

Christian Antognini, Trivadis AG

# Edition-Based Redefinition (Teil 1 von 2)

**Der Upgrade kritischer Applikationen kann sehr schwierig sein. Eines der Hauptprobleme hinsichtlich der Verfügbarkeit ist, dass lange Ausfallzeiten nicht regelmäßig eingeplant werden können. Aus diesem Grund ist es für solche Applikationen wünschenswert, Online Upgrades zu implementieren. Dies erfordert zum einen, dass sowohl die fragliche Applikation, als auch jegliche Software, die von der Applikation genutzt wird (z.B. die Datenbank), Online-Upgrades unterstützen. Oracle hat dieses Problem schon seit Jahren erkannt. Unglücklicherweise sind bis einschliesslich Oracle Database 11g Release 1 nur eine beschränkte Anzahl von Funktionalitäten hinsichtlich diesen Anforderungen implementiert. Erst mit Oracle Database 11g Release 2 hat sich diese Situation Gott sei Dank geändert. Mit der sogenannten «Edition-Based Redefinition» Funktionalität bietet die Oracle Datenbank nun endlich eine echte Unterstützung für Online Upgrades an. Das Ziel dieses Artikels ist es einen Überblick und einige Tipps zu diesem neuen Feature zu geben.**

## Konzept

In einer Oracle Datenbank wird ein Objekt durch seinen Namen, dem zugehörigen Benutzer (Schema) und seiner Edition identifiziert. Ziel des Ganzen ist, und das ist neu mit Oracle 11g Release 2, mehrere Kopien (Versionen) eines bestimmten Objekts zu unterstützen. Die Objekte, für die mehrere Kopien angelegt werden können, heissen «Editioned Objects». Objekte, die nicht mehrere Kopien haben können bzw. keine Edition haben, werden als «Noneditioned Objects» bezeichnet (Die EDITION Spalte in den Data Dictionary Views enthalten NULL als Wert). Das Ziel, zwei oder mehrere Versionen eines bestimmten Objektes zu unterstützen, ist sehr einfach. Während eine Gruppe von Objekten durch die Applikation benutzt wird, kann eine andere Gruppe derselben Objekte nicht nur redefiniert, sondern auch für Testzwecke benutzt werden. Mit anderen Worten, es ist also möglich, eine Gruppe von Objekten zu redefinieren und in einer isolierten Umgebung zu testen, bevor diese live geschaltet werden.

Nicht alle Objekttypen unterstützen Editionen. Nur die folgenden Objekttypen sind «Editionable Object Types». D.h. sie unterstützen dieses neue Konzept:

- FUNCTION
- LIBRARY
- PACKAGE und PACKAGE BODY
- PROCEDURE
- SYNONYM (PUBLIC SYNONYM ist ein «Noneditionable Object Type»)
- TRIGGER
- TYPE und TYPE BODY
- VIEW

Im Wesentlichen sind dies alle Objekttypen, die durch die erweiterte Funktionalität des Datenbankkerns über PL/SQL Code oder externe Libraries genutzt werden. Die einzige Ausnahme sind die Views und die Synonyme. Man

beachte, dass weder Java bezogene Objekttypen noch datenspeichernde Objekttypen (z.B. Tabellen und Indizes) editionierbar sind.

Objekte, die keine Editionen unterstützen, werden «Noneditionable Objekttypen» genannt.

## Editionserstellung für einen Benutzer ermöglichen

Der erste Schritt für die Nutzbarkeit des Edition Based Redefinition Features ist, den Eigentümer der zu redefinierenden Objekte freizuschalten. Nur Benutzer mit Berechtigung zur Editionserstellung können Editionierte Objekte erstellen. Um dies zu ermöglichen, muss entweder die ENABLE EDITIONS Klausel beim Erstellen eines Benutzers angegeben werden oder für einen bereits bestehenden Benutzer des ALTER USER Befehls wie im folgenden Beispiel dargestellt, ausgeführt werden.

```
SQL> ALTER USER cha ENABLE EDITIONS;
```

Hierbei ist wichtig zu erwähnen, dass die Berechtigung zur Editionserstellung nicht wieder entzogen werden kann. Mit anderen Worten, so etwas wie eine DISABLE EDITIONS Klausel existiert nicht.

Zur Überprüfung, ob ein Benutzer die Möglichkeit hat Editionierbare Objekte zu erstellen, kann folgende Abfrage verwendet werden.

```
SQL> SELECT editions_enabled
2 FROM dba_users
3 WHERE username = 'CHA';

EDITIONS_ENABLED
-----
Y
```

Ist ein Benutzer für Editionierbare Objekte berechtigt, muss zur Identifizierung einer spezifischen Version von Datenbankobjekten eine Edition angelegt werden.

