Editions Based Redefinition: Tijdens de verbouwing gaat de verkoop door.

Lucas Jellema, Alex Nuijten - AMIS Services BV

In een vorige editie van Optimize hebben we bekeken wat voor nieuwe toeters en bellen er voor database ontwikkelaars in de Oracle 11g Release 2 zitten. Veel mooie en spectaculaire features hebben de revue reeds gepasseerd. Maar het klapstuk moet nog komen, en dat staat in dit artikel: Edition Based Redefinition.
In de Top 11 van Oracle 11g features zet Tom Kyte, welbekend Oracle goeroe, Edition Based Redefinition op plaats 2. En ook op nummer 1. Om te onderstrepen hoe belangwekkend hij Edition Based Redefinition wel niet acht.
We bespreken eerst de concepten van Edition Based Redefinition, vervolgens laten we aan de hand van code voorbeelden zien hoe Oracle dit concept heeft geïmplementeerd en als laatste laten we de ondersteuning zien die in SQL Developer hiervoor gemaakt is.

Werk aan de weg

Edition Based Redefinition is in het leven geroepen om geplande downtime van database applicaties tot nul te reduceren. Geplande downtime is de tijd dat een applicatie niet beschikbaar is voor eindgebruikers omdat er een upgrade of patch uitrol plaats vindt. Deze functionaliteit is ook goed te gebruiken om in plaats van "big-bang" applicatie-migraties, geleidelijk een migratie uit te voeren - waarbij verschillende versies van een applicatie tegelijk actief zijn. Op deze manier worden eindgebruikers niet gedwongen om direct na de release de laatste versie van een applicatie te gaan gebruiken.

De release van een nieuwe versie van een database applicatie heeft vaak nogal wat gevolgen. Nieuwe objecten - zoals views, triggers, tabellen, packages- worden aangemaakt. Bestaande objecten worden gewijzigd; bestaande data dient te worden geconverteerd. Hoe dan ook, de applicatie kan tijdelijk niet gebruikt worden. Vanaf de allereerste wijziging aan een bestaand object is de applicatie al niet meer beschikbaar: andere objecten kunnen worden geinvalideerd of er ontstaat logische inconsistentie tussen al wel en nog niet gemigreerde objecten. De ontstane downtime is uiteraard afhankelijk van de wijzigingen die worden doorgevoerd, maar kan van enkele minuten tot vele uren bedragen.

Als Rijkswaterstaat eenzelfde manier van upgraden van het wegennetwerk zou hanteren, dan zouden de dagelijkse files vele malen langer zijn dan nu het geval is. Een bestaande weg (de applicatie) zou afgesloten worden (downtime) om een verbetering uit te voeren. Gelukkig doen ze dit niet. Een nieuwe, verbeterde, weg wordt naast de bestaande weg gelegd en wanneer deze klaar is dan pas word de bestaande weg afgesloten en de nieuwe weg in gebruik genomen. Je merkt er haast niets van,... tenminste dat is de bedoeling. Eenzelfde soort mechanisme word ook door Oracle gebruikt met Edition Based Redefinition. Door de nieuwe applicatie naast de in gebruik zijnde applicatie te plaatsen en beschikbaar te maken wanneer deze volledig geinstalleerd is, verminderd de downtime van de applicatie aanzienlijk. Net als in het voorbeeld van de wegen; je hoeft de bestaande weg niet af te sluiten die kan gewoon in gebruik blijven - zo hoef je de bestaande applicatie niet af te sluiten, deze kan nog gewoon in gebruik blijven.
Nieuwe database sessies kunnen gebruik gaan maken van de nieuwe versie van de applicatie. Bestaande database sessie kunnen gewoon doorgaan met het gebruik van de "oude" bestaande applicatie. En alle versies van de applicaties werken tegen dezelfde data.

