EDITION BASED REDEFINITION - VISOKA DOSTUPNOST KOD NADOGRADNJE APLIKACIJA

EDITION BASED REDEFINITION – ONLINE APPLICATION UPGRADE

Alen Prodan
Login d.o.o.
Mihačeva draga b.b.
51000 Rijeka
+385 91 156 44 68
alen.prodan@login.hr
http://www.login.hr

SAŽETAK

Nadogradnja aplikacija u produkcijskom okruženju predstavlja znatan izazov. Sve je veći broj organizacija koje zahtijevaju visoku dostupnost kako bi osigurale kontinuitet poslovnih procesa. U tom smislu, aplikacijsko rješenje i baza podataka moraju imati mogućnost online nadogradnje.

Oracle RDBMS kroz postojeće tehnologije pruža visoku dostupnost za planirane ali i za neplanirane prekide u radu. Kao novina u tom nizu, u verziji Oracle 11.2 baze podataka uvedena je podrška za online nadogradnju aplikacija, kroz novo svojstvo pod nazivom edition-based redefinition. U radu je prikazan pregled novog svojstva baze podataka.

ABSTRACT

Upgrading applications in a production environment is a significant challenge. There is an increasing number of organizations that cannot tolerate any down time during application upgrades. To this aim, applications and underlying database systems must be able to support online upgrades. Oracle RDBMS already provides HA solutions for both planned and unplanned outages.

As of Oracle 11.2 there is a new feature, edition-based redefinition, that enables DBAs and developers to upgrade an application's database objects while the application is in use, ensuring minimal or no down time. The aim of this paper is to provide an overview of this new feature.

UVOD

Edition-based redefinition, novo svojstvo Oracle 11.2 baze podataka, omogućuje izmjene nad objektima aplikacijske sheme bez prekida dostupnosti aplikacije. Sve promjene izvode se unutar edicija, a edicije omogućuju da se promjene izvode u izoliranom okruženju. Svaka baza podataka ima najmanje jednu ediciju. Administrator baze podataka kreira novu ediciju kao nasljednika (child) postojeće. Promjene se izvode u okviru child edicije, dok korisnici nastavljaju koristiti aplikaciju unutar roditeljske (parent) edicije.

Kod nadogradnje aplikacija u živo uvijek se koristi **edition** funkcionalnost koja omogućuje kopiranje objekata unutar edicije i njihovu izmjenu u izolaciji. Ako svi tipovi objekata koje treba redefinirati podržavaju edicije to je ujedno i jedina potrebna funkcionalnost.

Tablice ne podržavaju edicije. Ako prilikom nadogradnje mijenjamo strukturu jedne ili više tablica, onda moramo koristiti **editioning view** funkcionalnost.

Ako korisnici na sustavu moraju biti u mogućnosti mijenjati podatke u tablicama dok se mijenja struktura tablica, potrebno je koristiti **forward crossedition trigger**. Ukoliko su istovremeno u upotrebi i stara i nova verzija (edicija) aplikacije, tada je potrebno koristiti i **reverse crossedition trigger**.

1. UPRAVLJANJE EDICIJAMA

Edicije su objekti unutar baze podataka koji ne pripadaju niti jednoj pojedinoj shemi, odnosno one nemaju vlasnika. Edicije su kreirane u jedinstvenom prostoru imena (namespace), a unutar baze

podataka može postojati istovremeno više edicija. Svaka baza podataka mora imati najmanje jednu ediciju, pa tako svaka novo-kreirana ili nadograđena baza podataka posjeduje jednu ediciju pod nazivom ORA$BASE.

1.1. Koji se objekti mogu pridružiti edicijama ?

Edicijski objekti (editioned objects) predstavljaju objekte unutar sheme čiji se tipovi objekta mogu pridružiti različitim edicijama, a shemi-vlasniku objekta je omogućeno upravljanje s edicijama. Objekt unutar sheme koji pripada tipu objekta koji se može pridružiti edicijama, ali vlasniku sheme nije omogućeno upravljanje edicijama predstavlja potencijalni objekt za verzioniranje. Svaka edicija može imati svoju privatnu kopiju edicijskog objekta i samo ta kopija je vidljiva unutar edicije.