## Erstellen von Editionen

Jede Datenbank muss mindestens eine Edition haben. Hierfür wird beim Anlegen einer neuen Datenbank eine Basis Edition, die sogenannte ORA\$BASE, erstellt. Eine Datenbank unterstützt mehrere Editionen. Editionen mit Hilfe einer Root Edition durch eine Hierarchie organisiert (Standardmässig ORA\$BASE) und einer Eltern-Kind-Beziehung mit den anderen Editionen. Zum Beispiel zeigt die Abbildung 1 den Fall einer Datenbank mit vier Editionen, die Basis-Edition und drei benutzerdefinierten Editionen (lassen Sie uns annehmen, dass jede der benutzerdefinierten Editionen sich auf ein Applikationsrelease bezieht, hier die Bezeichnungen REL1, REL2 und REL3).

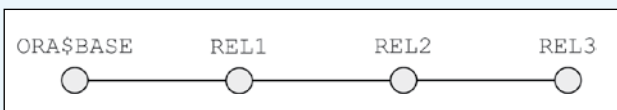


Abb. 1: Editionen werden über Hierarchien organisiert.

Für das Erstellen einer Edition muss der CREATE EDITION Befehl verwendet werden. Um dieses SQL Statement auszuführen, wird das CREATE ANY EDITION Systemprivileg benötigt. Für das Anlegen der Editionen aus dem Beispiel aus Abbildung 1 müssen folgende SQL Befehle ausgeführt werden. Man beachte wie die Eltern-Kind-Beziehung über die AS CHILD OF Klausel spezifiziert wird.

```

SQL> CREATE EDITION rel1 AS CHILD OF ora$base;
SQL> CREATE EDITION rel2 AS CHILD OF rel1;
SQL> CREATE EDITION rel3 AS CHILD OF rel2;
  
```

Standardmässig kann eine neue Edition nur über den anlegenden Benutzer oder durch den SYS Benutzer genutzt werden. Man kann die nötigen Privilegien an andere Benutzer über das USE ON EDITION Privileg vergeben.

Wie der folgende SQL Befehl zeigt, können die Privilegien entweder an einen bestimmten Benutzer oder an alle Benutzer über einen «GRANT TO PUBLIC» vergeben werden.

```

SQL> GRANT USE ON EDITION rel2 TO cha;
SQL> GRANT USE ON EDITION rel3 TO PUBLIC;
  
```

Das Anzeigen von Informationen hinsichtlich aller verfügbaren Editionen kann entweder über die View ALL\_EDITIONS oder die View DBA\_EDITIONS selektiert werden. Das folgende Beispiel zeigt eine hierarchische Abfrage, die für einen solchen Zweck genutzt werden kann. Man beachte, dass Editionen nicht zu einem bestimmten Schema gehören und deshalb die Views ALL\_EDITIONS und DBA\_EDITIONS keine Spaltenspezifizierung eines Benutzers zu einer Edition zeigen.

```

SQL> SELECT *
2 FROM all_editions
3 CONNECT BY parent_edition_name = PRIOR edition_name
4 START WITH parent_edition_name IS NULL
5 ORDER BY level;
  
```

EDITION_NAME	PARENT_EDITION_NAME	USABLE
ORA\$BASE		YES
REL1	ORA\$BASE	YES
REL2	REL1	YES
REL3	REL2	YES

Da eine Datenbank mehrere Editionen unterstützt, muss ein Benutzer (oder wahrscheinlicher eine Applikation) in der Lage sein, die Edition zu referenzieren, welche die korrekte Version der editionierten Objekte enthält. Mit anderen Worten, die so genannte Session Edition muss ausgewählt werden.

## Auswählen von Editionen

Die Auswahl einer Session Edition, APIs (z.B. OCI und JDBC) und Tools (z.B. SQL\*Plus und Data Pump) ermöglichen die Spezifizierung während der Erstellung einer neuen Datenbank. Bitte beachten Sie, dass wenn keine Edition angegeben wird, die Basis-Edition (definiert durch die Datenbankeinstellung des Parameters DEFAULT\_EDITION) genutzt wird.

Standardmässig ist die Einstellung auf ORA\$BASE gesetzt. Wie bei allen anderen Datenbankeigenschaften können diese über das ALTER DATABASE Kommando geändert werden.

```

SQL> ALTER DATABASE DEFAULT EDITION = rel1;

SQL> SELECT property_value
2 FROM database_properties
3 WHERE property_name = 'DEFAULT_EDITION';
  
```

PROPERTY_VALUE
REL1

Hierbei sind besonders zwei Punkte einer solchen Konfiguration hervorzuheben.

