**Детальный контроль доступа и контексты приложения**

### 

*Т.Кайт*

В этой статье рассмотрены два новых механизма Oracle: Детальный Контроль Доступа (Fine Grained Access Control) и Контексты Защищенных Приложений (Secure Application Contexts). При совместном использовании они обеспечивают новые качественные возможности для обеспечения информационной безопасности базы данных.

Как и в других статях этого цикла, в этом опусе будут рассмотрены:

* [**Для чего нужны новые возможности**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t1)
* [**Способы использования**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t2)
  + [**Пример 1. Реализация политики безопасности**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t3)
  + [**Важное предупреждение**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t4)
  + [**Пример 2. Использование контекста приложения**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t5)
  + [**Обработка ошибок и отладка**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t6)
* [**За и против**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t7)
* [**Демонстрационные скрипты**](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387.html#t8)

В различных изданиях Детальный Контроль Доступа может быть назван по-разному. Ниже перечислены его синонимы:

* Детальный Контроль Доступа (Fine Grained Access Control - техническое название)
* Виртуальная Частная База Данных (Virtual Private Database - рыночное название)
* Безопасность на Уровне Строк (Row Level Security - техническое название, идущее от того, что эту возможность реализуют PL/SQL-пакеты)

В двух словах, Детальный Контроль Доступа в Oracle - это возможность во время работы динамически присоединить предикат (предложения where) как к одному, так и ко всем запросам к таблице или представлению.

Таким образом, появилась возможность процедурной модификации запроса в процессе выполнения. Можно вычислить, **кто** выполняет запрос, **откуда**запрос выполняется, **когда** началось выполнение запроса, и сформировать предикат, соответствующий этим критериям. При использовании Контекстов Приложения можно незаметно через окружение (например, через роль, назначенную пользователю приложения) добавить дополнительную информацию, и получить доступ к ней через процедуру или предикат.

Примером Детального Контроля Доступа может служить политика безопасности, которая определяет, какие строки могут быть доступны различным группам пользователей. Политика безопасности формирует предикат, вид которого зависит от соединенного с базой пользователя и группы, к которой он относится. Детальный Контроль Доступа позволяет при вводе запроса "select \* from emp" различными пользователями преобразовать его к следующему виду:

|  |  |  |
| --- | --- | --- |
| **Пользователь** | **Запрос динамически  переписывается в** | **Замечания** |
| Служащий | select \*  from ( select \*  from emp  where ename = USER ) | Служащие могут видеть только свои записи |
| Менеджер | select \*  from ( select \*  from emp  where mgr = (select  empno from emp  where ename = USER)  or ename = USER) | Менеджеры могут видеть свои записи и записи тех, кто работает под их руководством. |
| Контролер | select \*  from (select \*  from emp  where deptno =  SYS\_CONTEXT  (‘OurApp’,‘Deptno’)) | Контролер может видеть любого в заданном отделе. Этот пример знакомит с синтаксисом получения переменных контекста приложения -встроенной функцией SYS\_CONTEXT(). |

***Почему используется эта возможность***

Есть много причин, чтобы использовать этот механизм. Наиболее распространенные из них:

* **Легко поддерживается**. Детальный Контроль Доступа позволяет иметь только 1 таблицу и 1 хранимую управляющую процедуру, которые заменят использование множества представлений. Создание множества представлений обычно приводит к увеличению числа объектов базы данных, так как для каждой группы пользователей требуется создание отдельного представления. Например, в описанном выше примере со служащими, менеджерами и контролерами в обычной системе необходимо создать 3 представления базы данных. Если потребуется еще одна группа пользователей, то придется добавить еще один набор представлений, которым надо будет управлять и поддерживать. Если политика безопасности изменится (то есть потребуется, чтобы менеджеры видели не только своих непосредственных подчиненных, но и на 2 уровня ниже), необходимо будет пересоздать представления базы данных, после чего все объекты, ссылающихся на эти представления, станут недействительными.
* **Осуществляется на сервере**. Учитывая сложность управления и поддержки большого количества представлений, разработчики раз за разом стремятся закладывать логику приложения в самое приложение. Приложения просматривают, кто присоединен к базе данных, что он запрашивает, и выполняют соответствующий запрос. Это защищает данные, ***но только тогда, когда доступ к ним осуществляется через данное приложение.*** Это снижает возможность использования средств выполнения запросов и генерации отчетов, а также других средств обработки данных. Повышается также вероятность получения искаженных данных, так как для того, чтобы сделать искажение, достаточно подключиться к базе данных через любое другое средство, отличное от рассматриваемого приложения, и запросить данные. Благодаря же включению в базу данных логики безопасности, то есть механизма, который определяет, какие данные может видеть пользователь, - ***вы******можете быть уверены, что данные будут защищены независимо от используемого средства доступа к ним, и обращение можно осуществлять с помощью любого средства, из которого возможен доступ к данным.***
* **Запрет на соединение с базой данных от имени обобщенных пользователей.** Благодаря Детальному Контролю Доступа каждый пользователь должен соединяться с базой данных под своим именем. В этом случае обеспечивается полная подотчетность - можно отслеживать действия на уровне пользователя. Раньше многие приложения при работе с различными представлениями данных для различных пользователей должны были применять обобщенных пользователей базы данных, соответственно выбираемым данным. Например, в вышеописанном случае служащий/менеджер/контролер в приложении должно быть создано три учетных записи. Каждый служащий должен использовать учетную запись “Служащий”. Каждый менеджер должен использовать учетную запись “Менеджер”. Каждый контролер должен использовать учетную запись “Контролер”. Это делает невозможным учитывать действия на уровне истинных пользователей.
* **Упрощение разработки приложения.**Детальный Контроль Доступа забирает логику безопасности из логики приложения. Для поддержки безопасности данных разработчик приложения может сконцентрироваться на самом приложении, а не на логике низкоуровневого доступа к данным. Так как Детальный Контроль Доступа полностью осуществляется на сервере, то приложения непосредственно наследуют эту логику. Раньше разработчики приложения должны были встраивать логику в приложение, делая приложение все более сложным, сначала для разработки и особенно сложным для его последующей поддержки. Если из приложения возможен доступ к данным, причем к одним и тем же данным и из нескольких точек приложения, то простейшее изменение политики безопасности может затронуть много дюжин модулей приложения. Благодаря применению Детального Контроля Доступа, изменения в политике безопасности **не влияют** на модули приложения.
* **Применение развитых средств разработки приложения.**Во многих средах политика безопасности по началу еще должным образом не определена и через некоторое время может измениться. Если происходит слияние компаний или другие структурные перемены или вводятся правила секретности, то политику безопасности необходимо изменить. Благодаря тому, что управление доступом осуществляется на уровне, близком к данным, можно создать условия для развития приложения с минимальным влиянием, и на него, и на средства разработки. Это является одной из причин для того, чтобы перейти к автоматическому использованию как новой логики, так и всех приложений и инструментов, позволяющих осуществлять доступ к базе данных со встроенной новой логикой.

**Способы использования этой возможности**

В Oracle существует два типа детального контроля доступа:

* **"Контекст" приложения**. Это пространство имен с набором пар соответствующих параметров переменная/значение. Например, в контексте, называемом ‘OurApp’, можно получить доступ к переменным ‘DeptNo’, ‘Mgr’ и так далее. Контексты приложения всегда связываются с некоторым PL/SQL-пакетом. **Единственный способ** **присваивания значений контекста -** **это** **вызов** **пакета.** Например, для получения переменной ‘DeptNo’ и установки ее значения в контексте ‘OurApp’ необходимо вызвать специальный пакет, связанный с контекстом ‘OurApp’. Этот пакет гарантирует корректную установку значений контекста ‘OurApp’ (вы сами так написали, поэтому корректная установка контекста гарантируется). В этом случае предотвращается установка значений контекста приложения злоумышленниками, которые в противном случае могли бы получить доступ к той информации, доступа к которой у них быть не должно. Любой пользователь может читать значения контекста приложения, но установить их может только пакет.
* **Политика безопасности.**Политика безопасности представляет собой просто функцию, построенную так, чтобы во время выполнения запроса она могла возвращать предикат для динамической фильтрации данных. Эта функция обычно использует значения контекста приложения для формирования и возврата корректного предиката (т.е. она просматривает, ‘кто’ присоединен к базе данных, ‘что’ он собирается сделать, и проверяет, есть ли у него привилегии для выполнения этих операций). Следует обратить внимание, что по отношению к пользователю SYS (или INTERNAL) никогда не используется политика безопасности, эти пользователи могут видеть все данные.

Для того, чтобы воспользоваться этой возможностью, разработчик, кроме стандартных ролей connect и resource, должен иметь следующие привилегии:

* **EXECUTE\_CATALOG\_ROLE.**Позволяет разработчику выполнять функции и процедуры пакета dbms\_rls. Другой вариант - можно, присоединившись как SYS, передать привилегию только на пакет: **grant execute on dbms\_rls to** **<учетная\_запись>**.
* **CREATE ANY CONTEXT:**Позволяет разработчику создавать контексты приложения.

Контексты приложения создаются простой SQL-командой

SQL> create context **OurApp** using **Our\_Context\_Pkg**;

OurApp - это имя контекста, а Our\_Context\_Pkg - это PL/SQL-пакет, через который устанавливаются значения контекста. Возможность использования Контекстов Приложения для Детального Контроля Доступа имеет большое значение по двум причинам:

* Обеспечивается гарантированный способ установки переменных пространства имен. Устанавливать значения этого контекста может только PL/SQL-пакет, связанный с контекстом. В этом случае гарантируется целостность значений контекста. Так как контексты предназначены для ограничения или разрешения доступа к данным, целостность значений контекста должна быть обеспечена.
* В SQL-запросе ссылки на значения Контекста Приложения трактуются как связанные переменные. Например, установка переменной ‘DeptNo’ контекста ‘OurApp’ и использование политики "deptno = SYS\_CONTEXT(‘OurApp’,’DeptNo’)" для возврата условия where не повлияют на частоту использования разделяемого sql-предложения, так как ссылка SYS\_CONTEXT подобна "deptno = :b1". Каждый может пользоваться значениями ‘Deptno’, но все будут повторно использовать один и тот же разобранный оптимизированный план запроса.

