Virtual Private Database in de praktijk

Centraal vastgelegde veiligheid op tabelniveau

Oracle positioneert de virtual private database (VPD) als een van haar belangrijkste security features in de database. Het is dan ook een zeer krachtig en conceptueel elegant mechanisme. Het biedt een zeer hoge mate aan veiligheid en is zelfs door de dba niet of nauwelijks te kraken. Het is standaard aanwezig in Oracle 8i en met een aantal uitbreidingen als optie in 9i.

Wat is een VPD ?

Het principe van een virtual private database is dat meerdere user community's dezelfde fysieke database gebruiken, maar iedere user community heeft uitsluitend toegang tot zijn eigen subset aan data. (figuur 1).
Deze datascheiding wordt in de database op tabelniveau afgedwongen en is volledig transparant voor de applicatie. De voordelen zijn enorm. Er hoeft maar 1 gezamenlijk owner schema opgezet te worden, en bovendien hoeft er in de applicatie niets gecodeerd te worden om de datascheiding te regelen. De logica m.b.t. het scheiden van de verschillende community's wordt op 'n centrale plaats vastgelegd in de database: de security policy. Op deze wijze is applicatie logica gescheiden van autorisatie logica. Uit oogpunt van onderhoudbaarheid en betrouwbaarheid is dit een absoluut winstpunt. Omdat de autorisatie regels in de database vastliggen, is het gevolg dat ook niet-applicatie software aan *dezelfde* security regels onderworpen is. Met andere woorden: ook met SQL*Plus en andere client tools (TOAD, MSAccess etc.) heb je uitsluitend toegang tot de rechtmatige eigen subset aan data. Het is dus gedaan met vrolijk data-shoppen in de database.

Toepassingen VPD

Het VPD-concept is erg geschikt voor web based applicaties, of voor applicatie hosting situaties waar verschillende user groepen onderscheiden moeten worden. Door een minimale ingreep kan er in een handomdraai een nieuwe usergroep gedefinieerd worden, zonder dat daarvoor code aanpassing nodig is. Maar er zijn ook andere toepassingen mogelijk. Schaf alle aparte demo- en training databases maar af en voeg ze samen (als VPD) in de productie database. Zo weet je tenminste zeker dat ze allen dezelfde applicatie code gebruiken. Of ga de decentrale satelliet systemen, die in wezen toch allemaal hetzelfde (zouden moeten) zijn, weer eens lekker centraliseren. Ieder z'n eigen VPD in één fysieke database met één centrale applicatie, en omlaag gaan de beheerkosten.

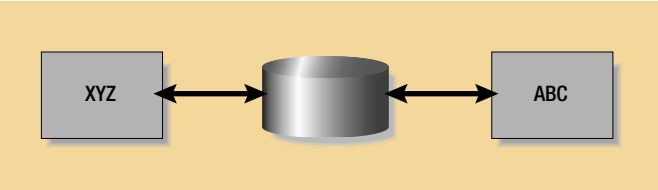
Geschiedenis

Eigenlijk is Virtual Private Database een slimme marketing term. De onderliggende techniek bestaat al langer, maar spreekt blijkbaar niet zo tot de verbeelding. Termen als fine grained access control (FGAC) en row level security (RLS) zijn eigenlijk benamingen voor hetzelfde. De crux zit 'm erin dat door het consequent en generiek toepassen van deze technieken de illusie van een eigen database gewekt wordt.
Vanaf Oracle8i zijn alle componenten voor de VPD standaard (gratis) aanwezig in het rdbms. In Oracle9i kun je optioneel (niet gratis) gebruik maken van Oracle Label Security (OLS), wat eigenlijk VPD is met diverse uitbreidingen en standaardisaties, ondersteund met grafische tools.

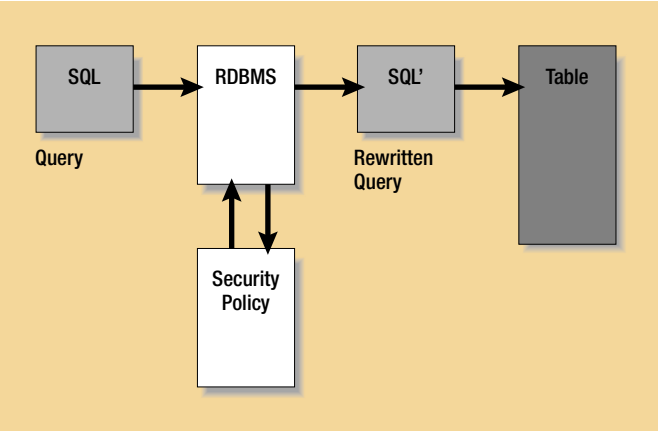
Rollen en privileges

De traditionele database rollen en privileges blijven natuurlijk onverwijld van kracht en zullen ook noodzakelijk blijven. Beide technieken vullen elkaar perfect aan. Rollen regelen de bevoegdheid op tabel niveau, policy's regelen de toegankelijkheid op rij niveau. De combinatie van beiden levert een ijzersterke security structuur op.

