# Valódi konszolidáció virtuális adatbázissal

A konszolidáció kifejezéssel, mint az IT költségek racionalizálásának módszerével szinte minden üzemeltetéssel foglalkozó szakember találkozott már. A hardver- és szoftvergyártók marketinganyagaiban rendre felbukkan ez a kifejezést, amit persze mindenki a saját maga módján értelmez.

Abban azonban megegyeznek, hogy az erőforrások jobb kihasználását, a korábbi szigetszerű, elosztott megoldások egyfajta koncentrációját teszi lehetővé. A konszolidáció annál hatékonyabb, minél magasabb szintet érint. Ezért a közös központi tároló kialakítása után logikus lépés az adatbázisok fürtbe (***Real Application Clusters***-be) való szervezése. Ez egy vagy több fizikai adatbázis több szerverről való egyidejű elérhetőségét teszi lehetővé. Az adatbázis fürtözés egyszerre szolgálja az adatbázis folyamatos elérhetőségét és a szerverek párhuzamos működése révén a teljesítmény igényeknek való rugalmas megfelelést. A konszolidáció csökkenti a költségeket, de a korábban különálló fizikai adatbázisok egy adatbázisban való egyesítése biztonsági kockázatot jelent. Csökkentésének egyik lehetséges eszköze a virtuális adatbázisok létrehozása. A korábban azonos szerkezetű több példányban létező adatbázisok egyetlen adatbázisban való összefésülése azt eredményezi, hogy a felhasználók egymás adataihoz hozzáférhetnek. A probléma megoldására több módszer kínálkozik.

Az alkalmazást módosítjuk, és minden *SQL* utasítást kiegészítünk egy szűrőfeltétellel. Ez a védelem, azonban csak ebben az alkalmazásban fog működni, vagyis minden további az adatbázist elérő alkalmazás módosítása szükséges.

Következő, már alkalmazás független megoldás, adatbázis nézetek létrehozása. Vagyis minden felhasználónak, vagy felhasználói csoportnak létre kell hozni egy a szűrőfeltételt tartalmazó nézetet. A táblák átnevezésével és a nézetekre az eredeti táblanévvel megegyező szinonimák létrehozásával az alkalmazás elől elrejthetjük a változtatást. A megoldás hátránya, hogy nagyon nehézkessé válhat a rengeteg nézet adminisztrálása.

Az előző kettőnél elegánsabb és alkalmazás független megoldás az *Oracle* adatbázis kezelő ***Virtual Private Database (VPD)*** technológiájának a használata. A *VPD* segítségével elérhető, hogy minden felhasználó ugyan azt az alkalmazást, és annak beépített hozzáférés szabályozását használva, mégsem látja egymás adatait. A gyakorlatban ez azt jelenti, hogy a felhasználók ugyan azon utasítást kiadva az adatbázis táblák más-más soraihoz férnek hozzá, akár lekérdezésről, akár adat manipulációs (***insert***, ***update***, ***delete***) műveletről legyen szó.

A *VPD*-t emiatt szoktuk sorszintű hozzáférés szabályozást lehetővé tevő eszköznek mondani.

A *VPD* működésének alapja, hogy minden táblához és azon belül minden utasításhoz egy szűrőfeltételt lehet rendelni. Ez utóbbit egy *PL/SQL* nyelven írt tárolt eljárás állítja elő.

## Példaprogram

Tegyük fel, hogy egy cég szeretné az adatainak egy részét a partnerei számára hozzáférhetővé tenni. Mindezt úgy, hogy minden partner a csak rá vonatkozó adatokat lássa. A példában két táblát fogunk használni. Mind kettő az RENDELÉSEK sémában található.