**Пример 1. Реализация политики безопасности**

Если требуется, чтобы политика безопасности позволяла пользователю, не являющемуся RLS\_ADMIN, видеть только такие строки, “владельцем” которых он является, то необходимо выполнить команду:

**EXECUTE\_CATALOG\_ROLE, CREATE ANY CONTEXT**

**Connect, resource,**

Создание от имени RLS\_ADMIN

SQL> create function my\_security\_function --example 1

2 ( p\_schema in varchar2,p\_object in varchar2 )

3 return varchar2

4 as

5 begin

6 if ( user = 'RLS\_ADMIN' ) then

7 return '';

8 else

9 return 'owner = USER';

10 end if;

11 end;

12 /

Функция создана.

Предикат "where owner = USER" будет динамически добавляться ко всем запросам по таблице, с которой связана эта функция, что значительно уменьшает количество строк, доступных пользователю. Предикат NULL (пусто) возвращается только в том случае, когда на данный момент к базе данных присоединен пользователь RLS\_ADMIN. Пустой возвращаемый предикат выглядит как "1=1" или "TRUE".

Для того, чтобы связать эту функцию с таблицей, необходимо использовать PL/SQL-процедуру "dbms\_rls.add\_policy". Например, имеется следующая таблица:

SQL> create table my\_table

2 ( datavarchar2(30),

3 **OWNER** varchar2(30) default USER

4 )

5 /

Table created.

SQL> grant all on my\_table to public

2 /

Grant succeeded.

**Надо добавить**

**create public synonym data\_table for data\_table;**

SQL> insert into my\_table (data)

2 values ('Некоторые данные')

3 /

1 row created.

SQL> insert into my\_table ( data, owner )

2 values ( 'Some Data Owned by SCOTT', 'SCOTT' )

3 /

1 row created.

SQL> commit

2 /

Commit complete.

SQL> select \* from my\_table

2 /

DATA OWNER

-------------------------------- ---------

Некоторые даные RLS

Некоторые данные владелеца SCOTT SCOTT

Create public synonym data\_table for data\_table;

Политику "My\_Security\_Policy" следует подключать

следующим образом:

Теперь привяжем написанную ранее функцию защиты к этой таблице с помощью сле-  
дующего обращения к пакету **DBMS\_RLS**:

Не RLS, а RLS\_ ADMIN

SQL> begin

2 dbms\_rls.add\_policy

3 ( object\_schema => 'RLS',

4 object\_name => 'MY\_TABLE',

5 policy\_name => 'MY\_POLICY',

6 function\_schema => 'RLS',

7 policy\_function => 'My\_Security\_Function',

8 statement\_types => 'select,insert,update,delete',

9 update\_check => TRUE );

10 end;

11 /

begin

dbms\_rls.add\_policy

( object\_schema => **'RLS\_admin'**, --alt user

object\_name => 'MY\_TABLE',

policy\_name => 'MY\_POLICY',

function\_schema => **'RLS\_admin'**, --alt user

policy\_function => 'My\_Security\_Function',

statement\_types => 'select, insert, update, delete' ,

update\_check => TRUE );

end;

PL/SQL procedure successfully completed.

**Только RLS\_ ADMIN увидит все записи**

Теперь все DML-предложения, относящиеся к таблице EMP, будут иметь предикат, возвращаемый связанной функцией my\_security\_function, независимо от источника, вызвавшего DML-операцию (т.е. независимо приложения, получающего доступ к данным). Посмотрим на это в действии:

SQL> connect rls/rls

Connected.

SQL> select \* from my\_table

2 /

DATA OWNER

----------------------- ------------

Некоторые данные RLS

**RLS\_ ADMIN/ RLS\_ ADMIN**

DATA OWNER

------------------------------ ------------------------------

Some Data RLS\_ADMIN

Some Data Owned by SCOTT SCOTT

Connect Prof/prof

DATA OWNER

------------------------------ ------------------------------

0 rows selected

Итак, полученный результат показывает, что строки отфильтрованы надлежащим образом - текущий пользователь RLS может видеть только свои строки - он является их владельцем. Строки, владельцем которых является SCOTT, стали невидимы. Присоединимся теперь как учетная запись RLS\_ADMIN:

SQL> connect rls\_admin/rls\_admin

Connected.

SQL> select \* from rls.my\_table

2 /

DATA OWNER

--------------------------------- ----------------

Некоторые данные RLS

Некоторые данные владелеца SCOTT SCOTT

Результат показывает, что учетная запись RLS\_ADMIN может видеть все данные, какие пожелает. Присоединимся опять учетной записью RLS и посмотрим, что произойдет при попытке создания данных, которые нельзя 'увидеть' (пользователь не является их владельцем):

SQL> connect rls/rls

Connected.

SQL> insert into my\_table ( data )

2 values ( 'Некоторые новые данные' )

3 /

1 row created.

SQL> insert into my\_table ( data, owner )

2 values ('Некоторые новые данные

3 владелеца SCOTT', 'SCOTT' )

4 /

**insert into my\_table ( data, owner )**

**\***

**ERROR at line 1:**

**ORA-28115: нарушение политики с опцией проверки**

Ошибка ORA-28115 возникает, так как при добавлении политики было указано:

…

9 update\_check => TRUE );

…

по аналогии с созданием представления с включенной возможностью "CHECK OPTION". Такая политика позволяет создавать только те данные, которые можно выбрать. По умолчанию можно создавать данные, которые выбрать нельзя.

Если подсоединится SCOTT, SYSTEM или SYS?

SYS и RLS\_ADMIN имеют одинаковые привилегии просматривать все записи, а SYSTEM -нет

**Важное предупреждение Кэширование курсоров**

Существует одна важная особенность реализации вышеописанной предикатной функции безопасности - фактически во время заданной сессии эта функция **обязана** **возвращать постоянный предикат - это существенно.** Если посмотреть на описанную выше функцию еще раз, видно логику:

3 as

4 begin

5 if ( user = 'RLS\_ADMIN' ) then

6 return '';

7 else

8 return 'owner = USER';

9 end if;

10 end;

Эта предикатная функция либо не возвращает никакого предиката, либо возвращает "owner = USER". Во время заданной сессии она **постоянно** будет возвращать один и тот же предикат. Ситуация, когда получен предикат "owner = USER", а затем в **этой же сессии** - пустой предикат "", возникнуть не может. Для того, чтобы понять, почему это **крайне необходимо** для корректного проектирования приложений с Детальным Контролем Доступа, следует понять, когда предикат связывается с запросом и как различные среды обрабатывают эту ситуацию.

### (Контекст сеанса в Oracle

### Часть 1: предопределенные контексты <http://citforum.ru/database/oracle/session_context/>)

|  |
| --- |
| Часть 2: создание своими руками http://citforum.ru/database/oracle/session\_context/part2.shtml |

* Database SQL Reference

<http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions165.htm>

## Синтаксис

Синтаксис функции Oracle / PLSQL SYS\_CONTEXT:

SYS\_CONTEXT( namespace, parameter, [ length ] )

### Параметры или аргументы

namespace пространство имен Oracle, которое уже было создано. Если пространство имен ‘USERENV’ используется, могут быть возвращены атрибуты описания текущей сессии Oracle.  
parameter допустимый атрибутом, который был установлен с помощью процедуры DBMS\_SESSION.SET\_CONTEXT.  
length не является обязательным. Это длина возвращаемого значения в байтах. Если этот параметр не указан или если недопустипый ввод, функция SYS\_CONTEXT по умолчанию будет 256 байт.

Смотри в приложении допустимые значения…………. http://oracleplsql.ru/sys\_context-function.html

Предположим, что написана следующая предикатная функция:

SQL> create or replace function rls\_examp

2 ( p\_schema in varchar2, p\_object in varchar2 )

3 return varchar2

4 as

5 begin

6 if ( sys\_context( 'myctx', 'x' ) is not null )

7 then

8 return 'x > 0';

9 else

10 return '1=0';

11 end if;

12 end;

/

Это показывает, что если атрибут "x" контекста установлен, то предикат должен иметь значение "x > 0". Если атрибут "x" контекста не установлен, то предикат должен быть "1=0". При создании таблицы T, добавьте в нее данные, политику и контекст так, как показано ниже:

SQL> **create table t** ( x int );

Table created.

SQL> **insert into t** values ( 1234 );

1 row created.

SQL> begin

2 dbms\_rls.**add\_policy**

3 ( object\_schema => user, object\_name => '**T**',

4 policy\_name => 'T\_POLICY',

5 function\_schema => user,

6 policy\_function => 'rls\_examp',

7 statement\_types => 'select' );

8 end;

9 /

PL/SQL procedure successfully completed.

SQL> create or replace procedure

2 **set\_ctx**( p\_val in varchar2 )

3 as

4 begin

5 dbms\_session.set\_context( 'myctx', 'x', p\_val );

6 end;

7 /

Procedure created.

SQL> create or replace context **myctx** using **set\_ctx**;

Context created.

Такая политика означает, что если контекст установлен, можно будет увидеть 1 строку. Если контекст не установлен, ни одной строки не будет видно. Действительно, если провести тест в SQLPLUS, непосредственно выполняя SQL, то получится следующий результат:

SQL> exec set\_ctx( null );

PL/SQL procedure successfully completed.

SQL> select \* from t;

no rows selected

SQL> exec set\_ctx( 1 ) ;

PL/SQL procedure successfully completed.

