Herr der Daten: APEX, VPD und Data Redaction – die Gefährten

Dr. Thomas Petrik, Sphinx IT Consulting

Die Macht des Datenschatzes eines Unternehmens hat nicht nur auf die klassischen Hacker eine magische Anziehungskraft, allzu oft sind es unkontrollierte interne Datenflüsse, die massiven Schaden verursachen. Ein modernes Security-Konzept sollte daher im Kern der Datenhaltung – also in der Datenbank selbst – ansetzen. Virtual Private Database (VPD) und Data Redaction sind seit Langem integrierte Bestandteile der Oracle Enterprise Edition und ermöglichen den fein granulierten Schutz der Daten auf Row- und Column-Level mit höchster Effizienz und (bei korrekter Implementierung) unabhängig von der Applikation.

Der Oracle REST-APEX-Database (RAD) Stack

VPD und Data Redaction als feingranulare Autorisierungstechnologien in einer Oracle-Datenbank basieren im Wesentlichen darauf, dass sich die Zugriffserlaubnis auf Spalten oder Zeilen eines Datenbestandes aus dem momentanen Session- Context ergibt. An erster Stelle steht dort natürlich die Information, welcher User gerade auf die Daten zugreifen möchte. In 2-Tier-Architekturen ist dies einfach: Der Session User (den wir sehr einfach über die USER-Funktion oder aus sys_context('userenv', 'session_user' im Zugriff haben) ist hier das wichtigste Kriterium.

In einer 3-Tier-Architektur (und dazu gehört auch APEX) stellt sich dies etwas komplizierter dar, weil die Datenbank-

verbindung über einen technischen User im Rahmen eines Connection Pools hergestellt wird und somit der eigentliche End-User aus dem erweiterten Session Context zu ermitteln ist. Dazu kommt die Problematik, dass eine logische APEX-Session (definiert durch die APEX-Session-ID) nicht zwangsläufig innerhalb derselben Datenbanksession abgehandelt wird. Es ist das Wesen eines Connection Pools, dass eine Datenbank-Session nach einem Idle-Timeout von wenigen Sekunden an den Pool retourniert wird und der logischen APEX-Session bei Bedarf eine neue Datenbank-Session zugeteilt wird. Der SessionContext geht beim Zurückgeben der DB-Session an den Pool verloren und muss bei Zuteilung einer neuen DB-Session wiederhergestellt werden.

Wir haben es also mit einer „Stateless"-Connection zu tun, gleichgültig ob wir über Page Rendering oder AJAX-Calls sprechen. *Abbildung 1* zeigt diese von Oracle als „RAD" bezeichnete Architektur, in diesem Fall dargestellt mit ORDS als Standalone Middle Tier – mit dem integrierten Jetty Webserver einerseits und einem Connection Pool zur Datenbank, der den APEX_PUBLIC_USER als technischen User nutzt.

APEX Session Handling & Autorisierung

Doch wie ist es möglich, mit dem APEX_PUBLIC_USER auf das eigentliche Applikationsschema zuzugreifen? Ein Blick auf die Privilegien dieses technischen Users zeigt schnell, dass es keinerlei Grants auf das Applikationsschema (früher auch als „Parsing User" bezeichnet) gibt, ebenso wenig existieren irgendwelche Grants auf das APEX-Repository – zumindest nicht direkt.

Wird hier auf magische Weise der eigentliche User geändert? Ja und Nein. Ein Blick auf die V$SESSION zeigt, dass alle Sessions stets unter dem APEX_PUBLIC_USER laufen *(siehe Abbildung 2)*. Dieser hat jedoch Zugriff auf eine Reihe von Packages im APEX-Repository (APEX_240100.WWV_FLOW_* – es handelt sich in diesem Fall um die Version APEX 24.1), wobei die Execute-Privilegien über Grants an den PUBLIC User und die Zugriffe über Public-Synonyme erfolgen.

Diese WWV_FLOW-Packages ihrerseits verwenden das Definer Rights Package SYS.WWV_DBMS_SQL_APEX_240100 aus dem SYS-Schema und dieses wiederum nutzt ein Oracle Package, das undokumentiert ist: DBMS_SYS_SQL. Darin findet sich die Prozedur PARSE_AS_USER, der – analog zu dem wohl bekannten und dokumentierten Package-Call DBMS_SQL.parse – ein SQL-Statement übergeben werden kann, zusätzlich jedoch auch eine beliebige User-ID, unter der dieses ausgeführt werden soll. Oracle hat mit dieser Methode seit jeher einen sudo-Mechanismus in der Datenbank implementiert, der sich für DBAs als äußerst nützlich erweisen kann. *Listing 1* zeigt ein Beispiel einer Prozedur, wie man diese Funktionalität bequem kapseln könnte, um so Database Links in beliebigen Schemata anzulegen, ohne den in zahlreichen Blogs beschriebenen Weg von Scheduler-Jobs gehen zu müssen.

Doch kommen wir wieder zurück zum Session-Handling. Für Row- und Column-Level-Security benötigen wir den tatsächlichen User, der sich in der APEX-Applikation eigeloggt hat. Diesen finden wir in der V$SESSION in den Feldern CLIENT_INFO und CLIENT_IDENTIFIER, wobei ersteres zusätzlich die Workspace-ID enthält und zweiteres die APEX Session-ID. Beide Werte stehen auch über den USERENV-Context zur Verfügung. Zusätzlich existiert auch noch ein Secure Application Context APEX$SESSION, der (unter anderem) den APP_USER beinhaltet.

