Редакции объектов в Oracle как инструмент внесения изменений в БД и в приложения

Владимир Пржиялковский
Преподаватель технологий Oracle
prz@yandex.ru
www.open-oracle.ru

Июль 2010 г.
Стилистическая правка: май 2012 г.

Да разве б не цвела земля афинская,
Когда бы так же рассуждали граждане
И постоянно не искали нового?

Аристофан, Женщины в народном собрании

Краткое описание

В версии Oracle 11.2 для некоторых видов объектов хранения в БД была введена возможность иметь одновременно несколько «редакций» (editions). Целью ставилось усовершенствовать процесс внесения изменений в схему данных так, чтобы допустить в некоторых случаях отладку нового варианта приложения, в расчете на новую структуру данных, без останова работы ранее написанных программ. Техника использования редакций объектов рассматривается в статье на примерах.

Введение

С версии 11.2 для некоторых видов хранимых объектов в Oracle можно заводить разные «редакции» (editions) и переключаться между ними в работе, моделируя одновременное наличие нескольких версий прикладного программного обеспечения при его разработке или переделке. Речь не идет о «редакциях данных», и на таблицы такая техника не распространяется. Она применима к объектам следующих видов:

VIEW
SYNONYM
PROCEDURE
FUNCTION
TRIGGER
PACKAGE/PACKAGE BODY
TYPE/TYPE BODY
LIBRARY.

Основное применение техники редакций объектов можно видеть в области и поддержки и развития приложения. Она позволяет в некоторых случаях выполнять часть работ по внесению изменений в существующее прикладное ПО и в БД без останова использования уже существующей системы и отлаживать необходимые нововведения впараллель основной работе.

Хотя техника редакций объектов хранения не распространяется на данные в таблицах БД, подготовить приложение к переходу на новые структуры имеющихся таблиц иногда помогают версии представлений.

В версии Oracle 11.2 техника редакций объектов воплощена в своем начальном варианте, по всей видимости, не окончательном, и в будущем будет усовершенствована.

Далее рассматривается несколько примеров создания и использования версий объектов в Oracle.

Подготовка схемы для редакций объектов

Ниже приводятся команды заведения в SQL*Plus схемы для объектов разных редакций и выполнения необходимых сопутствующих действий.

```
CONNECT / AS SYSDBA
CREATE USER yard IDENTIFIED BY pass;

GRANT CONNECT, RESOURCE, CREATE VIEW TO yard;

CREATE TABLE yard.emp AS SELECT * FROM scott.emp;
```

В схеме YARD появилась таблица EMP с той же структурой, что и одноименная в схеме SCOTT и с теми же данными (но без ограничений целостности).

Создание редакций для объектов и управление ими

Управление редакциями регулируется привилегиями CREATE/ALTER/DROP ANY EDITION. Слово ANY в названиях напоминает о внесхемном характере редакций: они создаются на уровне БД вообще. Формально редакции приписаны пользователю SYS.

Если права создания редакций объектов и управления ими требуется выдать пользователю YARD, администратору БД следует продолжить:

```
GRANT CREATE ANY EDITION, DROP ANY EDITION TO yard;
```

Узнать действующую в данный момент редакцию можно из контекста сеанса USERENV (он встроен в СУБД и не требует для обращения к своим значениям никаких особых полномочий):

```
SQL> CONNECT yard/pass
Connected.
SQL> SELECT SYS_CONTEXT ( 'USERENV', 'CURRENT_EDITION_NAME' ) FROM dual;

SYS_CONTEXT('USERENV','CURRENT_EDITION_NAME')
--------------------------------------------------------------------
ORA$BASE
```

ORA$BASE — это встроенная в БД и умолчательно действующая редакция, опираясь на которую администратор может создавать последовательность (а в будущих версиях Oracle, возможно, дерево) собственных редакций. Имя умолчательно установленной для БД редакции можно прояснить запросом:

```
SELECT property_value
FROM   database_properties
WHERE  property_name = 'DEFAULT_EDITION'
;
```

Примеры создания редакций:

```
CREATE EDITION app_release_1;
CREATE EDITION app_release_2 AS CHILD OF app_release_1;
-- и так далее
```

В первом случае редакция APP_RELEASE_1 была создана на основе исходной умолчательной редакции ORA$BASE, во втором — как следует из текста команды.