SQL> select \* from t;

X

----------

1234

Таким образом, **выбрались те данные**, которые ожидались. Динамический предикат работает так, как ожидалось. В действительности же, если использовать PL/SQL (Pro\*C или приложения, написанные на OCI, а также многие другие исполняемые среды) обнаруживается, что вышеописанный результат неверен. Создадим, например, небольшую PL/SQL-процедуру:

SQL> create or replace procedure dump\_t

2 ( some\_input in number default NULL )

3 as

4 begin

5 dbms\_output.put\_line

6 ( '\*\*\* Результат работы SELECT \* FROM T' );

7

8 for x in (select \* from t ) loop

9 dbms\_output.put\_line( x.x );

10 end loop;

11

12

13 if ( some\_input is not null )

14 then

15 dbms\_output.put\_line

16 ('\*\*\* Результат работы другого SELECT \* FROM T');

17

18 for x in (select \* from t ) loop

19 dbms\_output.put\_line( x.x );

20 end loop;

21 end if;

22 end;

23 /

Procedure created.

В первый раз простой "select \* from T" в этой процедуре выполняется, когда входной параметр не задан, и во второй раз, когда задано его некоторое значение. Давайте выполним эту процедуру и посмотрим результат:

**SQL> -- Включим вывод на экран**

**SQL> -- результат dbms\_output.put\_line**

SQL> set serveroutput on

**SQL> -- отменим установку контекста**

**SQL> -- присвоим X значение NULL**

SQL> exec set\_ctx( NULL )

PL/SQL procedure successfully completed.

**SQL> -- выполним процедуру. Заметьте, что**

**SQL> -- some\_input по умолчанию может быть NULL.**

**SQL> -- Выполнится только 1-ый select \* from t.**

**SQL> -- Как и ожидалось, выбрано НОЛЬ строк, так как**

**SQL> -- использовался предикат 1=0**

SQL> exec dump\_t

\*\*\* Результат работы SELECT \* FROM T

PL/SQL procedure successfully completed.

**SQL> -- Теперь установим значение контекста**

**-- в ненулевое значение.**

SQL> exec set\_ctx( 1 )

PL/SQL procedure successfully completed.

**SQL> -- Так как таблица t содержит 1 строку со**

**SQL> -- значением 1234, а предикат должен быть "x > 0",**

**SQL> -- когда этот атрибут установлен, то для получения**

**SQL> -- данных можно выполнить запрос к таблице T.**

**SQL> -- Чтобы убедиться в том, что результат может**

**SQL> -- оказаться неверным, выполним процедуру dump\_t с**

**SQL> --некоторым НЕНУЛЕВЫМ входным параметром.**

**SQL> -- В этом случае выполнятся оба select \* from T**

**SQL> -- Следует обратить внимание на то, что при первом**

**SQL> -- выполнении "select \* from T" никакие данные**

**SQL> -- не возвращаются!**

**SQL> -- А при втором - возвращаются!**

**SQL> -- Почему? Смотрите далее**

SQL> exec dump\_t( 0 )

\*\*\* Результат работы SELECT \* FROM T

\*\*\* Результат работы другого SELECT \* FROM T

1234

PL/SQL procedure successfully completed.

Итак, при запуске процедуры с атрибутом контекста "x", установленным в значение null, получен ожидаемый результат (так как в этой сессии процедура была запущена первый раз). Затем контекстный атрибут "x" был установлен в ненулевое значение, и результат получился "противоречивый". Первый select \* from t в процедуре снова не возвратил ни одной строки - он, скорее всего, все еще использует предикат "1=0". Второй запрос (тот, что в первый раз не выполнялся) возвратил, казалось бы, корректный результат - он, как и ожидалось, использует предикат "x > 0",.

Почему первый запрос в этой процедуре не использовал предикат, который предполагался? Это произошло из-за оптимизации, называемой "кэширование курсора". На самом деле PL/SQL и многие другие исполняемые среды не закрывают курсор по команде “закрыть”. Вышеописанный пример может быть легко воспроизведен, например, в Pro\*C, если опцию предкомпилятора "release\_cursor" оставить в значении по умолчанию NO. Если тот же самый код перекомпилировать с опцией release\_cursor=YES, то программа Pro\*C будет вести себя более похоже на запросы в SQLPLUS. Предикат, используемый DBMS\_RLS, связывается с запросом во время фазы PARSE. Первый запрос "select \* from T" разбирается во время первого выполнения хранимой процедуры - когда предикат действительно был равен "1=0". Инструмент PL/SQL кэширует этот разобранный курсор. Во второй раз при выполнении хранимой процедуры PLSQL просто повторно использует разобранный курсор из первого "select \* from T", при этом разобранный запрос имеет предикат "1=0" - предикатная функция в этот момент вообще не вызывалась. Так как процедуре передаются также некоторые входные данные, PLSQL выполнил второй запрос. Этот запрос, однако, уже не является открытым и разобранным, поэтому он разбирается во время его выполнения - когда контекстный атрибут НЕ ПУСТОЙ. Второй "select \* from t" использует связанный предикат "x>0". Отсюда и противоречивость. Так как в общем случае контроль за кэшированием этих курсоров не осуществляется, то предикатную функцию безопасности, возвращающую более 1 предиката за сессию, следует **во что бы то ни стало избегать.** В противном случае в будущем придется с большим трудом отыскивать ошибки приложения. В следующем примере я продемонстрирую, как построить предикатную функцию безопасности, которая не сможет возвратить более одного предиката за сессию. В этом случае гарантируется, что:

* Результаты одного запроса не противоречат результатам другого при работе с Детальным Контролем Доступа.
* Изменить предикат в середине сессии не удастся, иначе можно получить неопределенные непредсказуемые результаты.
* Предприняты меры для того, чтобы убедить политику безопасности в том, что имеющийся предикат - единственный для пользователя, и не пытаться возвратить предикат, специализированный под пользователя, работающего в текущем окружении.

Я настоятельно советую создавать все нетривиальные предикатные функции так же, как показано в следующем примере. Это позволит создавать хорошо работающие, предикабельные приложения, поддерживающие Детальный Контроль Доступа.

Следует добавить, что в некоторых случаях изменение предиката в середине сессии желательно. Для достижения наилучших результатов клиентские приложения, имеющие доступ к объектам, которые поддерживаются политикой, позволяющей изменять предикаты в середине сессии **должны быть написаны в особой форме**. Например, во избежание кэширования курсора в PLSQL необходимо написать приложение, полностью использующее динамический sql. Если используется этот динамический предикатный метод, то необходимо иметь в виду, что результаты будут зависеть от того, как написано клиентское приложение. Поэтому не следует применять политику безопасности с использованием этой возможности. Мы не будем рассматривать использование всех возможностей DBMS\_RLS, а лучше сконцентрируемся на ее конкретном использовании - для защиты данных.

**Пример 2. Использование контекстов приложения**

Необходимо, например, реализовать Политику Безопасности в подсистеме “Кадры” (Human Resources Security Policy ). В этом примере будут использоваться таблицы EMP и DEPT демонстрационного пользователя SCOTT/TIGER и добавится еще одна таблица, которая позволит назначить человека на должность контролера. Далее перечислены требования:

* Менеджер отдела может:
  + ЧИТАТЬ свою запись, записи всех сотрудников, которые ему подчиняются и всех сотрудников, которые в свою очередь подчиняются этим подчиненным (по иерархии)
  + ОБНОВЛЯТЬ записи всех служащих, которые подчиняются ему непосредственно.
* Служащий может:
  + ЧИТАТЬ свою запись.
* Контролер может:
  + ЧИТАТЬ все записи по отделу, в котором он работает (В этом приложении контролер работает в одном из отделов)
  + ОБНОВЛЯТЬ любую запись заданного отдела
  + ВСТАВЛЯТЬ в заданный отдел.
  + УДАЛЯТЬ из заданного отдела.

Как было сказано ранее, приложение будет использовать существующие таблицы EMP и DEPT пользователя SCOTT и добавочную таблицу HR\_REPS для связи контролера с отделом. Схема будет выглядеть следующим образом:

**SQL> -- создадим демонстрационную схему.**

**SQL> -- Она основывается на таблицах EMP и DEPT,**

**SQL> -- владелец которых scott. Добавим в схему**

**SQL> -- описатель RI (идентификатор контролера) и**

**SQL> -- переименуем значения поля ENAME в таблице EMP**

**SQL> -- так, чтобы они соответствовали именам**

**SQL> -- пользователей тестируемой базы данных**

**SQL> -- (например: пользователю RLS\_KING соответствует**

**SQL> -- имя пользователя RLS\_KING в таблице emp)**

Дайте права

Rls\_admin

Ot scott

Grant select on emp to rls\_admin

SQL> create table dept as select \* from scott.dept;

Table created.

SQL> alter table dept add constraint dept\_pk primary

2 key(deptno);

Table altered.

SQL> create table emp\_base\_table as

2 select \* from scott.emp;

Table created.

SQL> update emp\_base\_table

2 set ename = 'RLS\_' || ename;

14 rows updated.

SQL> alter table emp\_base\_table

2 add constraint emp\_pk primary key(empno);

Table altered.

SQL> alter table emp\_base\_table

2 add constraint emp\_fk\_to\_dept

3 foreign key (deptno) references dept(deptno);

Table altered.

**SQL> -- создадим индексы, которые будут использоваться**

**SQL> -- функцией контекста приложения для повышения**

**SQL> -- производительности.**

**SQL> -- Необходимо быстро определить, является ли**

**SQL> -- некоторый пользователь mgr (менеджером) отдела.**

**SQL> -- Кроме того, необходимо быстро конвертировать имя**

**SQL> -- пользователя в empno**

SQL> create index emp\_mgr\_deptno\_idx

2 on emp\_base\_table(mgr);

Index created.

SQL> create unique index emp\_ename\_idx

2 on emp\_base\_table(ename);

Index created.

**SQL> -- Кроме того, создадим представление EMP**

**SQL> -- на основе запроса**

**SQL> -- select \* from emp\_base\_table.**

**SQL> -- К этому ПРЕДСТАВЛЕНИЮ**

**SQL> -- будет применена политика, и через него**

**SQL> -- приложения будут**

**SQL> -- запрашивать/вставлять/обновлять**

**SQL> -- и так далее.**

**grant create any view to rls; -- иначе не идет**

SQL> create view emp

2 as select \* from emp\_base\_table;

View created.