1. Wenn Sie die Basis-Datenbank Edition ändern, vergibt die Datenbank automatisch das USE ON EDITION Privileg für die spezifizierte Edition an PUBLIC. Dies ist notwendig, weil die Basis-Datenbank Edition für alle Benutzer verfügbar sein muss.
2. Das Konzept der Basis-Datenbank Edition ist hauptsächlich aus Kompatibilitätsgründen für Applikationen gedacht, die während eines Aufbaus einer Datenbankverbindung nicht in der Lage sind eine bestimmte Edition zu setzen. Deshalb sollte die Session Edition wenn möglich immer zum Verbindungszeitpunkt spezifiziert werden.

Ist die Datenbankverbindung erst einmal hergestellt, kann die Session Edition selbstverständlich über ein ALTER SESSION Kommando geändert werden. Beachten Sie, dass dieses SQL Kommando nicht ausgeführt werden kann, wenn die aktuelle Session noch eine offene Transaktion beinhaltet. Das folgende Kommando zeigt, wie REL1 als Session Edition gesetzt wird.

```
SQL> ALTER SESSION SET EDITION = rel1;
```

Ebenfalls noch wichtig zu erwähnen ist, das vorherige SQL Kommando muss ein Top-Level SQL Kommando sein, daher kann es nicht in PL/SQL ausgeführt werden. Wenn Sie also die Session Edition über PL/SQL ändern wollen (z.B. in einem Logon Trigger), müssen Sie die SET\_EDITION\_DEFERRED Prozedur aus dem DBMS\_SESSION Paket verwenden.

Zwei Methoden sind zur Anzeige der verwendeten Session Edition verfügbar.

1. Nur für die aktuelle Session kann das SESSION\_EDITION\_NAME Attribut aus dem USERENV Kontext benutzt werden.
2. Für alle Sessions kann die V\$SESSION View genutzt werden.

Die folgenden Select-Befehle zeigen die unterschiedlichen Abfragemöglichkeiten:

```
SQL> SELECT sys_context('USERENV','SESSION_EDITION_NAME') edition_name
2 FROM dual;

EDITION_NAME
-----
ORA$BASE

SQL> SELECT DISTINCT object_name AS edition_name
2 FROM v$session, dba_objects
3 WHERE session_edition_id = object_id;

EDITION_NAME
-----
ORA$BASE
```

Wenn beides, der Eigentümer der editionierten Objekte sowie die korrekte Edition gesetzt ist, ist es an der Zeit die editionierten Objekt selbst anzulegen.

ANZEIGE

## Profitieren Sie jetzt von unserem Oracle-Know-how!

- **Ökonomisch:** Wir helfen Ihnen, Oracle wirtschaftlich zu lizenzieren.
- **Effizient:** Wir kümmern uns um Ihre Wartungsverträge - weltweit.
- **Loyal:** Wir vertreten Ihre Interessen gegenüber Oracle.



Kontaktieren Sie jetzt unseren Oracle-Spezialisten Christian Rügger:  
christian.ruegger@softwareone.com, Tel. +41 (0)844 44 55 44

**ORACLE® PARTNER**

software**ONE**  
The Software Licensing Experts™

## Erstellen Editionierter Objekte

Editionierte Objekte werden genauso wie alle anderen Objekte angelegt. Das einzige, was gemacht werden muss – neben der Tatsache, dass ein Benutzer verwendet wird, der auch editionierte Objekte besitzen darf – ist die Edition auszuwählen, mit der das Objekt assoziiert werden soll. Beispielsweise kann das folgende SQL Kommando für die Erstellung einer Prozedur und die Zuordnung zur Edition REL1 verwendet werden.

```
SQL> ALTER SESSION SET EDITION = rel1;

SQL> CREATE OR REPLACE PROCEDURE hello IS
2 BEGIN
3   dbms_output.put_line('Hello from REL1');
4 END;
5 /
```

Eine andere Methode der Erstellung von editionierten Objekten ist, die Editionen für einen Benutzer freizuschalten, der bereits Objekte besitzt. In diesem Fall sind alle Objekte automatisch mit der Root Edition assoziiert (nicht mit der Basis-Edition, welche meiner Meinung nach sinnvoller wäre).

Das Ziel der Erstellung von editionierten Objekten ist diese anschließend auch redefinieren zu können. Also schauen wir uns an, wie eine solche Operation ausgeführt werden kann.

