Edition Based Redefinition for Zero Downtime PL/SQL Changes

Chris Saxon
Oracle
UK

Keywords:

Edition-based redefinition, PL/SQL, 11g, 12c

Introduction

Releasing code changes to PL/SQL in heavily executed packages on databases with high uptime requirements brings many challenges. It's not possible to gain the DDL lock to compile a package while it's being executed, so how do you do so without downtime?

In 11g Release 2, Oracle introduced edition-based redefinition (EBR). This allows you to deploy multiple versions of PL/SQL objects in the same database. Using EBR, you can release PL/SQL easily without downtime.

This paper discusses earlier methods to tackling this problem and shows how EBR is superior. It steps through the process of creating and using editions. It also investigates some of the downsides to using EBR and techniques you can use to mitigate these.

Background

Prior to 11g Release 2 you can only deploy one version of each PL/SQL object in an Oracle database. To deploy PL/SQL changes to production without downtime brings the following challenges:

- It's not possible to get the DDL lock necessary to compile the package while other users are executing it
- The database code may be in an inconsistent state during the release
- Users are able to access the updated code as soon as it's compiled. This leaves you no time to perform post-release testing in isolation
- You must give careful consideration for how you will rollback changes in an emergency

The simplest resolution to these issues is to have two (or more) versions of an object available at any given time. Prior to Oracle 11.2 it isn't possible to have multiple instances of one object in a single schema. You could build your own solutions to do this such as:

- Renaming PL/SQL objects for each deployment (e.g. package_v1 becomes package_v2 then package_v3 etc.)
- Replicate your application across multiple databases
- Not using stored PL/SQL!

Each of these approaches has a number of downsides.

Appending version numbers to PL/SQL objects is simple in principle. This places a burden on the development team however. They must keep track of the current live number and ensure any conflicts in version numbers are resolved prior to release. If there is a significant amount of concurrent development this can add a sizeable overhead.

Renaming also adds complexities when you have PL/SQL objects that reference each other. For example, you may have a PL/SQL procedure that calls a package. This, in turn, uses a query based on a view. If you need to change the view, then you create a new copy of it, view2. You must then update the package to call view2. So this change is not instantly visible to end users, you need to create a new version of it, package2. This logic requires you to rename the procedure too. This applies even if there are no functional changes to the procedure or package!

Replicating your application across more than one database overcomes the challenges with renaming PL/SQL objects. This comes at the cost of additional hardware, licenses and software to support the replication. It also places significant extra effort on the release team to determine how to deploy the changes. For many these costs are too high for this to be a workable solution.

Using anonymous PL/SQL blocks directly in application code overcomes the issues with renaming and replication. You can deploy applications and test them in isolation before you make them accessible to end users. This comes at the cost of the benefits of compiled PL/SQL. These include code reuse, dependency management and native compilation. Finally it reduces the value of the PL/SQL, as multiple applications are unable to call the code directly.

Starting in 11.2, Oracle introduced edition-based redefinition. This allows you to deploy two different versions of the same PL/SQL object within one schema. Using this you can deploy changes into a new edition and verify them before making the new code available to the end users. This allows you to compile a private copy of PL/SQL without the need to rename objects or use multiple databases.

Technical Discussion and Examples

To use EBR you must first prepare your database. The first step is to editions enable your database users. You can do this with the following command:

```
alter user <username> enable editions;
```

Note this is a one-way operation. It is impossible editions disable a user. If you want to rollback this command, you must restore the user from a backup or re-create it.

Not all database objects are editionable (see appendix 1 for a list of editionable objects). The above command will fail if you attempt to editions enable a user that owns editionable objects with non-editionable dependents. A common example of this is any database with public synonyms. In 11.2, public synonyms are non-editionable. If you have any on PL/SQL objects the statement above will fail. You can overcome this by adding the force option when enabling editions:

```
alter user <username> enable editions force;
```

Take care when doing this as it can result in non-editionable dependencies that you're unable to validate. For example you'll be unable to validate any public synonym pointing to the target user, so must replace them with private synonyms. For other scenarios when you're using 11g (e.g.

materialized views or virtual columns that call PL/SQL functions), you'll need to redesign your application to use EBR.

There are several improvements to EBR in 12c. Public synonyms are editionable. Materialized views and virtual columns both include options to specify which edition they should resolve PL/SQL dependencies to. You can also make individual objects non-editionable if necessary. For example, you can make a PL/SQL package called by a materialized view non-editionable.