Objekti unutar sheme koje nije moguće verzionirati spadaju u ne-edicijski tip objekta (noneditioned objects). Edicija ne može imati svoju vlastitu kopiju ne-edicijskog objekta. Ne-edicijski objekti su identični i na podjednak način vidljivi unutar svih edicija. Tipovi objekata koji podržavaju edicije su:

- funkcije (FUNCTION)
- biblioteke (LIBRARY)
- paketi (PACKAGE i PACKAGE BODY)
- procedure (PROCEDURE)
- sinonimi (SYNONYM), ali ne i public sinonimi
- okidači (TRIGGER)
- tipovi (TYPE i TYPE BODY)
- pregledi (VIEW)

Tipovi objekata vezani uz Javu, kao i tipovi objekata u koje pohranjujemo podatke (tablice, indeksi i sl.) ne podržavaju edicije.

Tipovi objekata koji ne podržavaju edicije ne mogu ovisiti o tipu objekta koji se može verzionirati, pa tako npr. function based indeks ne može ovisiti o funkciji koja je verzionirana, ili pak materijalizirani view ne može ovisiti o običnom viewu ukoliko je on pridružen ediciji.

Edicijski objekt je jednoznačno identificiran atributima OBJECT_NAME, OWNER i EDITION_NAME. Ne-edicijski objekt je jednoznačno identificiran atributima OBJECT_NAME i OWNER, dok EDITION_NAME atribut ima vrijednost NULL.

1.2. Omogućavanje korištenja edicija unutar aplikacijske sheme

Prvi korak u korištenju edition-based redefinition mehanizma je njegovo aktiviranje za shemu-vlasnika objekata koje želimo redefinirati. To je moguće napraviti koristeći ENABLE EDITIONS klauzulu CREATE USER, odnosno ALTER USER naredbe. Npr:

```
alter user test enable editions;
```

U stupcu EDITIONS_ENABLED *_USERS porodice viewova moguće je vidjeti za koje je sheme aktivirano korištenje edicija. Aktiviranje edicija je neopoziva operacija, odnosno ne postoji odgovarajuća DISABLE EDITIONS naredba.

1.3. Kreiranje edicija, nasljeđivanje i aktualizacija objekata

Svaka baza podataka mora imati barem jednu ediciju, a svaka nova instalacija ili nadograđena instalacija sa starije verzije na Oracle 11.2 ima osnovnu ediciju ORA$BASE.

Edicije su organizirane u roditelj-dijete (parent-child) hijerarhiju uz ograničenje da svaka edicija može imati samo jedno dijete (child). Prilikom pokušaja kreiranja više od jedne edicije za istog roditelja sustav javlja grešku:

```
ORA-38807: Implementation restriction: an edition can have only one child
```

Za kreiranje edicija koristimo CREATE EDITION naredbu, za koju je potrebno imati CREATE ANY EDITION sistemsku privilegiju. Ukoliko želimo kreirati više edicija koje su međusobno u relaciji, npr. ORA$BASE => E1 => E2, to možemo učiniti pomoću sljedećih SQL naredbi:

```
SQL> create edition e1 as child of ora$base;
SQL> create edition e2 as child of e1;
```

Novo-kreiranu ediciju prvotno može koristiti samo korisnik koji ju je kreirao i SYS korisnik. Korisniku koji je kreirao ediciju sustav automatski pridružuje USE ON EDITION privilegiju sa GRANT opcijom. Kako bi i drugi korisnici na sustavu mogli koristiti novu ediciju, potrebno im je također dodijeliti USE ON EDITION privilegiju za odgovarajuću ediciju:

```
SQL> grant use on edition e1 to test;
SQL> grant use on edition e2 to test;
```

Alternativno je moguće ediciju učiniti dostupnom svim korisnicima na sustavu dodijeljujući USE privilegiju PUBLIC shemi:

```
SQL> grant use on edition e1 to public;
```