Wenn die logische APEX-Session (identifiziert durch die Session-ID in der URL) nun nach einiger Zeit der Inaktivität wieder eine Datenbank-Session aus dem Connection Pool anfordert, wird der Rest der Session-Information über die View APEX_WORKSPACE_SESSIONS, die ihrerseits auf der Tabelle WWV_FLOW_SESSIONS$ basiert, geholt und der Session Context erneut gesetzt. Auf diese Art und Weise garantiert APEX, dass in jeder Situation der End-User über den SYS_CONTEXT abfragbar und somit nutzbar für ein darüberliegendes Security Framework bleibt – ganz gleich, ob es sich um ein Page Rendering oder einen asynchronen AJAX-Call handelt.

Sobald eine Connection an den Pool retourniert wird, setzt APEX den Session Context durch Aufruf von DBMS_SESSION.modify_package_state (DBMS_SESSION.reinitialize) komplett zurück.

APEX In-Database Security

Die Autorisierung für den Datenzugriff erfolgt grundsätzlich auf 3 Ebenen:

- Object Level
- Row Level (RLS)
- Column Level (CLS)

```
PROCEDURE exec_sql_as_user (p_sql IN CLOB, p_username IN VARCHAR2)
    IS
        -- execute a statement as any user
        v_cur   INTEGER;
        v_uid   INTEGER;
    BEGIN
        IF p_username IS NULL
        THEN
            EXECUTE IMMEDIATE p_sql;
        ELSE
            SELECT user_id
              INTO v_uid
              FROM dba_users
             WHERE username = p_username;
            v_cur := DBMS_SQL.open_cursor;
            sys.DBMS_SYS_SQL.parse_as_user (v_cur
                                            ,p_sql
                                            ,DBMS_SQL.native
                                            ,v_uid);
            DBMS_SQL.close_cursor (v_cur);
        END IF;
END;
```

Listing 1: „sudo" für den DBA

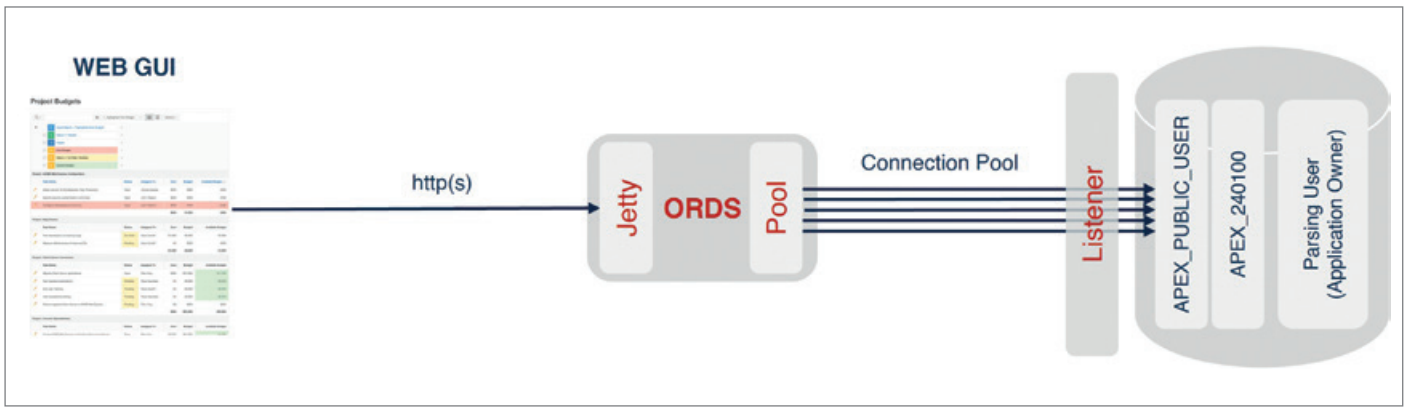

Abbildung 1: Oracle_RAD-Architektur (Quelle: Thomas Petrik)

