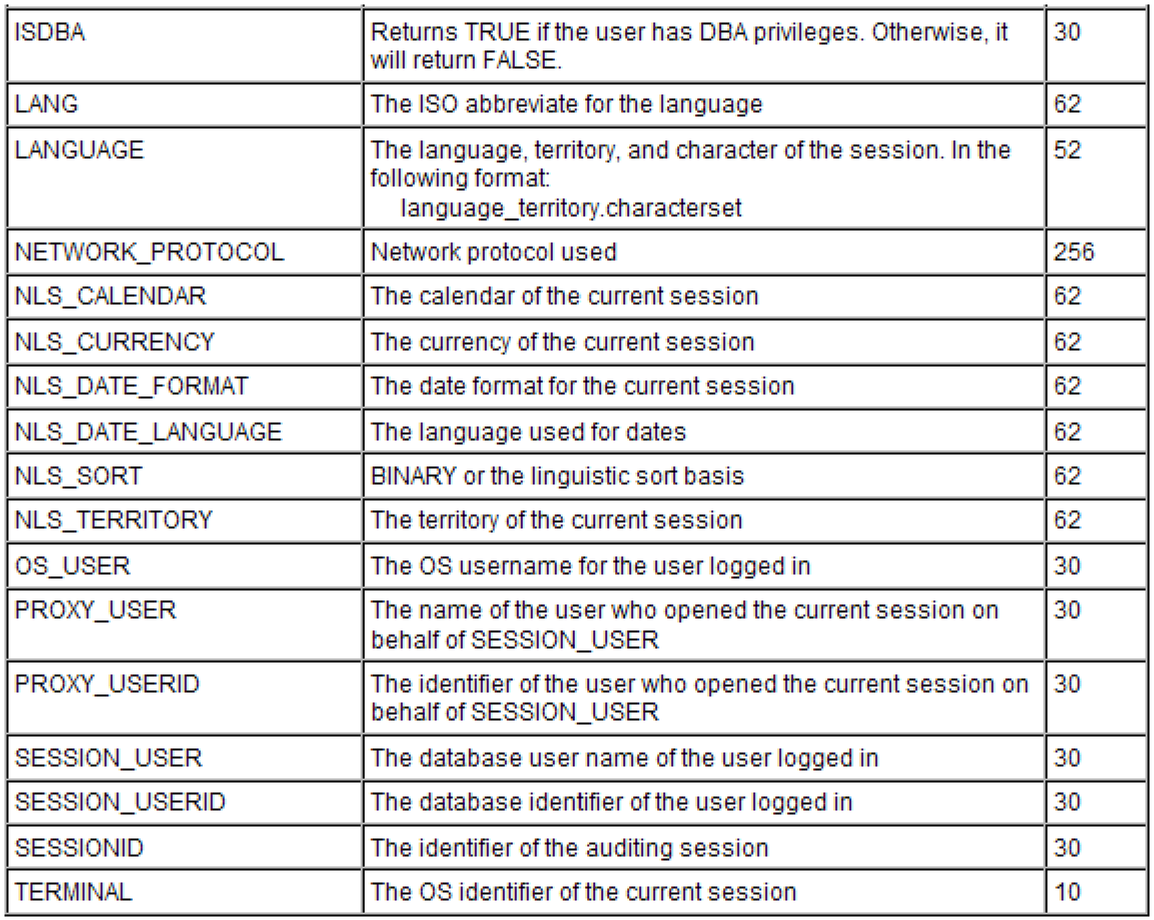
  sys_context(namespace, parameter, [length])
- *namespace* is an Oracle namespace that has already been created. If the namespace is 'USERENV', attributes describing the current Oracle session can be returned.
- *parameter* is a valid attribute that has been set using the DBMS_SESSION.set_context procedure.
- *length* is optional. It is the length of the return value in bytes. If this parameter is omitted or if an invalid entry is provided, the sys_context function will default to 256 bytes

USERENV namespace valid parameters



Parameter	Explanation	Return Length
AUDITED_CURSORID	Returns the cursor ID of the SQL that triggered the audit	N/A
AUTHENTICATION_DATA	Authentication data	256
AUTHENTICATION_TYPE	Describes how the user was authenticated. Can be one of the following values: Database, OS, Network, or Proxy	30
BG_JOB_ID	If the session was established by an Oracle background process, this parameter will return the Job ID. Otherwise, it will return NULL.	30
CLIENT_IDENTIFIER	Returns the client identifier (global context)	64
CLIENT_INFO	User session information	64
CURRENT_SCHEMA	Returns the default schema used in the current schema	30
CURRENT_SCHEMAID	Returns the identifier of the default schema used in the current schema	30
CURRENT_SQL	Returns the SQL that triggered the audit event	64
CURRENT_USER	Name of the current user	30
CURRENT_USERID	Userid of the current user	30
DB_DOMAIN	Domain of the database from the DB_DOMAIN initialization parameter	256
DB_NAME	Name of the database from the DB_NAME initialization parameter	30
ENTRYID	Available auditing entry identifier	30
EXTERNAL_NAME	External of the database user	256
GLOBAL_CONTEXT_MEMORY	The number used in the System Global Area by the globally accessed context	N/A
HOST	Name of the host machine from which the client has connected	54
INSTANCE	The identifier number of the current instance	30