Informacije o postojećim edicijama dostupne su kroz ALL_EDITIONS i DBA_EDITIONS viewove. Unutar sesije moguće je u jednom trenutku koristiti samo jednu ediciju. Prilikom kreiranja, child edicija nasljeđuje od svoje roditeljske (parent) edicije sve edicijske objekte koji su vidljivi parent ediciji. Ti nasljeđeni objekti su isključivo vidljivi i oni još uvijek ne pripadaju child ediciji. Kako bi nasljeđeni objekt pripao child ediciji potrebno ga je aktualizirati. To je postupak koji se događa u trenutku kada korisnik u child ediciji koristeći DDL naredbu (osim DROP naredbe) izmjeni ili kompilira nasljeđeni objekt. Nakon toga, originalna verzija objekta nasljeđena od roditeljske edicije nije više vidljiva unutar child edicije. Navedeno će biti najrazumljivije prikažemo li to kroz primjer. U primjeru će biti korištena shema AE koju smo prethodno kreirali, uključili mogućnost korištenja edicija (ENABLE EDITIONS), te dodijelili odgovarajuće privilegije.
Kao prvi korak, moramo kreirati dvije testne edicije E1 i E2:

```
SQL> create edition e1 as child of ora$base;

Edition created.

SQL> create edition e2 as child of e1;

Edition created.
```

Zatim, postavljamo E1 kao tekuću ediciju, te u toj ediciji kreiramo PL/SQL proceduru naziva TEST, ujedno je testiramo izvršavanjem:

```
SQL> alter session set edition = e1;

SQL> create or replace procedure test
  2  is
  3  begin
  4   dbms_output.put_line('Edicija E1');
  5  end;
  6  /

Procedure created.

SQL> set serveroutput on size 100000
SQL> exec test;
Edicija E1

PL/SQL procedure successfully completed.
```

Vidimo da je procedura uspješno završila i ispisala poruku: “Edicija E1“. Sada ćemo unutar iste sesije postaviti ediciju E2 kao tekuću ediciju, te ćemo probati izvršiti ponovno proceduru TEST.

```
SQL> alter session set edition = e2;

Session altered.

SQL> exec test;
Edicija E1

PL/SQL procedure successfully completed.
```

Vidimo da sustav ispisuje poruku “Edicija E1“, što znači da unutar E2 child edicije koristimo nasljeđenu proceduru čija se originalna kopija nalazi unutar E1 roditeljske edicije. U nastavku testa, izvršiti ćemo redefiniranje (izmjenu) TEST procedure unutar E2 edicije, kako bi izvršili mehanizam aktualizacije, kao rezultat kojega će se ediciji E2 pridružiti privatna kopija procedure TEST:

```
SQL> create or replace procedure test
  2  is
  3  begin
  4   dbms_output.put_line('Edicija E2');
  5  end;
  6  /

Procedure created.

SQL> exec test;
Edicija E2

PL/SQL procedure successfully completed.
```

Na temelju poruke koju ispisuje procedura TEST vidljivo je da se je izvršila kopija TEST procedure pridružene ediciji E2. Na kraju primjera, unutar iste sesije, ponovno ćemo referencirati originalnu proceduru unutar roditeljske E1 edicije, jednostavnim postavljenjem E1 edicije kao tekuće edicije unutar sesije:

```
SQL> alter session set edition = e1;

Session altered.

SQL> exec test;
Edicija E1

PL/SQL procedure successfully completed.
```

Na kraju primjera zanimljivo je i provjeriti činjenicu da zaista imamo vlastitu kopiju TEST procedure unutar E1 i E2 edicije, koristeći USER_OBJECTS_AE view:

```
SQL> select object_name, object_type, edition_name
  2  from user_objects_ae
  3  /

OBJECT_NAME                    OBJECT_TYPE         EDITION_NAME
------------------------------ ------------------- --------------
TEST                           NON-EXISTENT        ORA$BASE
TEST                           PROCEDURE           E1
TEST                           PROCEDURE           E2
```

1.4. Izbor tekuće edicije unutar sesije

Svaka sesija unutar baze podataka može istovremeno koristiti samo jednu ediciju. Edicija koju sesija koristi u nekom trenutku u vremenu nazivamo tekućom edicijom (current edition). Prilikom kreiranja sesije, tekuća edicija ujedno predstavlja i ediciju sesije (session edition).