**Der 2. Teil dieses Artikels folgt in Ausgabe 2/2011 ■**

### Contact

Trivadis AG

Christian Antognini

E-Mail:

christian.antognini@trivadis.com

ANZEIGE

## Wir bleiben für Sie wach

Basic



Advanced



Enterprise



### Maintenance & Support, das SLA von Edorex

Wenn es darum geht, Ihre Datenbanken im Auge zu behalten, können Sie auf uns Vertrauen. Installation, Betrieb und Pflege liegen uns am Herzen, buchen Sie das passende Angebot unter [www.edorex.ch](http://www.edorex.ch)

**edorex**  
creating solutions

Edorex Informatik AG | Unterdorfstrasse 5 | 3072 Ostermundigen | 031 930 10 70 | [info@edorex.ch](mailto:info@edorex.ch) | [www.edorex.ch](http://www.edorex.ch)

Christian Antognini, Trivadis AG

## Edition-Based Redefinition (Teil 2 von 2)

### Redefinieren editio- nierter Objekte

Für die Redefinition eines Objekts ist der erste Schritt eine neue Edition zu erstellen und, wenn dieser Schritt nicht durch den Eigentümer des editierten Objekts ausgeführt wird, die notwendigen Privilegien zu erteilen. Die einzelnen Schritte sind in den vorherigen Abschnitten bereits beschrieben. Wird eine neue Edition erstellt, werden alle Objekte, die mit der Eltern-Edition assoziiert sind, von der Kind-Edition übernommen. Aber Vorsicht, die Objekte sind nicht mit der Kind-Edition assoziiert. Sie wurden nur für die Kind-Edition sichtbar gemacht. Beispielsweise ist basierend auf der in Abbildung 1 dargestellten Hierarchie, eine in der REL1 Edition erstellte Prozedur ebenfalls in den REL2 und REL3 Editionen sichtbar (vorausgesetzt, dass sie nicht entfernt wurde).

Für die Redefinition eines editierten Objekts muss die neue Edition ausgewählt und anschliessend das editierte Objekt entweder geändert, hinzugefügt, entfernt oder im Fall eines «Stored Programs» durch reguläre DDL Kommandos neu kompiliert werden. Beispielsweise wird durch die folgenden SQL Kommandos die vorher in der REL1 Edition erstellte Prozedur in der REL2 Edition geändert und schliesslich in der REL3 Edition gelöscht.

```
SQL> ALTER SESSION SET EDITION = rel2;

SQL> CREATE OR REPLACE PROCEDURE hello IS
2 BEGIN
3   dbms_output.put_line('Hello from REL2');
4 END;
5 /

SQL> ALTER SESSION SET EDITION = rel3;

SQL> DROP PROCEDURE hello;
```

Jetzt, da die Prozedur redefiniert wurde, sind zwei unabhängige Objekte im Data Dictionary gespeichert. Infolgedessen kann nun ausgewählt werden, welches von beiden Objekten man verwenden möchte. Für diese Zuordnung muss die entsprechende Edition, wie im folgenden Beispiel dargestellt, ausgewählt werden. Bitte beachten Sie, dass USER\_OBJECTS (das Gleiche gilt für DBA\_OBJECTS und ALL\_OBJECTS) die Version anzeigt, die mit der Session Edition assoziiert ist.

```
SQL> ALTER SESSION SET EDITION = rel1;

SQL> SELECT edition_name
2 FROM user_objects
3 WHERE object_name = 'HELLO';

EDITION_NAME
-----
REL1

SQL> EXECUTE hello
Hello from REL1

SQL> ALTER SESSION SET EDITION = rel2;

SQL> SELECT edition_name
2 FROM user_objects
3 WHERE object_name = 'HELLO';

EDITION_NAME
-----
REL2

SQL> EXECUTE hello
Hello from REL2

SQL> ALTER SESSION SET EDITION = rel3;

SQL> SELECT edition_name
2 FROM user_objects
3 WHERE object_name = 'HELLO';

no rows selected
```

ANZEIGE

DIGICOMP

ORACLE  
UNIVERSITY

# Oracle-Seminare für Profis

[www.digicomp.ch/oracle](http://www.digicomp.ch/oracle)