**SQL> -- создадим таблицу для управления HR\_REPS.**

**SQL> -- Для этого будет использоваться**

**SQL> -- INDEX ORGANIZED TABLE, так как всегда будет**

**SQL> -- выполняться запрос только такого типа,**

**SQL> -- как**

**SQL> -- "select \* from hr\_reps**

**SQL> -- where username = X and deptno = Y".**

**SQL> -- В использовании таблицы нет необходимости,**

**SQL> -- достаточно использовать только индекс.**

SQL> create table hr\_reps

2 ( username varchar2(30),

3 deptnonumber,

4 primary key(username,deptno)

5 )

6 organization index;

Table created.

**SQL> -- Свяжем HR Reps с отделами.**

**SQL> -- KING может видеть все отделы.**

SQL> insert into hr\_reps values ( 'RLS\_JONES', 10 );

SQL> insert into hr\_reps values ( 'RLS\_BLAKE', 20 );

SQL> insert into hr\_reps values ( 'RLS\_CLARK', 30 );

SQL> insert into hr\_reps values ( 'RLS\_KING', 10 );

SQL> insert into hr\_reps values ( 'RLS\_KING', 20 );

SQL> insert into hr\_reps values ( 'RLS\_KING', 30 );

SQL> insert into hr\_reps values ( 'RLS', 10 );

SQL> commit;

Commit complete.

Теперь, когда для приложения созданы таблицы EMP, DEPT и HR\_REPS, напишем процедуру, которая позволит установить значения контекста приложения. Контекст приложения будет состоять из двух элементов данных - EMPNO пользователя, присоединенного в данный момент, и его роль (одна из EMP, MGR или HR\_REP). Процедура формирования динамического предиката будет использовать роль, хранящуюся в контексте приложения для того, чтобы определить, как должно выглядеть условие where для конкретного пользователя.

Для осуществления этой задачи используются таблицы EMP\_BASE\_TABLE и HR\_REP. Тогда возникает вопрос "зачем использовать таблицу EMP\_BASE\_TABLE и представление EMP, если можно просто выполнить select \* from emp\_base\_table?" По двум причинам:

**1.** Данные таблицы служащих используются для реализации политики безопасности.

**2.** Эта таблица считывается при создании контекста приложения.

Для того, чтобы прочитать данные о служащем, необходимо установить значения контекста приложения, для того, чтобы установить значения контекста приложения, необходимо прочитать данные о служащем. Это проблема курицы и яйца. Одним из вариантов ее решения является создание представления, которым будут пользоваться все приложения (представление EMP) и реализация безопасности этого представления. Для осуществления этой задачи политика безопасности будет пользоваться исходной таблицей EMP\_BASE\_TABLE. Из таблицы EMP\_BASE\_TABLE можно узнать, кто является менеджером некоторого отдела и кто подчиняется некоторому пользователю. Приложение и конечные пользователи никогда не будут обращаться к таблице EMP\_BASE\_TABLE - она будет использоваться только политикой безопасности. После этого создается процедура установки значений контекста:

**SQL> -- это процедура, "несущая ответственность" за**

**SQL> -- создание контекста приложения. Она реализует**

**SQL> -- политику путем установки привилегий,**

**SQL> -- выданных пользователю в контексте приложения.**

SQL> create or replace procedure

2 set\_role( p\_roleName in varchar2 default null )

3 as

4 l\_empno number;

5 l\_cnt number;

6 l\_roleName varchar2(40) default

upper(p\_roleName);

7 begin

8 if ( sys\_context( 'Hr\_App\_Ctx', 'RoleName' )

is NOT NULL )

9 then

**10 -- прерывание запроса. Изменение роли требует**

**11 -- изменения предиката, связанного с запросом. Из-за**

**12 -- кеширования курсоров клиентском приложении нельзя**

**13 -- гарантировать, что не существует других уже**

**14 -- разобранных запросов с предикатами из текущей**

**15 -- роли. Если, например, роль уже установлена в**

**16 -- значение MGR и разобрано несколько запросов, а**

**17 -- теперь попытаться изменить роль на EMP, то**

**18 -- разобранные запросы все еще будут использовать**

**19 -- предикаты MGR, а не EMP.**

20 raise\_application\_error(-2000,'Роль уже установлена');

21 end if;

**22 -- Так как используется пользователь СЕССИИ, а**

**23 -- не ТЕКУЩИЙ пользователь и, кроме того, empno**

**24 -- текущего пользователя используется достаточно**

**25 -- часто, то он будет храниться в этом Контексте.**

**26 -- Пользователь СЕССИИ - это имя присоединенного**

**27 -- в данный момент пользователя. ТЕКУЩИЙ же ,**

**28 -- пользователь обладает привилегиями на выполнение**

**29 -- запроса, им может быть владелец процедуры,**

30 **-- т.е. не присоединенный пользователь!!!**

31 select empno into l\_empno

32 from emp\_base\_table

33 where ename=

sys\_context('userenv', 'session\_user');

34 dbms\_session.set\_context

('Hr\_App\_Ctx','Empno',l\_empno);

35

36 if ( l\_roleName = 'EMP' )

37 then

**38 -- любой может воспользоваться ролью EMP**

39 dbms\_session.set\_context

('Hr\_App\_Ctx','RoleName','EMP');

40 elsif ( l\_roleName = 'MGR' )

41 then

**42 -- проверим, пользователь - MGR (менеджер), и**

**43 -- если нет, то выдадим сообщение об ошибке и прервем**

44 **-- выполнение. Пользователь может попытаться снова.**

45 select count(\*) into l\_cnt

46 from dual

47 where exists

48 ( select NULL

49 from emp\_base\_table

50 where mgr = to\_number

(sys\_context('Hr\_App\_Ctx','Empno'))

51 );

52 if ( l\_cnt = 0 )

53 then

54 raise\_application\_error

( -20002, 'Вы не менеджер' );

55 end if;

56 dbms\_session.set\_context

( 'Hr\_App\_Ctx', 'RoleName', 'MGR' );

57 elsif ( l\_roleName = 'HR\_REP' )

58 then

**59 -- проверим, является ли пользователь HR\_REP, если**

**60 -- нет, выдадим сообщение об ошибке и прервем**

61 -- выполнение.**Пользователь может попытаться снова.**

62 select count(\*) into l\_cnt

63 from dual

64 where exists

65 ( select NULL

66 from hr\_reps

67 where username =

sys\_context( 'userenv', 'session\_user' )

68 );

69

70 if ( l\_cnt = 0 )

71 then

72 raise\_application\_error( -20002, 'Вы не контролер' );

73 end if;

74 dbms\_session.set\_context

('Hr\_App\_Ctx','RoleName','HR\_REP');

75 else

76 raise\_application\_error( -20003, 'Роль

77 '||l\_roleName||' невозможно распознать' );

78 end if;

79 end;

80 /

Procedure created.

SQL> grant execute on set\_role to public

2 /

Grant succeeded.

Итак, до этого момента сделано: создана процедура, которая принимает имя роли в виде параметра. В начале этой процедуры обеспечивается, чтобы атрибут RoleName еще не был установлен. Так как в политике безопасности будут возвращаться различные предикаты, зависящие от значения RoleName, то нельзя разрешать пользователю изменять его роль, если она уже установлена. Если допустить изменение роли, то может возникнуть проблема, связанная с кэшированием курсора и ‘old’-предикатами. Далее посмотрим на EMPNO текущего пользователя. Эта процедура выполняет две операции:

1. Проверяет, является ли пользователь служащим - при получении ошибки "NO DATA FOUND", становится известным, что он не служащий. Поэтому значения его контекста никогда не будут установлены, а сам пользователь не увидит никаких данных.

2. Помещает часто используемое значение в контекст приложения. Теперь через EMPNO текущего пользователя можно быстро получить доступ к таблице EMP - это будет реализовано в предикатной функции, описанной ниже.

Далее процедура переходит к проверке, разрешено ли текущему пользователю получить роль, которую он запрашивает. Каждый, кто запрашивает роль EMP, может установить ее. Только тот, кто действительно управляет другими людьми, может установить роль MGR. Только тот, о ком есть данные в таблице HR\_REPS, может установить роль HR\_REP.

Далее создадим объект контекста приложения базы данных и свяжем его с только что созданной процедурой SET\_HR\_APP\_DEPT:

**SQL> -- Создание контекста приложения. Имя контекста -**

**SQL> -- HR\_APP\_CTX. Процедура, с которой он связан**

**SQL> -- в данном случае - это SET\_ROLE**

SQL> create or replace context Hr\_App\_Ctx using SET\_ROLE

2 /

Context created.

Итак, теперь созданы контекст с именем Hr\_App\_Ctx и процедура для его установки. Важно иметь в виду, что, так как контекст связан с процедурой Set\_Role, то она является единственным средством установки значений контекста. Если, например, попытаться установить в этом контексте RoleName с целью получения доступа к запрещенным данным, то обнаружится, что сделать это нельзя:

**SQL> REM Выполнение следующих далее операций будет**

**SQL> REM ПРЕРВАНО. Это показывает, что**

**SQL> REM процедура dbms\_session.set\_context**

**SQL> REM может установить контекст Hr\_App\_Ctx только**

**SQL> REM через процедуру SET\_ROLE**

SQL> exec dbms\_session.set\_context

( 'Hr\_App\_Ctx', 'RoleName', 'MGR' );

BEGIN dbms\_session.set\_context

( 'Hr\_App\_Ctx', 'RoleName', 'MGR' ); END;

\*

ERROR at line 1:

**ORA-01031: привилегий недостаточно**

ORA-06512: at "SYS.DBMS\_SESSION", line 55

ORA-06512: at line 1

Этот атрибут контекста отвечает за безопасность. Значения контекста можно установить только через процедуру. В этом случае гарантируется, что, если значение и существует, то оно проверено и установлено. Теперь для тестирования логики процедуры попытаемся выполнить хранимую процедуру под различными пользователями и посмотрим, какие роли можно установить и какие значения существуют в контексте.

Создать этого пользователя.