Kao što je bilo vidljivo u prethodnom primjeru, ediciju je unutar sesije moguće jednostavno mijenjati koristeći ALTER SESSION SET EDITION SQL naredbu. Koristeći SYS_CONTEXT funkciju na jednostavan način možemo provjeriti trenutnu ediciju ili ediciju sesije:

```
SQL> select sys_context('USERENV', 'SESSION_EDITION_NAME') from dual;

SYS_CONTEXT('USERENV','SESSION_EDITION_NAME')
--------------------------------------------------------------------------------
E1

SQL> select sys_context('USERENV', 'CURRENT_EDITION_NAME') from dual;

SYS_CONTEXT('USERENV','CURRENT_EDITION_NAME')
--------------------------------------------------------------------------------
E1
```

Ukoliko unutar sesije nije posebno izabrana edicija koja će se koristiti, sustav automatski dodjeljuje sesiji osnovnu database ediciju (default database edition), a prema osnovnim postavkama to je ORA$BASE edicija. Osnovnu (default) ediciju moguće je promijeniti uz pomoć ALTER DATABASE naredbe, kao što slijedi:

```
SQL> alter database default edition = e2;

SQL> select property_name, property_value
 2  from database_properties
 3  where property_name = 'DEFAULT_EDITION';

PROPERTY_NAME                  PROPERTY_VALUE
------------------------------ --------------------
DEFAULT_EDITION                E2
```

Važno je znati da kao popratni efekt sustav automatski dodijeljuje USE privilegiju nad default database edicijom PUBLIC shemi.

Na razini instance moguće je provjeriti ediciju koja se koristi unutar pojedine sesije pomoću V$SESSION viewa:

```
SQL> select s.sid, s.username, d.object_name
  2  from v$session s, dba_objects d
  3  where s.session_edition_id = d.object_id
  4  /

       SID USERNAME                       OBJECT_NAME
---------- ------------------------------ --------------------
       132                                E2
        22 ALEN                           E2
```

1.5. Povlačenje (retiring) i brisanje (dropping) edicija

Nakon uspješno izvedene nadogradnje, nova edicija postaje dostupna za korištenje svim korisnicima na sustavu. Stara edicija se povlači iz upotrebe čime prestaje njezina generalna dostupnost korisnicima sustava. Edicija se povlači iz upotrebe jednostavnim oduzimanjem USE privilegije (revoke) nad edicijom korisnicima kojima je ta privilegija dodijeljena. Više detalja o tome koji

korisnici raspolažu navedenom USE privilegijom moguće je saznati iz DBA_TAB_PRIVS sistemskog viewa:

```
SQL> select grantee, table_name, privilege
  2  from dba_tab_privs
  3  where table_name in ('E1', 'E2');

GRANTEE                        TABLE_NAME                     PRIVILEGE
------------------------------ ------------------------------ ---------
AE                             E1                             USE
AE                             E2                             USE
```

Ediciju koja nam više nije potrebna, možemo također i izbrisati sa sustava. Za to koristimo DROP EDITION privilegiju. Ukoliko su ediciji koju želimo obrisati pridruženi objekti, potrebno je izvršiti DROP EDITION naredbu sa CASCADE klauzulom:

```
SQL> drop edition e2;
drop edition e2
*
ERROR at line 1:
ORA-38811: need CASCADE option to drop edition that has actual objects


SQL> drop edition e2 cascade;

Edition dropped.
```

2. UPRAVLJANJE EDITIONING VIEWOVIMA

Tablice spadaju u skupinu ne-edicijskih objekata i stoga nije moguće koristiti edicije za redefiniranje njihove strukture. Umjesto toga, moguće je za svaku tablicu kreirati editioning view, te u aplikaciji zamijeniti referenciranje tablica sa referencama prema editioning viewovima. Editioning viewovi na takav način tvore svojevrsno sučelje između aplikacije i fizičkih struktura sheme.