USERENV namespace valid parameters



ISDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.	30
LANG	The ISO abbreviate for the language	62
LANGUAGE	The language, territory, and character of the session. In the following format: language_territory.characterset	52
NETWORK_PROTOCOL	Network protocol used	256
NLS_CALENDAR	The calendar of the current session	62
NLS_CURRENCY	The currency of the current session	62
NLS_DATE_FORMAT	The date format for the current session	62
NLS_DATE_LANGUAGE	The language used for dates	62
NLS_SORT	BINARY or the linguistic sort basis	62
NLS_TERRITORY	The territory of the current session	62
OS_USER	The OS username for the user logged in	30
PROXY_USER	The name of the user who opened the current session on behalf of SESSION_USER	30
PROXY_USERID	The identifier of the user who opened the current session on behalf of SESSION_USER	30
SESSION_USER	The database user name of the user logged in	30
SESSION_USERID	The database identifier of the user logged in	30
SESSIONID	The identifier of the auditing session	30
TERMINAL	The OS identifier of the current session	10

Oracle VPD - Example

2. Attach the policy function to my_table

```
execute dbms_rls.add_policy (object_schema => ‘Alice’,
                             object_name => ‘my_table’,
                             policy_name => ‘my_policy’,
                             function_schema => ‘Alice’,
                             policy_function => ‘sec_function’,
                             statement_types => ‘select, update, insert’,
                             update_check => TRUE );
```

- The VPD security model uses the Oracle *dbms_rls* package (RLS stands for row-level security)
- update_check: Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting update_check to TRUE causes the server to also check the policy against the value after insert or update.

DBMS_RLS.ADD_POLICY syntax

```
DBMS_RLS.ADD_POLICY (
        object schema IN VARCHAR2 NULL,
        object_name IN VARCHAR2,
        policy_name IN VARCHAR2,
        function_schema IN VARCHAR2 NULL,
        policy_function IN VARCHAR2,
        statement_types IN VARCHAR2 NULL,
        update_check IN BOOLEAN FALSE,
        enable IN BOOLEAN TRUE,
        static_policy IN BOOLEAN FALSE,
        policy_type IN BINARY_INTEGER NULL,
        long_predicate IN BOOLEAN FALSE,
        sec_relevant_cols IN VARCHAR2,
        sec_relevant_cols_opt IN BINARY_INTEGER NULL);
```

Oracle VPD - Example

3. Bob accesses my_table

```
select * from my_table;

   => select * from my_table where owner = ‘bob’;
   : only shows the rows such that owner is ‘bob’

insert into my_table values(‘Some data’, ‘bob’); OK!

insert into my_table values(‘Other data’, ‘alice’); NOT OK!
   = because of the check option.
```

Policy Commands

- ADD_POLICY – creates a new policy

- DROP_POLICY – drops a policy

```
DBMS_RLS.DROP_POLICY (
        object schema       IN VARCHAR2 NULL,
        object_name         IN VARCHAR2,
        policy_name         IN VARCHAR2);
```

- ENABLE_POLICY – enables or disables a fine-grained access control policy

```
DBMS_RLS.ENABLE_POLICY (
        object schema       IN VARCHAR2 NULL,
        object_name         IN VARCHAR2,
        policy_name         IN VARCHAR2,
        enable              IN BOOLEAN      );
```

Enable - TRUE to enable the policy, FALSE to disable the policy

Column-level VPD

- Instead of attaching a policy to a whole table or a view, attach a policy only to security-relevant columns
  - Default behavior: restricts the number of rows returned by a query.
  - Masking behavior: returns all rows, but returns NULL values for the columns that contain sensitive information.
- Restrictions
  - Applies only to ‘select’ statements
  - The predicate must be a simple Boolean expression.

Column-level VPD: Example

- Suppose Alice has (is the owner of) the following table.

  Employees (e_id number(2), name varchar2(10), salary number(3));

e_id	Name	Salary
1	Alice	80
2	Bob	60
3	Carl	99

- Policy: Users can access e_id's and names without any restriction. But users can access only their own salary information.

Column-level VPD: Example

1. Create a policy function

```
Create function sec_function (object_schema varchar2, object_name
varchar2)
Return varchar2
As
    user VARCHAR2(100);
Begin
    user := SYS_CONTEXT(‘userenv’, ‘SESSION_USER’);
    return ‘name = ‘ || user;
End;
```

Column-level VPD: Example

2. Attach the policy function to Employees (default behavior)

```
execute dbms_rls.add_policy (object_schema => ‘Alice’,
                             object_name => ‘employees’,
                             policy_name => ‘my_policy’,
                             function_schema => ‘Alice’,
                             policy_function => ‘sec_function’,
                             sec_relevant_cols=>’salary’);
```

Column-level VPD: Example

3. Bob accesses table Employees (with the default behavior). REMEMBER: default behavior restricts the number of rows returned by a query

a) select e_id, name from Employee;

e_id	Name
1	Alice
2	Bob
3	Carl

b) select e_id, name, salary from Employee;

e_id	Name	Salary
2	Bob	60

Column-level VPD: Example

2’. Attach the policy function to Employees (masking behavior)

```
execute dbms_rls.add_policy (object_schema => ‘Alice’,
                             object_name => ‘employees’,
                             policy_name => ‘my_policy’,
                             function_schema => ‘Alice’,
                             policy_function => ‘sec_function’,
                             sec_relevant_cols=>’salary’,
                     sec_relevant_cols_opt=>dbms_rls.ALL_ROWS);
```

Column-level VPD: Example

3. Bob accesses table Employees (with masking behavior). REMEMBER: Masking behavior returns all rows, but returns NULL values for the columns that contain sensitive information.

```
select e_id, name from Employee;
```

e_id	Name
1	Alice
2	Bob
3	Carl

```
select e_id, name, salary from Employee;
```

e_id	Name	Salary
1	Alice	
2	Bob	60
3	Carl	

Multiple Policies

- It is possible to associate multiple policies with a database object.
  - The policies are enforced with AND syntax.
  - For example, suppose table T is associated with {P1, P2, P3}.
  - When T is accessed by query Q = select A from T where C.
  - Q' = select A from T where C $\wedge$ (c1 $\wedge$ c2 $\wedge$ c3).