Het VPD-concept is erg geschikt voor web based applicaties, of voor applicatie hosting situaties



Figuur 1.



Figuur 2.

Policy routine

Het hart van een VPD is de security policy. Dit is een PL/Sql package functie die als resultaat een textstring oplevert in de vorm van een conditie. Nadat de policy functie eenmaal is toegekend aan een tabel, zal voor elk sql-statement naar deze tabel de policy routine doorlopen worden. De door de routine afgegeven textstring wordt dan door het rdbms toegevoegd aan het sql-statement, als extra conditie van de WHERE clause. Er vindt dus een dynamische query rewrite plaats op server niveau. (figuur 2)

Voorbeeld

We nemen een tabel met productgegevens van verschillende werkmaatschappijen.

Wmy	Prodnr	Prodnaam	Prijs
ABC	001	Potlood	0,34
ABC	002	Gum	1,25
ABC	003	Schrift	2,00
XYZ	004	Potlood	0,36
XYZ	005	Schrift	2,10

De gebruikers mogen alleen de gegevens van hun eigen werkmaatschappij zien, behalve de productmanager, die mag alle gegevens zien. De bijbehorende policy routine zou dan de volgende functionaliteit bevatten:

- User van werkmaatschappij ABC ⟶ conditie is: (WMY = 'ABC')
- User van werkmaatschappij XYZ ⟶ conditie is: (WMY = 'XYZ')
- User is productmanager ⟶ conditie is: (1=1)
- Overigen ⟶ conditie is: (1=2)

De grap is nu dat één en hetzelfde sql-statement een verschillend resultaat oplevert afhankelijk van wie het uitvoert.

Sql statement	User-type	Resultaat rijen
Select * from producten where prodnaam = 'Potlood';	wmy ABC	ABC 001 Potlood 0,34
	wmy XYZ	XYZ 004 Potlood 0,36
	Productmanager	ABC 001 Potlood 0,34 XYZ 004 Potlood 0,36
	Onbekend	-

Vooropgesteld nu dat alle tabellen de kolom WMY bevatten, dan kan deze policy routine op alle tabellen worden aangebracht. Het effect is dan dat iedere werkmaatschappij uitsluitend toegang heeft tot zijn eigen data. Dit geldt eveneens voor het muteren van data.

Policy stability

Omdat de policy routine een zelfgeschreven PL/Sql functie is, zijn de mogelijkheden onbegrensd. Zo is het eenvoudig mogelijk om af te dwingen dat het systeem alleen tijdens kantoor uren gebruikt mag worden. De routine geeft dan buiten kantoor uren gewoon een NULL-conditie af (1=2). Het beveiligen van extra gevoelige informatie is op deze wijze ook zeer eenvoudig.

Het toekennen van meerdere policy routines aan 'n tabel is toegestaan. Je zou bijvoorbeeld verschillende routines kunnen toekennen voor SELECT enerzijds, en mutaties (INSERT, DELETE, UPDATE) anderzijds. Hiermee kun je bewerkstelligen dat je wel de data van alle werkmaatschappijen kunt lezen, maar dat je alleen de data van je eigen werkmaatschappij kunt muteren. Een andere interessante (noodzakelijke) optie is de CHECK OPTION, waarmee policy stability kan worden afgedwongen. Kort gezegd komt het erop neer dat men de data niet zodanig kan muteren (of toevoegen) dat het buiten de eigen virtuele database terecht zou komen. In ons voorbeeld kan een user van werkmaatschappij ABC dus geen rijen toevoegen met wmy=XYZ. Per default staat de check option op true, en ik zou ook niet direct een situatie weten waarin je het anders zou willen.

Applicatie Context

Vanaf 8i is er het krachtige principe van application contexts geïntroduceerd. Een applicatie context is een verzameling session-variabelen waarvan de waarde gezet kan worden tijdens het aanloggen aan de database. Dit kan bijvoorbeeld door een logon trigger worden geregeld. (figuur 3). De logon trigger zoekt de betreffende user op in een user tabel, en zet vervolgens de context variabelen op de juiste waarde:

```
DBMS_SESSION.set_context('my_appl', 'wmy', l_wmy);
DBMS_SESSION.set_context('my_appl', 'admin', l_admin);
```

