

**Специальность 10.05.01 «Компьютерная безопасность»,
Специализация «Математические методы защиты информации»
Уровень высшего образования – специалитет**

Дисциплина: Основы построения защищенных баз данных.

Лабораторная работа №5.

**Администрирование Oracle. Разграничение доступа на уровне строк.
VIRTUAL PRIVATE DATABASE или VPD.**

1. Учебные цели:

- Отработать вопросы управления экземпляром Oracle в части реализации стратегии «виртуальной частной базы данных».
- Освоить приемы работы с библиотекой DBMS_RLS для настройки разграничения доступа на уровне строк.

2. Требования к результатам обучения основной образовательной программы, достигаемые при проведении лабораторной работы:

- Уметь использовать возможности современных систем для решения задач администрирования и защиты баз данных.
- Владеть средствами приложений СУБД Oracle в части реализации стратегии «виртуальной частной базы данных».

3. Перечень материально-технического обеспечения

ПЭВМ с проигрывателем виртуальных машин, виртуальная машина с установленной СУБД Oracle.

4. Краткие теоретические сведения и задания на исследование. Задания выделены рамками и синим шрифтом. Результаты лабораторной работы представляются в виде файла, содержащего копии экрана, показывающие этапы выполнения заданий.

Понятие *виртуальная частная база данных* (virtual private database, VPD), появилось в версии Oracle 8.1.

Оно означает возможность ограничить конкретным сеансом доступное множество строк в таблице, чтобы каждый сеанс, обращаясь формально к одной и той же таблице, имел доступ в ней («видел») только положенные строки.

Сам термин VPD рекламный, его технический эквивалент, фигурирующий наравне в документации – *детальный контроль доступа* [к строкам] (fine grained access control, FGAC).

Пользоваться VPD можно напрямую, через пакет SYS.DBMS_RLS, а так же неявно и не догадываясь об этом, поскольку это средство положено в основу другого, label security, реализующего известную модель *мандатного доступа* применительно к строкам таблицы из приложения.

Виртуальная частная база данных (Virtual Private Database – VPD) позволяет осуществлять контроль доступа на уровне строк, дополняя возможности, предоставляемые ролями и представлениями. VPD может гарантировать, что при доступе через Интернет клиенты, выполняющие банковские операции в оперативном режиме, видят только свои счета. Организации, предоставляющие Web-хостинг, могут сопровождать данные многих компаний в одной и той же БД Oracle, разрешая каждой компании видеть только свои данные.

Вместо того, чтобы создавать систему безопасности в каждом приложении, имеющем доступ к данным, ее можно выстроить один раз на сервере БД. Безопасность в этом случае

выше, так как она гарантируется базой данных и не зависит от того, каким образом пользователь осуществляет доступ к данным. Она больше не может быть нарушена при применении пользователем инструментов выполнения произвольных запросов (ad hoc query) или новых инструментов создания отчетов. VPD – ключевая технология, позволяющая организациям строить Web-приложения для хостинга. Многие приложения Oracle, включая Oracle SalesOnline.com и Oracle Portal, сами используют VPD для обеспечения разделения данных при реализации хостинга.

VPD включается путем установления связи одной или нескольких политик безопасности с таблицами и представлениями. Прямой или не прямой доступ к таблице с соответствующей прикрепленной к ней политикой безопасности заставляет базу данных обратиться к функции, реализующей эту политику. Такая функция возвращает условие доступа в виде предиката (предложение WHERE), которое добавляется базой данных в пользовательскую команду SQL. В результате динамически модифицируется доступ пользователя к данным.

Политики VPD могут быть статическими, контекстно-зависимыми или динамическими. Статические и контекстно-зависимые политики, новые для Oracle Database 10g, могут значительно повысить производительность выполнения запроса, потому что данные запроса и политик кэшируются. До Oracle Database 10g все политики были динамическими; другими словами, функция policy выполнялась каждый раз, когда происходил разбор SQL-оператора, содержащего целевую таблицу VPD. Статические политики разбираются один раз при входе в систему и остаются в кэше на протяжении всего сеанса, независимо от контекста приложения.

Определение и использование терминов

В контекстно-зависимых политиках функция политики запоминается при синтаксическом анализе оператора на время, пока контекст приложения не изменяется. Например, политика, которая обеспечивает бизнес-правило, согласно которому «сотрудники видят только собственную историю зарплат, а менеджеры – все зарплаты своих сотрудников». Если сотрудник, выполняющий запрос, не изменился, функция политики не вызывается снова, таким образом, снижается нагрузка на БД из-за политики VPD.

Дифференцированный контроль доступа (fine-grained access control – FGAC) позволяет использовать функции для реализации политик безопасности и связывать эти политики безопасности с таблицами, представлениями и синонимами.

Контекст приложения (application context) – функциональная возможность, позволяющая разработчикам приложений определять, устанавливать и получать доступ к атрибутам приложения, а затем использовать эти атрибуты для предоставления значений предикатам политик дифференцированного контроля доступа. Используя отдельно контекст приложения, разработчики могут определять, устанавливать и получать доступ к атрибутам приложения. При этом контекст приложения выступает в качестве кэша данных. Такое использование устраняет повторяющиеся накладные расходы, вызываемые запросами к базе данных, которые производятся каждый раз, когда приложению требуются атрибуты.

Атрибуты приложения (application attributes) определяются внутри контекста приложения и используются политиками дифференцированного контроля доступа.

Общее описание виртуальной частной базы данных

Виртуальная частная база данных (VPD) объединяет возможности сервера по дифференцированному контролю доступа и контекст приложения в базе данных Oracle. VPD позволяет строить приложения, использующие политики безопасности на уровне строк. Когда пользователь прямо или косвенно обращается к таблице, представлению или синониму, связанному с политикой безопасности VPD, сервер динамически модифицирует пользовательскую команду SQL (добавляет еще одно условие выбора строк). Такое изменение производится на основе предложения WHERE, возвращаемого функцией, реализующей

политику безопасности. Используя условие, которое может быть возвращено функцией, база данных динамически модифицирует команду, выполняя это прозрачно для пользователя.

В качестве примера реализации контроля доступа на уровне строк можно привести корзину покупателя в Web-магазине, в которой видны только товары конкретного покупателя.

Контекст приложения – функциональная возможность, позволяющая разработчикам приложений определять и задавать значения атрибутов, получать доступ к атрибутам приложения, а затем использовать их для предоставления значений предикатам политик дифференцированного контроля доступа.

Примечание. Дифференцированный контроль доступа и контекст приложения могут быть реализованы как отдельные автономные опции. VPD основывается на совместном применении этих опций.

Контекст приложения создается с использованием команды `create context` из пакета `DBMS_RLS`. Функция, используемая для возврата предикатов для применения политики, создается как любая другая функция, за исключением того, что она имеет два обязательных параметра и возвращает `VARCHAR2`.

Контекст приложения.

Используя команду `create context`, можно определить имя контекста приложения, имя библиотеки, в которой будут помещаться функции и процедуры, и имя функции, используемые для определения этого контекста в пользовательской сессии.

Пример:

```
create context hr_security using vpd.emp_access;
create or replace package emp_access as
procedure set_security_parameters;
end;
```

В этом примере имя контекста – `HR_SECURITY`. Библиотека, которая используется для настройки характеристик или атрибутов для пользователя во время сеанса, называется `EMP_ACCESS`.

Процедура `SET_SECURITY_PARAMETERS` будет вызвана при входе в систему. Никакие другие процедуры не могут изменить атрибуты сеанса, потому что контекст `HR_SECURITY` связан только с `EMP_ACCESS`. Это обеспечивает безопасный контекст приложения, который не может быть изменен пользователем или любым другим процессом после подключения к базе данных.

В типовой библиотеке, используемой для реализации контекста приложения и получения информации о сессии пользователя, используется встроенный контекст `USERENV`.