I dok je nad običnim viewovima moguće kreirati jedino INSTEAD OF vrstu triggera, editioning viewovi podržavaju sve tipove triggera koje je moguće definirati nad tablicama osim INSTEAD OF triggera i crossedition triggera, a potonji su po svojoj naravi i tako privremenog karaktera. Mogućnost kreiranja triggera nad editioning viewovima i mogućnost da se editioning viewovi mogu pridruživati edicijama stvara privid kao da su i same fizičke tablice verzionirane. Postoji i ograničenje - nije moguće kreirati indekse ili constrainte nad editioning viewovima, već ih je jedino moguće kreirati direktno nad pripadajućim tablicama.

Editioning viewovi nam omogućuju da eksponiramo podskup kolona iz pripadajućih tablica, te omogućuju definiranje zamjenskih naziva (alias) za te kolone, čime efektivno putem viewa mapiramo fizičke nazive kolona iz tablice sa logičkim nazivima kolona iz viewa.

Za kreiranje editioning viewa, potrebno je koristiti EDITIONING ključnu riječ u sklopu CREATE VIEW naredbe. Nadalje, editioning view može dozvoliti samo čitanje, pa ga je onda potrebno kreirati sa WITH READ ONLY klauzulom, u suprotnom dovoljno je izostaviti WITH READ ONLY klauzulu. Primjer kreiranja editioning viewa, sa WITH READ ONLY klauzulom:

```
create editioning view test_ev
as
select id, naziv from tab
with read only;
```

Za vrijeme nadogradnje aplikacije, editioning viewovi mogu biti postavljeni u modalitet da dozvoljavaju samo čitanje ili pak da dozvoljavaju i čitanje i pisanje. Jednostavnija je varijanta u kojoj editioning viewovi dozvoljavaju samo čitanje, iako se na taj način smanjuje funkcionalnost aplikacije. Ako okolnosti ne dozvoljavaju onemogućavanje izmjena podataka za vrijeme trajanja nadogradnje, potrebno je kreirati crossedition triggere.

3. UPRAVLJANJE CROSSEDITION TRIGGERIMA

Crossedition triggere koristimo u slučaju kada su ispunjena dva uvjeta. Prvo, struktura tablica mora biti redefinirana za vrijeme nadogradnje aplikacije. Drugo, nije moguće privremeno dozvoliti samo čitanje podataka postavljajući editioning viewove u read-only modalitet rada.

Redefiniranjem tablice mijenjamo njezinu strukturu, međutim potrebno je povesti brigu i o podacima. Prvo, podaci koji se modificiraju putem aplikacije za vrijeme trajanja nadogradnje moraju biti pohranjeni u nove izmjenjene strukture tablice. Drugo, već postojeći podaci pohranjeni u redefiniranoj tablici moraju također biti konvertirani u novu strukturu.

Sa prvim problemom možemo se uloviti u koštac korištenjem forward crossedition triggera unutar nove edicije. Svrha takvog triggera je transformirati redak iz stare strukture retka u novu. Iako se trigger kreira u novoj ediciji, on se okida samo ukoliko se izvrši DML naredba u sesiji koja koristi prethodnu ediciju u odnosu na onu u kojoj je trigger definiran. Stoga je prilikom kreiranja triggera potrebno eksplicitno navesti da želimo napraviti forward crossedition trigger, koristeći FORWARD CROSSEDITION klauzulu. Trigger također mora biti kreiran u statusu disabled, što je važno kako bi spriječili mogućnost da invalidni trigger negativno utječe na dostupnost aplikacije.

Drugi problem rješavamo transformacijom već pohranjenih podataka, za što nam na raspolaganju stoji više mogućnosti. Najlakši način je da izvršimo trigger za svaki postojeći redak u redefiniranoj tablici, uz nužnu opasku da to može biti vrlo spora operacija. U tu svrhu koristimo DBMS_SQL paket, sa novom verzijom PARSE funkcije, koja koristi APPLY_CROSSEDITION_TRIGGER parametar. U dosadašnjem tekstu bilo je riječi o propagiranju izmjena generiranih od strane aplikacije koja radi nad starom edicijom u izmjenjene strukture unutar nove edicije. No što se događa ako istovremeno dolazi do izmjena podataka kroz staru i novu ediciju aplikacije ? Ta situacija je tipična za scenarij nadogradnje “u živo“. Za taj specifičan scenarij dodatno koristimo i reverse crossedition trigger. Svrha takvog triggera je da izvodi obrnute operacije u odnosu na forward crossedition trigger. Drugim riječima, njegova namjena je transformacija retka iz nove strukture u staru. I u ovom slučaju trigger kreiramo u novoj ediciji. Reverse crossedition trigger se izvršava samo kod izvođenja DML naredbe unutar sesije koja koristi ediciju unutar koje je trigger definiran ili nasljednika te edicije. Zbog toga je potrebno eksplicitno navesti da se želi kreirati reverse crossedition trigger koristeći REVERSE CROSSEDITION klauzulu unutar CREATE TRIGGER naredbe.