Editions

Hoe heeft Oracle het principe - meerdere applicatie-versies naast elkaar draaien - nu geïmplementeerd? De introductie van Editions maakt dit mogelijk. Editions zijn geïsoleerde omgevingen waarin objecten aangemaakt en gewijzigd kunnen worden zonder invloed op elkaar te hebben. Om een object te kunnen identificeren in de Oracle database was tot nu toe de combinatie van schema, naam en object type doorslaggevend. Editions voegt daar een extra dimensie toe. Het is dus niet meer voldoende om te weten dat een package in het SCOTT schema staat met een bepaalde naam, maar je dient ook aan te geven in welke Edition deze staat. De meeste objecten zijn "editionable", sommige niet - later meer hierover. Op het moment dat je naar Oracle 11gR2 upgrade dan maak je automatisch gebruik van Editions. De basis edition die altijd in 11gR2 aanwezig is heet ORA$BASE.

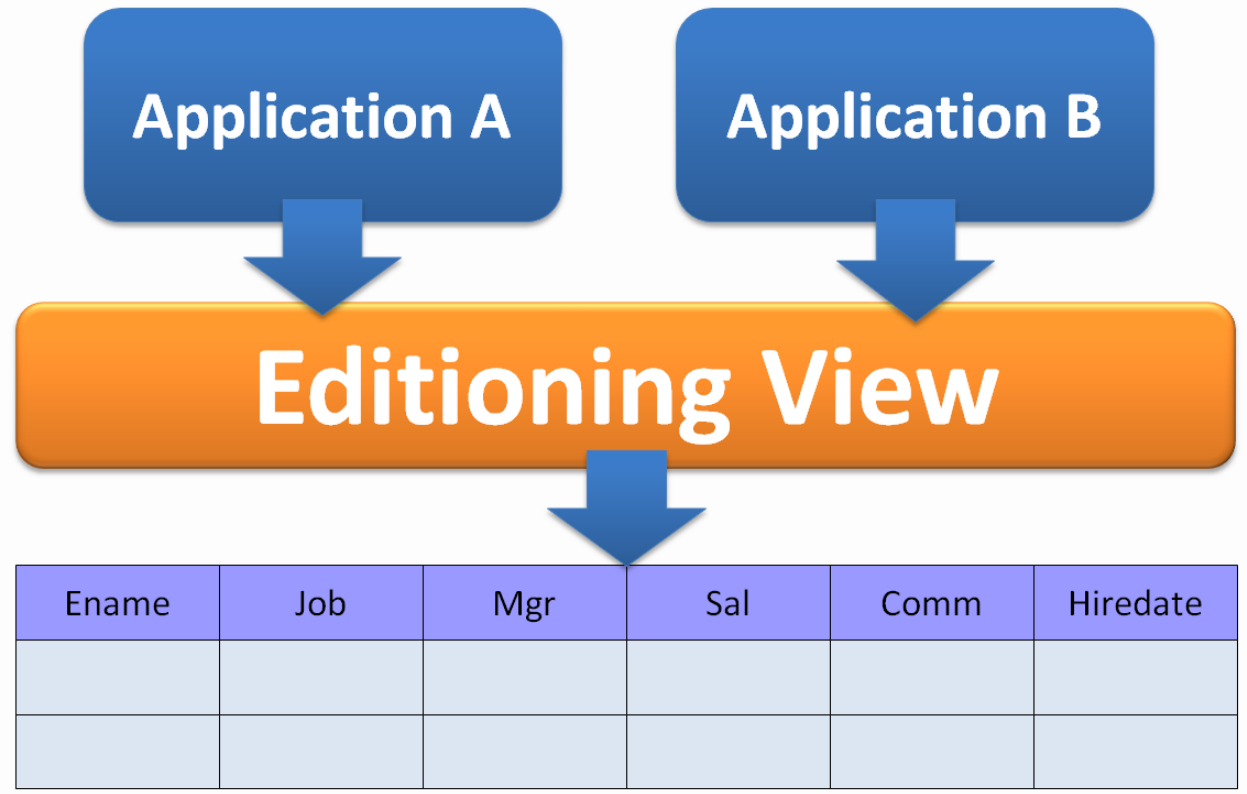
Nieuwe editions worden aangemaakt met de volgende syntax

```
CREATE EDITION release_2 AS CHILD OF ORA$BASE;
```