Параметры `USERENV`:

<code>CURRENT_SCHEMA</code>	Схема по умолчанию для текущей сессии
<code>DB_NAME</code>	Имя БД, определенное параметром инициализации <code>DB_NAME</code>
<code>HOST</code>	Имя хост-машины, с которой подключен пользователь
<code>IP_ADDRESS</code>	IP-адрес, с которого подключен пользователь
<code>OS_USER</code>	имя операционной системы, с которой подключен пользователь
<code>SESSION_USER</code>	имя пользователя, аутентифицированного БД

Например, следующие вызовы `SYS_CONTEXT` будут получать имя пользователя и `IP_ADDRESS` сессии

```
declare
username varchar2(30);
ip_addr varchar2(30);
begin
    username := SYS_CONTEXT('USERENV','SESSION_USER');
    ip_addr := SYS_CONTEXT('USERENV','IP_ADDRESS');
end;
```

Точно так же функцию SYS_CONTEXT можно использовать в запросе SQL:

```
select SYS_CONTEXT('USERENV','SESSION_USER') username from dual;
```

Используя некоторую комбинацию контекста USERENV и информации авторизации в базе данных, мы используем DBMS_SESSION.SET_CONTEXT, чтобы присвоить значения параметрам в приложении.

Контекст, который мы создадим:

```
dbms_session.set_context('HR_SECURITY','SEC_LEVEL','HIGH');
```

В этом примере переменная контекста приложения SEC_LEVEL установлена в HIGH в контексте HR_SECURITY. Значение может быть назначено на основе ряда условий, включая таблицу, на основе которой назначаются уровни безопасности на основе идентификатора пользователя.

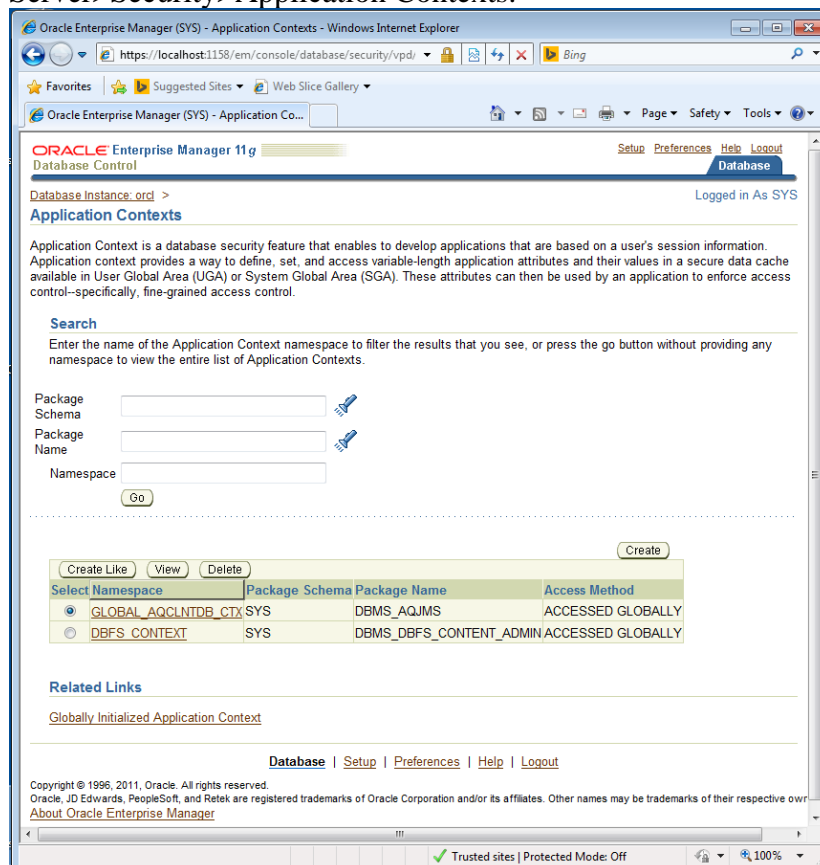
Чтобы убедиться, что переменные контекста установлены для каждого сеанса, мы можем использовать триггер входа в систему для вызова процедуры, связанной с контекстом. Как упоминалось ранее, переменные в контексте могут быть установлены или изменены только в назначенном пакете.

Вот пример триггера входа в систему, который вызывает процедуру для настройки контекста:

```
create or replace trigger vpd.set_security_parameters
after logon on database
begin
    vpd.emp_access.set_security_parameters;
end;
```

В этом примере процедура SET_SECURITY_PARAMETERS будет делать необходимые вызовы DBMS_SESSION.SET_CONTEXT.

В Oracle Enterprise Manager настройка контекстов возможна в разделе Server>Security>Application Contexts.




Database Instance: orcl > Application Contexts > Logged in As SYS

Create Application Contexts

Specify a name for the Application Context namespace, the associated PL/SQL package that sets/resets the attributes of context attributes and the type of the Application Context. Show SQL Cancel OK

* Namespace

* Package Name 
Example: Package Schema Name Package Name

Type

Choose the type of Application Context below. Depending upon where the context data is stored and how updates can be done there are two types of Application Contexts- Secure session based (or Local) Application Context (where context data is stored in the database user session (UGA) and can only be set by the PL/SQL package in that command) and Non session based (or Global) Application Context (where context data is stored in the SGA (System Global Area)).

- ☒ Secure session based - Accessed Locally
Context attributes can be initialized from a user's session information but can only be set by the package associated with the application context namespace.
- ☐ Secure session based - Initialized Externally
External resources like an OCI interface, a job queue process, etc. provide values for context attributes initialization but cannot bypass the designated PL/SQL packages to set the context.
- ☐ Secure session based - Initialized Globally
Based upon the user's identity context attributes are initialized from values in a centralized location (Oracle Internet Directory) which stores list of users and their application context.
- ☐ Non session based - Accessed Globally
Used for multi tiered environment users accessing databases through connection pools, this application context can be shared among trusted sessions.

Driving Context

Optionally provide driving context attributes below. A driving application context implements Partitioned Fine Grained Security by securely determining policy groups to be enabled for the user session, by setting the policy groups name into an application context.

Each of the following Schema Objects are associated multiple policy groups. At execution time, the server retrieves the name of the policy group to be activated from the attribute value of this context. When driving context is not set, policies in all policy groups will be enforced.

Add

Select Attribute	Schema	Object	Object Type
No Driving Context Found			

Реализация политики безопасности

После того, как будет создана инфраструктура для настройки политики безопасности, следующим шагом будет определение функций, используемых для генерации предиката, который будет прикреплен к каждому оператору выбора (или команды DML) из защищенных таблиц. Функция, используемая для реализации предиката, имеет два аргумента: владелец защищаемого объекта и имя объекта в схеме владельца. Одна функция может обрабатывать генерацию предикатов только для одного типа операций, например SELECT, или может быть применима ко всем командам DML, в зависимости от того, как эта функция связана с защищенной таблицей.

В Oracle Enterprise Manager настройка VDP возможна в разделе Server>Security>Virtual Private Database Policies.

Database Instance: orcl > Logged in As SYS

Virtual Private Database Policies


Virtual Private Database Policies enable to build applications that enforce row-level security policies at the object (table, view or synonym) level by attaching security policies to objects and dynamically appending predicates (WHERE clauses) to SQL statements that query them.


Policy Advanced

Security Policies can be applied to Tables, Views or Synonyms (Synonyms to tables and views only) to provide row level security also known as Fine Grained Access Control (FGAC).

Search

Specify an object name to list the policies associated with it. Optionally provide a policy name to filter the data that is displayed in the result.

Schema Name 

Object Name 

Policy Name

Go

Selection Mode Single

Create

Select	Policy	Object Name	Schema	Object Type	Policy Group	Enabled
<input checked="" type="radio"/>	ForeignKey745_TAB\$xd_dp	ForeignKey745_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	ForeignKey745_TAB\$xd_up	ForeignKey745_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	ForeignKey745_TAB\$xd_sp	ForeignKey745_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	DefaultStyle757_TAB\$xd_dp	DefaultStyle757_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	DefaultStyle757_TAB\$xd_up	DefaultStyle757_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	DefaultStyle757_TAB\$xd_sp	DefaultStyle757_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	ViewPoint3d762_TAB\$xd_dp	ViewPoint3d762_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	GridFile735_TAB\$xd_dp	GridFile735_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	GridFile735_TAB\$xd_up	GridFile735_TAB	MDSYS	TABLE	SYS_DEFAULT	✓
<input type="radio"/>	GridFile735_TAB\$xd_sp	GridFile735_TAB	MDSYS	TABLE	SYS_DEFAULT	✓

Create Like View Enable/Disable Delete Previous 1-10 of 36 Next 10 >

Database Instance: orcl > Virtual Private Database Policies > Logged in As SYS

Create Policy

General

*Policy Name

*Object Name

Policy Type: STATIC

☒ Enabled
Check this box to enable the policy after creation

Policy Function

Specify a policy function to return a predicate for filtering the data. The function can also reside in a package.

