DOYENSYS

Winner of **Great Indian Workplace 2018**
Emerging Enterprises Category

Online Patch Application Using Edition Based Redefinition

Sundaravel Ramasubbu, Senior Consultant – DBA practice

Nov 2015

Introduction:

This whitepaper explains concepts of edition based redefinition (EBR) in detail and challenges in the online application upgrade and how oracle 11gR2 meets these challenge.

These challenges must be met in order to do the Online Application Upgrade.

Changed database objects/code should not disturb the live users of pre-upgrade application.

Transaction done by the users of the pre-upgrade application must be reflected in the post-upgrade application.

In case of hot rollover, transaction done by the users of the post-upgrade application must be reflected in the pre-upgraded application.

Oracle Database version 11gR2 meets these challenges by introducing the new concept called Edition Based Redefinition.

Code changes are installed in the privacy of a new edition.

Data changes are made safely by writing only to new columns or new tables not seen by an old edition. This will be accomplished by an Editioning view.

A crossedition trigger propagates data changes made by the old edition into the new edition's columns, or (in hot-rollover) vice-versa.

Editions

Introduction:

- ✓ An Edition is a nonschema object.
- ✓ Listed in the DBA_OBJECTS catalog view.
- ✓ Like Object type dictionary, they appear to be owned by SYS.
- ✓ From Version 11gR2, there will be one edition by default with the name ORA$BASE.

Editioned or Noneditioned objects:

Edtioned object:

- ✓ If a Schema object that has both an editionable type and edition-enabled owner is called editioned object
- ✓ It has own copy an editioned object.
- ✓ It's only visible to the edition.
- ✓ Views, synonyms and all kind of PL/SQL objects are editionable object types.

Nonedtioned object:

- ✓ Schema object that has noneditionable type.
- ✓ It's identical in and visible to all editions.
- ✓ Table, Java class etc are Noneditionable object types.

Create Edition:

- ✓ We can create an edition as the child of an existing edition. First edition we create on the database is child of ORA$BASE edition.

```
SQL>create edition e2
```

DOYENSYS

Editions

Retiring Edition:

- ✓ After making the new edition (Upgraded Version) available it's very important to make sure no user will use old edition (Pre-Upgraded Version).
- ✓ It can be achieved by revoking "Use" privilege on the old edition from every user.

SQL>select 'revoke '||PRIVILEGE||' on '||table_name||' from '||grantee||';' FROM DBA_TAB_PRIVS WHERE TABLE_NAME = '&OldEditionNme' ;

Dropping Edition:

- ✓ It's safe to drop the edition if you want to rollback the application upgrade.
- ✓ You can drop the edition if the following conditions are met.
  - ✓ The edition is either the root edition or a leaf edition
  - ✓ If edition is the root edition, It has no editioned objects that are inherited by its child edition.
  - ✓ When the edition is not in use.
  - ✓ When the edition is not database default edition.
- ✓ Syntax to drop the edition

SQL>Drop edition EditionName cascade; ## which drops the edition and drops the actual objects.

Editioning Views:

Introduction

- ✓ An editioning view, as a special kind of view, is editionable
- ✓ Why it's special kind of view? Here is the answer.
    - ✓ In ordinary view (Non-editioning view), the only type trigger that we can define, that is "INSTEAD OF" trigger.
    - ✓ In an editioning view, we can define every type of trigger that we can define on a table.
    - ✓ However we can't add constraints and Indexes to an editioning view.

Creating an editioning view

- ✓ Create editioning view using the below syntax.

SQL>CREATE EDITIONING VIEW ed_orders_view (o_id, o_date, o_status) AS SELECT order_id, order_date, order_status FROM orders WITH READ ONLY;

- ✓ To create READ-ONLY editioning view, specify WITH READ ONLY clause.
- ✓ To create READ-WRITE editioning view just omit WITH READ ONLY clause.

Replacing an editioning view:

- ✓ Replace an editioning view use the syntax

SQL>CREATE OR REPLACE EDITIONING VIEW;

- ✓ Triggers defined on the replaced editioning view are retained.

Crossedition Triggers

Introduction

- ✓ Crossedition triggers are temporary.
- ✓ Crossedition triggers can be ordered with triggers defined on other tables.
- ✓ It's always editioned.

Forward crossedition triggers:

- ✓ User of pre-upgrade edition changes the table data, it will be reflect on Post-Upgrade edition. This is accomplished with forward crossedition triggers.

Reverse edition triggers:

- ✓ Reverse edition triggers ensures that when user of post-upgrade edition changes the table data, it will be reflect in the pre-Upgrade edition.

Creating a crossedition triggers:

- ✓ In order to create crossedition trigger used must be edition enabled.
- ✓ Following rules will apply to create crossedition triggers
  - ✓ It must be defined on table not view.
  - ✓ It must be a DML trigger.
  - ✓ Crossedition is always FORWARD unless we specify REVERSE clause in the syntax.
  - ✓ FOLLOWS clause is allowed only when creating a forward crossedition triggers.
  - ✓ PRECEDES clause is allowed only when creating a reverse crossedition triggers.

References:

http://docs.oracle.com/cd/E11882_01/appdev.112/e10471/adfns_editions.htm#ADFNS020

http://www.oracle.com/technetwork/database/features/availability/edition-based-redefinition-1-133045.pdf