Iedere edition is een opvolger (CHILD OF) van een vorige edition. Het is op dit moment niet mogelijk om meerdere opvolgers te hebben van een edition. Een nieuwe edition erft (verwijzingen naar) alle objecten van zijn voorganger. Dit houd in dat objecten in een edition blijven bestaan totdat ze worden verwijderddoor het uitvoeren van een DROP commando of met een CREATE OR REPLACE een nieuwe gewijzigde versie van het object word gemaakt.
Een database sessie is gekoppeld aan (speelt zich af binnen) een bepaalde edition. Dit kan de default edition zijn, maar dit kan eenvoudig worden gewijzigd.

```
alter session set edition = Release_2
```

Dit statement kan vanuit de client applicatie worden uitgevoerd, maar het is uiteraard ook mogelijk om dit vanuit een LOGON trigger te doen bij het starten van de sessie.



Versionering

Laten we eens een volledig code voorbeeld doorlopen. Beginnende in de default edition maken we een eenvoudige functie aan.

```
SQL> alter session set edition = ora$base;

Session altered.

SQL> create or replace
  2  function hello
  3     return varchar2
  4  is
  5  begin
  6     return 'versie 1';
  7  end hello;
  8  /

Function created.
```

Deze functie kunnen we als volgt aanroepen

```
SQL> select hello from dual;

HELLO
```

```
----------------------------

versie 1
```

Het resultaat is niet echt verassend. Met het volgende statement maken we een nieuwe edition, als opvolger van ORA$BASE.

```
SQL> CREATE EDITION release_2 AS CHILD OF ORA$BASE;

Edition created.
```

Deze edition - release_2 - erft alle objecten van de zijn voorgaande edition. In dit voorbeeld erft release_2 alle objecten van ORA$BASE. Alleen de function "hello". Wijzigen we nu de edition voor onze sessie met

```
SQL> ALTER SESSION SET EDITION = release_2;

Session altered.
```

dan zijn we overgestapt naar de nieuwe edition. Voeren we hetzelfde SELECT statement uit dan zouden we hetzelfde resultaat krijgen, immers de referentie naar de functie wordt geërfd van de ORA$BASE edition. Er is nu nog steeds maar één versie van de function hello.

Willen we een nieuwe versie van de function gaan maken, dan voeren we het volgende statement uit - in een andere edition dan waar de eerste versie van de functie zich bevindt:

```
SQL> create or replace
  2  function hello
  3     return varchar2
  4  is
  5  begin
  6     return 'versie 2';
  7  end hello;
  8  /

Function created.
```

Nu bestaan er twee versies van een functie met de naam "hello". Namelijk eentje in de default editon ORA$BASE en eentje in de edition Release_2. Als we nu het eerder genoemde SELECT statement uitvoeren - nog steeds in edition release_2 - dan krijgen we wel een ander resultaat.

```
SQL> select hello from dual;

HELLO
----------------------------

versie 2
```

Wisselen we nu naar ORA$BASE dan krijgen we weer het oorspronkelijke resultaat. De eerder aangemaakte versie van de function is dus niet overschreven door het CREATE OR REPLACE statement. Om deze functie te verwijderen in Release_2 voeren we een DROP statement uit. Eenmaal verwijderd, dan kunnen toekomstige opvolgende editions van Release_2 geen gebruik meer maken van deze function.

```
SQL> drop function hello;

Function dropped.
```

Zouden we na dit statement opnieuw de SELECT uitvoeren dan krijgen we een ORA-904 Invalid Identifier error omdat de function niet meer bestaat in de context van deze edition. Overigens: in edition ORA$BASE bestaat de function gewoon nog wel.

```
SQL> alter session set edition=ora$base
  2  /

Session altered.

SQL> select hello from dual;

HELLO
--------------------------------------

versie 1
```

De meeste objecten zijn "editionable", er kunnen verschillende versie van deze objecten bestaan in meerdere editons. Dit geld voor ondermeer Views, Packages (en Procedures en Functions), Triggers, Synonyms en (user defined) Types. Voor tabellen geld dit echter niet!