Wenn Sie alle Version, die im Data Dictionary gespeichert sind, anzeigen wollen, müssen entweder die Views USER\_OBJECTS\_AE, DBA\_OBJECTS\_AE oder die View ALL\_OBJECTS\_AE selektiert werden. Bitte beachten Sie im folgenden Beispiel, dass das mit REL3 assoziierte Objekt weiter verfügbar ist, aber als nicht existent (non-existent) markiert ist.

```
SQL> SELECT edition_name, object_type
2 FROM user_objects_ae
3 WHERE object_name = 'HELLO';
```

EDITION_NAME	OBJECT_TYPE
REL1	PROCEDURE
REL2	PROCEDURE
REL3	NON-EXISTENT

Zusammenfassend sind für das Redefinieren editionierter Objekte folgende typische Schritte auszuführen:

- Erstellen der neuen Edition und das Setzen einer Session Edition.
- Modifizieren der editionierten Objekte und das Sicherstellen der Gültigkeit aller Objekte.
- Prüfen, ob die Applikation wie erwartet mit der neuen Edition arbeitet.
- Dauerhaft auf die neue Edition wechseln.

Die bisher beschriebenen Techniken werden benutzt, um Objekte eines editionierbaren Objekttyps zu redefinieren. Das Redefinieren von Tabellen, diese sind nicht editionierbar, erfordert die Einführung eines anderen Konzeptes, dem so genannten View Editionieren.

## Views Editionieren

Tabellen sind nicht editionierbar, und daher kann man für die Strukturdefinition den Vorteil von Editionen nicht nutzen. Stattdessen kann jede Tabelle mit einer sogenannten «Editioning View» versehen werden und gleichzeitig können in der Applikation alle<sup>1</sup> Tabellenreferenzen durch Referenzen auf «Editioning Views» ersetzt werden. Vereinfacht gesagt kann eine (transparente) Ebene zwischen der Applikation und dem physikalischen Datenbankdesign erstellt werden.

Eine «Editioning View» ist eine normale View mit einigen speziellen Eigenschaften:

- Der Zweck einer «Editioning View» dient nur dem Selektieren von Daten und optional Aliase für eine bestimmte Untermenge von Spalten innerhalb einer Tabelle bereitzustellen. Operationen wie Joins, Subqueries, Set Operatoren, Aggregations- und hierarchische Abfragen werden nicht unterstützt.
- «Editioning Views» unterstützen DML Trigger. Demzufolge sollten die Trigger, die normalerweise auf Tabellenlevel spezifiziert werden, auf View Level präzisiert werden. Solche Trigger feuern nur, wenn DML Operationen auf «Editioning Views» abgesetzt werden. Mit anderen Worten, sie werden nicht ausgeführt, wenn DML Operationen auf die Tabelle selber abzielen, auf der wiederum die «Editioning View» basiert.
- Eine «Editioning View» unterstützt nicht INSTEAD OF Trigger.

Für die Erstellung einer «Editioning View» muss das Schlüsselwort EDITIONING dem CREATE VIEW Kommando hinzugefügt werden. Das folgende SQL Kommando zeigt beispielhaft, wie eine Tabelle durch eine «Editioning View» die Zuordnung der physikalischen Spaltennamen zu logischen Namen vornimmt. Zusätzlich ist ein DML Trigger mit der View selber assoziiert (beachten Sie, dass der gleiche Trigger bei einer normalen View nicht erstellt werden könnte).

```
SQL> DESCRIBE persons
Name                               Null?    Type
-----
ID                                  NOT NULL NUMBER(10)
FIRST_NAME                          NOT NULL VARCHAR2(100)
LAST_NAME                           NOT NULL VARCHAR2(100)
EMAIL                                NOT NULL VARCHAR2(100)

SQL> RENAME persons TO persons_tab;

SQL> CREATE EDITIONING VIEW persons AS
2 SELECT id, first_name AS firstname, last_name AS lastname, email
3 FROM persons_tab;

SQL> CREATE TRIGGER persons_bi_trg
2 BEFORE INSERT ON persons FOR EACH ROW
3 BEGIN
4     :new.id := persons_seq.nextval;
5 END;
6 /
```

<sup>1</sup> Es wird wahrscheinlich nicht möglich sein, alle Referenzen zu ersetzen. Z.B. kann ein TRUNCATE Befehl nicht auf eine View ausgeführt werden. So gesehen ist dies nicht unbedingt ein Problem. In Wirklichkeit arbeitet ein Truncate Befehl unabhängig von einer Tabellenstruktur.

Während eines Upgrades können die «Editioning Views» gegebenenfalls in den read-only oder read-write Modus gesetzt werden. Für die Vereinfachung eines Upgrades sollten alle auf Tabellen basierende «Editioning Views» in den read-only Modus gesetzt werden. Jedoch kann die Applikation die Daten von diesen Tabellen nur lesen und nicht ändern. Wenn der read-only Modus nicht in Frage kommt, muss überlegt werden, ob es eventuell notwendig ist sogenannte Crossedition Triggers einzusetzen.