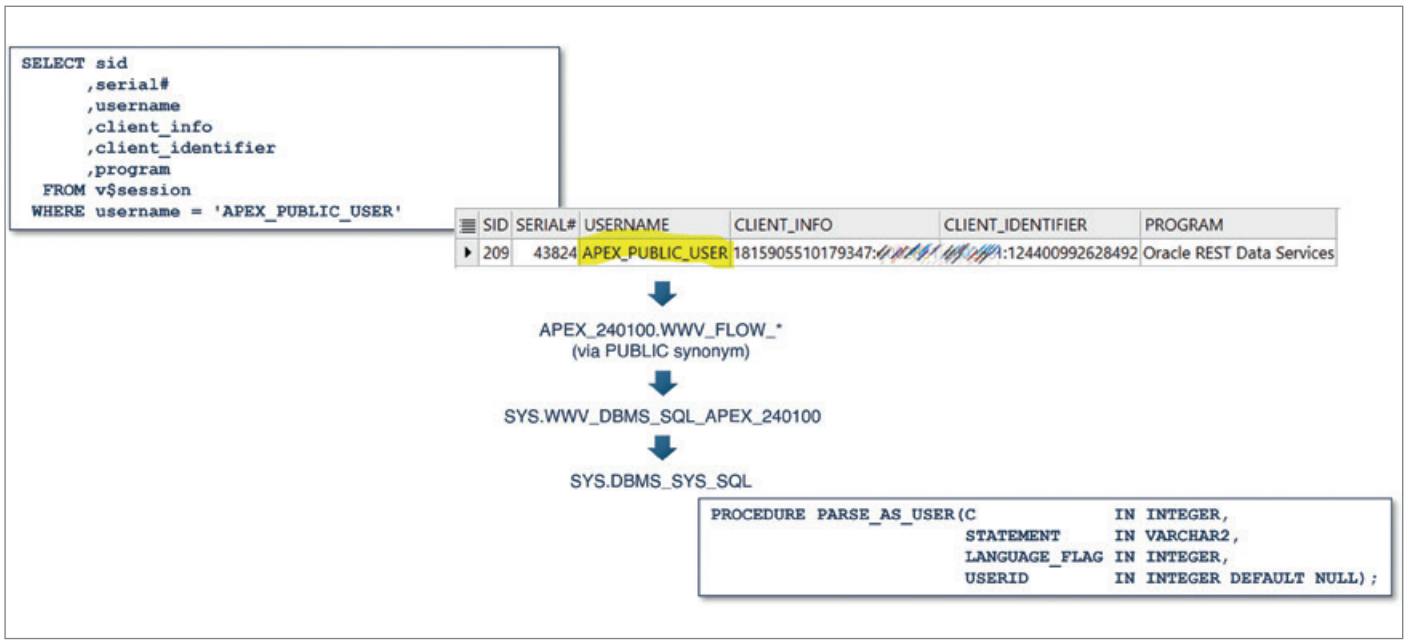

Abbildung 2: „sudo"-Funktion in einer APEX-Session (Quelle: Thomas Petrik)

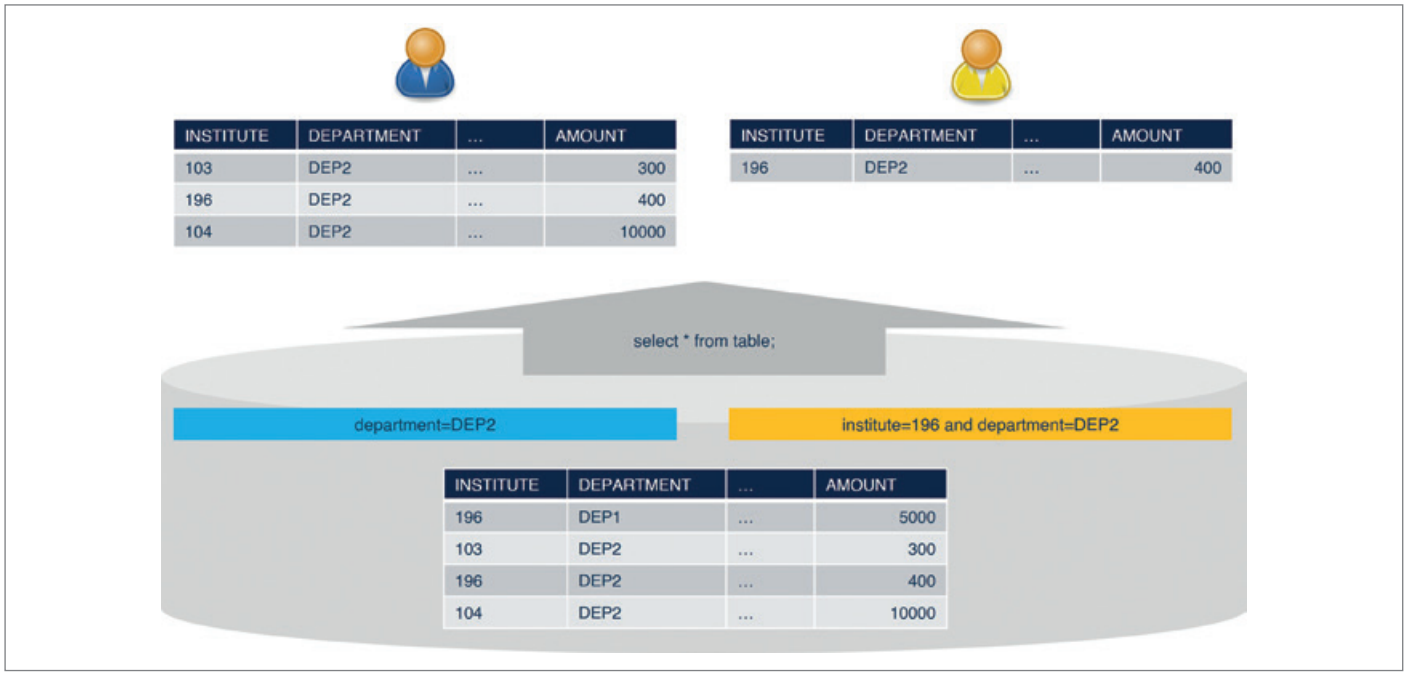

Abbildung 3: Row level Security mittels VPD (Quelle: Thomas Petrik)