- Different from Stonebraker's approach
  - The policies are enforced with OR syntax.
  - Q' = select A from T where C $\wedge$ (c1 $\vee$ c2 $\vee$ c3).

VPD Related Privileges

- Who can create VPD policies? That is, what privileges are needed to create a VPD policy on a database object?
  - <u>EXECUTE on the DBMS_RLS package</u> to attach a policy to an object
    - the package includes add_policy, drop_policy, enable_policy, and so on.
  - CREATE PROCEDURE to create a policy function
    - Not absolutely necessary as you can use somebody else's policy functions.
    - Does not need to have any privilege on the policy functions.
  - Does not require any object privilege on the target objects unless you are defining the policy function (explained later).

- Who can create application contexts?
  - <u>CREATE ANY CONTEXT</u> (there is no CREATE CONTEXT)
  - CREATE PROCEDURE
  - EXECUTE on the DBMS_SESSION package
  - Privileges on the objects that the setup functions access.

- Two classes of users are exempt from VPD policies.
  - SYS user is exempt by default.
  - Users with the EXEMPT ACCESS POLICY system privilege.

AC Based-on DB Content

- It is possible to define VPD policy functions without using the application context. Instead, we can directly query the database content from the policy functions.

- Alice: Employees(e_id number(2), name varchar2(10), salary number(2));
- Bob: Values(p_id number(2), value number(2));

- Users can access the record of any employee whose salary is less than the maximum value in Values.

AC Based-on DB Content

1. Create a policy function

```
create or replace function Policy_func (object_schema varchar2, object_name
   varchar2)
return varchar2
as
  cond varchar2(100);
  mxv number;
begin
  select max(value) into mxv from Bob.Values;
  cond := 'salary < ' || mxv;
  return (cond);
end Policy_func ;
```

2. Attach the function to Employee

```
execute dbms_rls.add_policy('alice', 'employees', ‘policy', ‘alice', ‘Policy_func',
   'select');
```

Discussion

- VPD provides a very powerful access control.
- It is difficult, if not impossible, to verify whether or not a particular user has access to a particular data item in a particular table in a particular state.
  - Such verification requires checking all policy functions.
  - As policy functions are too “flexible”, it is computationally impossible to analyze them.

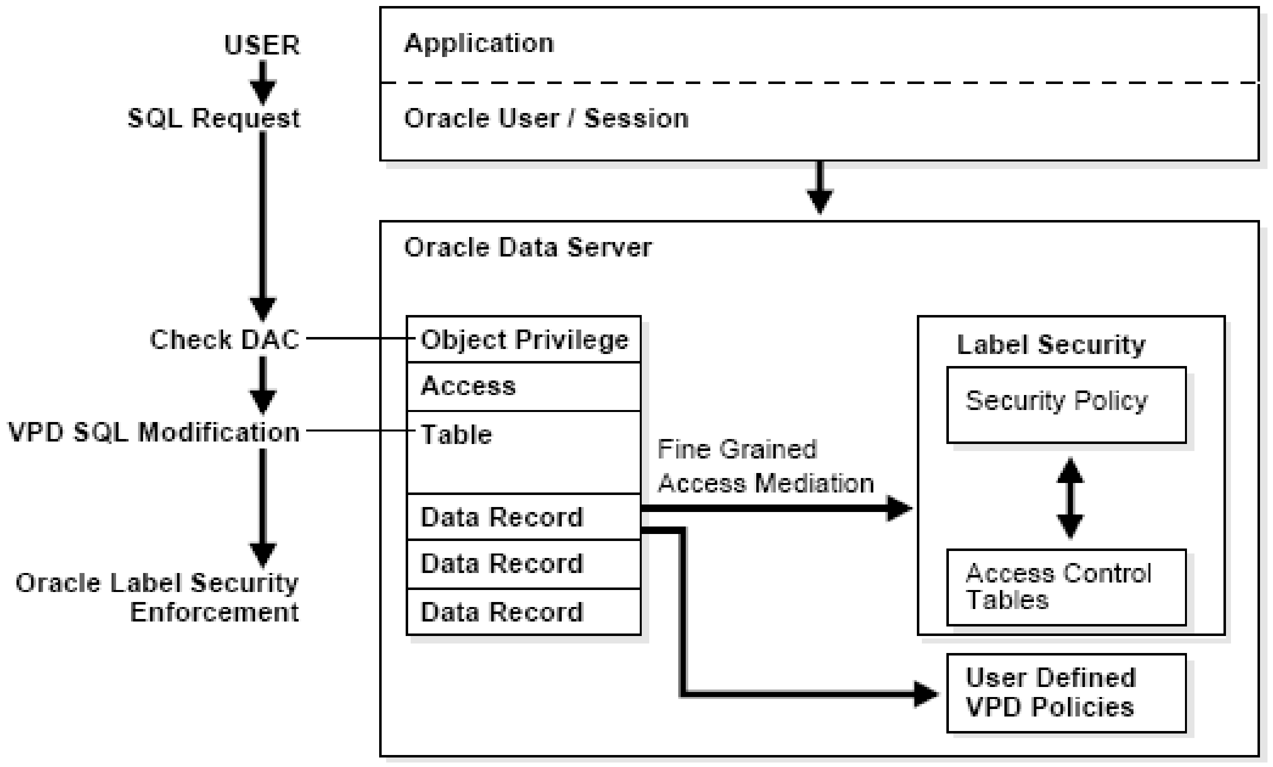
Case Study of Labelled Databases
Oracle Label Security