От имени sys

SQL> grant select on sys.v\_$context to rls\_smith;

Grant succeeded.

SQL> connect rls\_smith/rls\_smith

Connected.

SQL> set serveroutput on

SQL> show user

USER is "**RLS\_SMITH**"

SQL> exec rls.set\_role( 'Mgr' )

BEGIN rls.set\_role( 'Mgr' ); END;

\*

ERROR at line 1:

**ORA-20002: Вы не менеджер**

ORA-06512: at "RLS.SET\_ROLE", line 53

ORA-06512: at line 1

До настоящего времени можно было увидеть, что процедура доступна RLS\_SMITH, но не позволяет ему установить контекст ‘MGR’ до тех пор, пока он фактически не станет менеджером. Если теперь посмотреть на этот контекст через динамическое представление v$context, то можно увидеть:

SQL> select \* from v$context;

NAMESPACE ATTRIBUTE VALUE

---------- ---------- -----

HR\_APP\_CTX EMPNO 7369

что RLS\_SMITH действительно мог получить список номеров его служащих, но не атрибут RoleName. Теперь повторим операцию с RoleName, задав правильное значение для RLS\_SMITH:

SQL> exec rls.set\_role( 'emp' )

PL/SQL procedure successfully completed.

SQL> select \* from v$context;

NAMESPACE ATTRIBUTE VALUE

---------- ---------- -----

HR\_APP\_CTX ROLENAME EMP

HR\_APP\_CTX EMPNO 7369

Можно увидеть, что теперь все работает, как ожидалось. RLS\_SMITH может получить номер его служащего и атрибут RoleName, установленный в контексте HR\_APP\_CTX. Для дальнейшего тестирования логики выполним:

SQL> exec rls.set\_role( 'emp' )

BEGIN rls.set\_role( 'emp' ); END;

\*

ERROR at line 1:

**ORA-20001: Роль уже установлена**

ORA-06512: at "RLS.SET\_ROLE", line 18

ORA-06512: at line 1

SQL> select \* from v$context;

NAMESPACE ATTRIBUTE VALUE

---------- ---------- -----

HR\_APP\_CTX ROLENAME EMP

HR\_APP\_CTX EMPNO 7369

Результат показывает, что логика не позволяет пользователю изменить его роль после того, как она уже установлена, далее будет видно, что в контексте остаются прежние значения. Это сообщение об ошибке не критично, оно значит только то, что нельзя изменить роль во время сессии.

Далее присоединимся различными пользователями, просмотрим результаты работы процедуры и рассмотрим различные способы получения значений контекста сессии:

List\_context <http://docstore.mik.ua/orelly/oracle/guide8i/ch08_04.htm>

### LIST\_CONTEXT: Obtaining the List of Defined Context Attributes

PROCEDURE DBMS\_SESSION.LIST\_CONTEXT (

list OUT DBMS\_SESSION.AppCtxTabTyp,

lsize OUT number);

where lsize is the number of elements in list , and list is an index-by table of records. Each record has this format:

TYPE DBMS\_SESSSION.AppCtxRecTyp IS RECORD (

namespace VARCHAR2(30),

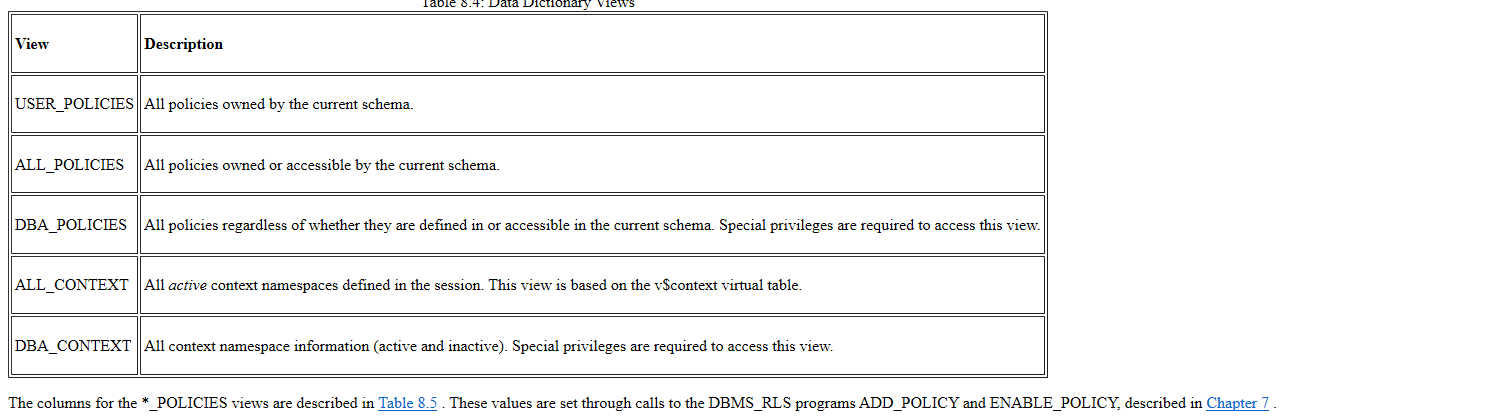
attribute VARCHAR2(30),

value VARCHAR2(4000));

where namespace and attribute have the meanings described for SYS\_CONTEXT.

Datele despre context

https://docstore.mik.ua/orelly/oracle/guide8i/ch08\_04.htm#ch08-53726



SQL> connect rls\_blake/rls\_blake

Connected.

SQL> exec rls.set\_role( 'mgr' )

PL/SQL procedure successfully completed.

SQL> set serveroutput on

SQL> declare

2 l\_AppCtx dbms\_session.AppCtxTabTyp;

3 l\_sizenumber;

4 begin

5 **dbms\_session.list\_context**

( l\_AppCtx, l\_size );

6 for i in 1 .. l\_size loop

7 dbms\_output.put( l\_AppCtx(i).namespace || '.' );

8 dbms\_output.put( l\_AppCtx(i).attribute || ' = ' );

9 dbms\_output.put\_line( l\_AppCtx(i).value );

10 end loop;

11 end;

/

Для RLS не проходит

**HR\_APP\_CTX.ROLENAME = MGR**

**HR\_APP\_CTX.EMPNO = 7698**

PL/SQL procedure successfully completed.

На этот раз присоединимся как RLS\_BLAKE, заведующий отделом 30. Когда RLS\_BLAKE вызывает процедуру Set\_Role с параметром RoleName = ‘MGR’, видно, что контекст установлен правильно: он - менеджер, и количество его служащих установлено. Кроме того, этот тест показывает, как просмотреть пары значений атрибутов в контексте сессии с помощью пакета dbms\_session.list\_context. Функции этого пакета может вызывать любой пользователь (так как происходит обращение к представлению sys.v$context, которое использовалось ранее), поэтому все пользователи могут использовать такой метод для проверки значений контекста сессии.

Теперь, так как контекст сессии установлен надлежащим образом, можно подготовиться к созданию процедуры политики безопасности. Эта процедура будет вызываться средствами базы данных во время их работы для формирования динамического предиката. Динамический предикат ограничивает такие возможности пользователя, как чтение и запись.

**SQL> -- создадим предикатный пакет**

**SQL> -- (пакет для генерации условия where) приложения HR.**

**SQL> -- Для создания уникального условия**

**SQL> -- where каждой операции**

**SQL> -- SELECT/INSERT/UPDATE/DELETE**

**SQL> -- должна соответствовать отдельная функция.**

SQL> create or replace package hr\_predicate\_pkg

2 as

3 function select\_function( p\_schema in varchar2,

4 p\_object in varchar2 ) return varchar2;

5

6 function update\_function( p\_schema in varchar2,

7 p\_object in varchar2 ) return varchar2;

8 function insert\_delete\_function

9 ( p\_schema in varchar2,

10 p\_object in varchar2 ) return varchar2;

11 end;

12 /

Для каждой операции Языка Манипулирования Данными (Data Manipulation Language - DML) напишем предикаты, несколько отличающиеся друг от друга. Каждая DML-операция будет подчиняться своим правилам. Это позволит предложению DELETE видеть набор данных (в данном примере меньший), отличающийся от набора данных, видимого при выполнении предложения SELECT. Далее показана реализация тела пакета HR\_PREDICATE\_PKG:

SQL> create or replace package body hr\_predicate\_pkg

2 as

3

4 g\_app\_ctx constant varchar2(30)

5 default 'Hr\_App\_Ctx';

6 g\_sel\_pred varchar2(1024) default NULL;

7 g\_upd\_pred varchar2(1024) default NULL;

8 g\_ins\_del\_pred varchar2(1024) default NULL;

9

10

11 function **select\_function**( p\_schema in varchar2,

12 p\_object in varchar2 ) return varchar2

13 is

14 begin

15 if ( g\_sel\_pred is NULL )

16 then

17 if ( sys\_context( g\_app\_ctx, 'RoleName' ) = 'EMP' )

18 then

19 g\_sel\_pred :=

20 'empno = sys\_context(''Hr\_App\_Ctx'',''EmpNo'')';

21 elsif ( sys\_context

22 ( g\_app\_ctx, 'RoleName' ) = 'MGR' )

23 then

24 g\_sel\_pred :=

25 ' empno in ( select empno

26 from emp\_base\_table

27 start with empno =

28 sys\_context(''Hr\_App\_Ctx'',''EmpNo'')

29 connect by prior empno = mgr)';

30 elsif (sys\_context(g\_app\_ctx,'RoleName') =

31 'HR\_REP')

32 then

33 g\_sel\_pred := 'deptno in

34 ( select deptno

35 from hr\_reps

36 where username =

37 sys\_context(''userenv'',''session\_user'') )';

38 else

39 raise\_application\_error

40 (-20005,'Роль не установлена');

41 end if;

42 end if;

43

44 return g\_sel\_pred;

45 end;

46

47 function **update\_function**( p\_schema in varchar2,

48 p\_object in varchar2 ) return varchar2

49 is

50 begin

51 if ( g\_upd\_pred is NULL )