* Policy Function
Example: Schema.Policy Function

☐ Long Predicate
Check this box to allow policy function to return a predicate with a length up to 32k. Default is 4k.

Enforcement

Select operation types to which the policy applies. It can be any combination of SELECT, INSERT, UPDATE, INDEX and DELETE.

☒ INSERT
☒ UPDATE
☒ DELETE
☒ SELECT
☐ INDEX
☐ Insert/Update Check (CHECK OPTION)
Check this to allow changes to the row if they are still visible to the user after update. Can be specified only if INSERT or UPDATE options are specified.

Overview Of Policy Types

STATIC - For this type of policies the policy function is executed once and then the predicate is cached in the SGA (System Global Area) for fast performance. Applies to only one object.

SHARED_STATIC - Same as STATIC except that the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared across multiple objects.

CONTEXT_SENSITIVE - For this type of policies, the server re-evaluates the policy function at statement execution time if it detects context changes. Server always executes the policy function on statement parsing and does not cache the value returned by the function. Applies to only one object.

SHARED_CONTEXT_SENSITIVE - For this type of policy the policy function is executed when the object to which it is associated is referenced for the first time in a database session. Predicates (WHERE clauses) are then cached in the private session memory UGA (User Global Area). Server first looks for a cached predicate and does not execute the policy function until session private application context changes occur. Shared across multiple objects.

DYNAMIC - For this type of policy server assumes the predicate may be affected by any system or session environment at any time. So the policy function is executed every time a policy protected database object is accessed. Applies to only one object.

Security Relevant Columns

Specify security relevant columns if the policy being created is meant to apply Column-Level Virtual Private Database (VPD).

☐ Enable Column Masking Behavior
Check this box to enable Column Masking Behavior of Column-Level Virtual Private Database.

Select Name

No items Found

Overview of Column Level VPD and Column Masking

Column-Level Virtual Private Database (VPD) enforces row level security policies only when a particular column or columns are accessed in the user's query. These columns containing sensitive information are marked as Security Relevant Columns. Column-level VPD applies to tables and views, but not to synonyms.

It can be configured to produce two distinct behaviors as follows:

- Default Behavior which restricts the number of rows returned for a query that references columns containing sensitive information.
- Column-Masking Behavior of Column-Level VPD displays all rows including those that reference sensitive columns unlike the default behavior of Column-Level VPD which restricts the number of rows returned. However, the sensitive columns display as NULL values.

[Database](#) |
 [Setup](#) |
 [Preferences](#) |
 [Help](#) |
 [Logout](#)

Copyright © 1996, 2011, Oracle. All rights reserved.
 Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
[About Oracle Enterprise Manager](#)

В следующем примере показано тело пакета, содержащее две функции: одна будет использоваться для управления доступом из операторов SELECT, а другая – для любых других операторов DML.

```
create or replace package body get_predicates is

function    emp_select_restrict(owner_name    varchar2,    object_name
varchar2)
return varchar2 is
ret_predicate varchar2(1000); -- part of WHERE clause
begin
    -- only allow certain employees to see rows in the table
    -- . . . check context variables and build predicate
    return ret_predicate;
end emp_select_restrict;
```

```

function      emp_dml_restrict(owner_name      varchar2,      object_name
varchar2)
return varchar2 is
ret_predicate varchar2(1000); -- part of WHERE clause
begin
    -- only allow certain employees to make changes to the table
    -- . . . check context variables and build predicate
    return ret_predicate;
end emp_dml_restrict;
end; -- package body

```

Каждая функция возвращает строку, содержащую выражение, которое добавляется в предложение where для оператора SELECT или команды DML. Пользователь или приложение никогда не увидят значение этого предложения WHERE, оно автоматически добавляется в команду во время разбора.

Разработчик должен убедиться, что функции всегда возвращают правильное выражение. В противном случае, любой доступ к защищенной таблице всегда будет неудачным, как в следующем примере:

```

SQL> select * from hr.employees;
select * from hr.employees
*
ERROR at line 1:
ORA-28113: policy predicate has error

```

Использование DBMS_RLS

Встроенный пакет DBMS_RLS содержит ряд процедур, которые DBA использует для поддержки политик безопасности, связанных с таблицами, представлениями и синонимами. Любой пользователь, которому нужно создавать или администрировать политики, должен иметь привилегии EXECUTE для пакета SYS.DBMS_RLS.

Процедуры пакета DBMS_RLS:

ADD_POLICY	Добавляет FGAC (дифференцированный контроль доступа) к объекту
DROP_POLICY	Удаляет политику FGAC объекта
REFRESH_POLICY	Повторно анализирует все кэшированные операторы, связанные с политикой
ENABLE_POLICY	Включает или отключает политику FGAC
CREATE_POLICY_GROUP	Создает группу политик
ADD_GROUPED_POLICY	Добавляет политику в группу политик
ADD_POLICY_CONTEXT	Добавляет контекст для текущего приложения
DELETE_POLICY_GROUP	Удаляет группу политик
DROP_GROUPED_POLICY	Удаляет политику из группы политик
DROP_POLICY_CONTEXT	Удаляет контекст для текущего приложения
ENABLE_GROUPED_POLICY	Включает групповую политику
DISABLE_GROUPED_POLICY	Отключает групповую политику
REFRESH_GROUPED_POLICY	Повторно анализирует все кэшированные операторы, связанные с групповой политикой

Далее будем использовать процедуры ADD_POLICY and DROP_POLICY. Синтаксис ADD_POLICY:

```

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 in Boolean false,
sec_relevant_cols    IN varchar2,
sec_relevant_cols_opt IN binary_integer null
);

```

Обратите внимание, что некоторые параметры имеют значения по умолчанию, и что реже используемые параметры находятся в конце списка аргументов. Это делает синтаксис для любого конкретного вызова DBMS_RLS.ADD_POLICY проще для написания и понимания в подавляющем большинстве случаев.

Параметры DBMS_RLS.ADD_POLICY:

object_schema	Схема, содержащая таблицу, представление или синоним защищаемого политикой. Если значение не определено, используется схема пользователя, который вызывает процедуру.
object_name	Имя таблицы, представления или синонима защищаемого политикой.
policy_name	Имя политики, которая будет добавлена к этому объекту, должно быть уникальным для каждого защищаемого объекта.
function_schema	Схема, которой принадлежит функция политики; если это значение не определено, используется схема пользователя, вызывающего процедуру.
policy_function	Имя функции, генерирующей предикат. Если функция является частью пакета, здесь также должно быть указано имя пакета (библиотеки).
statement_types	Типы операторов, к которым применяется политика. Допустимые значения, разделенные запятыми, могут быть любой комбинацией SELECT, INSERT, UPDATE, DELETE и INDEX. По умолчанию применяются все типы, кроме INDEX.
update_check	Для типов INSERT или UPDATE этот параметр является необязательным и по умолчанию имеет значение FALSE. Если его значение TRUE, политика также проверяется на наличие операторов INSERT или UPDATE, когда идет разбор SELECT или DELETE.
enable	Этот параметр по умолчанию имеет значение TRUE и указывает, включена ли политика.
static_policy	Если этот параметр равен TRUE, политика создает одинаковую строку предиката для любого, кто обращается к объекту, за исключением пользователя SYS или любого пользователя с привилегией EXEMPT ACCESS POLICY. По умолчанию FALSE.
policy_type	Переопределяет static_policy, если это значение не равно NULL. Допустимые значения: STATIC, SHARED_STATIC, CONTEXT_SENSITIVE, SHARED_CONTEXT_SENSITIVE и DYNAMIC.
long_predicate	Этот параметр по умолчанию имеет значение FALSE. Если это значение – TRUE, строка предиката может иметь длину до 32 КБ. В противном случае ограничение составляет 4000 байт.

sec_relevant_cols	Применяет VPD на уровне столбцов. Применяется только к таблицам и представлениям. Защищенные столбцы указываются в списке с разделителями-запятыми или пробелами. Политика применяется, только если указанные столбцы находятся в запросе или в выражении DML. По умолчанию применяется ко всем столбцам.
sec_relevant_cols_opt	Значением по умолчанию для этого параметра является NULL. В противном случае необходимо указать DBMS_RLS.ALL_ROWS, что потребует отображения строк в фильтрованном запросе VPD со значениями NULL, возвращаемыми для защищаемых столбцов.