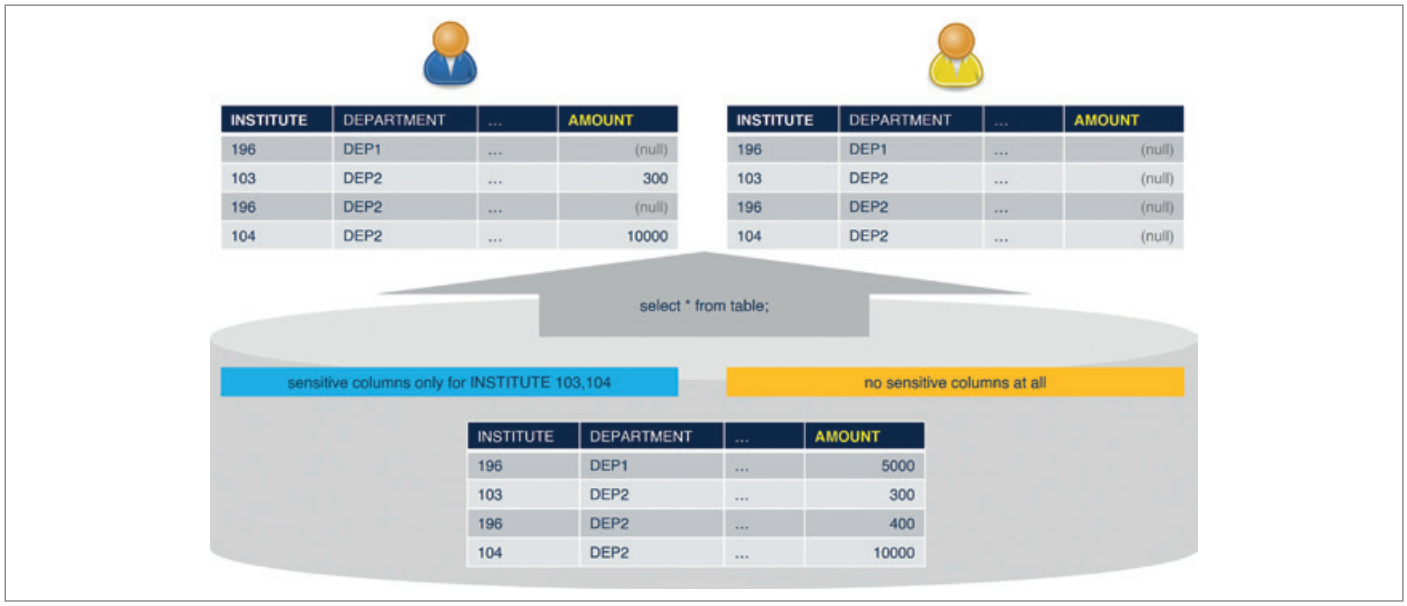

Abbildung 4: Column Level Security mittels VPD (Quelle: Thomas Petrik)

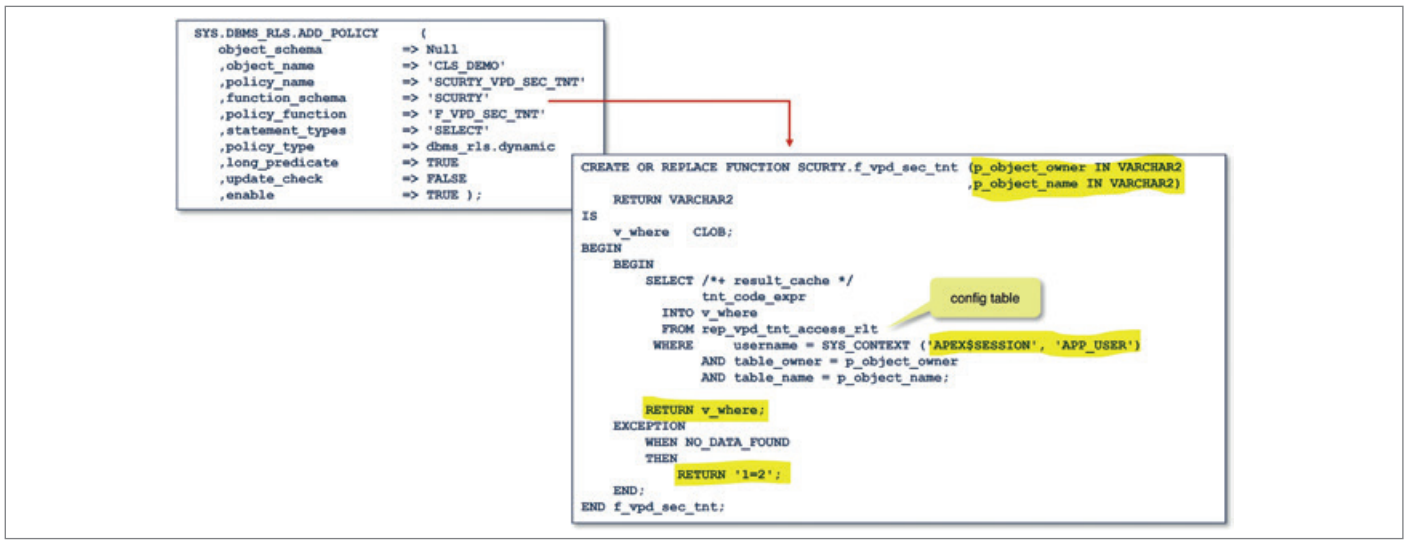

Abbildung 5: Funktionsweise der Row Level Security – Policy & Function (Quelle: Thomas Petrik)