Oracle label essential concepts

- Oracle Label Security enables row-level access control, based on the virtual private database technology of Oracle9i Enterprise Edition
- It controls access to the contents of a row by comparing that row's label with a user's label and privileges
- Administrators can add selective row-restrictive policies to existing databases
- Developers can add label-based access control to their Oracle9i applications

Oracle label architecture

Figure 1–2 Oracle Label Security Architecture



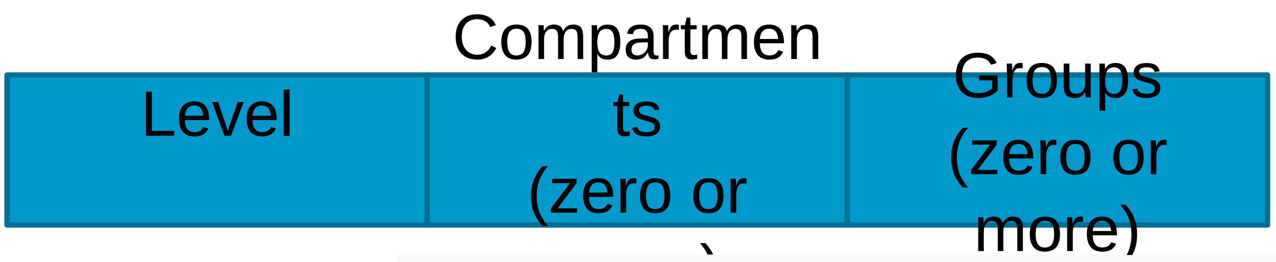
Label policy features

Oracle label controls the access to data by using 3 factors:

1. The label of the data row to which access is requested
2. The label of the user session requesting access
3. The policy privileges for that user session

Data Labels

- Every label contains three components:
  - a single level (sensitivity) ranking
  - zero or more horizontal compartments or categories
  - zero or more hierarchical group statements.



Example:
Confidential (10)
Highly Confidential (20)
Sensitive (30)

The more sensitive the information, the higher its level.
The less sensitive the information, the lower its level.

(Note: labels have a character form and a numeric form)

Data Labels *compartments*

- Compartments identify areas that describe the sensitivity of the labeled data, providing a finer level of granularity within a level.
- The compartment component is not hierarchical
- Example of departments:
  - Financial (it has Sensitive and Highly Confidential data)
  - Chemical (it has Sensitive data)
  - Operation (it has Sensitive, Highly confidential and Confidential data).

Single-level ranking	Compartmen ts (zero or more)	Groups (zero or more)

<u>Example:</u>
Confidential (10)
Highly Confidential (20)
Sensitive (30)

<u>Example:</u>
Financial
Chemical
Operation

Data Labels - compartments

<u>Levels:</u>

Sensitive	**Financial**	**Chemical**	**Operation**
HC	**Financial**		**Operation**
Confidential			**Operation**

Data Labels - compartments

If compartments are specified, then a user whose level would normally permit access to a row's data will nevertheless be prevented from such access unless the user's label also contains all the compartments appearing in that row's label.

Data Labels - groups

- The group component is hierarchical and is used to reflect ownership
- EXAMPLE: suppose one has two groups of users, Finance and Engineering. Users with the label Finance cannot access to data labeled Engineering (and vice versa), because they are "at the same level"
- Suppose that one has a group Board of Directors (BoD). Users in this group must be allowed to access the data of both Finance and Engineering group.
- To this end, one can establish a group hierarchy, where BoD is the group "father" of Finance and Engineering groups

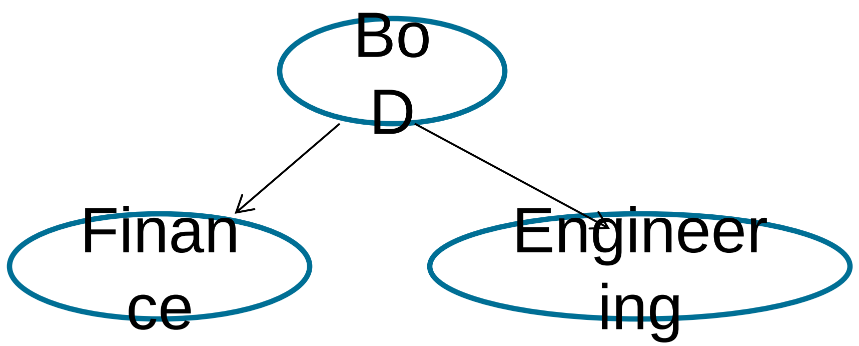
Data Labels – group example

Single-level ranking	Compartments (zero or more)	Groups (zero or more)