Использование параметра sec_relevant_cols удобно, когда вы не возражаете, что пользователи видят часть строк, но не видят информацию столбцов, содержащих конфиденциальную информацию (эти поля отображаются пустыми).

В нашем примере далее будем использовать первую политику безопасности, определим условия для фильтрации конфиденциальных данных для большинства сотрудников компании.

В следующем примере мы применяем политику с именем EMP_SELECT_RESTRICT к таблице HR.EMPLOYEES.

Схема (пользователь) VPD – владелец функций политики
get_predicates.emp_select_restrict.

Политика явно применяется к операторам SELECT в таблице, включен UPDATE_CHECK: команды обновления или удаления также будут проверяться при обновлении или вставке строк в таблицу.

```
dbms_ols.add_policy (
    object_schema => 'HR',
    object_name => 'EMPLOYEES',
    policy_name => 'EMP_SELECT_RESTRICT',
    function_schema => 'VPD',
    policy_function => 'get_predicates.emp_select_restrict',
    statement_types => 'SELECT',
    update_check => TRUE,
    enable => TRUE
);
```

Поскольку мы не установили static_policy, по умолчанию используется значение FALSE, что означает, что политика является динамической и проверяется каждый раз при анализе оператора select. Это единственное доступное поведение БД Oracle 10g и выше.

Использование процедуры ENABLE_POLICY – это простой способ временно отключить политику без необходимости перепривязывать политику к таблице позже:

```
begin
dbms_ols.enable_policy(
    object_schema => 'HR',
    object_name => 'EMPLOYEES',
    policy_name => 'EMP_SELECT_RESTRICT',
    enable => FALSE
);
end;
```

Если для одного объекта созданы несколько политик, между предикатами добавляется условие AND. Если нужно иметь условие ИЛИ между предикатами для нескольких политик, политика, скорее всего, должна быть пересмотрена. Логика для каждой политики должна быть объединена в одной политике с условием ИЛИ между всеми частями предиката.

Особенности виртуальной частной базы данных

Конфиденциальность может обеспечиваться и только для случая, когда команды обращаются или ссылаются на столбцы, существенные с точки зрения безопасности. В этом случае в процедуре DBMS_RLS.ADD_POLICY задается параметр SEC_RELEVANT_COLS (секретный столбец). Тогда политика будет применяться только в случае, если есть обращение к «секретному столбцу».

Если параметр SEC_RELEVANT_COLS не задан, то политика применяется во всех случаях обращения к объекту (таблице, представлению).

Гибкость всех типов реализуемых политик обеспечивается настройкой, проводимой для того, чтобы VPD соответствовала индивидуальным требованиям заказчика после развертывания его приложения. Такую настройку можно проводить путем применения неизменного предиката в статической политике или же динамически изменяемого предиката в нестатической политике.

Разделяемые политики позволяют применять единственную политику VPD ко многим объектам. Такая возможность снижает затраты администрирования.

Начиная с Oracle 10 можно использовать политики безопасности для операций сопровождения индексов, производимых с помощью команд DDL CREATE INDEX и ALTER INDEX. Это важно, так как пользователям требуется полный доступ к таблице для создания индекса. Поэтому пользователь, обладающий привилегиями сопровождения индекса, может видеть все строки данных, хотя не имеет при этом полного доступа к таблице при выполнении обычного запроса.

Аргумент LONG_PREDICATE процедуры DBMS_RLS.ADD_POLICY имеет по умолчанию значение FALSE, что означает, что функция политики может вернуть предикат, длиной до 4000 байтов. Установка аргумента LONG_PREDICATE в TRUE приводит к тому, что функция может вернуть текстовую строку предиката размером до 32 Кб.

Пример 1.

Хотя формально этого не требуется, при использовании VPD методологически правильно завести специального пользователя-администратора. Ниже ему предоставляется минимум полномочий, достаточных для примера.

Войдите в SQL Developer под именем SYS с правами SYSDBA.

Выполните скрипт создания пользователя vpd_admin с паролем vpd_admin и назначения ему необходимых прав:

```
CREATE USER vpd_admin IDENTIFIED BY vpd_admin
DEFAULT TABLESPACE sysaux TEMPORARY TABLESPACE temp QUOTA UNLIMITED
ON sysaux;
GRANT CREATE SESSION TO vpd_admin;
GRANT CREATE PROCEDURE TO vpd_admin;
GRANT CREATE TABLE TO vpd_admin;
GRANT EXECUTE ON dbms_ols TO vpd_admin;
```

Пример демонстрирует избирательность строк и значений с учетом столбцов.

Функция политики доступа к таблице SCOTT.EMP и вспомогательная таблица, регулирующая доступ, создаются в отдельной схеме VPD_ADMIN «администратора политик»

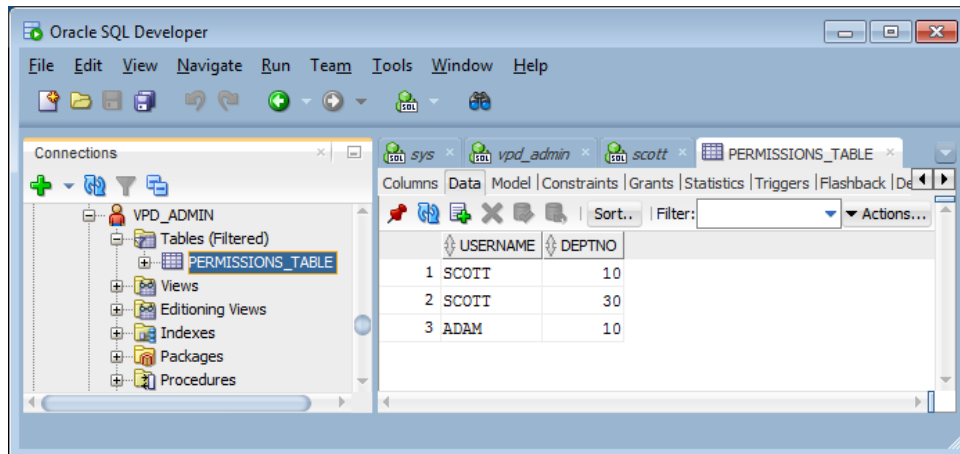
Войдите в SQL Developer под именем vpd_admin.

Выполните скрипт создания таблицы с данными, параметризующими доступ:

```

CREATE TABLE permissions_table
(
    username VARCHAR2(14),
    deptno NUMBER (2)
);
INSERT INTO permissions_table VALUES ('SCOTT', 10);
INSERT INTO permissions_table VALUES ('SCOTT', 30);
INSERT INTO permissions_table VALUES ('ADAM', 10);

```



Полагаем, что пользователю SCOTT будет разрешено работать с сотрудниками 10-го и 30-го отделов, а пользователю ADAM – с сотрудниками только 10-го.

От имени vpd_admin выполните скрипт создания функции, служащей предикатом доступа и задающей фильтр для строк при обращении к таблице SCOTT.EMP

```

CREATE OR REPLACE FUNCTION permissions_function
(
    obj_schema IN VARCHAR2
    ,obj_name IN VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN RETURN 'deptno IN (SELECT deptno FROM permissions_table ' ||
'WHERE username = USER)';
END;

```

Политику доступа к таблице SCOTT.EMP назовем EPOLICY.