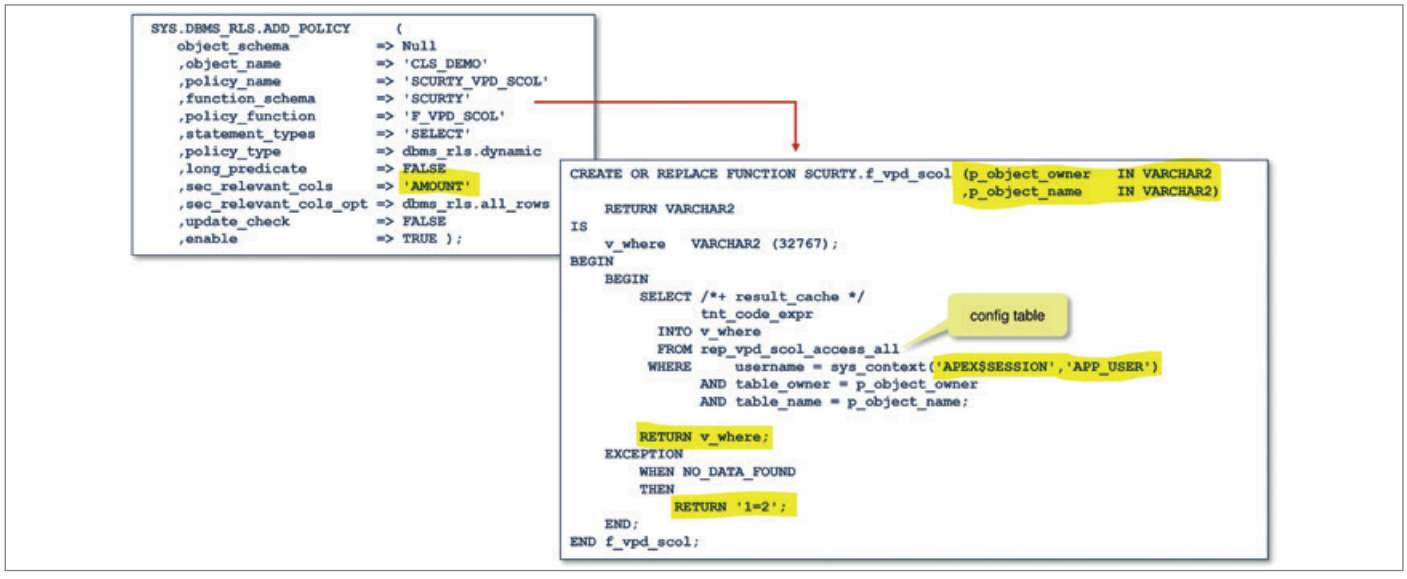

Abbildung 6: Funktionsweise der Column Level Security – Policy & Function (Quelle: Thomas Petrik)

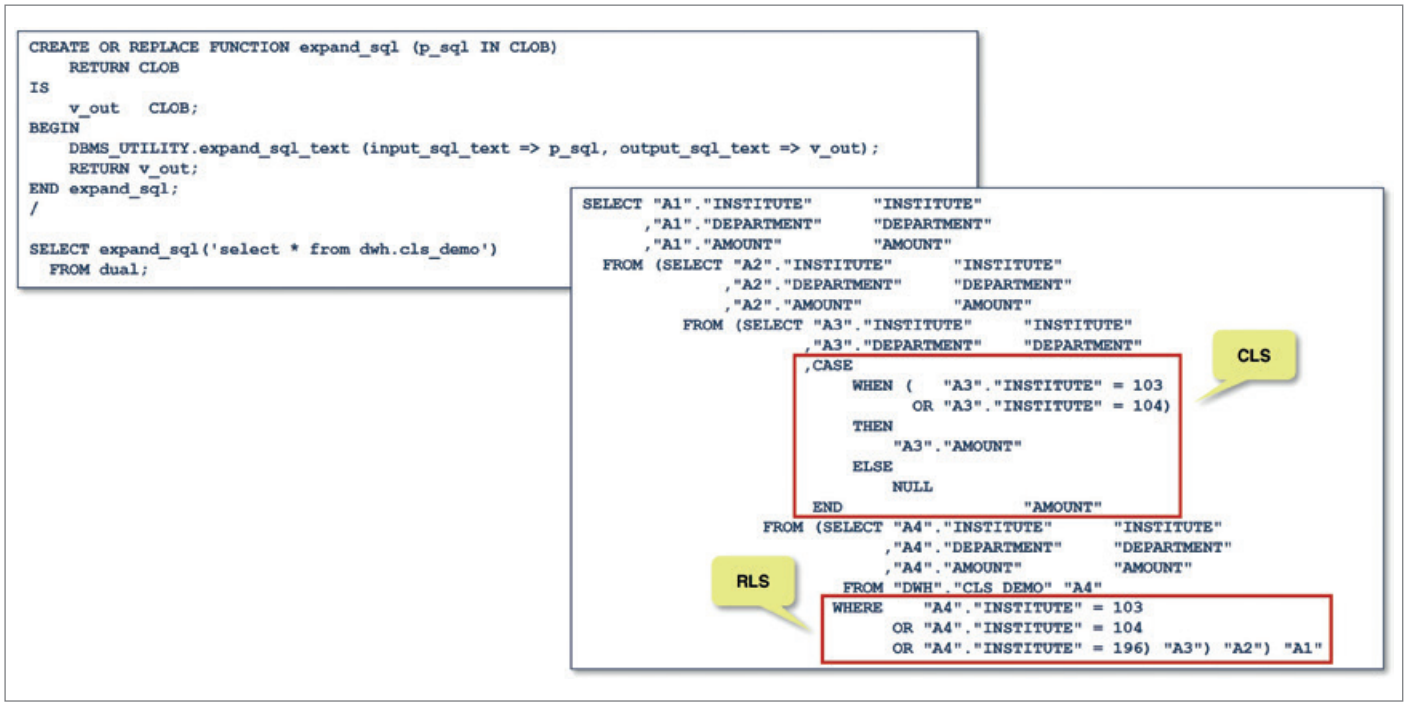

Abbildung 7: Statement Expansion nach Anwendung von RLS und CLS (Quelle: Thomas Petrik)