## Crossedition Trigger

Ein Crossedition Trigger wird verwendet, wenn zwei Anforderungen zu treffen:

1. Die Struktur muss während eines Upgrades redefiniert werden.
2. Die «Editioning View», die über einer zu redefinierenden Tabelle liegt, kann nicht in den read-only Modus gesetzt werden.

Nehmen wir mal an, dass eine Applikation den vollständigen Zugriff auf eine Tabelle, die eine Spalte beinhaltet, die wiederum Email Adressen speichert (wie die eine aus dem Beispiel im vorherigen Abschnitt), benötigt. Unsere Aufgabe ist nun, diese Spalte in zwei Spalten aufzuteilen: Eine Spalte beinhaltet den Empfängernamen und die andere Spalte beinhaltet den Domainnamen. Das Hinzufügen der beiden Spalten stellt in den meisten Fällen<sup>2</sup> für diese Tabelle kein Problem dar. Zum Beispiel kann man dafür das folgende SQL Kommando ausführen.

```
SQL> ALTER TABLE persons_tab ADD (
2   email_recipient VARCHAR2(100),
3   email_domain VARCHAR2(100)
4 );
```

Das Redefinieren dieser Tabelle ist ein guter Anfang. Aber was passiert mit den darin gespeicherten Daten? Grundsätzlich müssen für das korrekte Ausführen der Redefinition zwei Anforderungen erfüllt sein.

1. Es müssen die Daten, die während des Upgrades durch die Applikation geändert werden, in der neuen Struktur gespeichert werden.
2. Es müssen die Daten, die bereits in der zu redefinierenden Tabelle gespeichert sind, ebenfalls in die neue Struktur konvertiert werden.

Für die Erfüllung der ersten Anforderung kann man in der neuen Edition einen Forward Crossedition Trigger erstellen (man sollte niemals die alte Edition ändern). Das Ziel eines solchen Triggers ist, eine Zeile von der alten Struktur in die neue Struktur zu überführen. In dem hier diskutierten Beispiel erfüllt das folgende SQL Kommando diese Anforderung.

```
SQL> CREATE TRIGGER persons_fc_trg
2   BEFORE INSERT OR UPDATE ON persons_tab FOR EACH ROW
3   FORWARD CROSSEDITION
4   DISABLE
5   BEGIN
6   :new.email_recipient :=
7   regexp_substr(:new.email, '(.*)@', 1, 1, NULL, 1);
8   :new.email_domain :=
9   regexp_substr(:new.email, '@(.*)', 1, 1, NULL, 1);
10  END;
11  /
```

Beachten Sie, dass selbst wenn der Trigger in der neuen Edition erstellt wurde, dieser erst feuert, wenn ein DML Kommando durch eine Session aus einer Vorgänger-Edition, in der der Trigger erstellt wurde, ausgeführt wird. Aus diesem Grund muss durch die FORWARD CROSSEDITION Klausel explizit definiert werden, dass es sich um einen Forward Crossedition Trigger handelt. Beachten Sie bitte auch, dass dieser Trigger im Status «Disabled» erstellt werden sollte. Das ist insofern wichtig, dass ein ungültiger Trigger nicht die Verfügbarkeit der Applikation stört. Infolgedessen sollten Sie ihn nur aktivieren, wenn Sie sicher wissen, dass er gültig ist. Für die Erfüllung der zweiten Anforderung muss die Transformation auf die bereits gespeicherten Daten angewendet werden. Für dieses Szenario gibt es mehrere Möglichkeiten. Das einfachste aber nicht das Beste aus Performance-sicht ist, den Trigger für jede der bereits in der redefinierten Tabelle gespeicherten Zeile auszulösen. Das DBMS\_SQL Package stellt, wie im folgenden

<sup>2</sup> Einen Table Lock auf eine stark frequentierte Tabelle zu bekommen ist vielleicht problematisch. Demzufolge muss eventuell ein Timeout mit dem Initialisierungsparameter DDL\_LOCK\_TIMEOUT gesetzt werden. Für die Redefinition einer Tabelle ist vielleicht auch das DBMS\_REDEFINITION Package hilfreich.