От имени vpd_admin выполните скрипт создания политики.

```

BEGIN DBMS_RLS.ADD_POLICY
(
    POLICY_NAME => 'epolicy'
    ,OBJECT_SCHEMA => 'scott'
    ,OBJECT_NAME => 'emp'
    ,FUNCTION_SCHEMA => 'vpd_admin' ,POLICY_FUNCTION =>
'permissions_function'
);
END;

```

Пользователю scott предоставим права на запуск permissions_function:

```
GRANT EXECUTE ON permissions_function TO scott;
```

Проверка работы политики:

Разблокируйте пользователя scott, назначьте ему пароль tiger.

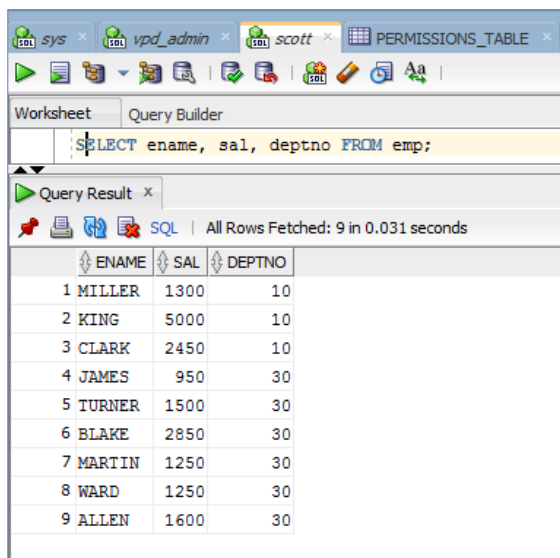
Проверьте, что таблица scott.emp содержит 14 строк (запрос к таблице от имени sys или графически, connection - sys).

Войдите в SQL Developer под именем scott.

Выполните запрос:

```
SELECT ename, sal, deptno FROM emp;
```

В выборке должны отобразиться только записи с номерами отделов 10 и 30.



	ENAME	SAL	DEPTNO
1	MILLER	1300	10
2	KING	5000	10
3	CLARK	2450	10
4	JAMES	950	30
5	TURNER	1500	30
6	BLAKE	2850	30
7	MARTIN	1250	30
8	WARD	1250	30
9	ALLEN	1600	30

Создайте пользователя ADAM, назначьте ему необходимые права.

Выполните запрос «SELECT ename, sal, deptno FROM emp;» от имени ADAM.

В выборке должно отобразиться 5 строк.

VPD на уровне столбцов.

Дополнительные параметры процедуры DBMS_RLS.ADD_POLICY.

Параметр SEC_RELEVANT_COLS (секретный столбец).

Рассмотрим пример, в котором VPD реализует бизнес-политику, в соответствии с которой только менеджер имеет доступ к важной информации о своих сотрудниках в таблице EMPLOYEES.

Пусть политика защищает только столбцы SALARY и COMMISSION_PCT таблицы EMPLOYEES:

```
BEGIN
  dbms_ols.add_policy(object_schema => 'hr',
    object_name => 'employees',
    policy_name => 'hr_policy',
    function_schema => 'hr',
    policy_function => 'hrsec',
    statement_types => 'select,insert',
    sec_relevant_cols=>'salary,commission_pct');
END;
```

Тогда для запроса вида

```
SQL> SELECT last_name FROM employees;
```

политика не будет работать, для любого пользователя запрос выдаст все строки таблицы.

А для запросов вида

```
SQL> SELECT last_name, salary FROM employees;
```

```
SQL> SELECT * FROM employees;
```

VPD применяет дифференцированный контроль доступа, определенный в функции политики.

От имени vpd_admin выполните скрипт удаления ранее использовавшейся политики.

```
BEGIN DBMS_RLS.drop_POLICY('scott','emp','epolicy'); end;
```

Теперь создадим новую политику с использованием секретного столбца.

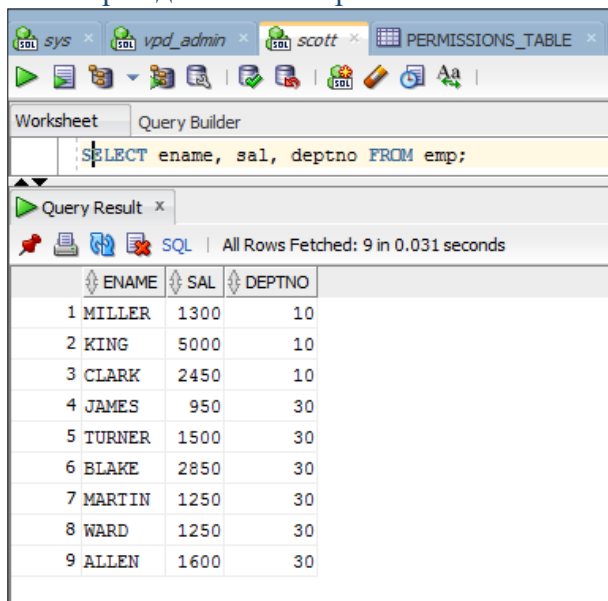
```
BEGIN DBMS_RLS.ADD_POLICY
(
    POLICY_NAME => 'epolicy'
  ,OBJECT_SCHEMA => 'scott'
  ,OBJECT_NAME => 'emp'
  ,FUNCTION_SCHEMA => 'vpd_admin'
  ,POLICY_FUNCTION => 'permissions_function'
  ,SEC_RELEVANT_COLS => 'sal, comm'
);
END;
```

Проверка:

От имени scott выполните запрос:

```
SELECT ename, sal, deptno FROM emp;
```

В выборке должны отображаться только записи с номерами отделов 10 и 30.

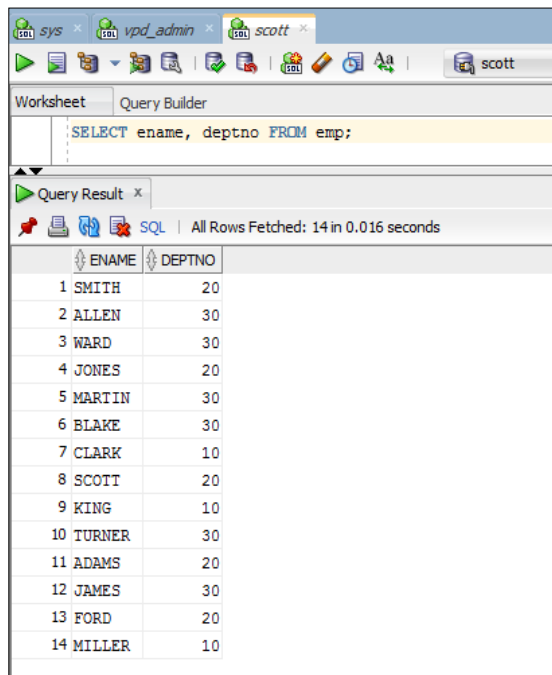


	ENAME	SAL	DEPTNO
1	MILLER	1300	10
2	KING	5000	10
3	CLARK	2450	10
4	JAMES	950	30
5	TURNER	1500	30
6	BLAKE	2850	30
7	MARTIN	1250	30
8	WARD	1250	30
9	ALLEN	1600	30

А вот что получим, если не обратимся к «секретному» столбцу с зарплатой.

От имени scott выполните запрос:

```
SELECT ename, deptno FROM emp;
```



The screenshot shows the Oracle SQL Developer interface. The 'Query Builder' tab is active, displaying the query 'SELECT ename, deptno FROM emp;'. Below the query, the 'Query Result' tab shows the results of the query. The results are displayed in a table with two columns: 'ENAME' and 'DEPTNO'. The table contains 14 rows of data, numbered 1 through 14. The data is as follows:

	ENAME	DEPTNO
1	SMITH	20
2	ALLEN	30
3	WARD	30
4	JONES	20
5	MARTIN	30
6	BLAKE	30
7	CLARK	10
8	SCOTT	20
9	KING	10
10	TURNER	30
11	ADAMS	20
12	JAMES	30
13	FORD	20
14	MILLER	10

Дополнительные параметры процедуры DBMS_RLS.ADD_POLICY.