Deze context variabelen blijven gedurende de gehele usersessie beschikbaar in memory en kunnen overal door de applicatie (en dus ook door de policy routine) opgevraagd worden. Hierdoor wordt de policy routine onderhoudsvrij. In plaats van hard gecodeerde werkmaatschappij codes, wordt er in ons voorbeeld door de policy routine simpelweg de waarde van de context variabele ingevuld:

```
l_wmy := sys_context('my_appl','wmy');
l_admin := sys_context('my_appl','admin');
IF l_admin = 'NO'
   THEN
l_conditie := 'WMY=' || l_wmy;
   ELSIF l_admin = 'YES'
THEN
        l_conditie := '1=1';
ELSE
       l_conditie := '1=2'
END IF;
return l_conditie;
```

Moet er een nieuwe werkmaatschappij bijkomen, dan hoeft er nergens code aangepast te worden. Simpelweg de nieuwe gebruikers met een aparte werkmaatschappij code toevoegen aan de user tabel, en voilà: er is een virtual private database bijgekomen!
Een applicatie context wordt ook wel namespace genoemd. Er kunnen meerdere namespaces in de database gedefinieerd zijn, dat is handig als er diverse applicaties dezelfde database gebruiken. Iedere applicatie gebruikt dan zijn eigen namespace (context), door bijvoorbeeld in sql-statements en PL/Sql packages de waarde van bepaalde variabelen uit de applicatie context op te halen.

Het mechanisme van application contexts is robuust en absoluut veilig. Iedere context is namelijk in de database geassocieerd met een package, en het is alleen dit package die de waarden van de context variabelen mag veranderen.

Praktijk toepassing

Bij Philips-CE wordt het VPD concept intensief toegepast in een applicatie die het SCM proces ondersteunt. De producten zijn onderverdeeld in productgroepen, en de gebruikersorganisatie is onderverdeeld in geografische regio's. Voor de duvel niet bang, heeft men dan ook meteen maar een 2-dimensionale VPD opgezet (figuur 4). Een online user heeft toegang tot 1 data segment; de policy routine geeft een gecombineerde conditie af:

```
pg='DVD' and region='LATAM'
```

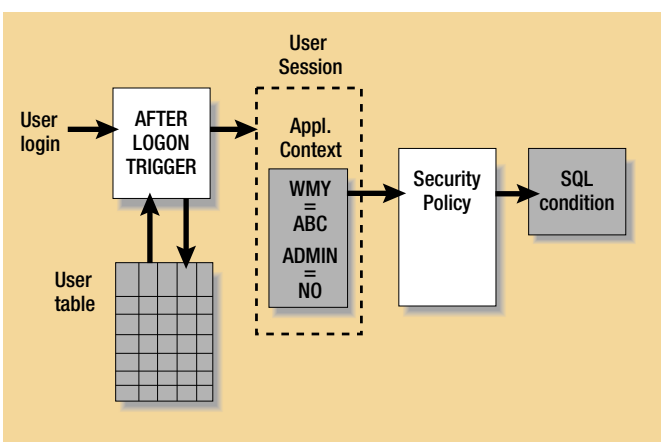
Een batch job verwerkt alle gegevens van 'n hele productgroep over alle regio's heen. De user waaronder de batch job draait is bepalend voor de productgroep; de policy routine geeft dan bijvoorbeeld de conditie:

```
pg='monitors'
```

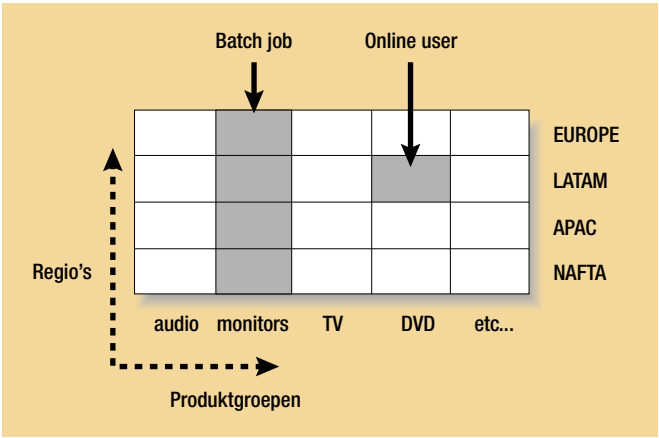
Alle tabellen (plm. 200 stuks) hebben 2 extra kolommen, REGION en PG. Deze kolommen worden automatisch gevuld door 'n insert trigger op iedere tabel:

```
:new.pg := sys_context('scm','pg');
:new.region := sys_context('scm','regio');
```

De users (plm. 300) worden onderhouden in een user tabel, waarbij voor iedere user is opgeslagen het type user, de regio en productgroep. Een logon trigger leest de user tabel en vult de juiste waarden in de context variabelen. De policy routine leest dan later deze context variabelen uit, en geeft de juiste conditie af.



Figuur 3.



Figuur 4.