<u>Example:</u>
Confidential (10)
Highly Confidential (20)
Sensitive (30)

<u>Example:</u>
Financial
Chemical
Operation

<u>Example:</u>
BoD
Finance
Engineering



Data Labels

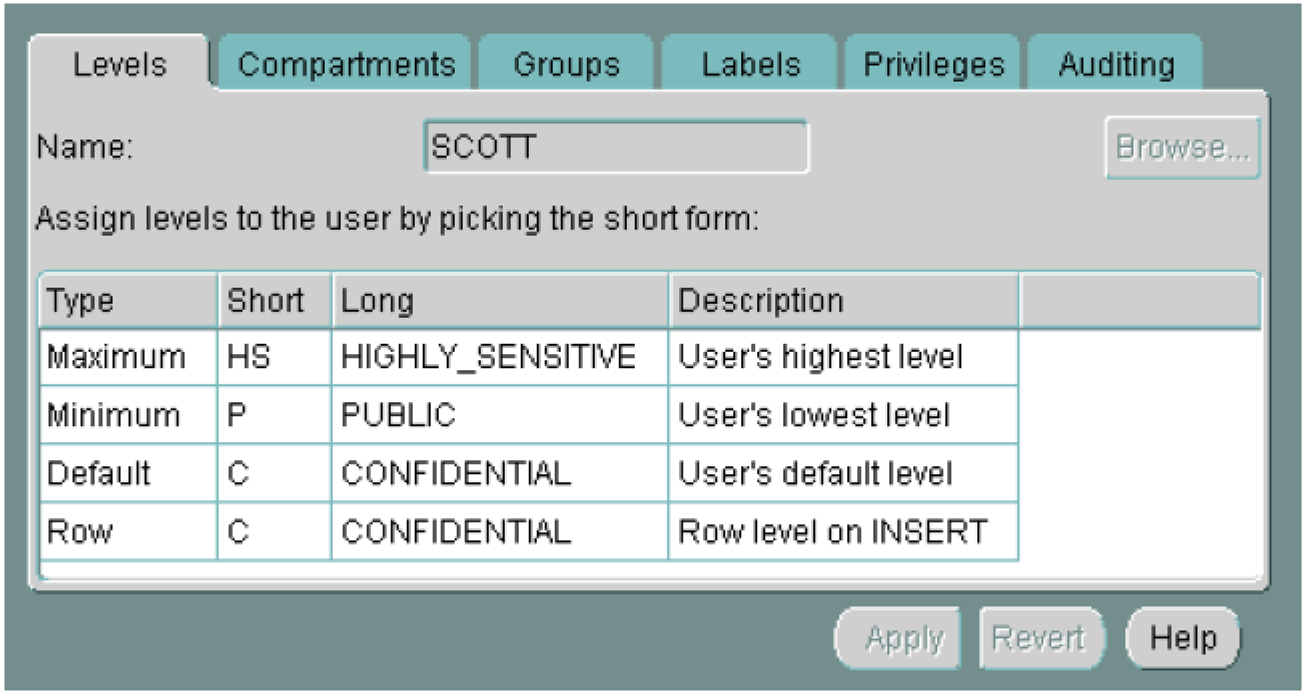
A label can be any one of the following four combinations of components:

- a single level component, with no groups or compartments, such as U::
- a level and a set of compartments with no groups, such as U:Alpha, Beta:
- a level and a set of groups with no compartments, such as U::FIN, ASIA
- a level with both compartments and groups, such as U:Beta,Psi:ASIA,FIN

User Labels

- A user label specifies that user's sensitivity level plus any compartments and groups that constrain the user's access to labeled data.
- Each user is assigned a range of levels, compartments, and groups, and each session can operate within that authorized range to access labeled data within that range.

User Labels and level authorizations



Name: SCOTT

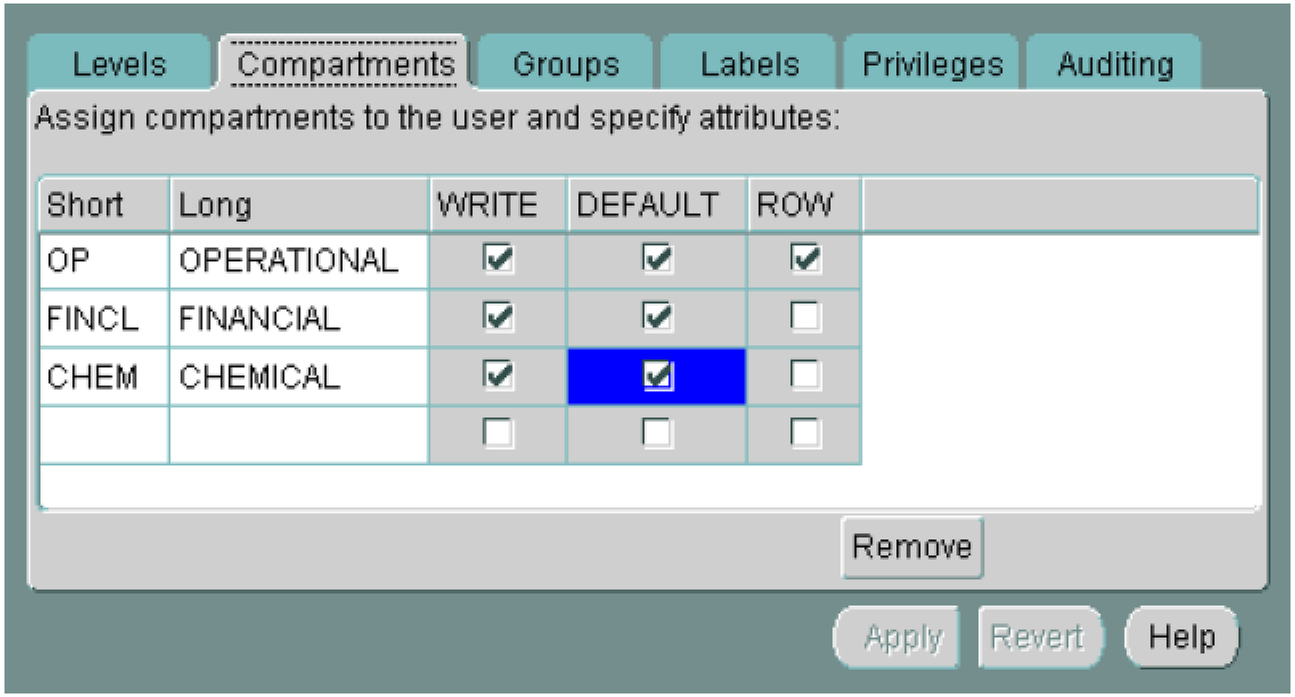
Assign levels to the user by picking the short form:

Type	Short	Long	Description
Maximum	HS	HIGHLY_SENSITIVE	User's highest level
Minimum	P	PUBLIC	User's lowest level
Default	C	CONFIDENTIAL	User's default level
Row	C	CONFIDENTIAL	Row level on INSERT

User Default Level: The level that is assumed by default when connecting to Oracle9*i*

User Default Row Level: The level that is used by default when inserting data into Oracle9*i*

User Labels and compartments

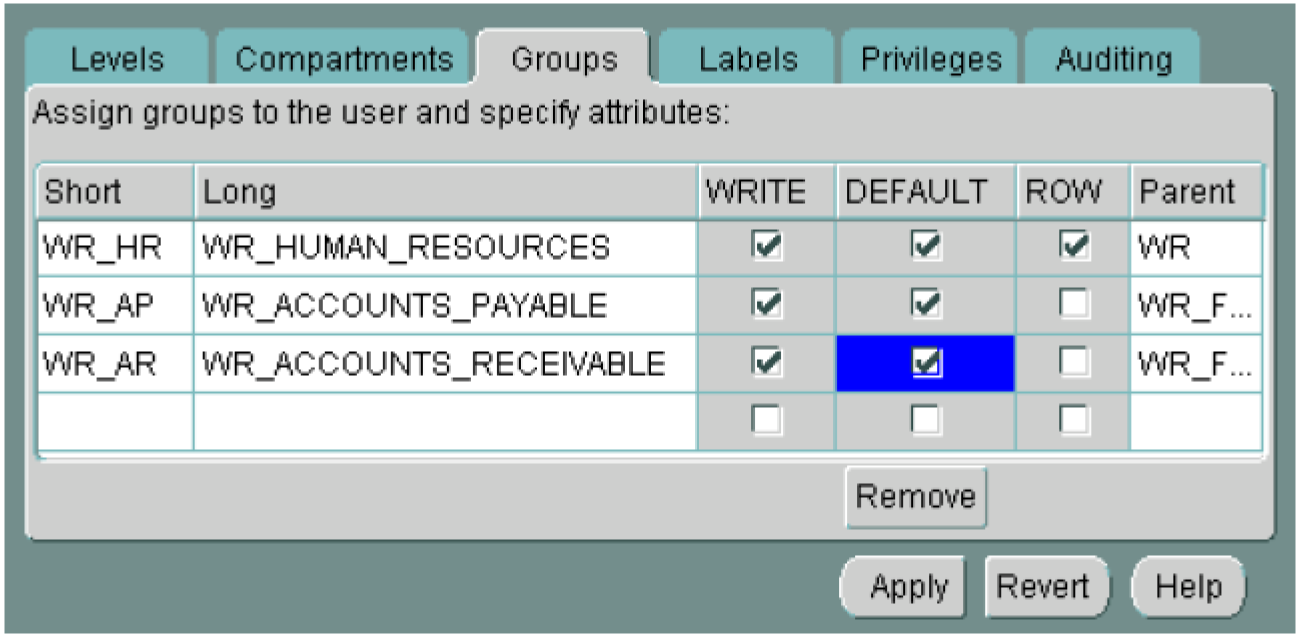


Assign compartments to the user and specify attributes:

Short	Long	WRITE	DEFAULT	ROW
OP	OPERATIONAL	✓	✓	✓
FINCL	FINANCIAL	✓	✓	☐
CHEM	CHEMICAL	✓	✓	☐
		☐	☐	☐

•The administrator specifies the list of compartments that a user can place in her session label.
•Write access must be explicitly given for each compartment
•The Row designation indicates whether the compartment should be used as part of the default row label for newly inserted data.
•A user cannot directly insert, update, or delete a row that contains a compartment that she does not have authorization to write.

User Labels and authorized groups



Assign groups to the user and specify attributes:

Short	Long	WRITE	DEFAULT	ROW	Parent
WR_HR	WR_HUMAN_RESOURCES	✓	✓	✓	WR
WR_AP	WR_ACCOUNTS_PAYABLE	✓	✓	☐	WR_F...
WR_AR	WR_ACCOUNTS_RECEIVABLE	✓	✓	☐	WR_F...
		☐	☐	☐	

•The administrator specifies the list of groups that a user can place in her session label.
•Write access must be explicitly given for each group listed.
•Row designation indicates whether the group should be used as part of the default row label for newly inserted data.