Параметр SEC_RELEVANT_COLS_OPT.

При операции SELECT будут выдаваться все строки, но значения в «секретных» столбцах некоторых строк (защищенных политикой) не отображаются.

От имени vpd_admin выполните скрипт удаления ранее использовавшейся политики.

```
BEGIN DBMS_RLS.drop_POLICY('scott','emp','epolicy'); end;
```

Теперь создадим новую политику с использованием секретного столбца.

От имени vpd_admin выполните скрипт.

```
BEGIN DBMS_RLS.ADD_POLICY
(
    POLICY_NAME => 'epolicy'
  ,OBJECT_SCHEMA => 'scott'
  ,OBJECT_NAME => 'emp'
  ,FUNCTION_SCHEMA => 'vpd_admin'
  ,POLICY_FUNCTION => 'permissions_function'
  ,SEC_RELEVANT_COLS => 'sal, comm'
  ,SEC_RELEVANT_COLS_OPT => DBMS_RLS.ALL_ROWS
);
END;
```

Проверка:

От имени scott выполните запрос:

```
SELECT ename, sal, deptno FROM emp;
```

The screenshot shows the SQL Developer interface with a query window containing the SQL statement: `SELECT ename, sal, deptno FROM emp;`. Below the query window, the 'Query Result' pane displays 14 rows of data. The columns are labeled ENAME, SAL, and DEPTNO. The data is as follows:

	ENAME	SAL	DEPTNO
1	SMITH	(null)	20
2	ALLEN	1600	30
3	WARD	1250	30
4	JONES	(null)	20
5	MARTIN	1250	30
6	BLAKE	2850	30
7	CLARK	2450	10
8	SCOTT	(null)	20
9	KING	5000	10
10	TURNER	1500	30
11	ADAMS	(null)	20
12	JAMES	950	30
13	FORD	(null)	20
14	MILLER	1300	10

Пример 2.

Будем реализовывать политику FGAC для таблицы HR.EMPLOYEES, чтобы ограничить доступ на основе статуса менеджера и номера отдела сотрудника. Сотрудник сможет увидеть только информацию – строку данных о себе, менеджер – о себе и всех подчиненных.

Войдите в SQL Developer под именем SYS с правами SYSDBA.

Создадим в базе данных пользователей, которые будут выбирать строки из таблицы HR.EMPLOYEES:

```
create user smavris identified by smavris702;
grant connect, resource to smavris;

create user dgrant identified by dgrant507;
grant connect, resource to dgrant;

create user kmourgos identified by kmourgos622;
grant connect, resource to kmourgos;
```

Пользователь KMOURGOS является менеджером всех клерков, а DGRANT - одним из подчинённых KMOURGOS. Пользователь SMAVRIS - это главный менеджер по кадрам для компании.

Далее предоставим привилегии SELECT для таблицы HR.EMPLOYEES всем пользователям БД, создадим таблицу HR.EMP_LOGIN_MAP поиска, которая сопоставляет идентификационные номера сотрудников с их учетной записью в базе данных. Процедура, которая устанавливает переменные контекста для пользовательской сессии, будет использовать эту таблицу для присвоения идентификационного номера сотрудника контекстной переменной, которая будет использоваться в политике функция для генерации предиката.

```
grant select on hr.employees to public;
create table hr.emp_login_map (employee_id, login_acct)
as select employee_id, email from hr.employees;
grant select on hr.emp_login_map to public;
```

Далее мы создадим учетную запись пользователя с именем VPD, которая имеет привилегии для создания контекстов и является владельцем функции политики:

```
create user vpd identified by vpd439;  
grant connect, resource, create any context, create public synonym  
to vpd;
```

Подключимся к БД под именем VPD и создадим контекст под названием HR_SECURITY, определим пакет и процедуру, используемые для установки контекста приложения:

```
create context hr_security using vpd.emp_access;  
create or replace package vpd.emp_access as  
    procedure set_security_parameters;  
end;
```

Помните, что процедуры в пакете VPD.EMP_ACCESS являются единственными процедурами, которые могут устанавливать контекстные переменные.

Тело пакета для VPD.EMP_ACCESS:

```
create or replace package body vpd.emp_access is  
--  
-- At user login, run set_security_parameters to  
-- retrieve the user login name, which corresponds to the EMAIL  
-- column in the table HR.EMPLOYEES.  
--  
-- context USERENV is pre-defined for user characteristics such  
-- as username, IP address from which the connection is made,  
-- and so forth.  
--  
-- for this procedure, we are only using SESSION_USER  
-- from the USERENV context.  
--  
procedure set_security_parameters is  
    emp_id_num number;  
    emp_login varchar2(50);  
begin  
    -- database username corresponds to email address in HR.EMPLOYEES  
    emp_login := sys_context('USERENV','SESSION_USER');  
    dbms_session.set_context('HR_SECURITY','USERNAME',emp_login);  
    -- get employee id number, so manager rights can be established  
    -- but don't bomb out other DB users who are not in the  
    -- EMPLOYEES table  
begin  
    select employee_id into emp_id_num  
    from hr.emp_login_map where login_acct = emp_login;  
    dbms_session.set_context('HR_SECURITY','EMP_ID',emp_id_num);  
exception  
    when no_data_found then  
        dbms_session.set_context('HR_SECURITY','EMP_ID',0);  
end;  
    -- Future queries will restrict rows based on emp_id  
end; -- procedure  
end; -- package body
```


В процедуре получаем схему пользователя, используя контекст БД USERENV, который включен по умолчанию для всех пользователей и присваивает значение переменной USERNAME во вновь созданном контексте HR_SECURITY.

Другой контекст HR_SECURITY – это переменная EMP_ID определяется путем поиска в таблице сопоставления HR.EMP_LOGIN_MAP.

Чтобы избежать ошибок в процедуре в случае, если зарегистрированный пользователь отсутствует в HR.EMP_LOGIN_MAP, присвоим EMP_ID значение 0, что приведет к отсутствию доступа к таблице HR.EMPLOYEES, когда предикат сгенерируется в функции политики.

На следующих шагах предоставим всем пользователям БД привилегии EXECUTE для пакета и создадим для него синоним, чтобы упростить вызов процедур.

```
grant execute on vpd.emp_access to PUBLIC;
create public synonym emp_access for vpd.emp_access;
```

Чтобы гарантировать, что контекст определен для каждого пользователя при входе в систему, подключимся к БД под учеткой SYS с правами SYSDBA и создадим триггер входа в систему, который определяет переменные в контексте:

```
create or replace trigger vpd.set_security_parameters
after logon on database
begin
    vpd.emp_access.set_security_parameters;
end;
```

Поскольку этот триггер срабатывает для каждого пользователя, который подключается к базе данных, важно, чтобы код был протестирован для всех пользователей БД. Если триггер завершается с ошибкой, обычные пользователи не смогут войти в БД.

Итак, у нас определены контекст, процедура, используемая для установки переменных контекста, и триггер, который автоматически вызывает процедуру. Войдя в систему как один из трех наших ранее определенных пользователей, мы можем запросить содержимое контекста:

Подключитесь к БД с именем smavris и выполните запрос:

```
select * from session_context;
```

NAMESPACE	ATTRIBUTE	VALUE
HR_SECURITY	USERNAME	SMAVRIS
HR_SECURITY	EMP_ID	203

2 rows selected.

Обратите внимание, что происходит, когда SMAVRIS пытается выдать себя за другого сотрудника:

```
begin dbms_session.set_context('HR_SECURITY','EMP_ID',100);end;
```

```
begin
```

```
*
```

```
ERROR at line 1:
```

```
ORA-01031: insufficient privileges
```

```
ORA-06512: at "SYS.DBMS_SESSION", line 94
```

```
ORA-06512: at line 2
```

Только пакет VPD.EMP_ACCESS может устанавливать или изменять переменные в контексте.

Заключительные шаги включают определение процедур, которые будут генерировать предикат, и назначение одной или нескольких из этих процедур таблице HR.EMPLOYEES.