Tabellen

Tabellen zijn niet "editionable". In eerste instantie lijkt dit heel erg vreemd: je voegt versionering toe aan de database - maar niet aan de belangrijkste objecten in die database?! Denk er echter even over na, en bekijk de vooroordelen. De data is het meest belangrijk en hiervan bestaat slechts één versie - wat zou het betekenen om meerdere versies van een employee of bestelling te hebben. Data hoeft niet en mag niet gekloond of gekopieerd te worden. Tabellen staan buiten de editions die we met Edition Based Redefinition creëren.

Maar hoe ga je dan om met wijzigingen in de tabelstructuur? Een nieuw type View is daarvoor essentieel: de Edtioning View. Een Editioning View is de nieuwe abstractie laag voor de applicatie. Iedere tabel krijgt een eigen een-op-een Editioning View. Alle verwijzingen naar tabellen in de applicatie en ook in triggers en packages, worden nu naar de Editioning Views gelegd. Alleen constraints en auditing policies verwijzen nog direct naar de onderliggende tabel.

Om de impact op bestaande applicatiecode zo klein mogelijk te houden is het handig om de tabel te hernoemen en de Editioning View de oorspronkelijke naam te geven van de tabel. In code:

```
SQL> ALTER TABLE EMP RENAME TO EMP_BASE;

Table altered.

SQL>
SQL> CREATE OR REPLACE EDITIONING VIEW EMP
  2  AS
  3  SELECT empno
  4       , ename
  5       , deptno
  6       , job
  7       , hiredate
  8       , sal
  9       , comm
 10       , mgr
 11    FROM EMP_BASE;

View created.
```

Editioning Views zijn een heel beperkt soort view. Editioning Views mogen alleen gebruikt worden voor data projectie (select een of meer kolommen uit een tabel, zonder WHERE clause), op basis van één tabel en ook zonder gebruik van kolomexpressies. De view mag wel kolommen hernoemen (bijvoorbeeld SAL AS SALARY) en ook kolommen weglaten. Daarnaast hoeft niet iedere versie van een Editioning View noodzakelijkerwijs tegen dezelfde tabel gedefinieerd te zijn… (dat is waarschijnlijk een verwarrende zo niet verontrustende gedachte die we hier verder niet uitwerken behalve door te zeggen dat in sommige gevallen versionering van data wél zinvol is - denk aan systeem-parameters, domein-waarden, schermteksten en error-messages; in die gevallen kan het handig zijn als de editioning view in de edition voor release 1 zijn data ophaalt uit de ene tabel terwijl de view in de release 2 edition de nieuwe set data queried uit een andere tabel).

Bij Editioning Views mogen wel DML triggers gedefinieerd worden. De triggers die eerst tegen een tabel waren gedefinieerd worden nu op de Editioning View gelegd. Na hercompilatie van de applicatie code (de "normale" views en packages) ben je klaar voor de toekomst. Vrijwel overal waar in de applicatie eerst de tabellen stonden, staan nu de editionable (multi-versie) Editioning Views. De tabellen zijn naar de achtergrond verdwenen en vormen het stabiele fundament onder alle releases van de applicatie.