This greatly simplifies the issue of non-editionable objects depending on editionable ones. If you're planning to use EBR on an 11.2 database and encounter these dependencies, in most cases it's easier to upgrade to 12c.

So what is the purpose of the force option? It is useful when you're edition enabling two or more users that have cross-schema dependencies. For example, if user A has a package that calls a procedure that user B owns. User B also has a package that calls a procedure owned by A. Here you need to force enable editions command for the first user. Once both users are editions enabled then you can validate the editionable objects in both schemas.

Once users are editions enabled, the next step is to create your new edition. You can do this with the following command:

```
create edition <edition name> [as child of <previous edition
name>];
```

The user issuing this command must have the "CREATE ANY EDITION" privilege. The "as child of" clause is optional. It's only possible to create a new edition as a child of the current leaf edition. New editions automatically inherit the code from the parent edition.

All databases come with the edition ORA$BASE. This is the default edition for all new databases. The first edition you create will be a child of this edition.

Once you've created an edition, you need to grant all the users that will use it privileges to do so:

```
grant use on edition <edition name> to <username>;
```

Once these steps are complete, you're ready to deploy your code into the new edition. To demonstrate this, take the following example. You have the following PL/SQL procedure:

```
create or replace procedure my_procedure as
begin
  dbms_output.put_line ('Hello World!');
end;
/
```

You wish to make the following change to it:

```
create or replace procedure my_procedure as
begin
  dbms_output.put_line ('Hello World! Nice to meet you');
end;
/
```

You want to do this in an online manner, so you're using EBR. You plan deploy the updated procedure into edition EDITION_1. This is the first edition you're creating, so it'll be a child of the edition ORA$BASE. To do this you execute the following:

```
create edition edition_1;
grant use on edition edition_1 to edition_user;
```

Once you've created the edition and granted your user use privileges on it you can connect to it via the following alter session command:

```
alter session set edition = edition_1;
```

You're now ready to compile the changes. All changes you make to editionable objects in this session will be in edition RELEASE_1. To revert to the previous edition, set the session parameter back:

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

After releasing your changes you can perform post-release testing. If you find there was a mistake in the release or you no longer want the new version for business reasons, you can rollback the changes by dropping the edition:

```
drop edition release_1 cascade;
```

Note this command requires the "DROP ANY EDITION" privilege. The cascasde option is necessary if you have made any changes to editionable database objects within the edition. If this isn't the case you can omit it. Dropping an edition also has the following conditions:

- There must be no users connected to the edition.
- You can only drop the first and last editions in a database - it is impossible to drop editions in the "middle", i.e. those with both a parent and a child edition.

For a complete worked example of this see appendix 2.

Once the release is complete you're ready to test it. If you wish to do this using the application itself then you need define the edition correctly. One way to do this is to set it in the connection string. Common methods for this are:

- Set the ORA_EDITION operating system environment variable to the desired edition for applications connecting using OCI
- For JDBC connections, set oracle.jdbc.editionName to the name of the target edition
- Link a database service to an edition. You can do this with the DBMS_SERVICE.MODIFY_SERVICE procedure, like so:

```
exec DBMS_SERVICE.modify_service(
  service_name   => 'DB11G.WORLD',
  edition        => 'EDITION_1',
  modify_edition => TRUE
);
```

For example, to connect to EDITION_1 using SQL*Plus in Windows do the following:

```
set ORA_EDITION = edition_1
sqlplus user/pass@database
```

You can verify which edition a session is using with the following query:

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

After you've finished your testing the final step is to make the edition available to all users. You can do this by granting use on the edition to public or by setting the new edition as the default for the database.

To grant public access to an edition, execute the following:

```
grant use on edition <edition name> to public;
```

To set the database's default edition, run:

```
alter database default edition = <edition name>;
```

This effectively grants use on the edition to public. Every database has one default edition. If a connection does not specify an edition, it will use the default. You can view the current default with this command:

```
select property_value
from   database_properties
where  property_name = 'DEFAULT_EDITION';
```

Once the new edition is publicly available, you should retire the old edition. You can do this by revoking database users' privileges on it:

```
revoke use on edition <edition name> from <username>;
```

Alternatively you can drop the old edition. To do so you must meet the following criteria:

- It must be the parent edition for the database
- There must be no users connected to the edition
- It must not be the default edition for the database
- The child edition has actualized all objects in the parent. This means that all objects must be compiled or dropped in the new edition

Once you've completed these steps the upgrade is finished.