52 then

53 if (sys\_context(g\_app\_ctx,'RoleName') = 'EMP')

54 then

55 g\_upd\_pred := '1=0';

56

57 elsif ( sys\_context(g\_app\_ctx,'RoleName') = 'MGR')

58 then

59 g\_upd\_pred :=

60 ' empno in ( select empno

61 from emp\_base\_table

62 where mgr =

63 sys\_context(''Hr\_App\_Ctx'',''EmpNo'') )';

64

65 elsif (sys\_context(g\_app\_ctx,'RoleName') = 'HR\_REP')

66 then

67 g\_upd\_pred := 'deptno in

68 ( select deptno

69 from hr\_reps

70 where username =

71 sys\_context(''userenv'',''session\_user'') )';

72 else

73 raise\_application\_error

74 (-20005,'Роль не установлена');

75 end if;

76 end if;

77

78 return g\_upd\_pred;

79 end;

80 function **insert\_delete\_function**

81 (p\_schema in varchar2,

82 p\_object in varchar2 ) return varchar2

83 is

84 begin

85 if (g\_ins\_del\_pred is NULL ) then

86 if (sys\_context(g\_app\_ctx,'RoleName')

87 in ('EMP','MGR'))

88 then

89 g\_ins\_del\_pred := '1=0';

90 elsif (sys\_context(g\_app\_ctx,'RoleName') = 'HR\_REP')

91 then

92 g\_upd\_pred := 'deptno in

93 ( select deptno

94 from hr\_reps

95 where username =

96 sys\_context(''userenv'',''session\_user'') )';

97 else

98 raise\_application\_error

99 (-20005,'Роль не установлена');

100 end if;

101 end if;

102 return g\_ins\_del\_pred;

103 end;

104 end;

/

select \* from emp where

deptno in

( select deptno

from hr\_reps

where username =

sys\_context('userenv','session\_user') );

Package body created.

Ранее, без Детального Контроля Доступа использование одной таблицы с описанными выше тремя предикатами можно было достичь при помощи 3-х представлений - по одному на каждую операцию SELECT, UPDATE и INSERT/DELETE. Детальный Контроль Доступа позволяет сократить количество объектов до одной таблицы с динамическим предикатом.

Вспомните логику, описанную ранее:

11 function select\_function( p\_schema in varchar2,

12 p\_object in varchar2 ) return varchar2

13 is

14 begin

15

16 if ( g\_sel\_pred is NULL )

17 then

**логика для присвоения значения g\_sel\_pred,**

**глобальной переменной,**

**объявленной в теле пакета**

42 end if;

43

44 return g\_sel\_pred;

45 end;

В этой функции переменная g\_sel\_pred устанавливается в непустое значение точно один раз за сессию. Если при предыдущем вызове этой предикатной функции предикат уже установлен - он просто возвращается снова. В этом есть два преимущества:

* Это быстрее, чем вычисление и создание предиката в процедуре для каждого разбираемого предложения. Размер исполняемого кода уменьшается.
* Предотвращается возможность возврата различных значений предиката в пределах одной сессии. Как отмечено в первом рассмотренном выше примере, результаты возврата различных предикатов в пределах одной сессии могут быть противоречивыми, а при использовании этой технологии предотвращается возникновение такой ситуации.

Последний этап этого процесса заключается в связывании предикатов с каждой DML-операцией и самой таблицей EMP. Далее показана реализация этой операции:

**SQL> -- Добавим политику к представлению EMP.**

**SQL> -- Каждая функция пакета HR\_PREDICATE\_PKG**

**SQL> -- связывается с таблицей для выполнения операций**

**SQL> -- SELECT/INSERTUPDATE/DELETE.**

**SQL> -- При INSERT и UPDATE установим флаг**

**SQL> -- 'update\_check' в значение TRUE.**

**SQL> -- Это очень похоже на создание**

**SQL> -- представления с 'CHECK OPTION'**

**SQL> -- В этом случае обеспечивается, что данные,**

**SQL> -- создаваемые в базе данных, - это те данные,**

**SQL> -- которые пользователь может увидеть.**

**Из файла runpolicy**

SQL> begin

2 dbms\_rls.add\_policy

3 ( object\_schema => 'RLS',

4 object\_name => 'EMP',

5 policy\_name => 'HR\_APP\_SELECT\_POLICY',

6 function\_schema => 'RLS',

7 policy\_function =>

8 'HR\_PREDICATE\_PKG.SELECT\_FUNCTION',

9 statement\_types => 'select' );

10 end;

11 /

PL/SQL procedure successfully completed.

SQL> begin

2 dbms\_rls.add\_policy

3 ( object\_schema => 'RLS',

4 object\_name => 'EMP',

5 policy\_name => 'HR\_APP\_UPDATE\_POLICY',

6 function\_schema => 'RLS',

7 policy\_function =>

8 'HR\_PREDICATE\_PKG.UPDATE\_FUNCTION',

9 statement\_types => 'update' ,

10 update\_check => TRUE );

11 end;

12 /

PL/SQL procedure successfully completed.

SQL> begin

2 dbms\_rls.add\_policy

3 ( object\_schema => 'RLS',

4 object\_name => 'EMP',

5 policy\_name =>

6 'HR\_APP\_INSERT\_DELETE\_POLICY',

7 function\_schema => 'RLS',

8 policy\_function =>

9 'HR\_PREDICATE\_PKG.INSERT\_DELETE\_FUNCTION',

10 statement\_types => 'insert, delete' ,

11 update\_check => TRUE );

12 end;

13 /

PL/SQL procedure successfully completed.

Таким образом, с каждой DML-операцией связывается различная предикатная функция. Когда пользователь запрашивает данные таблицы EMP, к запросу будет присоединяться предикат, сформированный пакетной функцией hr\_predicate\_pkg.select. Когда пользователь обновляет таблицу, будет использоваться пакетная функция update и так далее.

Теперь протестируем приложение. Для этого создадим пакет HR\_APP. Этот пакет представляет собой приложение. В него входят функции для выполнения:

* Запроса данных
* Обновления данных
* Удаления данных
* Вставки новых данных

Присоединимся различными пользователями с различными ролями и проследим за поведением нашего приложения. Это позволит увидеть, как работает Детальный Контроль Доступа.

Ниже показана спецификация приложения:

SQL> create or replace package hr\_app

2 as

3 procedure listEmps;

4

5 procedure updateSal;

6

7 procedure deleteAll;

8

9 procedure insertNew( p\_deptno in number );

10 end;

11 /

Package created.

Тело пакета:

SQL> create or replace package body hr\_app

2 as

3

4 procedure listEmps

5 as

6 l\_AppCtx dbms\_session.AppCtxTabTyp;

7 l\_size number;

8 begin

9 dbms\_output.put\_line

( '--- Контекст Сессии -------' );

10 dbms\_session.list\_context( l\_AppCtx, l\_size );

11 for i in 1 .. l\_size loop

12 dbms\_output.put( l\_AppCtx(i).namespace || '.' );

13 dbms\_output.put( l\_AppCtx(i).attribute || ' = ' );

14 dbms\_output.put\_line( l\_AppCtx(i).value );

15 end loop;

16 dbms\_output.put\_line

17 ('--Данные таблицы Emp, которые можно увидеть--');

18 for x in ( select ename, sal, dname

19 from emp, dept

20 where emp.deptno = dept.deptno )

21 loop

22 dbms\_output.put\_line

23 ( x.ename || ',' || x.sal || ',' || x.dname );

24 end loop;

25 end;

26

27 procedure updateSal

28 is

29 begin

30 update emp set sal = 9999;

31 dbms\_output.put\_line

32 ( sql%rowcount || ' строк обновлено' );

33 end;

34 procedure deleteAll

35 is

36 begin

37 delete from emp where empno

38 <> sys\_context( 'Hr\_app\_Ctx', 'EMPNO' );

39 dbms\_output.put\_line

( sql%rowcount || ' строк удалено' );

40 end;

41 procedure insertNew( p\_deptno in number )

42 as

43 begin

44 insert into emp ( empno, deptno, sal )

values ( 123, p\_deptno, 1111 );

45 end;

46

47 end hr\_app;

48 /

Package body created.

SQL> grant execute on hr\_app to public

2 /

Grant succeeded.

Итак, "приложение" создано. Процедура listEmps показывает все записи, которые можно увидеть через представление EMP. Процедура updateSal обновляет каждую запись, к которой можно получить доступ. Процедура deleteAll удаляет каждую запись, к которой можно получить доступ, за исключением записи, идентифицирующей пользователя. Процедура insertNew создает нового служащего в заданном отделе. Это приложение просто тестирует все DML-операции над представлением EMP, которые можно было бы выполнить.

Теперь присоединимся различными пользователями и протестируем функциональность приложения.

**SQL> -- теперь посмотрим,**

**SQL> -- что произойдет при попытке выполнить**

**SQL> -- что-нибудь без получения роли**

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии ---

HR\_APP\_CTX.EMPNO = 7876

--Данные таблицы Emp, которые можно увидеть--

BEGIN rls.hr\_app.listEmps; END;

\*

ERROR at line 1:

**ORA-28112: ошибка при выполнении функции политики**

ORA-06512: at "RLS.HR\_APP", line 18

ORA-06512: at line 1

Появилось сообщение об ошибке. Это сообщение возникло, потому что так написана предикатная функция:

function select\_function( p\_schema in varchar2,

p\_object in varchar2 ) return varchar2

is

begin

if ( g\_sel\_pred is NULL )

then

if ( sys\_context( g\_app\_ctx, 'RoleName' )

= 'EMP' )

then

…

elsif ( sys\_context( g\_app\_ctx, 'RoleName' )

= 'MGR' )

then

…

elsif ( sys\_context( g\_app\_ctx, 'RoleName' )

= 'HR\_REP' )

then

…

else

**raise\_application\_error**

**( -20005, 'Роль не установлена' );**

end if;

end if;

return g\_sel\_pred;

end;

Полученный результат - это результат выполнения raise\_application\_error в предикатной функции. Конечный пользователь получает сообщение об ошибке ORA-28112. Далее, в следующей секции, мы рассмотрим, как обнаружить эти ошибки и отладить их.