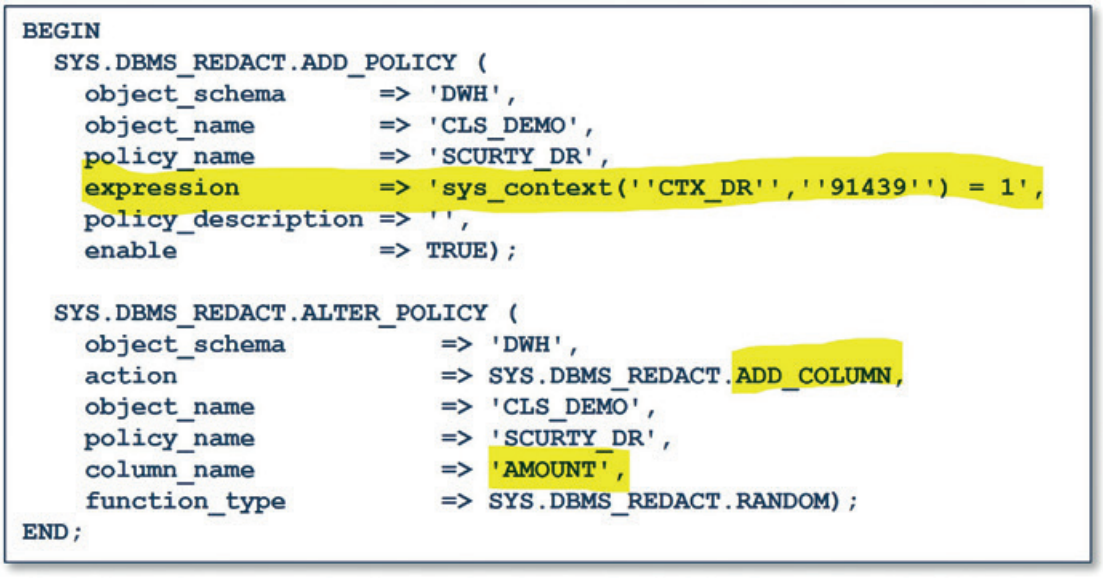

Abbildung 8: Anwendung einer Data Redaction Policy (Quelle: Thomas Petrik)

Object-Level-Security spielt in APEX keine Rolle, da der Parsing User (also der Applikations-User) ohnehin vollen Zugriff auf seine Objekte hat und die Frage, wer auf bestimmte Sichten über Tabellen oder Views zugreifen darf, tatsächlich über die Applikationsmasken geregelt wird.

Anders sieht die Lage für RLS und CLS aus: Diese Logik wird durch VPD und Data Redaction beigesteuert. Beide Technologien ermöglichen das dynamische und transparente Ausblenden von Zeilen beziehungsweise die Maskierung von Spalten, ohne dass dies in der Applikation berücksichtigt werden muss.

Hinweis:

Die Virtual Private Database ist ein kostenloses Feature der Enterprise Edition, wohingegen die Data Redaction eine kostenpflichtige Option ist.

Abbildung 3 zeigt plakativ den Effekt von Row-Level-Security, *Abbildung 4* jenen von Column-Level-Security. In beiden Fällen wird stets ein „select * from …“ abgesetzt, doch in Abhängigkeit vom End-User ein sehr unterschiedliches Ergebnis angezeigt. Im Fall der Data Redaction wird im Vergleich zur CLS Columns nicht auf NULL gesetzt, sondern durch eine (wählbare) Maskierung modifiziert.

Hinweis:

Es ist wichtig zu verstehen, dass die VPD ein Rewrite des Statements bewirkt noch bevor es zum Parsing kommt, wohingegen die Redaction am Resultset ansetzt, also nach dem Fetch der Daten. Dies hat zur Konsequenz, dass eine VPD tatsächlich nicht umgangen werden kann