Откомментировать редакцию в словаре-справочнике БД можно командой SQL COMMENT:

```
COMMENT ON EDITION app_release_1
```

```
  IS 'The first release of application'
;
```

Снимается комментарий, как обычно, указанием пустой строки ''. Просмотр комментариев выполняется через таблицу ALL_EDITION_COMMENTS.

Узнать существующие редакции в их взаимосвязи можно запросом к отдельной таблице:

```
SQL> SELECT * FROM all_editions;

EDITION_NAME                   PARENT_EDITION_NAME            USA
------------------------------ ------------------------------ ---
ORA$BASE                                                      YES
APP_RELEASE_1                  ORA$BASE                       YES
APP_RELEASE_2                  APP_RELEASE_1                  YES
```

Удалить из дерева (пока что — ветки) редакций можно только лист, свободный от подчиненных редакций:

```
DROP EDITION app_release_2;
```

Для того, чтобы пользователь Oracle мог не просто обращаться с редакциями объектов, но и формировать их, ему следует сообщить особое качество:

```
CONNECT / AS SYSDBA
ALTER USER yard ENABLE EDITIONS;
```

Качество ENABLE EDITIONS не изначальное, и неотъемлемое; буде оно раз выдано, *отменить его нельзя*. В результате все пользователи Oracle оказываются разделены на две категории: тех, кому разрешено формировать (строить) редакции, и тех, кому не разрешено. При том, возможен перевод пользователя из второй категории в первую, но никак не обратно. Удостовериться в наличие свойства ENABLE EDITIONS у пользователя можно по значению столбца EDITIONS_ENABLED (нового в версии 11.2) в таблице DBA_USERS (ее владелец SYS, и обычным пользователям она сама по себе не видна).

После выдачи последней команды каждый объект пользователя YARD, для которого разрешено создавать редакции, так или иначе будет привязан к какой-нибудь из нескольких редакций.

Настройка на работу с нужной редакцией

Чтобы пользователь Oracle имел право в конкретном сеансе работать с конкретной редакцией:

он должен иметь привилегию на работу с редакцией, выданную лично ему или, вместо этого, псевдопользователю PUBLIC (то есть всем вообще);
сеанс должен быть переключен на работу с этой редакцией.

Выдать пользователю личное общее разрешение на работу с объектами требуемой редакции можно примерно так:

```
GRANT USE ON EDITION app_release_1 TO scott;
```

USE — это привилегия на объекты вида EDITION, передаваемая кроме того через PUBLIC и через роли. Если редакцию, объявить в БД, как умолчательную, она автоматически полагается выданной для PUBLIC, то есть общедоступной, и не требует личных (или же ролевых) разрешений. По этой причине изначально частных разрешений на работу с ORA$BASE не требуется — оно есть у всех. То же самое случится с редакцией APP_RELEASE_1, если в какой-то момент выдать:

```
ALTER DATABASE DEFAULT EDITION = app_release_1;
```

На последнюю команду способен обладатель привилегии ALTER DATABASE (а ею сейчас обладают SYS и SYSTEM, но пока не YARD). Как только такая команда будет выдана, выполнять GRANT USE, как выше, для придания нужных полномочий пользователю SCOTT, не потребуется. Выдачей подобной команды может венчаться процесс отладки новых редакций объектов («перевод приложения на новую

редакцию»).

После получения разрешения (то есть привилегии) на работу с объектами конкретной редакции, пользователь Oracle приобретает право в рамках *отдельных* сеансов настраиваться на нее:

```
SQL> CONNECT scott/tiger
Connected.
SQL> SELECT SYS_CONTEXT ( 'USERENV', 'CURRENT_EDITION_NAME' ) FROM dual;

SYS_CONTEXT('USERENV','CURRENT_EDITION_NAME')
--------------------------------------------------------------------------------
ORA$BASE

SQL> ALTER SESSION SET EDITION = app_release_1;

Session altered.

SQL> SELECT SYS_CONTEXT ( 'USERENV', 'CURRENT_EDITION_NAME' ) FROM dual;

SYS_CONTEXT('USERENV','CURRENT_EDITION_NAME')
--------------------------------------------------------------------------------
APP_RELEASE_1
```

Код выше подтверждает то, что при открытии сеанса «сама собою» действует редакция, объявленная ранее умолчательной в БД.