4. PRIMJER NADOGRADNJE APLIKACIJSKE SHEME

U nastavku slijedi primjer nadogradnje aplikacije koji se temelji na korištenju edicija, editioning viewa, forward crossedition triggera, te reverse crossedition triggera. Primjer aplikacije na kojoj će biti prikazan postupak nadogradnje sastoji se od jedne tablice.

```
create table imenik
(
 id number,
 naziv varchar2(20),
 telefon varchar2(15)
);
```

Tablica sadrži telefonski imenik sa ukupno 5 slogova. U stupcu telefon upisani su brojevi telefona sa uključenim predbrojem.

```
SQL> select * from imenik;

        ID NAZIV                TELEFON
---------- -------------------- ---------------
         1 ivan ivić            051/111-2222
         2 pero perić           051/222-3333
         3 jurica jurić         051/333-4444
         4 mate matić           051/444-5555
         5 luka lukić           051/555-6666
```

Pretpostavimo da želimo redefinirati tablicu IMENIK na način da odvojimo predbroj od telefonskog broja korisnika. U novoj verziji predbroj će se nalaziti u zasebnom stupcu. Također, za vrijeme izvođenja nadogradnje, drugi korisnici moraju biti u mogućnosti ažurirati podatke iz tablice IMENIK. U tom scenariju biti će nam potrebna sva svojstva edition-based redefinition mehanizma: edicija, editioning view jer mijenjamo strukturu tablice, te crossedition triggeri, s obzirom da postoji potreba za istovremenim ažuriranjem podataka u tablici dok traje nadogradnja aplikacije.
Kao prvi korak, u sklopu postojeće aktivne edicije (E1) preimenovati ćemo tablicu IMENIK u IMENIK_TAB, kako bi mogli dodijeliti njezino ime editioning viewu:

```
alter table imenik rename to imenik_tab;
```

Zatim kreiramo editioning view pod nazivom IMENIK unutar postojeće edicije (E1):

```
create editioning view imenik
as
select id, naziv AS ime_prezime, telefon
from imenik_tab;
```

Idući korak kojega moramo poduzeti je kreirati novu ediciju E2, nasljednika postojeće edicije E1,

```
create edition e2 as child of e1;
```

te postaviti novu ediciju kao aktivnu ediciju unutar sesije:

```
alter session set edition = e2;
```

U ovom trenutku spremni smo pristupiti izmjeni fizičke strukture IMENIK_TAB tablice:

```
alter table imenik_tab add (predbroj varchar2(3), tel_broj varchar2(9));
```

Nakon izmjene fizičkih struktura IMENIK_TAB tablice, moramo redefinirati IMENIK editioning view, kako bi i on u novoj ediciji odražavo nastale promjene u fizičkoj strukturi tablice:

```
create or replace editioning view imenik
as
select id, naziv as ime_prezime, predbroj, tel_broj
from imenik_tab;
```

Ako izvršimo select iz imenik viewa u novoj ediciji, vidjeti ćemo da u ovom trenutku kolone predbroj i tel_broj imaju vrijednost NULL u sebi:

```
SQL> select * from imenik;

        ID IME_PREZIME          PRE TEL_BROJ
---------- -------------------- --- ---------
         1 ivan ivić
         2 pero perić
         3 jurica jurić
         4 mate matić
         5 luka lukić
```