PL/SQL Block dargestellt wird, eine neue Version der PARSE Funktion bereit. Beachten Sie, wie ein Dummy UPDATE Kommando dafür benutzt wird, den durch den APPLY\_CROSSEDITION\_TRIGGER Parameter spezifizierten Trigger auszulösen.

```
SQL> DECLARE
2   c INTEGER;
3   r INTEGER;
4 BEGIN
5   c := dbms_sql.open_cursor;
6   dbms_sql.parse(
7     c => c,
8     statement => 'UPDATE persons SET email_domain = email_domain',
9     language_flag => dbms_sql.native,
10    apply_crossedition_trigger => 'persons_fc_trg'
11  );
12  r := dbms_sql.execute(c);
13  dbms_sql.close_cursor(c);
14  COMMIT;
15 END;
16 /
```

Es ist essentiell, zur Vermeidung eines inkonsistenten Zustands zu verstehen, dass die erforderlichen Operationen, die diese beiden Anforderungen erfüllen sollen, in der folgenden Reihenfolge ausgeführt werden müssen:

- Aktivieren des Forward Edition Triggers.

```
SQL> ALTER TRIGGER persons_fc_trg ENABLE;
```

- Es muss auf jede Transaktion der redefinierten Tabelle gewartet werden. Entweder muss diese bestätigt oder zurückgesetzt sein. Zu diesem Zweck wurde eine neue Funktion, die sogenannte WAIT\_ON\_PENDING\_DML Funktion, innerhalb des DBMS\_UTILITY Packages hinzugefügt. Wenn dieser Schritt nicht ausgeführt wird, können die Transaktionen, die während des Aktivierens des Forward Edition Triggers geöffnet waren, in einen inkonsistenten Zustand gelangen.

```
SQL> DECLARE
2   r BOOLEAN;
3   scn INTEGER;
4 BEGIN
5   r := dbms_utility.wait_on_pending_dml(
6     tables => 'CHA.PERSONS_TAB',
7     timeout => NULL,
8     scn => scn
9   );
10 END;
11 /
```

- Start des PL/SQL Blocks, der den Forward Edition Trigger für jede Zeile auslöst.

Sobald diese Operationen abgeschlossen sind, müssen vielleicht auch einige neue Constraints erstellt werden. Beispielsweise sind in diesem Fall die neuen Spalten als NOT NULL gesetzt.

```
SQL> ALTER TABLE persons_tab MODIFY (
2   email_recipient NOT NULL,
3   email_domain NOT NULL
4 );
```

Das bisher aufgezeigte Beispiel zeigt, was getan werden kann, um die Änderungen, die von einer mit der alten Edition arbeitenden Applikation auf die von der neuen Edition genutzten Datenstrukturen zu propagieren. Eine weitere Situation, die berücksichtigt werden muss, ist was passiert, wenn aktuell beide, die alte und die neue Edition, gleichzeitig genutzt werden? Dies tritt auf, wenn ein sogenannter «Hot Rollover» auf die neue Edition erforderlich ist. Für die Erfüllung dieser neuen Anforderung muss ein anderer Typ von Trigger verwendet werden, der sogenannte Reverse Crossedition Trigger. Einfach gesprochen ist der Zweck eines solchen Triggers, das Gegenteil eines Forward Crossedition Triggers zu erreichen. Mit anderen Worten hat dieser die Funktion, eine Zeile der neuen in die alte Struktur zu überführen. Das folgende SQL Kommando zeigt ein Beispiel hierfür.

```
SQL> CREATE TRIGGER persons_rc_trg
2   BEFORE INSERT OR UPDATE ON persons_tab FOR EACH ROW
3   REVERSE CROSSEDITION
4   DISABLE
5   BEGIN
6     :new.email := :new.email_recipient || ',' || :new.email_domain;
7   END;
8   /
```

In diesem Fall muss der Trigger ebenfalls in der neuen Edition erstellt werden (auch hier gilt, dass man niemals die alte Edition ändern soll). Zu beachten ist ausserdem, dass ein Reverse Crossedition Trigger nur ausgelöst wird, wenn ein DML Kommando durch eine Session ausgeführt wird, in welcher der Trigger erstellt wurde (oder ein Nachfolger davon). Aus diesem Grund muss explizit über die REVERSE CROSSEDITION Klausel definiert werden, dass es sich um einen Reverse Crossedition Trigger handelt.

Die hier gezeigten Beispiele zu Crossedition Triggern sind sehr einfach. Wenn die Komplexität der Redefinition steigt, z.B. wenn SQL Kommandos innerhalb des Triggers ausgeführt werden müssen, sind verschiedene andere Fragen zu berücksichtigen. Da diese kurze Einführung nicht alle diese Details abdecken kann, wird empfohlen für zusätzliche Informationen die am Ende des Papiers gelisteten Dokumente zu referenzieren.