Az ÜGYFELEK tábla szerkezete:

```
Név             Üres?    Típus
--------------------- --------
ÜGYFÉL_AZ       NOT NULL NUMBER(6)
ÜGYFÉL_NÉV      VARCHAR2(45)
```

Az ÜGYFELEK tábla tartalma:

```
ÜGYFÉL_AZ ÜGYFÉL_NÉV
--------- -----------------
      102 SCOTT
      101 PALM
```

A RENDELÉSEK tábla szerkezete:

```
Név             Üres?    Típus
--------------------- --------
RENDELÉS_AZ     NOT NULL NUMBER(4)
RENDELÉS_DÁTUM  DATE
ÜGYFÉL_AZ       NOT NULL NUMBER(6)
```

A RENDELÉSEK tábla tartalma:

```
RENDELÉS_AZ RENDELÉS_DÁT
ÜGYFÉL_AZ
 ----------- ------------ -----
         100 07-FEBR. -26   102
         101 07-FEBR. -21   101
```

A hozzáférés szabályozásunknak az a célja, hogy minden ügyfél csak a saját rendelési adataihoz férjen hozzá. Jelen esetben a SCOTT felhasználó a RENDELÉSEK tábla azon sorához fog hozzáférni, amelynek az ÜGYFÉL_AZ mezőjének értéke 102.

A hozzáférés szabályozás tárolt eljárásai a SZABÁLY sémában találhatóak. Az ügyfél-azonosítót a felhasználók adatbázisba való bejelentkezésekor

a LOGON_TRIGGER segítségével rögtön meghatározzuk és egy szerver oldali környezeti változóban tároljuk, amely az egész bejelentkezés ideje alatt elérhető lesz.
Felhasználó ügyfél azonosítójának meghatározása a bejelentkezéskor:

```
CREATE OR REPLACE PACKAGE
➥"SZABÁLY"."RENDELÉSEK_BIZT_
➥CSOMAG" IS
  PROCEDURE "AZONOSÍTÓ";
end;
/

CREATE OR REPLACE PACKAGE BODY
➥"SZABÁLY"."RENDELÉSEK_BIZT_
➥CSOMAG" IS
  PROCEDURE "AZONOSÍTÓ"
IS
  "KÓD" NUMBER;
  BEGIN
    SELECT "ÜGYFÉL_AZ" INTO
    ➥"KÓD" FROM "RENDELÉSEK".
    ➥"ÜGYFELEK"
      WHERE "ÜGYFÉL_NÉV"=
      ➥SYS_CONTEXT('USERENV',
      ➥'SESSION_USER');
    DBMS_SESSION.SET_CONTEXT
    ➥('RENDELÉSEK_BIZT',
    ➥'ÜGYFÉL_KÓD',to_char
    ➥("KÓD"));
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_SESSION.SET_CONTEXT
      ➥('RENDELÉSEK_BIZT',
      ➥'ÜGYFÉL_KÓD','-1');
  END;
END "RENDELÉSEK_BIZT_CSOMAG"
/
```

A SYS_CONTEXT függvény a felhasználói munkafolyamat környezeti változóinak lekérdezését teszi lehetővé. A USERENV az alapértelmezett névtér. A SESSION_USER értéke az adatbázis felhasználó neve. A program tehát az ÜGYFELEK tábla alapján meghatározza az ügyfél azonosítót (ÜGYFÉL_AZ) és a RENDELÉSEK_BIZT névtérbe helyezi ÜGYFÉL_KÓD néven.
Ehhez a csomaghoz minden adatbázis felhasználónak végrehajtási jogot adunk:

```
grant execute on "RENDELÉSEK_
➥BIZT_CSOMAG" to public;
```

Ezután létrehozzuk az adatbázis LOGON_TRIGGER-t:

```
CREATE OR REPLACE TRIGGER
➥LOGON_TRIGGER
  AFTER LOGON
  ON DATABASE
BEGIN
  "SZABÁLY"."RENDELÉSEK_BIZT
  ➥_CSOMAG"."AZONOSÍTÓ";
END;
/
```