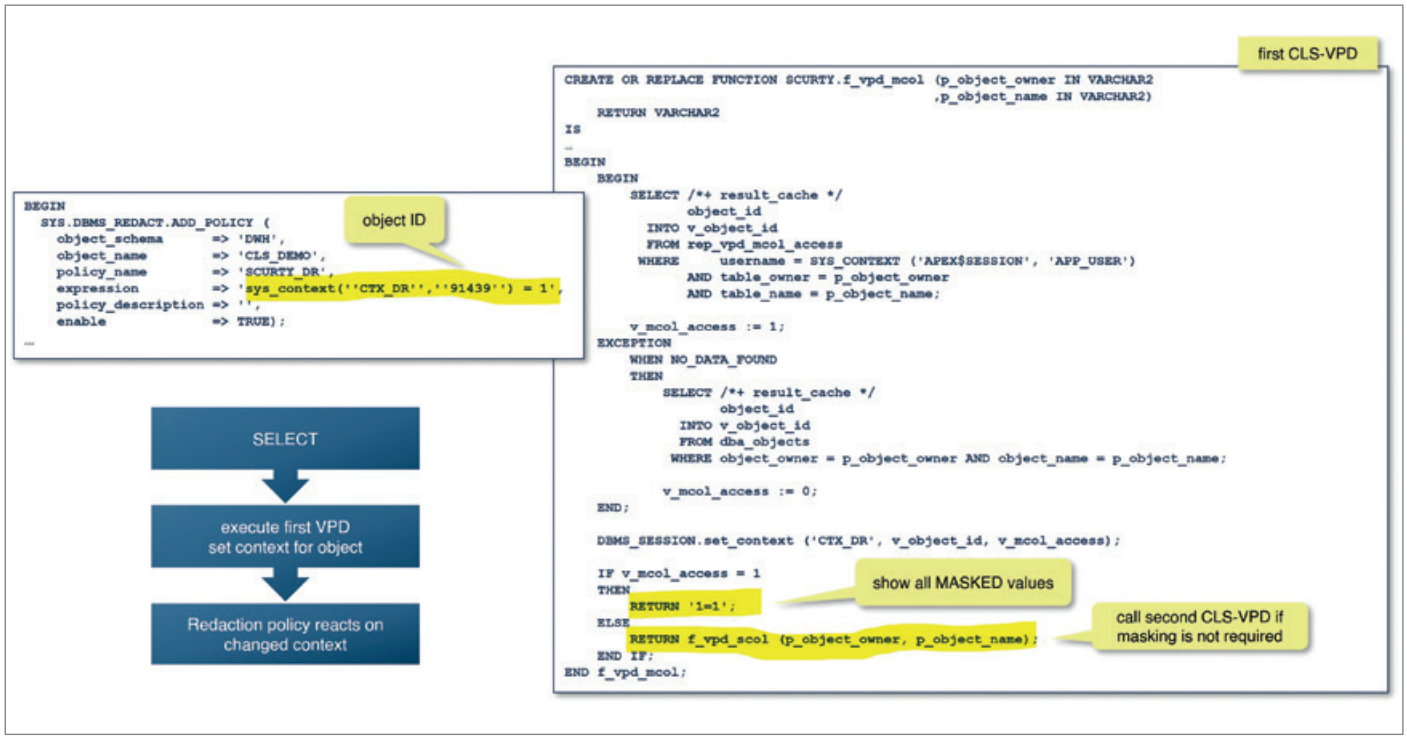

Abbildung 9: Dynamische Steuerung der data Redaction mittels vorgelagerter VPD (Quelle: Thomas Petrik)

im Gegensatz zur Redaction, die (direkten SQL-Zugriff auf die Daten vorausgesetzt) durch gezielten Einsatz von Where-Klauseln in einem Ausschlussverfahren umgangen werden kann. Im Kontext einer APEX-Applikation, wo es üblicherweise keinen Direktzugriff durch den End-User gibt, stellt dies allerdings kein Risiko dar.

Abbildung 5 und *Abbildung 6* geben ein vereinfachtes (aber funktionsfähiges) Beispiel für die Umsetzung von RLS- beziehungsweise CLS-VPDs: Einem Objekt (Table, View, Synonym) wird eine Policy assoziiert, die Policy selbst referenziert eine PL/SQL-Function, deren Aufgabe es ist, eine Where-Klausel zu retournieren, die besagt, welche Rows im Falle einer RLS-Policy im Resultset stehen sollen beziehungsweise im Fall einer CLS, in welchen Zeilen die Werte der in der CLS-Policy spezifizierten Column angezeigt werden dürfen. Ein Return-Value FALSE verhindert in beiden Fällen die Anzeige. Entscheidend ist hier die Kombination einer Konfigurationstabelle mit dem aktuellen Session Context, sodass die passende Where-Clause aus der Kombination des Objekts mit dem APP_USER oder anderen Merkmalen konstruiert werden kann.

DBMS_UTILITY.expand_sql_text zeigt, was der Optimizer vor dem Parsing aus dem Statement macht: Die Where-Clause der CLS wird zu einem Case-Statement verarbeitet wohingegen der Filter der RLS 1:1 angewandt wird *(siehe Abbildung 7)*.

Die Policy der Data Redaction *(siehe Abbildung 8)* sieht im Gegensatz zur VPD keine Verwendung von Funktionen vor, um eine dynamische Filterbedingung aus Kontext- und Objektinformation zu bauen. Lediglich einfache Expressions (wie im gezeigten Beispiel) unter Verwendung von SYS_CONTEXT können verwendet werden. Das ist grundsätzlich verständlich, da die Redaction – wie bereits erwähnt – erst am Resultset ansetzt, für eine Where-Clause ist es da schon zu spät.

Die dynamische Data Redaction

Müssen wir also bei Verwendung der Data Redaction tatsächlich auf die Flexibilität der VPD verzichten? Weder die Oracle-Dokumentation noch einer der zahlreichen Blogs liefern einen Ansatzpunkt, wie die Redaction Policy zur Laufzeit auf den Session Context und das jeweilige Objekt reagieren könnte.

Und doch gibt es hier einen sehr eleganten Ausweg, wenn man bedenkt, dass die VPD-Funktion zwingend stets vor dem Parsing ausgeführt wird (sofern eine „dynamic“ Policy zur Anwendung kommt), die Redaction Policy hingegen immer erst nach dem Fetch wirkt. Es liegt daher nahe, eine eigene VPD-Policy zu schreiben, die nach bewährter Methode eine Variable eines Secure Application Context umsetzt, auf die in weiterer Folge die Redaction im selben Statement reagieren kann. *Abbildung 9* zeigt diese Vorgehensweise, wobei im Application Context CTX_DR Object-IDs als Attribute verwendet werden, deren Wert dynamisch (wiederum auf Basis einer Konfigurationstabelle) auf 0 oder 1 gesetzt werden und dementsprechend in der Policy True oder False ergeben.

Hinweis:

In Kombination von Data Redaction und VPD kommt es (zumindest) in Oracle 19c häufig zu ORA-28094: SQL construct not supported by data redaction. Durch Setzen des Hidden Parameters strict_redaction_semantics" = False konnte dieser Fehler umgangen werden, ohne dass es zu Datenfehlern kam. Eine Nachfrage beim Oracle Support ist aber vor Anwendung in Produktivsystemen jedenfalls dringend zu empfehlen.