Using EBR to deploy PL/SQL changes is easy. Using it brings management overhead. The following discusses some of the practical challenges of using EBR and steps you can take to minimize these.

Currently there is little toolset support for editions. This means it's not always clear which edition you're connected to. If care is not taken, it's easy to deploy changes into the "wrong" edition. At best

this means new functionality isn't exposed. At worst it can cause releases to fail and leave applications in an inconsistent state. To overcome this issue, I recommend taking the following actions:

- Assign one person to manage the editions for a given database. Ensure they document which editions are currently live. This person should also inform the release team which edition to use for upcoming deployments. Clearly state the target release edition on all release documentation.
- Ensure release scripts always set the edition as the first step in any deployment. Combined with the previous step this reduces the risk of deploying changes to the incorrect edition.
- Minimize the time that multiple editions are live at any one time. Once old editions are deprecated, update the database's default edition and revoke access to old editions.

It's also not immediately obvious which objects are compiled in which edition. Help with this, Oracle introduced additional dictionary views. You can view which editions are currently present in a database with the following query:

```
select * from dba_editions;
```

You can also see information about objects across editions using *_AE (all editions) views (see appendix 3). For example, the USER_OBJECTS_AE view displays which edition each of the current user's objects were compiled in. If you've compiled version of a procedure in each edition then it will be listed once for each edition. Any dropped objects appear with the OBJECT_TYPE set to NON-EXISTENT.

Further Reading

EBR can also be used to manage online schema changes with zero downtime. You can do this using editioning views and cross-edition triggers. For further details on how to do this see Tom Kyte's Oracle Magazine articles, Edition-Based Redefinition, Part 2 (http://www.oracle.com/technetwork/issue-archive/2010/10-mar/o20asktom-098897.html) and Edition-Based Redefinition, Part 3 (http://www.oracle.com/technetwork/issue-archive/2010/10-may/o30asktom-082672.html)

Contact address:

Chris Saxon
Oracle
Oracle Parkway, Thames Valley Park (TVP)
Reading, Berkshire, RG6 1RA
United Kingdom

Phone: +44(0)118-924-0749
Email chris.saxon@oracle.com
Internet: https://blogs.oracle.com/sql

Appendix 1

The following lists the editionable objects:

11gR2:

- SYNONYM
- VIEW

All PL/SQL object types:

- FUNCTION
- LIBRARY
- PACKAGE and PACKAGE BODY
- PROCEDURE
- TRIGGER
- TYPE and TYPE BODY

12cR1 also includes:

- SQL translation profile
- PUBLIC SYNONYM

All other object types are noneditionable.

Appendix 2

Complete example of enabling and deploying editions in a database (note the setup must be run as system or other user with privileges to create users and create/drop editions):

```
create user edition_user identified by "editions";
grant create session, create procedure to edition_user;
alter user edition_user enable editions;
create edition edition_1;
grant use on edition edition_1 to edition_user;

conn edition_user/editions
set serveroutput on

create or replace procedure my_procedure as
begin
  dbms_output.put_line ('Hello World!');
End;
/

exec my_procedure;

Hello World!

alter session set edition = edition_1;

create or replace procedure my_procedure as
```

```
begin
  dbms_output.put_line ('Hello World! Nice to meet you');
end;
/

exec my_procedure;

Hello World! Nice to meet you

alter session set edition = ora$base;
exec my_procedure;

Hello World!

-- revert back to previous user
conn system
drop edition release_1 cascade;
```

Appendix 3

All editions dictionary views:

- *_ERRORS_AE
- *_OBJECTS_AE
- *_EDITIONING_VIEW_COLS_AE
- *_EDITIONING_VIEWS_AE
- *_SOURCE_AE
- *_VIEWS_AE

References

- 11gR2 Documentation: http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_editions.htm#ADFNS020
- 12cR1 Documentation: http://docs.oracle.com/database/121/ADFNS/adfns_editions.htm
- Oracle-BASE article: http://oracle-base.com/articles/11g/edition-based-redefinition-11gr2.php
- Tom Kyte's Oracle Magazine article, Edition-Based Redefinition, Part 1: http://www.oracle.com/technetwork/issue-archive/2010/10-jan/o10asktom-172777.html
- Edition-Based Redefinition Whitepaper, Bryn Llewellyn July 2009: http://www.oracle.com/technetwork/database/features/availability/edition-based-redefinition-1-133045.pdf