Deze oplossing met VPD's werkt feilloos. Het alternatief, database + applicatie dupliceren voor iedere productgroep (of regio) is afstotelijk en niet werkbaar. De batch programmatuur (PL/Sql packages) is immers identiek voor het hele systeem, evenals de online programmatuur (Forms). Het enig werkelijke probleem is niet technisch maar organisatorisch van aard. Iedere wijziging aan het datamodel of programmatuur geldt meteen voor het hele systeem (alle VPD's). Plannings technisch geeft dat bij grote wijzigingen soms hoofdbrekens. Niet alle locale eenheden kunnen hun interne organisatie gelijktijdig op poten hebben. Zodoende moet er soms gewacht worden tot iedereen zich ingesteld heeft op de nieuwe situatie. Dit dwingt tot standaardisatie in de werkwijze van lokale organisaties, en leidt dus tot kostenbesparing. Misschien wel het grootste winstpunt van de hele exercitie.

Performance

Kost al dit moois dan geen extra performance? Puur alleen het VPD principe beschouwend is het antwoord: nee. Wat maakt het immers uit, of de applicatie nu een extra conditie in ieder sql statement opneemt, of dat het systeem het automatisch voor je doet? Laat het dan maar aan het systeem over, dan weet je tenminste zeker dat het niet ergens vergeten wordt. Het zal wellicht wat extra CPU tijd vergen, en voor de rest staat alles in memory (SGA). De performance van een VPD zou best wel eens beter kunnen zijn. Een index op de VPD-discriminerende kolommen geeft welhaast een gegarandeerde index-hit, omdat toch ieder sql-statement deze kolommen zal specificeren met een harde waarde. Ook tests met partitioning tonen aan dat de optimizer keurig de juiste partitie kiest, indien de partitioning gebaseerd is op de VPD-discriminerende kolommen.

DBA

De dba moet even wennen aan het feit dat ook hij niet alle data zomaar kan bekijken in de database. Het is zelfs een beetje oppassen geblazen, want ook de export utility is onderworpen aan het VPD mechanisme. De volgende melding is even schrikken:

```
EXP-00079: Data in table "PRODUCTEN" is protected
Conventional path may only be exporting partial table.
. . exporting table          PRODUCTEN       1223 rows exported
```

Dit is eigenlijk gewoon een warning message, de export wordt wel gedaan maar is mogelijk niet compleet. Ook hier heeft de policy routine een extra conditie toegevoegd aan de tabel selectie. De enige user die buiten de VPD omgaat is SYS. Verder zijn er een aantal extra views en packages waar de dba mee te maken krijgt:

```
view DBA_CONTEXT bevat context (namespace) definities
view DBA_POLICIES bevat tabel/policy gegevens
package DBMS_RLS om policy op tabel te zetten of eraf te halen
```

Het tracen en analyseren van sql statements is een avontuur op zichzelf. De library cache bevat keurig de geëxpandeerde (rewritten) sql statements. Het blijkt dat herhaald uitvoeren van eenzelfde statement niet tot hard parses leidt. Met andere woorden: het eenmaal opgestelde executie plan wordt hergebruikt. Het kost echter wat moeite om het werkelijk uitgevoerde sql statement boven water te krijgen, daarvoor moet er een hidden parameter gezet worden. De view SQL$AREA laat alleen de oorspronkelijke sql-statements zien. Dit is een instinker, want in werkelijkheid wordt het geëxpandeerde statement uitgevoerd! Ook STATSPACK en TKPROF rapporteren steevast de oorspronkelijke sql-statements.

Tot slot

De indruk zou gewekt kunnen worden dat deze techniek alleen te gebruiken is bij de opzet van nieuwe systemen. Dat is onjuist. Het is zeker wel mogelijk om een bestaand systeem te migreren naar het VPD concept. Met een beetje geluk zelfs zonder de applicatie aan te passen. Als er nog geen discriminerende kolom bestaat dan zou die toegevoegd kunnen worden aan alle tabellen. Als de applicatie netjes geprogrammeerd is (geen select * from) dan kan dat probleemloos. Cruciaal is dan nog hoe de primary key's gevuld worden. Contructies als "select max(id)" zijn dodelijk wat dit betreft, dat geeft per VPD een verschillend resultaat. Worden key waarden daarentegen met sequences gegenereerd dan is alles in orde, want "select sequence.nextvalue" geeft een database-wide uniek volgnummer uit. Meer over VPD is na te lezen in de Oracle8i white paper over dit onderwerp: http://otn.oracle.com/deploy/security/oracle8i/pdf/vpd_wp6.pdf

René Beijk
is project dba bij Atos Origin, service line Business Application Solutions. Zijn e-mailadres rene.beijk@atosorigin.com staat open voor reacties.