Wijzigingen in de tabel definitie hebben geen directe invloed op de applicatie - aangezien de applicatie de tabel niet direct ziet of gebruikt. Stel dat in een nieuwe release van onze HRM-applicatie dringend behoefte is aan een language kolom, om de moerstaal van iedere employee vast te leggen. Aangezien er maar één tabel EMP bestaat, kunnen we niet anders dan deze kolom toevoegen aan de tabel.

```
SQL> ALTER TABLE EMP_BASE ADD (LANGUAGE VARCHAR2(2) NULL)
  2  /

Table altered.
```

Om de (nieuwe release van de) applicatie toegang te geven tot deze kolom moeten we een nieuwe versie aanmaken van de Editioning View EMP, in de Edition waarin de nieuwe

release wordt opgebouwd.

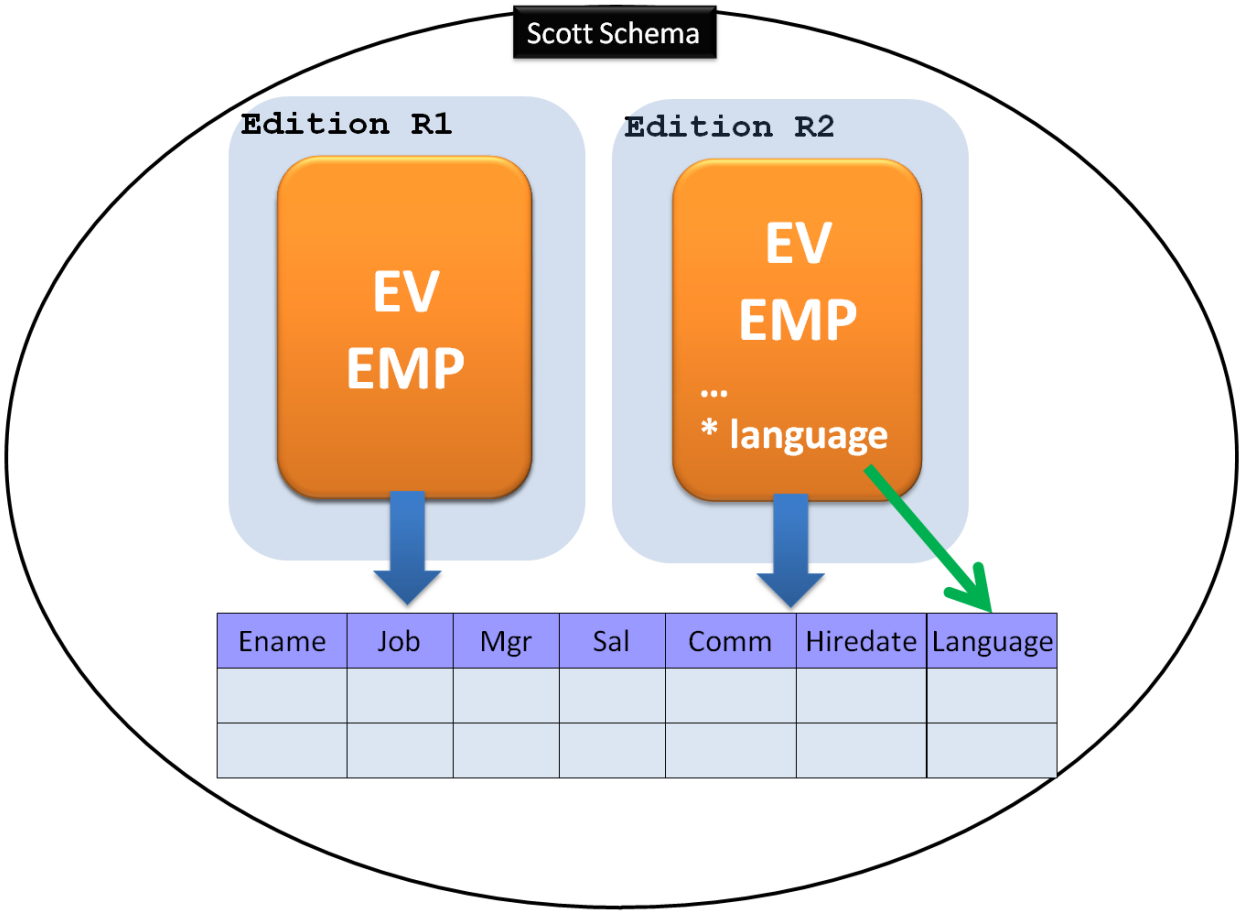
```
SQL> ALTER SESSION SET EDITION = release_2
  2  /

Session altered.

SQL> CREATE OR REPLACE EDITIONING VIEW EMP
  2  AS
  3  select empno
  4       , ename
  5       , language
  6    from emp_base
  7  /

View created.
```