Далее установим такую роль, чтобы можно было что-нибудь сделать, и попробуем выполнить эти же операции:

**SQL> -- Теперь установим корректную роль**

**SQL> -- и выполним что-нибудь**

SQL> exec rls.set\_role( 'emp' );

PL/SQL procedure successfully completed.

**SQL> -- посмотрим контекст и данные, которые**

**SQL> -- можно видеть - это только одна запись**

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии ----

HR\_APP\_CTX.ROLENAME = EMP

HR\_APP\_CTX.EMPNO = 7876

--Данные таблицы Emp, которые можно увидеть--

RLS\_ADAMS,1100,RESEARCH

PL/SQL procedure successfully completed.

**SQL> -- несмотря на то, что данные "видно"**

**SQL> -- их нельзя "изменить".**

SQL> exec rls.hr\_app.updateSal

0 rows updated

PL/SQL procedure successfully completed.

**SQL> -- нельзя удалить никакую информацию**

SQL> exec rls.hr\_app.deleteAll

0 rows deleted

PL/SQL procedure successfully completed.

**SQL> -- нельзя ничего создать**

SQL> exec rls.hr\_app.insertNew(20)

BEGIN rls.hr\_app.insertNew(20); END;

\*

ERROR at line 1:

**ORA-28115: нарушение политики с опцией проверки**

ORA-06512: at "RLS.HR\_APP", line 44

ORA-06512: at line 1

Итак, результат показывает, что можно видеть только ту запись, которая соответствует текущему пользователю, нельзя изменить какие бы то ни было данные, нельзя удалить записи, и вставка нового служащего также завершается неудачно. Происходит как раз то, что и предполагалось. В самом приложении, HR\_APP не делается ничего специально для выполнения этих правил, теперь это делает база данных.

Далее присоединимся как MGR и посмотрим, что произойдет:

**SQL> -- Присоединимся как менеджер**

SQL> connect rls\_jones/rls\_jones

Connected.

**SQL> -- Включим возможность вывода на экран из PLSQL**

SQL> set serveroutput on

**SQL> -- Для начала попробуем стать менеджером**

**SQL> -- мы являемся менеджером,**

**SQL> -- так как на этот раз нам разрешено**

**SQL> -- стать им**

SQL> exec rls.set\_role( 'mgr' )

PL/SQL procedure successfully completed.

**SQL> -- посмотрим контекст и данные, которые**

**SQL> -- можно видеть. На этот раз - более одной строки.**

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии -----

HR\_APP\_CTX.ROLENAME = MGR

HR\_APP\_CTX.EMPNO = 7566

--Данные таблицы Emp, которые можно увидеть--

RLS\_SMITH,800,RESEARCH

RLS\_JONES,2975,RESEARCH

RLS\_SCOTT,3000,RESEARCH

RLS\_ADAMS,1100,RESEARCH

RLS\_FORD,3000,RESEARCH

PL/SQL procedure successfully completed.

**SQL> -- Следующая операция показывает,**

**SQL> -- что некоторые записи можно**

**SQL> -- изменить. Затем снова выполним listEmps**

**SQL> -- для того, чтобы увидеть,**

**SQL> -- какие строки изменились**

**SQL> -- (только те, которые подчинены напрямую)**

SQL> exec rls.hr\_app.updateSal

2 rows updated

PL/SQL procedure successfully completed.

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии ----

HR\_APP\_CTX.ROLENAME = MGR

HR\_APP\_CTX.EMPNO = 7566

--Данные таблицы Emp, которые можно увидеть--

RLS\_SMITH,800,RESEARCH

RLS\_JONES,2975,RESEARCH

RLS\_SCOTT,9999,RESEARCH

RLS\_ADAMS,1100,RESEARCH

RLS\_FORD,9999,RESEARCH

PL/SQL procedure successfully completed.

**SQL> -- так как мы не являемся контролером, то,**

**SQL> -- согласно заданным правилам,**

**SQL> -- нельзя никого удалить**

SQL> exec rls.hr\_app.deleteAll

0 rows deleted

PL/SQL procedure successfully completed.

**SQL> -- так как мы не являемся контролером, то,**

**SQL> -- согласно заданным правилам,**

**SQL> -- нельзя никого вставить**

SQL> exec rls.hr\_app.insertNew(20)

BEGIN rls.hr\_app.insertNew(20); END;

\*

ERROR at line 1:

**ORA-28115: нарушение политики с опцией проверки**

ORA-06512: at "RLS.HR\_APP", line 44

ORA-06512: at line 1

SQL> @rls\_adams

**SQL> -- Присоединимся как служащий,**

**SQL> -- у которого нет способности управлять.**

SQL> connect rls\_adams/rls\_adams

Connected.

SQL> set serveroutput on

**SQL> -- Сначала попробуем стать менеджером**

**SQL> -- Мы не являемся менеджером, поэтому стать им**

**SQL> -- не разрешается**

SQL> exec rls.set\_role( 'mgr' )

BEGIN rls.set\_role( 'mgr' ); END;

\*

ERROR at line 1:

**ORA-20002: Вы не менеджер**

ORA-06512: at "RLS.SET\_ROLE", line 53

ORA-06512: at line 1

Таким образом, результат показывает, что нельзя получить роль, не предназначенную для текущего пользователя. Чтобы убедиться, что ни к каким данным нет доступа, попробуем теперь запросить что-нибудь, и посмотрим, что произойдет:

Таким образом, теперь нам, как MGR, можно:

* Просматривать не только свои данные. Видно всех, кто нам подчинен и тех, кто подчинен нашим подчиненным и так далее (по иерархии).
* Изменять некоторые данные. Точнее, можно изменять только те записи, которые относятся к нашим непосредственным подчиненным - что и требуется.
* Ни над какими данными все еще нельзя выполнить DELETE или INSERT - что и требуется

И, наконец, присоединимся как контролер и посмотрим на поведение приложения при работе с этой ролью:

**SQL> -- Присоединимся как контролер**

SQL> connect rls\_king/rls\_king

Connected.

**SQL> -- Подключим возможность вывода на экран из PLSQL**

SQL> set serveroutput on

**SQL> -- Для начала, попробуем стать контролером**

**SQL> -- Теперь мы являемся контролером, так как**

**SQL> -- стать им разрешено**

SQL> exec rls.set\_role( 'hr\_rep' )

PL/SQL procedure successfully completed.

**SQL> -- посмотрим контекст и данные, которые**

**SQL> -- можно видеть. На этот раз видно все**

**SQL> -- строки, так как пользователь - контолер.**

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии ----

HR\_APP\_CTX.ROLENAME = HR\_REP

HR\_APP\_CTX.EMPNO = 7839

--Данные таблицы Emp, которые можно увидеть--

RLS\_CLARK,2450,ACCOUNTING

RLS\_KING,5000,ACCOUNTING

RLS\_MILLER,1300,ACCOUNTING

RLS\_SMITH,800,RESEARCH

RLS\_JONES,2975,RESEARCH

RLS\_SCOTT,9999,RESEARCH

RLS\_ADAMS,1100,RESEARCH

RLS\_FORD,9999,RESEARCH

RLS\_ALLEN,1600,SALES

RLS\_WARD,1250,SALES

RLS\_MARTIN,1250,SALES

RLS\_BLAKE,2850,SALES

RLS\_TURNER,1500,SALES

RLS\_JAMES,950,SALES

PL/SQL procedure successfully completed.

**SQL> -- следующая операция показывает, ч**

**SQL> -- то можно изменить любую запись**

**SQL> -- в любом отделе, так как пользователь**

**SQL> -- для всех является контролером**

**SQL> -- далее снова запустим listEmps, чтобы увидеть,**

**SQL> -- какие строки изменились (все)**

SQL> exec rls.hr\_app.updateSal

14 rows updated

PL/SQL procedure successfully completed.

**SQL> -- так как пользователь - контролер,**

**SQL> -- то он может удалить кого-нибудь согласно**

**SQL> -- заданным правилам**

**SQL> -- При удалении ВСЕХ не удаляется 'я',**

**SQL> -- т.е. текущий пользователь**

SQL> exec rls.hr\_app.deleteAll

13 rows deleted

PL/SQL procedure successfully completed.

**SQL> -- так как пользователь - контролер, то он**

**SQL> -- может вставить кого-нибудь согласно**

**SQL> -- заданным правилам**

SQL> exec rls.hr\_app.insertNew(20)

PL/SQL procedure successfully completed.

**SQL> -- посмотрим на результат изменения,**

**SQL> -- удаления и последующей вставки**

SQL> exec rls.hr\_app.listEmps

--- Контекст Сессии ---

HR\_APP\_CTX.ROLENAME = HR\_REP

HR\_APP\_CTX.EMPNO = 7839

--Данные таблицы Emp, которые можно увидеть--

RLS\_KING,9999,ACCOUNTING

,1111,RESEARCH

PL/SQL procedure successfully completed.

На этом завершается тестирование трех ролей рассматриваемого примера. Все требования удовлетворены - безопасность данных обеспечена, и они стали прозрачными для приложения.

**Обработка Ошибок и Отладка**

Во время создания вышеописанного приложения я натолкнулся на некоторые ошибки и должен был его отлаживать. Так как Детальный Контроль Доступа работает на сервере, то при обнаружении ошибок и отладке приложения могут возникнуть сложности. Следующий раздел поможет успешной отладке и обнаружению ошибок.