Пример создания и использования разных редакций представления данных (view)

К настоящему моменту в БД образовались две редакции. Будем формировать их содержание редакциями объектов в схеме YARD. Создадим в последней две несложные редакции одного и того же представления данных — с выдачей сведений об отделе сотрудника, и без:

```
CONNECT yard/pass
ALTER SESSION SET EDITION = ora$base;
CREATE OR REPLACE EDITIONING VIEW emp_view
  AS
  SELECT empno, ename, deptno FROM emp
;
ALTER SESSION SET EDITION = app_release_1;
CREATE OR REPLACE EDITIONING VIEW emp_view
  AS
  SELECT empno, ename FROM emp
;
```

Настройку на редакцию ORA$BASE можно было выше не выполнять, потому что эта редакция умолчательная (это проверялось ранее), и автоматически действует в начале каждого сеанса.

В результате появились две редакции представления данных EMP_VIEW:

```
SQL> SELECT view_name, edition_name FROM user_editioning_views_ae;

VIEW_NAME                      EDITION_NAME
------------------------------ ------------------------------
EMP_VIEW                       ORA$BASE
EMP_VIEW                       APP_RELEASE_1
```

Редактируемые представления данных (editioning views) отличаются от обычных не только формальным словом EDITIONING при создании, но и некоторыми техническими свойствами. Они могут строиться на основе единственной таблицы, без фильтрации строк фразой WHERE и с отсутствием преобразований столбцов (в то же время, воспроизведение всех столбцов не обязательно). Есть и другие отличия, не востребованные настоящим примером.

Привилегии доступа к данным в разных редакциях выдаются по-отдельности:

```
ALTER SESSION SET EDITION = ora$base;
GRANT SELECT ON emp_view TO scott;
ALTER SESSION SET EDITION = app_release_1;
GRANT SELECT ON emp_view TO scott;
```

Вот как ниже SCOTT этими привилегиями пользуется:

```
SQL> CONNECT scott/tiger
Connected.
SQL> ALTER SESSION SET EDITION = ora$base;

Session altered.

SQL> SELECT * FROM yard.emp_view WHERE ROWNUM = 1;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7369 SMITH              20

SQL> ALTER SESSION SET EDITION = app_release_1;

Session altered.

SQL> SELECT * FROM yard.emp_view WHERE ROWNUM = 1;

     EMPNO ENAME
---------- ----------
      7369 SMITH
```

Теперь без отмены прежнего представления данных (и без прекращения работы с ним текущего приложения) открылась возможность независимо отлаживать приложение применительно к новому.

Упражнение. Отобрать у пользователя SCOTT привилегию на выборку данных из YARD.EMP_VIEW в редакции APP_RELEASE_1 и наблюдать результат попытки обращения.

Пример редакций процедур

Заведение разных редакций одной и той же процедуры в схеме со свойством EDITIONS_ENABLED = TRUE выглядит достаточно прозрачно. Так, для добавления данных о сотрудниках можно завести две редакции процедуры INSERT_EMPLOYEE следующим образом:

```
CONNECT yard/pass
ALTER SESSION SET EDITION = ora$base;
CREATE OR REPLACE PROCEDURE insert_employee (
  eno   NUMBER
, ename VARCHAR2
, dno   NUMBER
)
AS
BEGIN
INSERT INTO yard.emp_view ( empno, ename, deptno ) VALUES ( eno, ename,
dno )
;
END;
/
GRANT EXECUTE ON insert_employee TO scott;

ALTER SESSION SET EDITION = app_release_1;
CREATE OR REPLACE PROCEDURE insert_employee (
  eno   NUMBER
, ename VARCHAR2
)
AS
```

```
BEGIN
INSERT INTO yard.emp_view ( empno, ename ) VALUES ( eno, ename )
;
END;
/
GRANT EXECUTE ON insert_employee TO scott;
```

Проверка:

```
SQL> CONNECT scott/tiger
Connected.
SQL> ALTER SESSION SET EDITION = ora$base;

Session altered.

SQL> EXECUTE yard.insert_employee ( 1111, 'OBAMA', 10 )

PL/SQL procedure successfully completed.

SQL> ROLLBACK;

Rollback complete.

SQL> ALTER SESSION SET EDITION = app_release_1;

Session altered.

SQL> EXECUTE yard.insert_employee ( 1111, 'OBAMA' )

PL/SQL procedure successfully completed.

SQL> ROLLBACK;

Rollback complete.
```

Откат транзакций сделан (а) чтобы сохранить прежние данные, и (б) в первом случае — чтобы закрыть транзакцию перед переключением на новую редакцию.