De eerder gedefinieerde versie van deze view wordt niet gewijzigd. Voor gebruikers van deze view zal er niets veranderen. De gebruikers hebben nog geen weet van deze nieuwe versie van de Editioning View.
De grote vraag is nu: hoe gaan we om met de data in deze nieuwe kolom, bijvoorbeeld als er via release 1 van de applicatie nieuwe employees worden opgevoerd? Gebruikers van de bestaande applicatie weten niet dat deze kolom er is en hebben zelfs geen toegang tot deze kolom. Zij kunnen dus geen waarde vastleggen. Stel dat de nieuwe kolom verplicht is? Je zou een default waarde voor de kolom kunnen opnemen, en dat zou werken in dit geval. Maar als een bepaalde kolom in een nieuwe versie gesplitst wordt in twee kolommen - bijvoorbeeld ENAME wordt FIRST_NAME en LAST_NAME - dan zou het truukje met een default waarde niet opgaan.



Om ook hier een oplossing voor te hebben, heeft Oracle nieuwe triggers geintroduceerd. Triggers die over Editions heen gaan.
De basis is de logica waarmee een waarde voor deze kolom kan worden afgeleid, bijvoorbeeld uit de overige kolomwaarden in het record. Stel dat we een waarde voor de Language van een Employee kunnen bepalen op basis van de waarde van Job:

```
SQL> CREATE OR REPLACE FUNCTION
  2  GET_LANGUAGE( p_job in varchar2)
  3     return varchar2
  4  is
  5  begin
  6     return
  7        case p_job
  8          when 'MANAGER' then 'fr'
  9          else 'en'
 10        end;
 11  end;
 12  /

Function created.
```

Uiteraard is dit een vergezocht voorbeeld, maar het gaat om het idee. We kunnen dan in de nieuwste Edition een trigger aanmaken die speciaal bedoeld is voor het synchroniseren van DML in voorgaande editions met de huidige structuur van de tabel (en Editioning View) zoals die in deze edition gedefinieerd is. Deze trigger heet een *forward cross edition* trigger – omdat hij vanuit vorige edities voorwaarts naar de huidige editie DML complementeert.

```
SQL> CREATE OR REPLACE TRIGGER EMP_1_2_Fwd_Xed
  2  BEFORE INSERT ON EMP_BASE
  3  FOR EACH ROW
  4  FORWARD CROSSEDITION
  5  BEGIN
  6     :new.language := get_language(:new.job);
  7  END EMP_1_2_Fwd_Xed;
  8  /

Trigger created.
```

Deze trigger gaat af wanneer er een insert wordt uitgevoerd vanuit een Edition die voorafgaat aan de Edition waarin hij is aangemaakt. Als er dus een Employee wordt geinsert vanuit een eerdere Edition wordt deze trigger afgevuurd en wordt een waarde afgeleid voor de language kolom in de EMP_BASE tabel.

Dit mechanisme werkt goed voor nieuwe data, maar hoe ga je bestaande data converteren? Een uitbereiding in het package DBMS_SQL maakt het mogelijk om de forward cross edition te laten afgaan voor de bestaande data in de tabel.