Praktikable Umsetzungskonzepte

Wir haben gezeigt, dass die Kombination unterschiedlicher VPDs sowie der Data

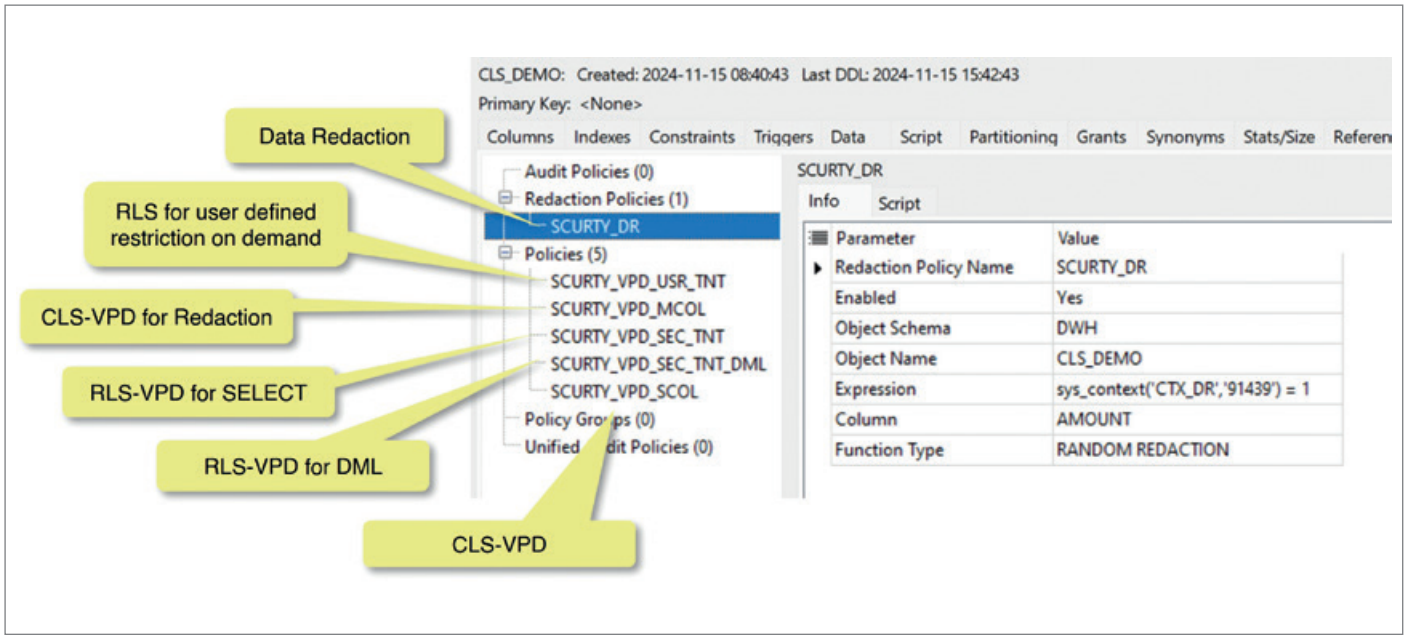

Abbildung 10: Überlagerung mehrerer Policies im SCURTY-Framework (Quelle: Thomas Petrik)

Redaction nahezu unbegrenzte Möglichkeiten für die Umsetzung fein granulierter Security bietet. Um in der Praxis allerdings eine wartbare Lösung ohne die Gefahr von Inkonsistenzen und Wildwuchs zu implementieren, bedarf es eines vollautomatisierten Frameworks, das lediglich durch ein vorgelagertes Identity Management (in den meisten Fällen ein Active Directory) gesteuert wird.

Die Sphinx IT bietet bereits seit vielen Jahren ein fertiges Framework unter dem Namen SCURTY an, das diese Technologien nutzt und folgende Ziele verfolgt:

1. Single Point Of Control (SPOC)
   Die Autorisierung liegt zentral in der Datenbank und kann somit vollkommen transparent von allen Applikationen genutzt werden.
2. Vereinfachung (Streamlining) der Applikationen
   Die Applikation muss sich nicht um die Zugriffslogik kümmern, gleichzeitig kann es zwischen Applikationen nie zu Inkonsistenzen kommen. Das Need-To-Know-Prinzip wird einheitlich garantiert.
3. Client Tool-Unabhängigkeit
   Security wird unabhängig vom verwendeten Tool oder der verwendeten Applikation.
4. Zentrales Audit
   Da die Autorisierung in der DB geregelt ist, kann dort auch zentral das Audit durchgeführt werden.
5. 100% Metadata Driven
   Keine Programmierung, rein deklaratives Customising unter ausschließlicher Verwendung der Features der Oracle Enterprise Edition.

Abbildung 10 zeigt ein Beispiel für die Überlagerung mehrerer Policies auf einem Objekt, wie sie vom SCURTY-Framework generiert werden.

Schlussfolgerungen

Für eine feingranulare Autorisierung bietet die Oracle Enterprise Edition mit Virtual Private Database und Data Redaction herausragende Features, die allerdings erst durch ein umfassendes Framework kombiniert und nutzbar gemacht werden müssen. Der Komplexitätsgrad ist nicht zu unterschätzen, die Umsetzung wird allerdings durch eine unglaubliche Flexibilität belohnt: vereinheitlichte und vor allem konsistente Autorisierung auf Zeilen- und Spaltenebene und eine bedeutende Entlastung der Applikationen, die sich mit diesem Thema überhaupt nicht mehr befassen müssen. Speziell APEX bietet von Haus aus die besten Voraussetzungen, um an ein derartiges Framework anzudocken.

Eine kurze Anmerkung zum Thema Performance darf zum Schluss auch nicht fehlen: Der korrekte Einsatz von VPDs mit dem entsprechendem Hintergrundwissen um die Funktionalität wird nie zu Performance-Problemen führen, wenngleich es natürlich die eine oder andere Falle zu umgehen gilt.

Über den Autor

Dr. Thomas Petrik arbeitet seit fast 3 Jahrzehnten mit Oracle-Datenbanken und befasst sich ebenso lange mit den Themen Security, Performance und Effizienz in der Betriebsführung (Automatisierung). Es ist kein Zufall, dass aus dieser Tätigkeit im Laufe der Zeit Services und Produkte entstanden sind, die aus der Praxis für die Praxis geschaffen wurden und erfolgreich von den Kunden der Sphinx IT Consulting eingesetzt werden.

Dr. Thomas Petrik
thomas.petrik@sphinx.at