Session Labels

- The *session label is the particular combination of level, compartments, and groups at* which a user works at any given time.
- The user can change the session label to any combination of components for which he is authorized.
- When a user writes data without specifying its label, a *row label is assigned* automatically, using the user's session label.

How Data Labels and User Labels Work Together

- Each Oracle Label Security user can only access data within the range of his or her own label authorizations.
- Each user has:
  - Maximum and minimum levels
  - A set of authorized compartments
  - A set of authorized groups (and, implicitly, authorization for any subgroups)
  - For each compartment and group, a specification of read-only access, or read/write access
- Example:
  - if a user is assigned a maximum level of Highly Confidential, then the user potentially has access to Highly Confidential, and Confidential data. The user has no access to Sensitive data.

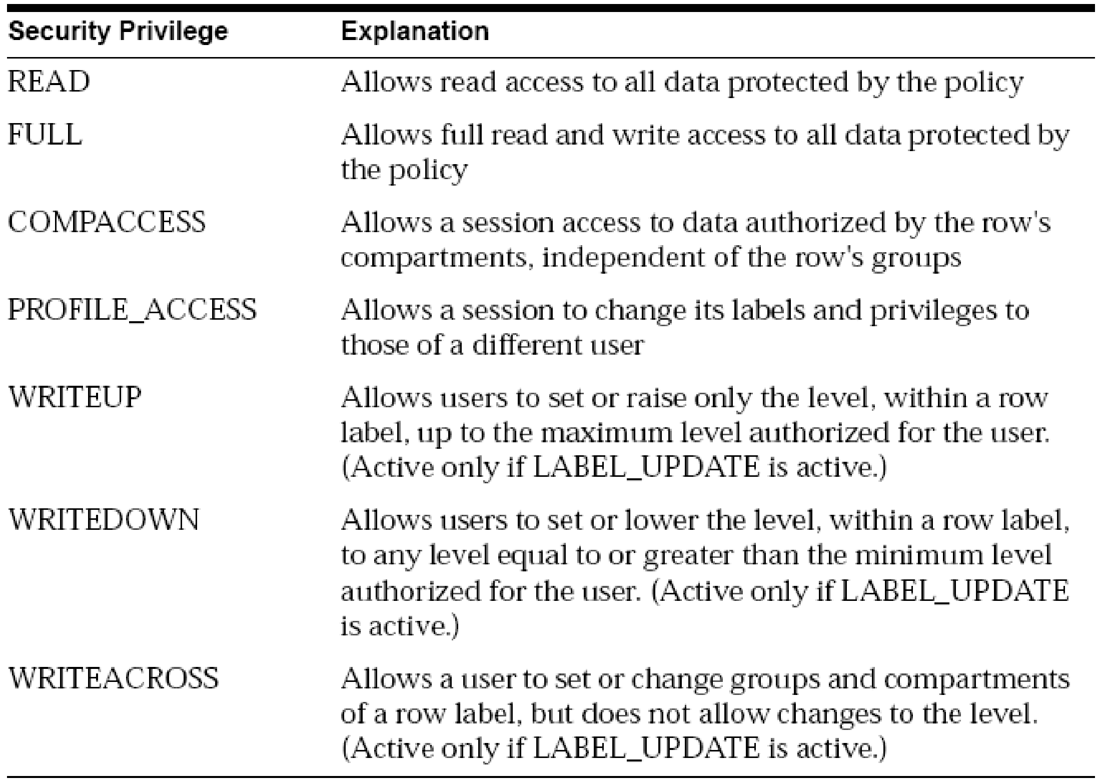
Policy Privileges

- The policy privileges enable a user or a stored program unit to bypass some aspects of the label-based access control policy
- The administrator can also authorize the user or program unit to perform specific actions, such as the ability of one user to assume the authorizations of a different user
- Privileges can be granted to program units, authorizing the procedure, rather than the user, to perform privileged operations

Privileges in Oracle Label Security Policies

Oracle Label Security supports special privileges that allow authorized users to *bypass certain parts of the policy.*

Table 3–3 Oracle Label Security Privileges



Security Privilege	Explanation
READ	Allows read access to all data protected by the policy
FULL	Allows full read and write access to all data protected by the policy
COMPACCESS	Allows a session access to data authorized by the row's compartments, independent of the row's groups
PROFILE_ACCESS	Allows a session to change its labels and privileges to those of a different user
WRITEUP	Allows users to set or raise only the level, within a row label, up to the maximum level authorized for the user. (Active only if LABEL_UPDATE is active.)
WRITEDOWN	Allows users to set or lower the level, within a row label, to any level equal to or greater than the minimum level authorized for the user. (Active only if LABEL_UPDATE is active.)
WRITEACROSS	Allows a user to set or change groups and compartments of a row label, but does not allow changes to the level. (Active only if LABEL_UPDATE is active.)