От имени пользователя VPD – владельца контекстных процедур, настроим пакет, который определяет предикаты.

```
create or replace package vpd.get_predicates as

-- note -- security function ALWAYS has two parameters,
-- table owner name and table name

function emp_select_restrict
  (owner_name varchar2, object_name varchar2) return varchar2;

-- other functions can be written here for INSERT, DELETE, and so forth.
end get_predicates;
```

```
create or replace package body vpd.get_predicates is
  function emp_select_restrict
    (owner_name varchar2, object_name varchar2) return varchar2 is

    ret_predicate varchar2(1000); -- part of WHERE clause
  begin
    -- only allow employee to see their row or immediate subordinates
    ret_predicate := 'EMPLOYEE_ID = ' ||
                     sys_context('HR_SECURITY','EMP_ID') ||
                     ' OR MANAGER_ID = ' ||
                     sys_context('HR_SECURITY','EMP_ID');
    return ret_predicate;
  end emp_select_restrict;

end; -- package body
```

Как только мы прикрепим функцию к таблице с помощью DBMS_RLS, она сгенерирует текстовую строку, которая может использоваться в предложении WHERE при каждом доступе к таблице. Строка всегда будет выглядеть примерно так:

«EMPLOYEE_ID = 124 OR MANAGER_ID = 124»

Как и в случае пакетов, которые устанавливают контекстную среду, нам нужно разрешить пользователям доступ к этому пакету:

```
grant execute on vpd.get_predicates to PUBLIC;
create public synonym get_predicates for vpd.get_predicates;
```

Наконец, что не менее важно, определим функцию политики для таблицы, используя процедуру DBMS_RLS.ADD_POLICY:

```
SQL> begin
  dbms_rls.add_policy (
    object_schema =>      'HR',
    object_name =>        'EMPLOYEES',
    policy_name =>        'EMP_SELECT_RESTRICT',
    function_schema =>    'VPD',
    policy_function =>    'get_predicates.emp_select_restrict',
    statement_types =>    'SELECT',
    update_check =>       TRUE,
    enable =>             TRUE
  );
end;
```

Сотрудник может получить доступ к таблице HR.EMPLOYEES, как и раньше, но он увидит только свою строку и строки подчиненных сотрудников, если таковые имеются.

Войдя в систему как KMOURGOS, попытаемся получить все строки таблицы HR.EMPLOYEES, но видим только KMOURGOS и его подчиненных:

```
select employee_id, first_name, last_name, email, job_id, salary,
manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
124	Kevin	Mourgos	KMOURGOS	ST_MAN	5800	100
141	Trenna	Rajs	TRAJS	ST_CLERK	3500	124
142	Curtis	Davies	CDAVIES	ST_CLERK	3100	124
143	Randall	Matos	RMATOS	ST_CLERK	2600	124
144	Peter	Vargas	PVARGAS	ST_CLERK	2500	124
196	Alana	Walsh	AWALSH	H_CLERK	3100	124
197	Kevin	Feeney	KFEENEY	SH_CLERK	3000	124
198	Donald	OConnell	DOCONNEL	SH_CLERK	2600	124
199	Douglas	Grant	DGRANT	SH_CLERK	2600	124

9 rows selected.

Для пользователя DGRANT:

```
select employee_id, first_name, last_name, email, job_id, salary,
manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
199	Douglas	Grant	DGRANT	SH_CLERK	2600	124

1 row selected.

Для пользователя SMAVRIS:

```
select employee_id, first_name, last_name, email, job_id, salary,
manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
203	Susan	Mavris	SMAVRIS	HR_REP	6500	101

1 row selected.

Но SMAVRIS – сотрудник отдела кадров и должен видеть все строки таблицы. Кроме того, SMAVRIS должен быть единственным, кто видит информацию о зарплате всех сотрудников.

В результате нам необходимо изменить функцию политики, чтобы SMAVRIS и другие сотрудники отдела кадров имели полный доступ к таблице HR.EMPLOYEES.

Чтобы облегчить доступ к таблице HR.EMPLOYEES сотрудникам отдела кадров, нам сначала нужно изменить нашу таблицу сопоставления, включив в нее столбец JOB_ID. Если столбец JOB_ID имеет значение HR_REP, сотрудник находится в отделе кадров. Сначала мы **отключим** действующую политику и создадим **новую таблицу** сопоставления:

От имени VDP выполним:

```
begin
    dbms_rls.enable_policy(
        object_schema => 'HR',
        object_name => 'EMPLOYEES',
        policy_name => 'EMP_SELECT_RESTRICT',
        enable => FALSE
    );
end;
```

От имени SYS выполним:

```
drop table hr.emp_login_map;
create table hr.emp_login_map (employee_id, login_acct, job_id)
    as select employee_id, email, job_id from hr.employees;
grant select on hr.emp_login_map to public;
```

Процедура VPD.EMP_ACCESS, которую мы используем для настройки переменных контекста, нуждается в добавлении другой переменной контекста, которая указывает уровень безопасности пользователя, обращающегося к таблице.

Мы изменим инструкцию SELECT и сделаем еще один вызов DBMS_SESSION.SET_CONTEXT следующим образом (см. тело пакета для VPD.EMP_ACCESS):

```
. . .
    emp_job_id varchar2(50);
. . .
        select employee_id, job_id into emp_id_num, emp_job_id
            from hr.emp_login_map where login_acct = emp_login;

        dbms_session.set_context('HR_SECURITY', 'SEC_LEVEL',
            case emp_job_id when 'HR_REP' then 'HIGH' else
            'NORMAL' end );
. . .
```

А именно, от имени VDP выполним:

```
create or replace package body vpd.emp_access is
procedure set_security_parameters is
emp_id_num number;
emp_login varchar2(50);
emp_job_id varchar2(50);
begin
    emp_login := sys_context('USERENV', 'SESSION_USER');
    dbms_session.set_context('HR_SECURITY', 'USERNAME', emp_login);
```

```

begin
    select employee_id into emp_id_num from hr.emp_login_map where
        login_acct = emp_login;
    select employee_id, job_id into emp_id_num, emp_job_id from
        hr.emp_login_map where login_acct = emp_login;

    dbms_session.set_context('HR_SECURITY','EMP_ID',emp_id_num);
    dbms_session.set_context('HR_SECURITY','SEC_LEVEL',case
        emp_job_id when 'HR_REP' then 'HIGH' else 'NORMAL' end );

    exception
    when no_data_found then
        dbms_session.set_context('HR_SECURITY','EMP_ID',0);
    end;
end; -- procedure
end; -- package body

```

От имени VDP:

Для каждого сотрудника с должностью HR_REP, контекстная переменная SEC_LEVEL устанавливается на HIGH вместо NORMAL. В нашей функции политики нужно проверить это новое условие следующим образом:

```

create or replace package body vpd.get_predicates is
    function emp_select_restrict
        (owner_name varchar2, object_name varchar2) return varchar2 is
        ret_predicate varchar2(1000); -- part of WHERE clause

    begin
        -- only allow employee to see their row or immediate subordinates,
        -- unless they have high security clearance
        if sys_context('HR_SECURITY','SEC_LEVEL') = 'HIGH' then
            ret_predicate := ''; -- no restrictions in WHERE clause
        else
            ret_predicate := 'EMPLOYEE_ID = ' ||
                sys_context('HR_SECURITY','EMP_ID') ||
                ' OR MANAGER_ID = ' ||
                sys_context('HR_SECURITY','EMP_ID');

            end if;
            return ret_predicate;
        end emp_select_restrict;

    end; -- package body

```

От имени VDP выполним включение политики (которую мы ранее выключили):

```

begin
    dbms_rls.enable_policy(
        object_schema => 'HR',
        object_name => 'EMPLOYEES',
        policy_name => 'EMP_SELECT_RESTRICT',
        enable => TRUE
    );
end;

```

Поскольку политика является динамической, предикат генерируется при каждом выполнении инструкции SELECT, поэтому нам не нужно обновлять политику.

Когда пользователь SMAVRIS, представитель HR, выполнит запрос, он увидит все строки в таблице HR.EMPLOYEES:

```
select employee_id, first_name, last_name, email, job_id, salary,
manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
100	Steven	King	SKING	AD_PRES	24000	
101	Neena	Kochhar	NKOCHHAR	AD_VP	17000	100
. . .						
204	Hermann	Baer	HBAER	PR_REP	10000	101
205	Shelley	Higgins	SHIGGINS	AC_MGR	12000	101
206	William	Gietz	WGIEZT	AC_ACCOUNT	8300	205
107 rows selected.						