Következő lépés a szűrőfeltételt meghatározó *PL/SQL* program létrehozása:

```
CREATE OR REPLACE PACKAGE
➥"SZABÁLY"."ÜGYFÉL_BIZT"  AS
  FUNCTION  "FELTÉTEL"(D1
  ➥VARCHAR2,D2 VARCHAR2)
  ➥RETURN VARCHAR2;
  PRAGMA RESTRICT_REFERENCES
  ➥("FELTÉTEL",WNDS);
END;
/

CREATE OR REPLACE PACKAGE BODY
➥"SZABÁLY"."ÜGYFÉL_BIZT"  AS
  FUNCTION  "FELTÉTEL"(D1
  ➥VARCHAR2,D2 VARCHAR2)
  ➥RETURN VARCHAR2
  IS
    D VARCHAR2(2000);
  BEGIN
    IF RTRIM(USER)<>D1 AND
    ➥RTRIM(USER)<>'SYS' AND
    ➥RTRIM(USER)<>'SYSTEM' THEN
      D:='"ÜGYFÉL_AZ"=
      ➥SYS_CONTEXT
      ➥(''RENDELÉSEK_BIZT'',
      ➥''ÜGYFÉL_KÓD'')';
    ELSE
      D:='1=1';
    END IF;
    RETURN D;
  END "FELTÉTEL";
END;
/
```

A PRAGMA RESTRICT_REFERENCES direktívával jelezzük, hogy ez a tárolt eljárás nem fogja módosítani az adatbázist, tehát lekérdezésben kívánjuk használni. A rendszer két paraméterrel (a séma nevével, amelyben a tábla van, és a tábla nevével) fogja a FELTÉTEL függvényt meghívni. A USER változó tartalma a felhasználó neve. A FELTÉTEL függvény visszatérési értékével fog az *SQL* utasítás WHERE része kiegészülni. Látható, hogy a logikailag mindig igaz, 1=1 értéket adja vissza, ha az utasítást kiadó felhasználó megegyezik a tábla tulajdonossal, a SYS, vagy a SYSTEM felhasználóval.
A szabály elkészítése után már csak meg kell határoznunk, hogy mely táblára és milyen adatbázis művelet esetén kívánjuk alkalmazni:

```
BEGIN DBMS_RLS.ADD_POLICY
➥('RENDELÉSEK','RENDELÉSEK',
➥'RENDELÉS_SZABÁLY','SZABÁLY',
➥'"ÜGYFÉL_BIZT"."FELTÉTEL"',
➥'SELECT');
END;
/
```

A RENDELÉSEK séma RENDELÉSEK táblájára létrehozzuk a RENDELÉS_SZABÁLY-t, amelynek érvényesítését a SZABÁLY séma ÜGYFÉL_BIZT csomagjának FELTÉTEL nevű függvénye fogja elvégezni, minden SELECT művelet esetében.

## A VPD működése

A SCOTT felhasználó bejelentkezése során az adatbázis LOGON_TRIGGER meghívja az AZONOSÍTÓ tárolt eljárást, amely az ÜGYFELEK tábla alapján meghatározza az ügyfél azonosítóját (102) és a RENDELÉSEK_BIZT névtér ÜGYFÉL_KÓD változójába tárolja. Ezután ha SCOTT lekérdezi a RENDELÉSEK táblát, akkor az FELTÉTEL függvény visszatérési értéke az "ÜGYFÉL_AZ"=102 lesz. Az eredeti

```
"select * from
➥RENDELÉSEK.RENDELÉSEK"

lekérdezés tehát a

"select * from
➥RENDELÉSEK.RENDELÉSEK where
➥ÜGYFÉL_AZ=102"
```

utasítássá alakul át. Az eredmény pedig ez lesz:

```
RENDELÉS_AZ RENDELÉS_DÁT
➥ÜGYFÉL_AZ
----------- ------------ ------
        100 07-FEBR. -26    102
```

Összefoglalva a bemutatott példa jól szemlélteti, hogy hogyan lehet az *Oracle VPD* technológiájával hatékonnyá tenni a konszolidációt a biztonsági kockázatok kezelése mellett.

*Mosolygó Ferenc*