Da bi se podaci u tim kolonama počeli pohranjivati, potrebno je postaviti pravila transformacije podataka iz stare u novu ediciju, a to činimo putem forward crossedition triggera:

```
create trigger imenik_fw_xed_trig
before insert or update on imenik_tab
for each row
FORWARD CROSSEDITION
disable
```

```
BEGIN
 :new.predbroj := substr(:new.telefon, 1, 3);
 :new.tel_broj := substr(:new.telefon, 5);
END;
```

Dobra praksa nalaže da se inicijalno trigger kreira u disable statusu dok se ne uvjerimo da je uredno kompiliran kako ne bi utjecali na rad postojećih korisnika. Jednom kada se uvjerimo da je sve u redu možemo ga postaviti u enable status:

```
alter trigger imenik_fw_xed_trig enable;
```

Forward crossedition trigger kojega smo upravo kreirali brinuti će se za transformaciju strukture redaka obuhvaćenih DML operacijama u staroj ediciji (E1). Međutim, većina postojećih redaka koji neće biti obuhvaćeni DML operacijama ostati će nepromijenjeni. Stoga, naš je idući korak transformacija strukture svih postojećih redaka u tablici. To ćemo učiniti uz pomoć DBMS_SQL.PARSE procedure koja od verzije Oracle servera 11.2 prima novi parametar APPLY_CROSSEDITION_TRIGGER. Tim parametrom određujemo naziv forward crossedition triggera koji će biti apliciran prilikom izvođenja SQL naredbe kroz DBMS_SQL paket. Prije nego izvršimo transformaciju pomoću DBMS_SQL paketa, potrebno je provjeriti da li ima postojećih otvorenih transakcija nad tablicom koju planiramo ažurirati. Za tu provjeru koristiti ćemo se pozivom na funkciju WAIT_ON_PENDING_DML koja dolazi u sklopu DBMS_UTILITY paketa. Funkcija čeka dok sve transakcije (osim onih koje je pokrenuo pozivatelj) koje imaju zaključane retke nad navedenom tablicom, a započele su prije navedenog SCN-a potvrde ili ponište svoje transakcije. Ako je SCN vrijednost koju predajemo NULL, sustav uzima tekući SCN.

```
declare
 r boolean;
 scn number;
begin
 r := dbms_utility.wait_on_pending_dml(
  tables=>'AE.IMENIK_TAB',
  timeout=>null,
  scn=>scn);
end;
```

Nakon što pozivom na DBMS_UTILITY.WAIT_ON_PENDING_DML provjerimo da nema otvorenih transakcija nad našom tablicom, možemo pristupiti transformaciji postojećih redaka:

```
declare
 cur number;
 nrows number;
begin
 cur := dbms_sql.open_cursor;
 dbms_sql.parse(
  c => cur,
  statement => 'UPDATE imenik SET id = id',
  language_flag => dbms_sql.native,
  apply_crossedition_trigger => 'imenik_fw_xed_trig',
  fire_apply_trigger => true
 );
 nrows := dbms_sql.execute(cur);
 dbms_sql.close_cursor(cur);

 commit;
end;
```

Iz primjera poziva DBMS_SQL.PARSE procedure možemo primjetiti dvije važne stvari. Putem statement parametra definirana je DML naredba kojom radimo promjene nad podacima. U ovom slučaju zapravo i ne mijenjamo ništa, jer stupcu ID dodijeljujemo već postojeću vrijednost. Putem

APPLY_CROSSEDITION_TRIGGER parametra naveli smo naziv forward crossedition triggera, a u ovom slučaju riječ je o IMENIK_FW_XED_TRIG triggeru.
Nakon što je završila transformacija, možemo ponoviti naš upit nad IMENIK viewom i provjeriti da li smo postigli željeni rezultat:

```
SQL> select * from imenik;

        ID IME_PREZIME          PRE TEL_BROJ
---------- -------------------- --- ---------
         1 ivan ivić            051 111-2222
         2 pero perić           051 222-3333
         3 jurica jurić         051 333-4444
         4 mate matić           051 444-5555
         5 luka lukić           051 555-6666
```