Во время разработки процедур Детального Контроля Доступа могут появиться четыре основных кода ошибок Oracle:

* **ORA-28110: функция политики или пакет <имя\_функции> содержит ошибки**. Это означает, что связанный с политикой пакет или функция содержат ошибки и не могут быть скомпилированы. Ошибки можно увидеть, если выполнить "show errors function <имя\_функции>" или "show errors package body <имя\_пакета>".
* **ORA-28112: ошибка при выполнении функции политики.** Возникает, если ошибка появляется во время выполнения предикатной функции. Это может произойти, например, когда при выполнении предложения SELECT INTO, находящегося внутри PL/SQL-функции, строки не найдены, и для этой ситуации нет обработчика исключений. Функция распространяет исключение NO\_DATA\_FOUND обратно в точку вызова (в ядро базы данных), и база данных инициирует ошибку ORA-28112.
* **ORA-28113: предикат политики содержит ошибки.** Эта ошибка возникает, когда предикатная функция успешно возвращает условие where, но при его добавлении к SQL-запросу внутри него обнаруживаются ошибки. Например, в том случае, когда возвращается условие where типа "x = 5", а таблица, с которой оно ассоциируется, не имеет столбца "x", буден получен код ошибки ORA-28113.
* **ORA-28106: входное значение аргумента #2 неверно.** Эта ошибка возникает при обращении к dbms\_session.set\_context, если имя атрибута не является правильным идентификатором Oracle. Имена атрибутов контекста приложения должны быть правильными идентификаторами (т.е. их можно использовать для назначения имен столбцов таблиц или PL/SQL-переменных). Необходимо только изменить имя атрибута. Например, в приложении могут использоваться атрибуты ‘SEL’, ‘INS’, ‘UPD’ и ‘DEL’ вместо ‘SELECT’, ‘INSERT’ и так далее, потому что ‘SELECT’ не является правильным именем идентификатора Oracle.

При написании предикатных функций я часто пользуюсь одной утилитой - это пакет ‘debug’. Этот пакет, автором которого является Кристофер Бек (Christopher Beck) из Oracle, позволяет вставить в код предложения команду ‘print’. Кроме того, этот пакет позволяет широко использовать предложения типа:

create function foo …

as

…

begin

**debug.f( ‘Вход в процедуру foo’ );**

if ( some\_condition ) then

l\_predicate := ‘x=1’;

end if;

**debug.f**

**( ‘Переход к возврату предиката "%s"’, l\_predicate );**

return l\_predicate;

end;

Таким образом, работа процедуры debug.f похожа на с-функцию printf, а сама она использует пакет UTL\_FILE. На сервере базы данных она создает управляемые программистом файлы трассировки. Файлы трассировки содержат отладочные предложения, которые можно использовать для просмотра выполненных действий при выполнении кода. Так как программный код находится в ядре базы данных, отладка может оказаться сложной. Наличие файлов трассировки может сэкономить много времени. Скрипты, которые можно загрузить (см. далее в этом же разделе) содержат отладочный пакет и комментарии по его установке и использованию.

**За и Против**

Существует много За эту возможность и совсем немного Против. Фактически, сложно вообще найти хотя одно Против этой возможности. Как бы то ни было, они перечислены ниже:

|  |  |
| --- | --- |
| **За** | **Против** |
| **Упрощает разработку Приложения** –переносит управление доступом из приложения на уровень данных. | **Отладка может оказаться сложной, так как** Детальный Контроль Доступа осуществляется в фоновом режиме. Для этой цели более подходят пакеты типа ‘debug’, о которых идет речь в секции диагностики и отладки. |
| **Гарантирует полную защиту информации базы данных**. Независимо от средства доступа к данным, гарантируется, что политика безопасности подключена и не может быть проигнорирована. |  |
| **Допускает значительные изменения политики безопасности**без влияния на клиентские приложения. |  |
| **Упрощает управление объектами базы данных**. Уменьшается общее число объектов базы данных, необходимых для поддержки приложения. |  |
| **Хорошо работает**. Использование контекстов приложения позволяет воспользоваться преимуществами разделяемого SQL. |  |

**Скрипты**

Для того, чтобы получить все скрипты, используемые в этой статье, загрузите ([zip-файл](file:///C:\Documents%20and%20Settings\Sveta\%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D1%81%D1%82%D0%BE%D0%BB\audit2014\www.eManual.ru_1387_%D0%A2%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C\1387\article.zip) ). Пожалуйста, непременно прочитайте файл README.TXT, входящий в состав архива. Zip-файл можно открыть под Windows с помощью WinZip версии 6.0 и выше.

## Примечание

Допустимые параметры для пространства имен ‘USERENV’: (Обратите внимание, что не все параметры действительны во всех версиях Oracle)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Explanation** | **Oracle 9i** | **Oracle 10g** | **Oracle 11g** |
| ACTION | Returns the position in the module | No | Yes | Yes |
| AUDITED\_CURSORID | Returns the cursor ID of the SQL that triggered the audit | Yes | Yes | Yes |
| AUTHENTICATED\_IDENTITY | Returns the identity used in authentication | No | Yes | Yes |
| AUTHENTICATION\_DATA | Authentication data | Yes | Yes | Yes |
| AUTHENTICATION\_METHOD | Returns the method of authentication | No | Yes | Yes |
| AUTHENTICATION\_TYPE | Describes how the user was authenticated. Can be one of the following values: Database, OS, Network, or Proxy | Yes | No | No |
| 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. | Yes | Yes | Yes |
| CLIENT\_IDENTIFIER | Returns the client identifier (global context) | Yes | Yes | Yes |
| CLIENT\_INFO | User session information | Yes | Yes | Yes |
| CURRENT\_BIND | Bind variables for fine-grained auditing | No | Yes | Yes |
| CURRENT\_SCHEMA | Returns the default schema used in the current schema | Yes | Yes | Yes |
| CURRENT\_SCHEMAID | Returns the identifier of the default schema used in the current schema | Yes | Yes | Yes |
| CURRENT\_SQL | Returns the SQL that triggered the audit event | Yes | Yes | Yes |
| CURRENT\_SQL\_LENGTH | Returns the length of the current SQL statement that triggered the audit event | No | Yes | Yes |
| CURRENT\_USER | Name of the current user | Yes | No | No |
| CURRENT\_USERID | Userid of the current user | Yes | No | No |
| DB\_DOMAIN | Domain of the database from the DB\_DOMAIN initialization parameter | Yes | Yes | Yes |
| DB\_NAME | Name of the database from the DB\_NAME initialization parameter | Yes | Yes | Yes |
| DB\_UNIQUE\_NAME | Name of the database from the DB\_UNIQUE\_NAME initialization parameter | No | Yes | Yes |
| ENTRYID | Available auditing entry identifier | Yes | Yes | Yes |
| ENTERPRISE\_IDENTITY | Returns the user’s enterprise-wide identity | No | Yes | Yes |
| EXTERNAL\_NAME | External of the database user | Yes | No | No |
| FG\_JOB\_ID | If the session was established by a client foreground process, this parameter will return the Job ID. Otherwise, it will return NULL. | Yes | Yes | Yes |
| GLOBAL\_CONTEXT\_MEMORY | The number used in the System Global Area by the globally accessed context | Yes | Yes | Yes |
| GLOBAL\_UID | The global user ID from Oracle Internet Directory for enterprise security logins. Returns NULL for all other logins. | No | No | Yes |
| HOST | Name of the host machine from which the client has connected | Yes | Yes | Yes |
| IDENTIFICATION\_TYPE | Returns the way the user’s schema was created | No | Yes | Yes |
| INSTANCE | The identifier number of the current instance | Yes | Yes | Yes |
| INSTANCE\_NAME | The name of the current instance | No | Yes | Yes |
| IP\_ADDRESS | IP address of the machine from which the client has connected | Yes | Yes | Yes |
| ISDBA | Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE. | Yes | Yes | Yes |
| LANG | The ISO abbreviate for the language | Yes | Yes | Yes |
| LANGUAGE | The language, territory, and character of the session. In the following format: language\_territory.characterset | Yes | Yes | Yes |
| MODULE | Returns the appplication name set through DBMS\_APPLICATION\_INFO package or OCI | No | Yes | Yes |
| NETWORK\_PROTOCOL | Network protocol used | Yes | Yes | Yes |
| NLS\_CALENDAR | The calendar of the current session | Yes | Yes | Yes |
| NLS\_CURRENCY | The currency of the current session | Yes | Yes | Yes |
| NLS\_DATE\_FORMAT | The date format for the current session | Yes | Yes | Yes |
| NLS\_DATE\_LANGUAGE | The language used for dates | Yes | Yes | Yes |
| NLS\_SORT | BINARY or the linguistic sort basis | Yes | Yes | Yes |
| NLS\_TERRITORY | The territory of the current session | Yes | Yes | Yes |
| OS\_USER | The OS username for the user logged in | Yes | Yes | Yes |
| POLICY\_INVOKER | The invoker of row-level security policy functions | No | Yes | Yes |
| PROXY\_ENTERPRISE\_IDENTITY | The Oracle Internet Directory DN when the proxy user is an enterprise user | No | Yes | Yes |
| PROXY\_GLOBAL\_UID | The global user ID from Oracle Internet Directory for enterprise user security proxy users. Returns NULL for all other proxy users. | No | Yes | Yes |
| PROXY\_USER | The name of the user who opened the current session on behalf of SESSION\_USER | Yes | Yes | Yes |
| PROXY\_USERID | The identifier of the user who opened the current session on behalf of SESSION\_USER | Yes | Yes | Yes |
| SERVER\_HOST | The host name of the machine where the instance is running | No | Yes | Yes |
| SERVICE\_NAME | The name of the service that the session is connected to | No | Yes | Yes |
| SESSION\_USER | The database user name of the user logged in | Yes | Yes | Yes |
| SESSION\_USERID | The database identifier of the user logged in | Yes | Yes | Yes |
| SESSIONID | The identifier of the auditing session | Yes | Yes | Yes |
| SID | Session number | No | Yes | Yes |
| STATEMENTID | The auditing statement identifier | No | Yes | Yes |
| TERMINAL | The OS identifier of the current session | Yes | Yes | Yes |

## Применение

**Функцию SYS\_CONTEXT** можно использовать в следующих версиях Oracle / PLSQL:

* Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i, Oracle 8i

Bibliografia

Citforum

<http://citforum.ck.ua/database/oracle/session_context/>

<http://citforum.ck.ua/database/oracle/session_context/part2.shtml>