Privileges in Oracle Label Security Policies

- READ
- A user with READ privilege can read all data protected by the policy, regardless of his authorizations or session label. The user does not even need to have label authorizations.
- A user with READ privilege can *write to any* data rows for which he or she has write access, based on any label authorizations.
    - ➢ useful for system administrators who need to export data, but who should not be allowed to change data

- FULL
- The FULL privilege has the same effect and benefits as the READ privilege, with one difference: a user with FULL privilege can also *write to all the data.*

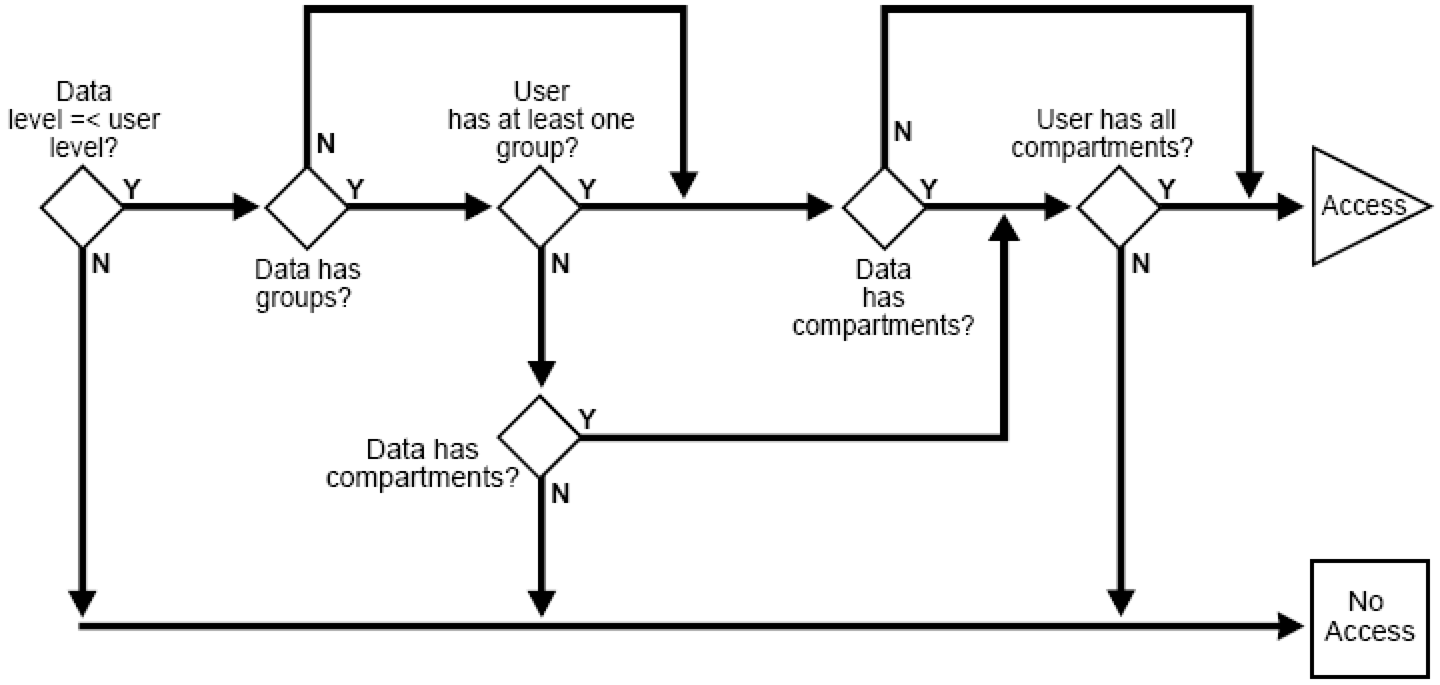
Privileges in Oracle Label Security Policies

COMPACCESS

- The COMPACCESS privilege allows a user to access data based on the row label's compartments, independent of the row label's groups.
- If a row label has no compartments, then access is determined by the group authorizations. However, when compartments do exist, and access to them is authorized, then the group authorization is bypassed.

Privileges in Oracle Label Security Policies

Figure 3–9 Label Evaluation Process for Read Access with COMPACCESS Privilege



Privileges in Oracle Label Security Policies

- PROFILE_ACCESS
- The PROFILE_ACCESS privilege allows a session to change its session labels and session privileges to those of a different user.
- This is a very powerful privilege, since the user can potentially become a user with FULL privileges.
- This privilege cannot be granted to a trusted stored program unit.

Privileges in Oracle Label Security Policies

- Once the label on a row has been set, Oracle Label Security privileges are required to modify the label. These privileges include WRITEUP, WRITEDOWN, and WRITEACROSS.
- WRITEUP
  - The WRITEUP privilege enables the user to raise the level of data within a row, without compromising the compartments or groups. The user can raise the level up to his or her maximum authorized level. He can raise the level above his current session level, but cannot change the compartments.

Privileges in Oracle Label Security Policies

- Once the label on a row has been set, Oracle Label Security privileges are required to modify the label. These privileges include WRITEUP, WRITEDOWN, and WRITEACROSS.
- WRITEDOWN
    - The WRITEDOWN privilege enables the user to lower the level of data within a row, without changing the compartments or groups. The user can lower the level to any level equal to or greater than his or her minimum authorized level.
- WRITEACROSS
    - The WRITEACROSS privilege allows the user to change the compartments and groups of data, without altering its sensitivity level.

Documentation

Oracle® Label Security Administrator’s Guide 10*g* *Release* 10*g* *Release 2 (10.2)* B14267-02 http://www.oracle.com/pls/db102/to_pdf?pathname=network.102%2Fb14267.pdf&remark=portal+%28Administration%29

Next Lecture

- Overview of Cryptography