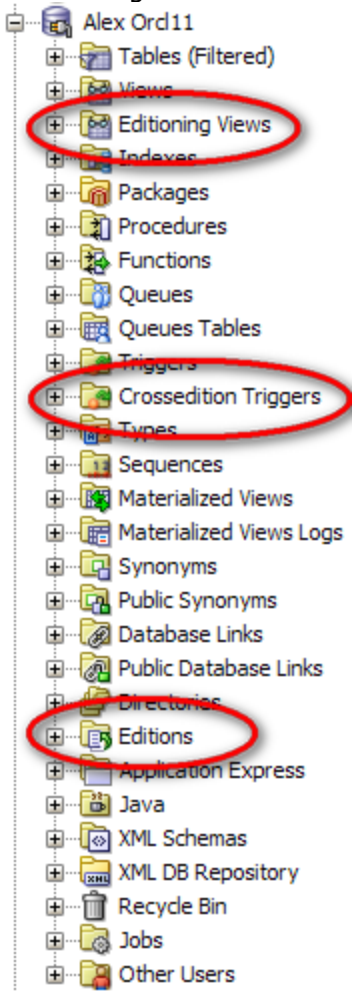
```
SQL> DECLARE
  2     c NUMBER := DBMS_SQL.OPEN_CURSOR();
  3     x NUMBER;
  4  BEGIN
  5     DBMS_SQL.PARSE
  6       ( c => c
  7       , Language_Flag => DBMS_SQL.NATIVE
  8       , Statement => 'UPDATE EMP_BASE
  9                 SET EMPNO = EMPNO'
 10       , Apply_Crossedition_Trigger => 'EMP_1_2_Fwd_Xed'
 11       );
 12     x := DBMS_SQL.EXECUTE(c);
 13     DBMS_SQL.CLOSE_CURSOR(c);
 14  end;
 15  /

PL/SQL procedure successfully completed.
```

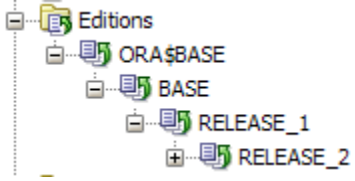
Een zelfde soort mechanisme is er ook om nieuwe data die gecreëerd wordt in de nieuwste Edition gesynchroniseerd te houden met oudere Editions. Eigenlijk is dit het tegengestelde van de forward cross edition trigger, namelijk de reverse cross edition trigger.

SQL Developer 2

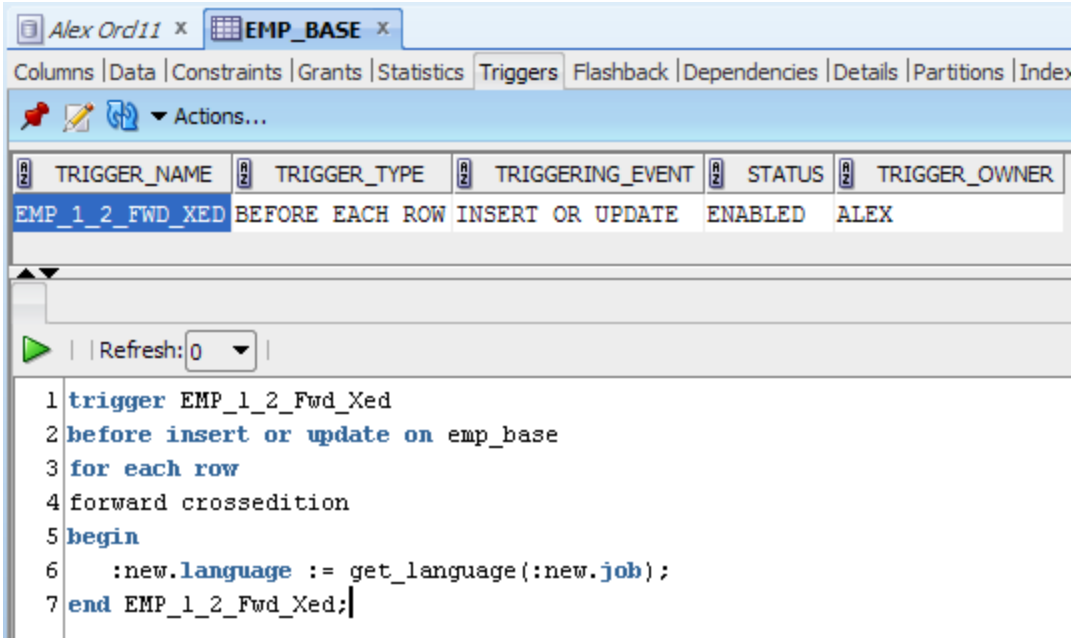
SQL Developer, het gratis ontwikkeltool van Oracle - de opvolger van SQL*Plus, biedt ondersteuning voor Edition Based Redefinition. Als je met SQL Developer connect naar een Oracle 11gR2 database dan krijg je er in de navigator een paar extra folders erbij.