Пример редакций триггерных процедур

Теперь для добавления в БД данных о сотрудниках создадим две редакции триггерных процедур. Это делается аналогично обычным процедурам. Прикладной смысл триггерных процедур в данном случае состоит в нормализации имен сотрудников перед помещением в базу.

```
CONNECT yard/pass
ALTER SESSION SET EDITION = ora$base;

CREATE OR REPLACE TRIGGER empl_insert
BEFORE INSERT ON emp_view
FOR EACH ROW
BEGIN
   :new.ename := INITCAP ( :new.ename );
END;
/
GRANT INSERT ON emp_view TO scott;
```

Обратите внимание, что для редактируемых представлений (EDITIONING VIEW) в триггерных процедурах *не действует* привязка к событию INSTEAD OF, как для обычных представлений, а вместо этого BEFORE и AFTER, как для основных таблиц. Это одно из проявлений особости редактируемых представлений от обычных.

Вторая редакция:

```
ALTER SESSION SET EDITION = app_release_1;

CREATE OR REPLACE TRIGGER empl_insert
BEFORE INSERT ON emp_view
FOR EACH ROW
BEGIN
   :new.ename := LOWER ( :new.ename );
END;
/
GRANT INSERT ON emp_view TO scott;
```

Проверку можно выполнить следующей последовательностью команд в SQL*Plus:

```
CONNECT scott/tiger
ALTER SESSION SET EDITION = ora$base;
INSERT INTO yard.emp_view VALUES ( 1111, 'OBAMA', 10 );
SELECT * FROM yard.emp_view WHERE empno = 1111;
ROLLBACK;

ALTER SESSION SET EDITION = app_release_1;
INSERT INTO yard.emp_view VALUES ( 1111, 'OBAMA' );
SELECT * FROM yard.emp_view WHERE empno = 1111;
ROLLBACK;
```

Перекрестные триггерные процедуры для разных редакций

Когда отлаживается работа приложения с новой редакцией объектов БД, какое-то время обе редакции объектов (старая и новая) сосуществуют. Сложность в том, что работа с новой редакцией не должна портить данные, с которыми продолжает иметь дело старый вариант приложения. Если планируемые изменения в схеме однозначно взаимообратимы с исходным состоянием, помочь в этом способны перекрестные триггерные процедуры для разных редакций (межредакционные триггерные процедуры; crossedition triggers, CET).

Рассмотрим пример подготовки к изменению структуры таблицы EMP в схеме YARD. Предположим, требуется хранить в БД самостоятельно сведения о должностях, как например максимальную зарплату и тому подобное. Ради этого придется завести отдельную новую таблицу с данными о должностях, а из таблицы EMP изъять столбец с названием должности сотрудника, и поставить ему на замену ссылку на сведения о должностях. Пока новая редакция приложения не будет объявлена основной, старый столбец придется какое-то время сохранять.

Подобное разбиение одной таблицы сотрудников на две — сотрудников и, отдельно, должностей — очевидно обратимо, так что на время отладки будет удобно воспользоваться межредакционными триггерными процедурами. Они будут отвечать при работе со старой редакцией за параллельное внесение изменений в новые структуры, а при работе с новой редакцией — в старые, обеспечивая в БД возможность предоставления как старого, так и нового «взгляда» на данные.

Подготовка таблиц

Создадим таблицу должностей, и добавим в таблицу сотрудников ссылку, при том, что столбец с названием должности оставим до будущего перехода на новую редакцию приложения. Выполним в SQL*Plus:

```
CONNECT yard/pass
CREATE TABLE job (
  jobid  NUMBER   ( 2 ) PRIMARY KEY
, jname  VARCHAR2 ( 9 )
, maxsal NUMBER   ( 7, 2 )
);
INSERT INTO job ( jobid, jname, maxsal ) VALUES ( 1,   'ANALYST', 3500 );
INSERT INTO job ( jobid, jname, maxsal ) VALUES ( 2,     'CLERK', 2000 );
INSERT INTO job ( jobid, jname, maxsal ) VALUES ( 3,   'MANAGER', 3000 );
INSERT INTO job ( jobid, jname, maxsal ) VALUES ( 4, 'PRESIDENT', 6000 );
```

```
INSERT INTO job ( jobid, jname, maxsal ) VALUES ( 5,  'SALESMAN', 2000 );

ALTER TABLE emp ADD ( jobno NUMBER ( 2 ) REFERENCES job ( jobid ) );

UPDATE emp SET jobno = ( SELECT jobid FROM job WHERE jname = emp.job );
```