Как и следовало ожидать, уровень безопасности SMAVRIS в контексте HR_SECURITY – HIGH:

```
select sys_context('HR_SECURITY','SEC_LEVEL') from dual;
```

```
SYS_CONTEXT('HR_SECURITY','SEC_LEVEL')
-----
HIGH
```

Пользователь DGRANT по-прежнему может видеть только свою строку в таблице, потому что его уровень безопасности в контексте HR_SECURITY равен NORMAL:

```
select employee_id, first_name, last_name, email, job_id, salary,
manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
199	Douglas	Grant	DGRANT	SH_CLERK	2600	124
1 row selected.						

```
select sys_context('HR_SECURITY','SEC_LEVEL') from dual;
```

```
SYS_CONTEXT('HR_SECURITY','SEC_LEVEL')
-----
NORMAL
```

Чтобы выполнить требование о том, что только сотрудники отдела кадров могут видеть информацию о зарплате, нам нужно внести небольшое изменение в функцию политики и включить политику с ограничениями на уровне столбцов:

```
dbms_rls.add_policy (
    object_schema => 'HR',
    object_name => 'EMPLOYEES',
    policy_name => 'EMP_SELECT_RESTRICT',
    function_schema => 'VPD',
    policy_function => 'get_predicates.emp_select_restrict',
    statement_types => 'SELECT',
    update_check => TRUE,
```

```

enable => TRUE,
sec_relevant_cols => 'SALARY',
sec_relevant_cols_opt => dbms_ols.all_rows
);

```

Последний параметр, SEC_RELEVANT_COLS_OPT, указывает константу пакета DBMS_OLS.ALL_ROWS, чтобы указать, что мы хотим видеть все строки в результатах нашего запроса, но с пустыми ячейками в конфиденциальных столбцах (в данном случае SALARY). Иначе строки из запросов, которые содержат столбец SALARY, не будут отображаться.

Измените функцию политики так, чтобы отображались все строки запроса, но конфиденциальные данные в столбце SALARY будут скрыты. Проверьте, что политика работает корректно для пользователей DGRANT, SMAVRIS, KMOURGOS.

Отладка VPD политики.

Даже если нет ошибок типа «ORA-28113: policy predicate has error» или «ORA-00936: missing expression», может быть очень полезно увидеть фактический предикат, сгенерированный во время синтаксического анализа оператора. Есть несколько способов отладки ваших предикатов, все имеют свои преимущества и недостатки.

Первый метод использует динамические представления производительности V\$SQLAREA и V\$VPD_POLICY. Как видно из названий, V\$SQLAREA содержит операторы SQL, которые в настоящее время находятся в общем пуле, а также текущую статистику выполнения. В представлении V\$VPD_POLICY перечислены все политики, применяемые в настоящее время в базе данных, а также предикат. Объединение двух таблиц, как в следующем примере, дает нам информацию, необходимую нам для отладки любых проблем, которые у нас возникают с результатами запроса:

```

SQL> select s.sql_text, v.object_name, v.policy, v.predicate
       from v$sqlarea s, v$vpd_policy v
       where s.hash_value = v.sql_hash;

```

SQL_TEXT	OBJECT_NAME	POLICY	PREDICATE
select employee_id, first_name, last_name, email, job_id, salary, manager_id from hr.employees	EMPLOYEES	EMP_SELECT_RESTRICT	EMPLOYEE_ID = 199 OR MANAGER_ID = 199

Если мы добавим соединение к V\$SESSION в этом запросе, мы сможем определить, какой пользователь запускал SQL. Это особенно важно, когда к оператору SQL не применяется предикат, и мы можем проверить вывод, что один из сотрудников отдела кадров выполнил запрос.

У этого метода есть и обратная сторона: если база данных очень занята, команды SQL могут быть замещены в shared pool другими командами SQL, прежде чем вы получите возможность выполнить этот запрос.

Другой метод использует команду alter session для создания файла трассировки в виде простого текста, содержащего большую часть информации из предыдущего запроса. В примере даны команды для настройки трассировки:

```

SQL> begin dbms_ols.refresh_policy; end;
PL/SQL procedure successfully completed.

```

```
SQL> alter session set events '10730 trace name context forever,
level 12';
Session altered.
```

Событие 10730 определено для отслеживания предикатов политики RLS. Другими общими событиями, которые можно отследить, являются 10029 и 10030 для входа / выхода из сеанса, 10710 для отслеживания доступа к индексам растровых изображений и 10253 для имитации ошибок записи в журнал повторов. Как только сеанс изменен, пользователь DGRANT запускает свой запрос:

```
SQL> select employee_id, first_name, last_name, email, job_id,
salary, manager_id from hr.employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	MANAGER_ID
199	Douglas	Grant	DGRANT	SH_CLERK	2600	124

1 row selected.

Here's a look at the bottom part of the trace file located in the directory specified by the initialization parameter USER_DUMP_DEST (DIAGNOSTIC_DEST in Oracle Database 11g):

Вот последние строки файла трассировки, расположенного в каталоге, указанном в параметре инициализации USER_DUMP_DEST (DIAGNOSTIC_DEST в Oracle Database 11g):

Trace file

```
/u01/app/oracle/diag/rdbms/dw/dw/trace/dw_ora_31128.trc
```

```
Oracle Database 11g Enterprise Edition
```

```
Release 11.1.0.6.0 - Production
```

```
With the Partitioning, OLAP, Data Mining and
```

```
Real Application Testing options
```

```
ORACLE_HOME = /u01/app/oracle/product/11.1.0/db_1
```

```
System name: Linux
```

```
Node name: dw
```

```
Release: 2.6.9-55.0.2.0.1.EL
```

```
Version: #1 Mon Jun 25 14:24:38 PDT 2007
```

```
Machine: i686
```

```
Instance name: dw
```

```
Redo thread mounted by this instance: 1
```

```
Oracle process number: 40
```

```
Unix process pid: 31128, image: oracle@dw (TNS V1-V3)
```

```
*** 2007-08-12 12:48:37.852
```

```
*** SESSION ID:(120.9389) 2007-08-12 12:48:37.852
```

```
*** CLIENT ID:() 2007-08-12 12:48:37.852
```

```
*** SERVICE NAME:(SYS$USERS) 2007-08-12 12:48:37.852
```

```
*** MODULE NAME:(SQL*Plus) 2007-08-12 12:48:37.852
```

```
*** ACTION NAME:() 2007-08-12 12:48:37.852
```

```
-----
Logon user      : DGRANT
```

```
Table/View     : HR.EMPLOYEES
```

```
Policy name    : EMP_SELECT_RESTRICT
```

```
Policy function : VPD.GET_PREDICATES.EMP_SELECT_RESTRICT
```

```
RLS view       :
```



```
SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "EMAIL", "PHONE_NUMBER",  
"HIRE_DATE", "JOB_ID", "SALARY", "COMMISSION_PCT", "MANAGER_ID",  
"DEPARTMENT_ID" FROM "HR"."EMPLOYEES"  
"EMPLOYEES" WHERE (EMPLOYEE_ID = 199 OR MANAGER_ID = 199)
```

Исходный оператор SQL пользователя и добавленный предикат четко отображаются в файле трассировки. Недостатком использования этого метода является то, что, хотя пользователь может иметь доступ к динамическим представлениям производительности, разработчик обычно может не иметь доступа к каталогу дампа пользователя на самом сервере. В результате при попытке отладки проблем предиката может потребоваться участие администратора базы данных.

Обязательно отключите трассировку после завершения отладки, чтобы уменьшить накладные расходы и дисковое пространство, связанные с операциями трассировки (или просто выйдите из системы!):

```
SQL> alter session set events '10730 trace name context off';  
Session altered.
```

Время на выполнение лабораторной работы – 2 часа.