Binnen de Editions folder zie je een overzicht van de Editions die gemaakt zijn. Door rechts te klikken op de Edition van je keuze is het eenvoudig wisselen van de ene Edition naar de andere. Na wisseling van Edition zijn alleen die Editioning Views en Crossedition Triggers te zien die voor de gekozen Editions van toepassing zijn.



Het is niet eenvoudig te zien in welke Edition je aan het werk bent, misschien een verbetering voor een toekomstige versie van SQL Developer. De source code van de Crossedition Triggers is niet te zien bij de Edition zelf, wel bij de tabel waar deze trigger bijhoort.



Conclusie

Edition Based Redefinition is een revolutionair nieuw mechanisme dat zeker voor organisaties met een strikte eisen voor de beschikbaarheid van (database) applicaties interessante mogelijkheden biedt. Daarnaast is EBR interessant in situaties waarin verschillende gebruikersgroepen – binnen of buiten de organisatie – verschillende versies van dezelfde applicatie willen gebruiken, maar wel van dezelfde data of in elk geval dezelfde tabellen gebruik willen maken.
Toepassing van EBR voor Packages en Views is betrekkelijk eenvoudig te realiseren. Volledig overgaan op EBR ook voor tabellen is een wat complexere stap die wellicht in een tweede fase kan worden opgepakt.
Het enige tool op de markt dat ondersteuning biedt voor Edition Based Redefinition, Oracle SQL Developer, maakt het gemakkelijk om het overzicht met EBR te bewaren. Toch zijn er nog wat kleine bugs in het tool die naar verwachting in toekomstige releases worden opgelost.
Signalen vanuit Oracle geven overigens aan dat er de komende periode nog verdere uitbreidingen, verfijningen en bijstellingen van het EBR mechanisme te verwachten zijn – zoals de mogelijkheid om database links te versioneren en meerdere kind-edities te creëren onder een ouder-editie (een beetje zoals branches in versiebeheer). Oracle's eigen ervaringen met de toepassing van EBR voor Fusion Applications zijn voor verdere ontwikkelingen van deze functionaliteit van groot belang.
Edition Based Redefinition wordt door Oracle gezien als basis-functionaliteit voor de database en is dan ook beschikbaar in alle edities (!) - standaard, enterprise en ook straks in de 11gR2 release van de gratis XE editie van de database.
Mochten je handen nu beginnen te jeuken om eens aan de slag te gaan met Edition Based Redefinition, houd dan de AMIS agenda in de gaten. Begin juli organiseert AMIS in Nieuwegein een hands on sessie waar Edition Based Redefinition centraal staat. Ook niet te missen is de ODTUG Preview - voor degene die niet naar ODTUG Kaleidoscope in Washington gaan - op 8 juni die plaatsvindt bij AMIS in Nieuwegein. Meer informatie via

www.amis.nl.

Zie ook: http://tinyurl.com/39bv78b voor visuele ondersteuning bij de voorgaande uitleg van Edition Based Redefinition.