Переименуем таблицу сотрудников, и отдадим ее старое имя двум редакциям представления:

```
ALTER TABLE emp RENAME TO emp_tab;

ALTER SESSION SET EDITION = ora$base;
CREATE OR REPLACE EDITIONING VIEW emp
  AS
  SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno FROM emp_tab
;
GRANT INSERT, UPDATE, DELETE, SELECT ON emp TO scott;

ALTER SESSION SET EDITION = app_release_1;
CREATE OR REPLACE EDITIONING VIEW emp
  AS
  SELECT empno, ename, jobno, mgr, hiredate, sal, comm, deptno FROM
emp_tab
;
GRANT INSERT, UPDATE, DELETE, SELECT ON emp TO scott;
```

Создание перекрестных межредакционных триггерных процедур

Одна из создаваемых ниже триггерных процедур отвечает за правку данных, необходимую для работы старой редакции приложений, во время работы нового, а вторая наоборот. Существенно, что транслироваться обе перекрестные процедуры должны в действии новой редакции:

```
ALTER SESSION SET EDITION = app_release_1;

CREATE OR REPLACE TRIGGER cross_forward_job
BEFORE INSERT OR UPDATE ON emp_tab
FOR EACH ROW
FORWARD CROSSEDITION
BEGIN
  SELECT jobid INTO :new.jobno FROM job WHERE :new.job = jname;
END;
/

CREATE OR REPLACE TRIGGER cross_reversed_job
BEFORE INSERT OR UPDATE ON emp_tab
FOR EACH ROW
REVERSE CROSSEDITION
BEGIN
  SELECT jname INTO :new.job FROM job j WHERE :new.jobno = jobid;
END;
/
```

Проверку способен организовать следующий код:

```
CONNECT scott/tiger
ALTER SESSION SET EDITION = ora$base;
INSERT INTO yard.emp ( empno, ename, job ) VALUES ( 1111, 'OBAMA',
'CLERK' );
COMMIT;
ALTER SESSION SET EDITION = app_release_1;
INSERT INTO yard.emp ( empno, ename, jobno ) VALUES ( 2222, 'LADEN', 2 );
COMMIT;
```

Как и раньше, если транзакция успела изменить какие-нибудь данные в БД, для настройки на новую редакцию ее потребуется сначала закрыть. В результате получим:

```
SQL> SELECT * FROM yard.emp WHERE empno IN ( 1111, 2222 );

     EMPNO ENAME        JOBNO      MGR HIREDATE      SAL    COMM   DEPTNO
---------- ------- ----------- -------- --------- ------- ------- --------
      1111 OBAMA             2
      2222 LADEN             2

SQL> ALTER SESSION SET EDITION = ora$base;

Session altered.

SQL> SELECT * FROM yard.emp WHERE empno IN ( 1111, 2222 );

     EMPNO ENAME   JOB            MGR HIREDATE      SAL    COMM   DEPTNO
---------- ------- --------- -------- --------- ------- ------- --------
      1111 OBAMA   CLERK
      2222 LADEN   CLERK
```

При работе со старой редакцией воспроизводится поведение старой таблицы EMP, а при работе с новой — с тою же таблицей, но в новом варианте.

Дополнительные замечания по технологии

Приведенные примеры перекрестных триггерных процедур были намерено упрощены. В жизни в них следовало бы предусмотреть реакцию на указание в качестве нового значения отсутствующей должности. Предположим, что в старом приложении подобная обработка не программировалась, то есть в БД добавлялась ровно то название должности, которое было указано в INSERT/UPDATE. Тогда для сохранения поведения старой реакции приложения следовало бы на возникающую в SELECT … INTO ... FROM job ошибку NO_DATA_FOUND среагировать добавлением новой записи в таблицу JOB. Придется решить технический вопрос о поставке значений в JOBID; это может потребовать употребления генератора последовательности (sequence) и прочих усложнений.

При решении перейти на новую редакцию приложения перекрестные триггерные процедуры и редакционные представления данных следует удалить, а освободившееся имя EMP вернуть основной таблице:

```
CONNECT yard/pass
ALTER SESSION SET EDITION = ora$base;
DROP VIEW emp;

ALTER SESSION SET EDITION = app_release_1;
DROP VIEW emp;
DROP TRIGGER cross_reversed_job;
DROP TRIGGER cross_forward_job;

ALTER TABLE emp_tab RENAME TO emp;
ALTER TABLE emp DROP COLUMN job;

CONNECT / AS SYSDBA
ALTER DATABASE DEFAULT EDITION = app_release_1;
```