Kao što možemo vidjeti iz rezultat prethodnog upita, transformacija je uspješno izvedena. Međutim, naš test ne bi bio potpun kada ne bi testirali transformaciju u oba smjera: iz stare u novu ediciju (forward crossedition trigger), te iz nove u staru ediciju (reverse crossedition trigger). Za potonju nam je potrebno kreirati reverse crossedition trigger kako slijedi:

```
create trigger imenik_rv_xed_trig
before insert or update on imenik_tab
for each row
REVERSE CROSSEDITION
disable
BEGIN
 :new.telefon := :new.predbroj||'/'||:new.tel_broj;
END;

alter trigger imenik_rv_xed_trig enable;
```

Prvo ćemo izvršiti insert u novoj ediciji, kako bi testirali novo-kreirani reverse crossedition trigger:

```
insert into imenik (id, ime_prezime, predbroj, tel_broj) values (100, 'testni
korisnik', '051', '123-4567');

commit;
```

Zatim ćemo otvoriti zasebnu sesiju kroz SQL*Plus, te ćemo postaviti tekuću ediciju unutar sesije na staru ediciju E1, a potom ćemo upitom iz IMENIK viewa, provjeriti da li je korektno prikazan netom insertirani redak:

```
SQL> conn ae/ae
Connected.
SQL> alter session set edition = e1;

SQL> select * from imenik;

        ID IME_PREZIME          TELEFON
---------- -------------------- ---------------
         1 ivan ivić            051/111-2222
         2 pero perić           051/222-3333
         3 jurica jurić         051/333-4444
         4 mate matić           051/444-5555
         5 luka lukić           051/555-6666
       100 testni korisnik      051/123-4567

6 rows selected.
```

Kao što možemo vidjeti redak sa vrijednosti ID = 100 postoji i ispravno je prezentiran u strukturi retka iz E1 edicije. Sada ćemo unutar te iste sesije sa tekućom edicijom E1 insertirati još jedan redak kako bi testirali forward crossedition trigger:

```
insert into imenik (id, ime_prezime, telefon) values (101, 'testni korisnik2',
'051/765-4321');

commit;
```

Na kraju ćemo provjeriti u sesiji sa tekućom edicijom E2, da li je netom insertirani redak transformiran u skladu sa pravilima definiranim putem forward crossedition triggera:

```
SQL> alter session set edition = e2;

Session altered.

SQL> select * from imenik;

        ID IME_PREZIME          PRE TEL_BROJ
---------- -------------------- --- ---------
         1 ivan ivić            051 111-2222
         2 pero perić           051 222-3333
         3 jurica jurić         051 333-4444
         4 mate matić           051 444-5555
         5 luka lukić           051 555-6666
       100 testni korisnik      051 123-4567
       101 testni korisnik2     051 765-4321

7 rows selected.
```

Na temelju rezultata gornjeg upita možemo se uvjeriti u točnost rada forward crossedition triggera, te konstatirati da su i u ovom slučaju svi retci ispravno transformirani.

ZAKLJUČAK

U Oracle Database 11g Release 2, postoje dva ozbiljna ograničenja koja se odnose na hijerarhije edicija. Prvo ograničenje odnosi se na činjenicu da svaka edicija može imati samo jednog nasljednika. Drugo ograničenje odnosi se na to da baza podataka može imati samo jednu izvornu (root) ediciju.[1] U današnje vrijeme nije rijetkost da se više različitih aplikacija konsolidira unutar iste baze podataka (npr. financije i crm). Postojanja više izvornih (root) edicija, pružilo bi mogućnost da svaka od tih aplikacija može imati svoju vlastitu hijerarhiju verzija.

Unatoč navedenim ograničenjima, edition-based redefinition predstavlja jednu od najvažnijih novih funkcionalnosti baze podataka bez koje su nadogradnje sustava u živo znatno teže izvedive.

LITERATURA

Oracle Database Advanced Application Developer's Guide 11g Release 2 (E10471-05), March 2010
Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (E10577-05), Sept. 2009
Oracle Database SQL Language Reference 11g Release 2 (E10592-04), October 2009
Antognini, Christian: Edition-Based Redefinition, January 2010

[1] Antognini, Christian: Edition-Based Redefinition, January 2010.