## Entfernen von Editionen

Um eine Edition zu entfernen, muss das DROP EDITION Kommando verwendet werden. Für die Ausführung dieses Kommandos wird das DROP ANY EDITION Privileg benötigt. Zusätzlich müssen folgende Bedingungen erfüllt sein:

- Die Edition darf nicht die letzte Datenbank-Edition sein.
- Die Edition ist nicht die Basis-Datenbank Edition.
- Die Edition wird nicht als Session Edition verwendet.
- Die Edition ist entweder Root oder Kind der Hierarchie.
- Wenn die Edition Kind der Hierarchie ist, und sie noch assoziierte Objekte besitzt, dann muss die CASCADE Option spezifiziert werden.
- Wenn die Edition Root der Hierarchie ist, darf die nachfolgende Edition keine Objekte der Root Edition übernehmen. Mit anderen Worten, müssen alle übernommenen Objekte durch die nachfolgende Edition redefiniert werden.

Das Entfernen einer nicht mehr genutzten Edition ist optional. Statt eine nicht mehr genutzte Edition zu entfernen, kann sie auch über das Widerrufen des USE ON EDITION Privilegs für alle Benutzer ausser Dienst gestellt werden.

## Einschränkungen

In Oracle Database 11g Release 2 gibt es zwei wichtige Implementierungseinschränkungen, die sich auf die Hierarchien, die mit Editionen erstellt wurden, beziehen:

- Jede Edition kann höchstens eine nachfolgende Edition haben. Dem-

zufolge ist eine Hierarchie, wie auf der linken Seite in Abbildung 2 gezeigt, nicht erlaubt.

- Jede Datenbank hat nur eine, und wirklich nur eine, Root Edition. Demzufolge kann eine Datenbank keine zwei Hierarchien, wie auf der rechten Seite in Abbildung 2 gezeigt, beinhalten.

Während die erstgenannte Einschränkung nicht ausschlaggebend ist, kann die letztere kritisch sein. In der Tat ist es nicht erlaubt unabhängige Editionen für jede Applikation, die die gleiche Datenbank für Konsolidierungszwecke nutzt, zu verwenden. Für das Beispiel in Abbildung 2 würde die Idee sein, eine Hierarchie für die CRM Applikation zu nutzen und eine andere für die HR Applikation.



Abb 2: Beispiele für nicht zulässige Hierarchien

Die übrigen wichtigen Einschränkungen beziehen sich auf Indizes und Constraints. Da beide nur auf einer Tabelle erstellt werden können, gibt es keine Möglichkeit den Vorteil von Editionen für diese zu nutzen. Im Falle von neuen Indizes kann ihre Auswirkung, die sie unsichtbar («Invisible») gesetzt werden, limitiert werden. Mit Constraints hingegen ist es eventuell nicht möglich eine Definition zu verwenden, die korrekt mit mehr als einer Edition funktioniert. Die einzige generelle Ausnahme bezieht sich auf Check Constraints. Tatsächlich kann für diese ihre Logik auf der Session Edition basieren.

## Schlussfolgerung

Obwohl es immer noch einige Einschränkungen gibt, offeriert Edition-Based Redefinition komplett neue Features, die Online Upgrades möglich machen. Nichtsdestotrotz darf die Komplexität solcher Upgrades nicht unterschätzt werden. ■

## Contact

Trivadis AG

Christian Antognini

E-Mail:

christian.antognini@trivadis.com

## Referenzen

- Oracle White Paper, Edition-Based Redefinition in Oracle Database 11g Release 2
- Oracle Database 11g Release 2 documentation, Advanced Application Developer's Guide
- Oracle Database 11g Release 2 documentation, SQL Language Reference
- Oracle Database 11g Release 2 documentation, PL/SQL Packages and Types Reference

ANZEIGE

# Wir bleiben für Sie wach

**Basic**



**Advanced**



**Enterprise**



### Maintenance & Support, das SLA von Edorex

Wenn es darum geht, Ihre Datenbanken im Auge zu behalten, können Sie auf uns Vertrauen. Installation, Betrieb und Pflege liegen uns am Herzen, buchen Sie das passende Angebot unter [www.edorex.ch](http://www.edorex.ch)

**edorex**  
creating solutions

Edorex Informatik AG | Unterdorfstrasse 5 | 3072 Ostermundigen | 031 930 10 70 | [info@edorex.ch](mailto:info@edorex.ch) | [www.edorex.ch](http://www.edorex.ch)